FREE Studio Plus

Operating Guide

Original instructions

9MA10256.04
12/2023

eliwell™
by Schneider Electric

Legal Information

The information provided in this document contains general descriptions, technical characteristics and/or recommendations related to products/solutions.

This document is not intended as a substitute for a detailed study or operational and site-specific development or schematic plan. It is not to be used for determining suitability or reliability of the products/solutions for specific user applications. It is the duty of any such user to perform or have any professional expert of its choice (integrator, specifier or the like) perform the appropriate and comprehensive risk analysis, evaluation and testing of the products/solutions with respect to the relevant specific application or use thereof.

The Schneider Electric brand and any trademarks of Schneider Electric SE and its subsidiaries referred to in this document are the property of Schneider Electric SE or its subsidiaries. All other brands may be trademarks of their respective owner.

Schneider Electric does not grant any right or license for commercial use of the document or its content, except for a non-exclusive and personal license to consult it on an "as is" basis.

Schneider Electric reserves the right to make changes or updates with respect to or in the content of this document or the format thereof, at any time without notice.

To the extent permitted by applicable law, no responsibility or liability is assumed by Schneider Electric and its subsidiaries for any errors or omissions in the informational content of this document, as well as any non-intended use or misuse of the content thereof.

Table of Contents

Safety Information

Important Information

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 or Eliwell 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

Document Scope

This document describes how to use the FREE Studio Plus software to configure, program, and commission applications for supported logic controllers.

Validity Note

The information in this document is applicable **only** for FREE Studio Plus products.

This document has been updated for the release of FREE Studio Plus V1.6.0.

Related Documents

Title of documentation	Reference number
FREE Smart Logic Controller	9MA10251
FREE Advance Logic Controller	9MA10291
EWCM 9000 PRO (HF)	CRCTA-00
FREE Advance 7/18 IO - Instruction Sheet	9IS54609
FREE Advance 28/42 IO - Instruction Sheet	9IS54473
FREE Advance 28/42 IO isolated - Instruction Sheet	9IS54655
FREE EVE6000/EVE10200 Expansion Module 12/28 IO - Instruction Sheet	9IS54578
FREE AVP1000 Display Color Touchscreen - Instruction Sheet	9IS54479
FREE AVP1000 Display Color Touchscreen Flush Mounting - Instruction Sheet	9IS54608
FREE AVK1000000500 Monochrome Display - Instruction Sheet	9IS54800
FREE EVS Plugin - Instruction Sheet	9IS54405
EWCM 9000 PRO (HF) - Instruction Sheet	9IS54760
FREE Optima - Instruction Sheet	9IS5487300
FREE Optima Logic Controller - Hardware Guide	9MA10312

You can download these technical publications, the present document and other technical information from our website www.eliwell.com.

Product Related Information

⚠WARNING

LOSS OF CONTROL

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

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

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

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

- Only use software approved by Eliwell 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 the information contained herein, 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
IEC 61131-2:2007	Programmable controllers, part 2: Equipment requirements and tests.
ISO 13849-1:2023	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

Standard	Description
ISO 14119:2013	Safety of machinery - Interlocking devices associated with guards - Principles for design and selection
ISO 13850:2015	Safety of machinery - Emergency stop - Principles for design
IEC 62061:2021	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 61784-3:2021	Industrial communication networks - Profiles - Part 3: Functional safety fieldbuses - General rules and profile definitions.
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.

Getting Started with FREE Studio Plus

What's in This Part

Getting Started with FREE Studio Plus

What's in This Chapter

Introduction to FREE Studio Plus

Overview

The FREE Studio Plus software makes it possible to create and customize IEC 61131-3 programs for various types of applications. You can download FREE Studio Plus from Eliwell web site download center. It is intended for applications in HVAC&R.

Features

FREE Studio Plus 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, FREE Studio Plus includes:

- Textual source code editors programming languages:
  - Instruction List (IL) , page 271
  - Structured Text (ST), page 280
- Graphical source code editors programming languages:
  - Ladder Diagram (LD), page 278
  - Function Block Diagram (FBD), page 274
  - Sequential Function Chart (SFC), page 289
- 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.

FREE Studio Plus Software Component

FREE Studio Plus 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

FREE Studio Plus is divided into four tabs:

- **Configuration**
- **Programming**
- **Display**
- **Commissioning**

For more information about the tabs of FREE Studio Plus, refer to Tabs, page 25.

System Requirements and Supported Devices

System Requirements

Overview

FREE Studio Plus can be installed on a personal computer having the following characteristics.

Operative System:

- Windows 10 64 Bit
- Windows 11 64 Bit

Hardware requirements:

- Processor Pentium 1.6 GHz or later
- RAM Memory 1 GB; 2 GB preferred
- Hard Disk 1 GB of free space
- Peripherals Mouse or compatible pointing device
- Peripherals USB interface
- Web access: Web registration requires Internet access

FREE Studio Plus requires Administrator rights to be installed.

For further information, contact your Eliwell support center.

Supported Devices

Logic Controllers

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

Reference	Description	Hardware Guide
FREE Optima	FREE Optima Logic Controller	*FREE Optima Logic Controller – Hardware Guide*
AV•••••••5•500	FREE Advance AV••••••5•500 Logic Controller	*FREE Advance Logic Controller - Hardware Guide*
AV••••••6•500	FREE Advance AV••••••6•500 Logic Controller	
SMP•••• / SMD•••• /SMC••••	FREE Smart Logic Controller	*FREE Smart Logic Controller - Hardware Guide*
EV•7500	FREE Evolution Logic Controller	*FREE Evolution Logic Controller - Hardware Guide*
EWCM 9000 PRO (HF)	EWCM 9000 PRO (HF) Logic Controller	*EWCM 9000 PRO - User Guide*

Expansion Modules

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

Reference	Description	Hardware Guide
SME4500	FREE Smart Expansion SME4500	*FREE Smart Logic Controller - Hardware Guide*
SME3200	FREE Smart Expansion SME3200	

Reference	Description	Hardware Guide
SME5500	FREE Smart Expansion SME5500	
EVE4200	FREE Expansion EVE4200	*FREE Evolution Logic Controller - Hardware Guide*
EVE7500	FREE Expansion EVE7500	
EVE6000000500	FREE Expansion EVE6000	*FREE Advance Logic Controller- Hardware Guide*
EVE1020000500	FREE Expansion EVE10200	
EP4000000B0	EWCM Expansion EP4000	*EWCM 9000 - Hardware Guide*

Electronic Expansion Valve Driver

For more information about the electric expansion valve drivers and their compatibility, refer to the following guide:

Reference	Description	Hardware Guide
EVEVD••000500	FREE EVEVD Electronic Expansion Valve Driver	*FREE EVEVD Electronic Expansion Valve Driver - Hardware Guide*

Displays

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

Reference	Description	Hardware Guide
OTDLED	FREE Optima Remote Display LED	*FREE Optima Logic Controller – Hardware Guide*
AVP100G0P0500	FREE_AVP Color Touchscreen remote display flush mounting white	*FREE Advance Logic Controller - Hardware Guide*
AVP100W0P0500	FREE_AVP Color Touchscreen remote display flush mounting gray	
AVP11000W0500	FREE_AVP Color Touchscreen remote display vertical mounting with built-in temperature sensor	
AVP12000W0500	FREE_AVP Color Touchscreen remote display vertical mounting with built-in temperature and humidity sensors	
AVP13000W0500	FREE_AVP Color Touchscreen remote display vertical mounting with built-in temperature, humidity, and presence (PIR) sensors	
AVK1000000500	FREE AVK1000 Monochrome Display Graphic	
SKP10	FREE Smart Display SKP10	*FREE Smart Logic Controller - Hardware Guide*
SKP22	FREE Smart Terminal SKP22	
SKW22	FREE Smart Wall thermostat without backlight SKW22	
SKW22L	FREE Smart Wall thermostat with backlight SKW22L	
EVK1000	FREE Evolution Terminal EVK1000	*FREE Evolution Logic Controller - Hardware Guide*

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.

FREE Smart PC Connection

FREE Smart 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.

FREE Advance PC Connection

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

- Type Mini-B USB (DEVICE). Used to connect AV••••••6•500 / AV•••••••5•500 to a PC via Mini-B/A USB cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.
- Type micro-B USB (DEVICE). Used to connect AVP1•0•••0500 to a PC via micro-B/A USB cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.

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 AV••••••6•500 / AV•••••••5•500 can also be supplied through the USB cable with limited functionalities related to debugging, commissioning, downloading and uploading with FREE Studio Plus.

For more details, refer to *FREE Advance Logic Controller- Hardware Guide*.

EWCM 9000 PRO PC Connection

EWCM 9000 PRO (HF) can be connected to a PC through the USB port and a USB cable:

- Type Mini-B USB (DEVICE). Used to connect EWCM 9000 PRO (HF) to a PC via Mini-B/A USB cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.
- Type micro-B USB (DEVICE). Used to connect EWCM 9000 PRO (HF) to a PC via micro-B/A USB cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.

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 EWCM 9000 PRO (HF) can also be supplied through the USB cable with limited functionalities related to debugging, commissioning, downloading and uploading with FREE Studio Plus.

For more details, refer to *EWCM 9000 PRO - User Guide EWCM 9000 PRO - User Guide*.

FREE Evolution PC Connection

FREE Evolution 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 FREE Evolution to a PC via Mini-B/A USB cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.

FREE Optima PC Connection

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

USB (Type C). Used to connect FREE Optima to a PC via USB (Type A) / USB (Type C) cable for debugging, commissioning, downloading, and uploading with FREE Studio Plus.

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

For more details, refer to *FREE Optima Logic Controller – Hardware Guide*.

FREE Evolution/FREE Advance Programming Converters

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

Alternatively, if there is an RS-232 serial port, FREE Evolution/FREE Advance can be connected to the PC using an RS-485/RS-232 adapter.

FREE Evolution/FREE Advance Communication Modules

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

NOTE: Not available with FREE Panel EVP.

For more details, refer to communication modules description, page 87.

FREE Studio Plus Basics

Creating Projects with FREE Studio Plus

Overview

FREE Studio Plus is a graphical programming tool designed to configure, develop, and commission programs for logic controllers.

FREE Studio Plus Terminology

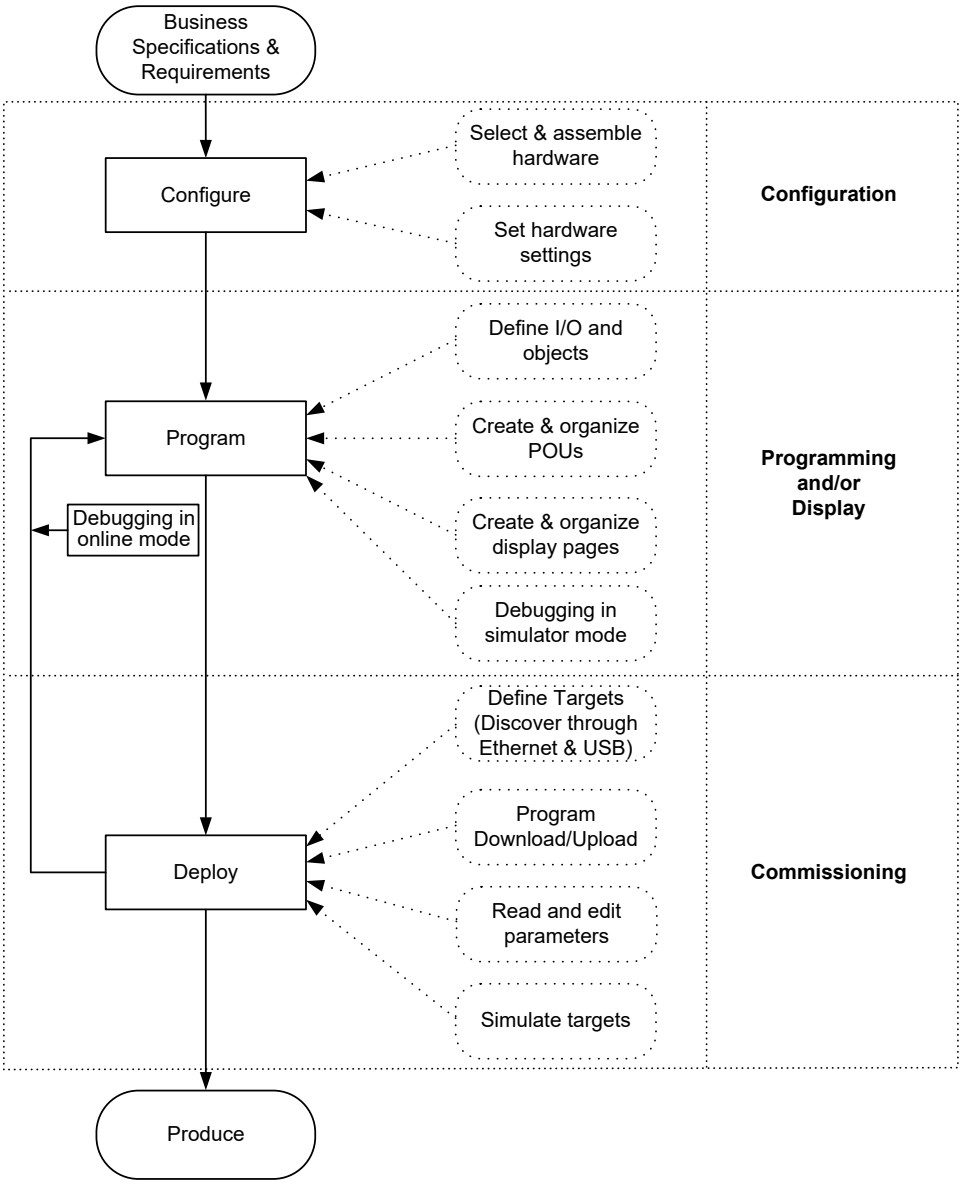
FREE Studio Plus uses the following terms:

- **Project**: A FREE Studio Plus 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 FREE Studio Plus

Introduction

The following diagram presents the typical stages of developing a project in FREE Studio Plus:



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 FREE Studio Plus.

FREE Studio Plus can operate in the following modes:

- Offline mode
- Online mode
- Simulator mode

Offline Mode

FREE Studio Plus operates in offline mode when no physical connection to a logic controller has been established.

In offline mode, you configure FREE Studio Plus to match the hardware components you are targeting, then develop your application.

Online Mode

FREE Studio Plus 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. FREE Studio Plus 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, page 212 and Triggers, page 215.

Simulator Mode

FREE Studio Plus 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 FREE Studio Plus simulates a connection to a logic controller and the expansion modules to run and test the program.

For more information, refer to Simulation, page 186.

Software Interface

What's in This Chapter

Overview

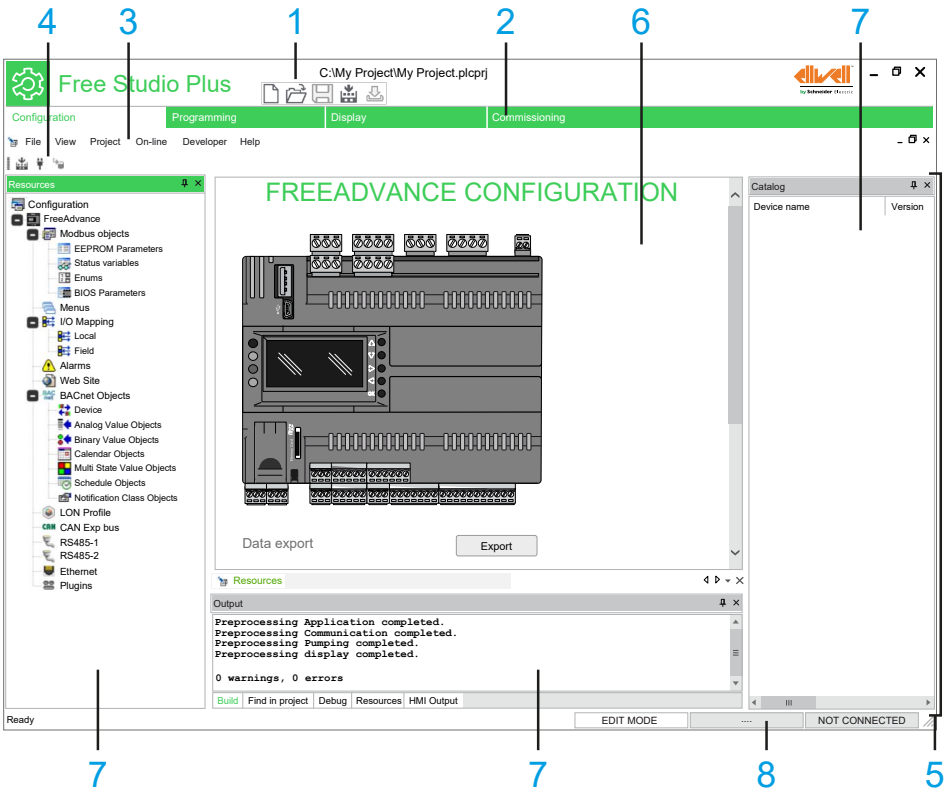
Welcome Window

The **Welcome** window is always displayed when you launch FREE Studio Plus. Use this window to select and open an existing project or create a project, page 41.

Main Window

The FREE Studio Plus main window provides access to menus and commands, windows, and toolbars.

The illustration presents the FREE Studio Plus main window:



Item	Description
1	Project toolbar, page 25
2	Tabs, page 25
3	Menu bar, page 26
4	Toolbars, page 34
5	Workspace, page 35
6	Editor window

Item	Description
7	Tool windows, page 34
8	Status bar, page 35

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, page 41.
	**Open project**: opens a project saved on your computer. For more information, refer to Open an Existing Project, page 45.
	**Save project**: saves modifications to the current project. For more information, refer to Save a Project, page 44.
	**Build all**: compiles your project. For more information, refer to Build All, page 48.
	**Download all**: downloads onto the target. For more information, refer to Download and Upload Applications, page 48.

Tabs

FREE Studio Plus consists of four development environments for programming controllers:

- **Configuration**, page 51

  For creating networks. It is the entry point of the software.
- **Programming**, page 91

  For creating and managing libraries, controller applications and diagnostics.

  It includes a **Simulation mode**, page 186, dedicated to software developers, for running and testing controller applications without a logic controller and expansion modules physically connected.
- **Display**, page 318

  For creating the graphical interface on built-in displays and remote displays.
- **Commissioning**, page 405

  For downloading the project to the connected logic controller device and modifying device parameters from a serial port.

Menu Bars

Overview

FREE Studio Plus has several different menus. Most of them depend on the context. They may or may not appear depending on the tab or the action you are performing.

Tab	Menu
All	File, page 28
	View, page 33
	Help, page 28
Partially common	Edit, page 27
	Project, page 30
	On-line, page 29
	Options, page 29
	Window, page 34
Configuration	Developer, page 27
Programming	Debug, page 26
	Scheme, page 31
	Variables, page 32
	Tools, page 32
Display	Page, page 29
Commissioning	Parameters, page 30
	Recipes, page 31
	Target, page 32

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).

Developer Menu

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

Command	Icon	Key	Description
Data export	-	-	Export data to a CSV file, page 46.
Import EDS	-	-	Import EDS (Electronic Data Sheet) file, page 73.
Run Modbus custom Editor	-	-	Run Modbus custom editor, page 82.
Build Web site	-	-	Generate a website to manage the target device remotely, page 66.
Generate EDE files	-	-	Generates the Engineering Data Exchange (EDE) files in CSV format (in order to be used by SCADA BACnet supervisor).
Build Televis Driver	-	-	Televis Driver Generation, page 433

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...	-	-	Allows to manage the existing bookmarks.

Command	Icon	Key	Description
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 user to move to a specific line in the source code editor.
Find…		**Ctrl+F**	Asks user 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 user to draw logical wires to connect pins.
Watch mode		-	Editing mode which allows user to add variables to any debugging tool.

File Menu

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

Command	Icon	Key	Description
New project		-	Creates a new project, page 41.
Open project		Ctrl+O	Opens an existing project, page 45.
Save project		Ctrl+S	Saves the current open project., page 44
Save project as…	-	-	Saves the current open project specifying new name, location and extension, page 44.
Close project	-	-	Closes the open project, page 46.
Options…	-	-	Opens the Program options dialog box, page 39.
Print…		Ctrl+P	Prints the document of the currently active window, page 43.
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 FREE Studio Plus.

Help Menu

This menu gives access to features allowing you to read documentation about hardware or some specific FREE Studio Plus 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.
About...	-	-	Credits and version information.

On-line Menu

Command	Icon	Key	Description
Set up communication...	-	-	Lets you set the properties of the connection to the target, page 176.
Connect		-	Starts the communication with the device, page 181.
Download code		**F5**	Download the configuration and PLC application of the project to the device, page 183.
Download options…	-	-	Allows to set the properties of the source code downloaded to the target.
Force image upload	-	-	If the target device is connected, allows to upload the image file.
Force debug symbols upload	-	-	If the target device is connected, allows to 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.

Options Menu

This menu gives access to features allowing to manage the commissioning options:

Command	Icon	Key	Description
Logging options…	-	-	Select what types of information are recorded in the event log.
Refresh interval…	-	-	Set refresh interval for monitor, graph, and auto-refresh (ms).

Page Menu

This menu gives access to features allowing to add objects on pages:

Command	Icon	Key	Description
New Static		-	Add a new static object.
New Image		-	Add a new image object.

Command	Icon	Key	Description
New Animation		-	Add a new animation object.
New Edit		-	Add a new edit box.
New Button		-	Add a new button object.
New Custom Ctrl		-	Add a new custom control.
New TextBox		-	Add a new text box.
New Progress		-	Add a new progress bar.
New Chart		-	Add a new chart object.
New Trend		-	Add a new trend object.

Parameters Menu

This menu gives access to features allowing to manage project parameters:

Command	Icon	Key	Description
Read selected	-	Ctrl+R	Read selected value.
Write selected	-	Ctrl+W	Write selected value.
Write default values	-	-	Reset the selected value to default.
Refresh page	-	-	Updates the parameters and their values in the **Resources** window.
Select all	-	Ctrl+A	Select the parameters from the **Resources** window
Read all	-	Ctrl +Shift+R	Read the values of the **Resources** window.
Write all	-	Ctrl +Shift +W	Write the values of the **Resources** window.
Write all default values	-	-	Reset the values of the **Resources** window to default.
Search parameters…	-	-	Search for a specific parameter using filters.
Connected mode	-	Ctrl+T	Simulates the connection of the target device.
Auto refresh mode	-	-	Changing parameter values in the application are written automatically to the target device.
Export to text file…	-	-	Export the parameters of the **Resources** window as a *.csv text file.
Import values from other project…	-	-	Apply parameter values from another project to the current project.

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, page 48.
Select target...	-	-	Lets you select a new target for the project, page 47.
Refresh current target	-	-	Allows to update the target file for the same version of the target.
Option	-	-	Allows to enable **Installer Pro project** and/or **Televis driver generation** features.

Recipes Menu

This menu gives access to features allowing you to manage recipes:

Command	Icon	Key	Description
Add recipe	-	-	Add a new recipe.
Import recipe	-	-	Import an existing recipe from the hard drive disk or a removable media.
Delete recipe	-	-	Delete the selected recipe.
Export recipe	-	-	Export an existing recipe to the hard drive disk or a removable media.
Set recipe values	-	-	Applying the change in the value of the selected parameter to the value of the recipe.
Delete parameters from recipe	-	-	Delete one or more selected parameters from the actual recipe folder.
Write all recipe values	-	-	Download the recipes to the target device.

Scheme Menu

Command	Icon	Key	Description
Network	-	-	Adds new networks.
New	-	-	List of new networks.
Top		-	Adds a new network at the top.
Bottom		-	Adds a new network at the bottom.
Before		-	Adds a new network at the before the current selection.
After		-	Adds a new network at the after the current selection.
Label	-	-	Adds a new network label.
Object	-	-	Adds new objects.
New	-	-	List of new objects.
Function	-	-	Adds a new function.
Function Block	-	-	Adds a new function block.
Variable		Shift+V	Adds a new variable.
Constant		Shift+K	Adds a new constant.
Return		Shift+R	Adds a new return.
Jump		Shift+J	Adds a new jump.

Operator	-	-	Adds a new operator.
Comment		Shift+M	Adds a new comment.
Instance name	-	-	Modifies the name of an instance of a function block.
Open source		-	Displays the source code of the selected object.
Auto connect		-	Connects blocks automatically.
Delete invalid connection	-	Ctrl+M	Removes invalid connections of deleted blocks.
Increment pins		Ctrl + '+'	Increments the number of input pins of some operators and embedded functions.
Decrement pins		Ctrl + '-'	Decrements the number of input pins of some operators and embedded functions.
Enable EN/ENO pins		E	Displays the enable input and output pins.
Object properties		-	Shows the properties of the selected object.

Target Menu

This menu gives access to features allowing you to set the communication settings:

Command	Icon	Key	Description
Communication settings	-	-	Select and configure the protocol to use to communicate with the target device., page 412

Tools Menu

Command	Icon	Key	Description
Custom tools	-	-	Displays up to 16 user defined commands, page 40.

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.

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, page 36.
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, page 37.
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.
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, page 38. (**Esc** to exit from this mode).

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.

Toolbars

FREE Studio Plus has several toolbars dedicated to software tabs:

Toolbars	Dedicated tabs
Main	All
Project	**Programming**, page 95
Debug	
FBD	
SFC	
LD	
Network	
Configuration	**Configuration**, page 53
HMI Page	**Display**, page 323
HMI Project	
HMI Profiles	
Commissioning	**Commissioning**, page 407

To manage toolbars, refer to Toolbars Management, page 36.

Tool Windows

FREE Studio Plus has several tool windows dedicated to software tabs:

Tool windows	Dedicated tabs
Local variables	**Configuration**, page 51
Project	**Programming**, page 91
Watch	
Properties Window	
Oscilloscope	
PLC run-time status	
Operators and blocks	
Library Tree	
Output	**Configuration**, page 51
Cross Reference	**Programming**, page 91
Resources	**Configuration**, page 51

Tool windows	Dedicated tabs
Catalog	
HMI Project	**Display**, page 318
HMI Properties	
HMI Actions	
HMI Vars and Parameters	
HMI Templates	
Commissioning	**Commissioning**, page 405
Commissioning Watch	
Commissioning Oscilloscope	

To manage tool windows, refer to Tool Windows Management, page 37.

Status Bar

Overview

The status bar is located at the bottom right of the FREE Studio Plus 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, page 211
- Connection Status, page 182
- Application Status, page 182

Software Interface 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.

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:

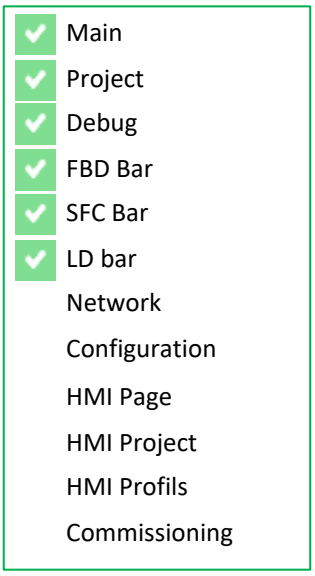
Step	Action
1	Click **File > Options…**. The **Program options** dialog box appears.
2	In **Tool windows** area, click **Reset bars positions** button. A new dialog box appears.
3	Click **OK** button.
4	In **Program options** dialog box, click **OK** button.
5	In the project toolbar, click **Save project** icon.
6	Click **File > Exit**.
7	Restart FREE Studio Plus.

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

Toolbars

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: 
3	Click the Toolbar name you want to show/hide.

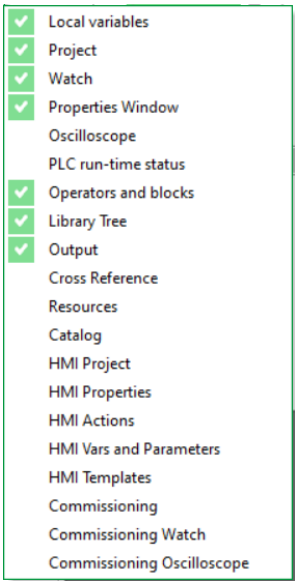
Move

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**. 
2	Select the toolbar to show or hide

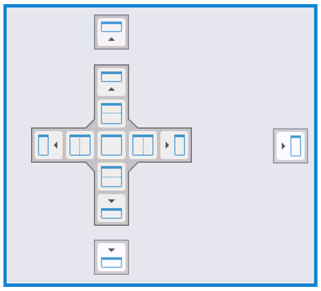
Undock Tool Windows

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, reclick the pin button when the tool window is displayed.

Window Management

Overview

FREE Studio Plus allows you to navigate between the opened source code editor windows.

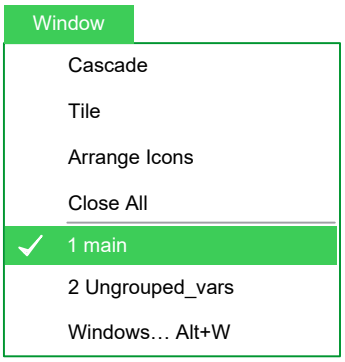
Document Tab

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

FREE Studio Plus 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**:

  It is just possible to choose, in the **Color theme** list, the **Standard** color theme.
- **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, page 112.

Managing Projects

What's in This Chapter

Create a New Project

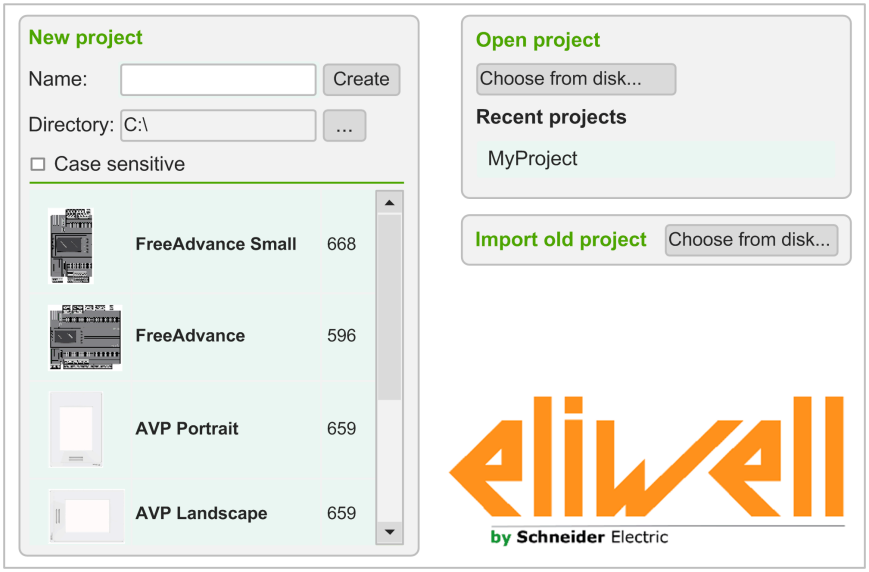
Overview

There are two ways to create a new project:

- In the Welcome page, page 41.
- In the New project window, page 43.

Welcome Page

When FREE Studio Plus starts, the **Welcome** page appears:



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, page 101.
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, page 17.
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, page 101. **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 FREE Studio. FREE Studio Plus will proceed to convert the old program. However, incompatibilities may exist between FREE Studio and FREE Studio Plus.

⚠WARNING

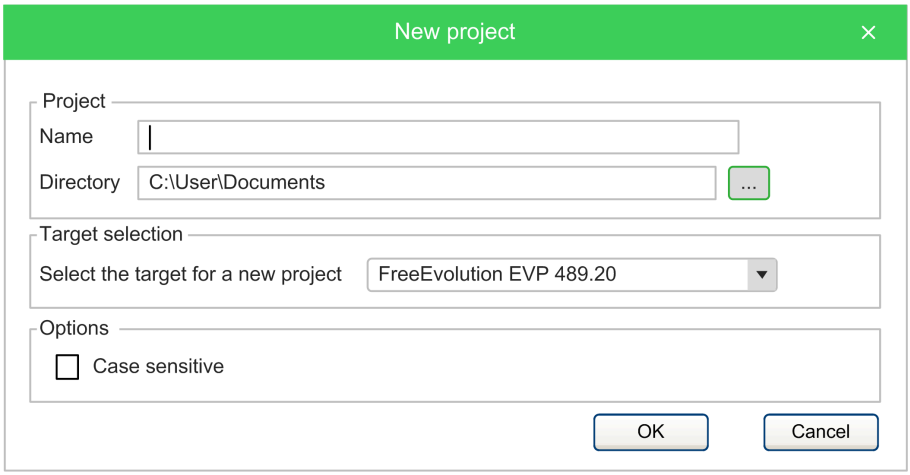
UNINTENDED EQUIPMENT OPERATION

- 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:



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, page 101.
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, page 17.
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, page 101. **NOTE:** This option is not compliant with IEC 61131-3 standard.
5	Click **OK** button.

Print a Project

Print the Project

FREE Studio Plus 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

FREE Studio Plus 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

FREE Studio Plus 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 FREE Studio Plus 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, page 46 of the project file.
5	Click **OK** button.

AutoSave

FREE Studio Plus 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, page 39.

Backup Copies

FREE Studio Plus includes a backup feature of the previous version of the project on which you are working.

When you explicitly save the project, FREE Studio Plus 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, page 39.

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.

FREE Studio Plus cannot modify elements of a project when at least one of the following conditions holds:

- FREE Studio Plus 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, FREE Studio Plus asks you to choose between saving and discarding them.

Then the **Welcome** page, page 41 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 FREE Studio Plus.

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**.

FREE Studio Plus 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 FREE Studio Plus.
- It can require a password when opening the file in FREE Studio Plus.
- Its size is reduced because it is a binary file.

Export CSV Files

Overview

FREE Studio Plus 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:

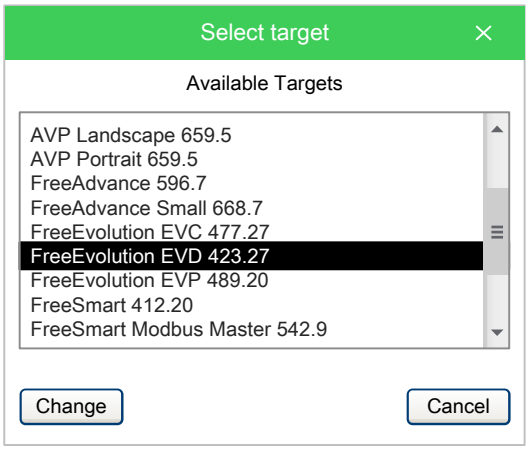
Step	Action
1	Click **Configuration** tab.
2	In the **Resources** window, click the target device.
3	In the main box, click **Export** button.
4	In **Data Export** window, select data you want to export and click **OK** button.
5	In the **Save As** dialog box, choose the name of your file and click **Save** 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, FREE Studio Plus 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, page 94.

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	USB C port	RS-485 port Modbus	Ethernet
	DMI programming cable	MFK programming stick	USB A Memory key	USB A/ USB Mini-B Cable	USB A/ USB C Cable	USB/RS-485 adapter	Ethernet Cable
FREE Smart	✓	✓	-	-	-	-	-
FREE Evolution	-	-	✓	✓	-	✓	✓(1)
A-V••••••50-500	-	-	-	✓	-	✓	✓(1)
A-V•••••••6•-500	-	-	✓	✓	-	✓	✓
FREE Optima	-	-	-	-	✓	✓	-

(1) An Ethernet communication module must be connected to the controller, except for FREE Panel EVP 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, page 176
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 Connect to a Device, page 181 and On-Line Status, page 182
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 into service until the file transfer has completed successfully.

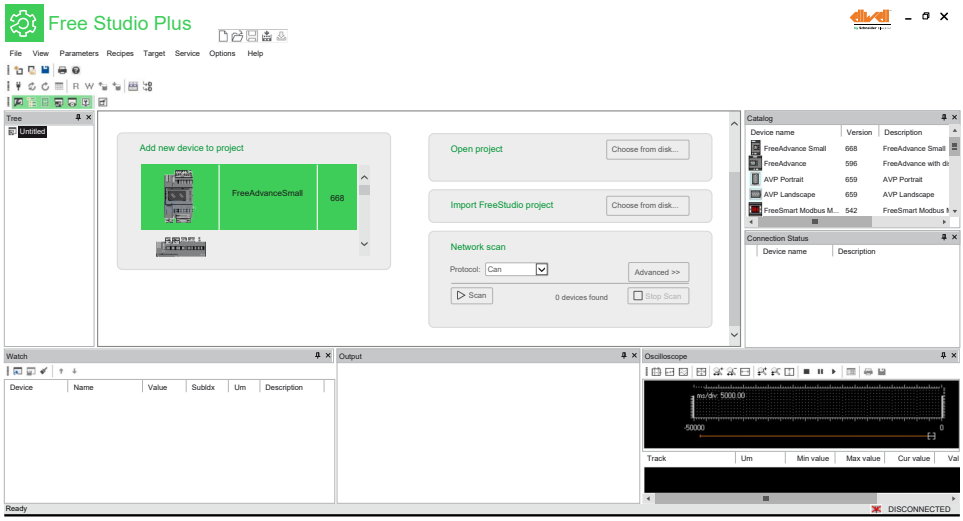
Failure to follow these instructions can result in equipment damage.

Installer Software

Description

A stand-alone software, **Installer** software, is delivered in addition with FREE Studio Plus.

You can run it via the Windows start menu or the icon on the desktop after installing FREE Studio Plus.



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.
- Detect on which port the target device is connected (via the **Network scan** area).
- Manage parameters files of the project (import and export throughout controllers).
- Create a programming USB key to upload projects in various controllers.
- Generate and preview an HTML page with project parameters.

The **Installer** software is dedicated for maintenance use. The project code cannot be modified.

Close FREE Studio Plus

Overview

To exit FREE Studio Plus, click the **Close** button in the top right-hand corner of the FREE Studio Plus window.

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

Configuration

What's in This Part

The Configuration Tab

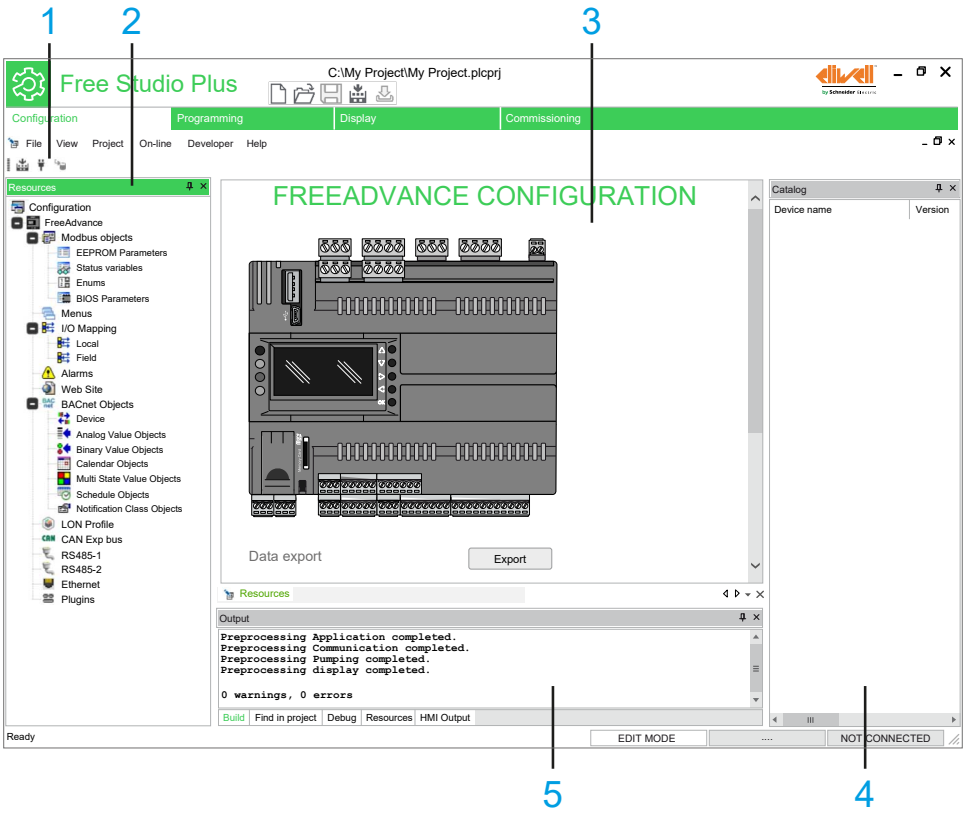
What's in This Chapter

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:



Item	Description	
1	Toolbar	This toolbar shows the tools in form of icons. For more information, refer to Toolbars, page 53.
2	**Resources** window	This window shows the configurable parameters of the device. For more information, refer to Content of the Resources Window, page 54.
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, page 35. For more information, refer to Download and Upload Applications, page 48.

Menu Bar

Overview

The menu bar of **Configuration** tab is composed of these menus:

- File, page 28
- View, page 33
- Project, page 30
- On-line, page 29
- Developer, page 27
- Help, page 28

Toolbar

Introduction

The toolbar appears at the top of the FREE Studio Plus window to provide access to frequently used functions.

For generalities of toolbars, refer to Toolbars description, page 34.

Configuration Toolbar

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

Icon	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.

Managing Resources Elements

What's in This Chapter

Overview

Resources Window

Overview

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

- Define parameters and variables.
- Create and configure the embedded website (for FREE Evolution and FREE Advance 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, page 57		Shows the picture of the target device and allows you to configure some settings.
Modbus objects, page 58		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, page 62		Manages menus where you can group the Modbus objects that are shown in **Commissioning**.
I/O Mapping, page 64		Defines the links between variables and physical I/O of the target device.
Alarms, page 65		Defines alarm variables which status must be managed by the developer.
Web Site, page 66		Defines website pages to monitor the device from a web browser.
BACnet Objects		Configures the BACnet objects.
LON Profile		Configures the LonWorks protocol.

Item	Icon	Description
CAN Exp bus, page 68		Configures the CAN Expansion bus.
RS-485-1, page 78		Configures the first RS-485 port.
RS-485-2, page 78		Configures the second RS-485 port.
Master Modbus RTU, page 78		Configure the RS-485 port. It only applies to FREE Smart Modbus master/slave.
Ethernet, page 85		Configures the Ethernet port.
Plugins, page 87		Configures protocols using communication modules.
Help		Shows LED reference for the developer. For FREE Smart 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 %SW 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
FREE Evolution	CAN expansion bus	On-board For I/O expansion and remote display
	RS-485	On-board Modbus RTU (master/slave)
	Ethernet	Optional with communication module: Modbus TCP (Server), BACnet IP (Server), HTTP
	Plugins	Optional modules available separately
FREE Panel EVP	CAN expansion bus	On-board For I/O expansion
	RS-485	On-board Modbus RTU (master/slave) or BACnet MS/TP (Server)
	Ethernet	On-board Modbus TCP (client/server), BACnet IP (Server), FTP, HTTP
AV••••••50500	CAN expansion bus	On-board For I/O expansion and remote display
	RS-485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS-485-2	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
AV•••••••6•500	CAN expansion bus	On-board For I/O expansion and remote display
	RS-485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS-485-2	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
EWCM 9000 PRO (HF)	CAN expansion bus	On-board For I/O expansion and remote display
	RS-485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS-485-2	On-board

Target	Communication Bus	Description
		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
FREE Optima	CAN expansion bus	On-board For I/O expansion and remote display
	RS-485-1	On-board Modbus RTU (Slave only) or BACnet MS/TP (Server)
	RS-485-2	On-board Modbus RTU (master/slave) or BACnet MS/TP (Server)

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

NOTE: On the RS-485 protocol, you can also connect generic Modbus devices.

Providing HMI Pages

AV•••••••6•500 supports HMI Remote so its pages can be downloaded and displayed in **Display** tab for FREE Evolution displays.

This feature is not supported by FREE Panel EVP. No linked device can upload HMI pages from FREE Panel EVP 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.

FREE Smart/FREE Evolution 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 FREE Smart.
- 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 FREE Smart.
- Export parameters and variables defined in **.csv** format.

  For more information, refer to Export CSV Files, page 46.
- Consult the hardware guide of the device by clicking the  icon.

FREE Advance 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, page 46.
- Consult the hardware guide of the device by clicking the  icon.

FREE Optima 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, page 46.
- Consult the hardware guide of the device by clicking the  icon.

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, page 89.

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.

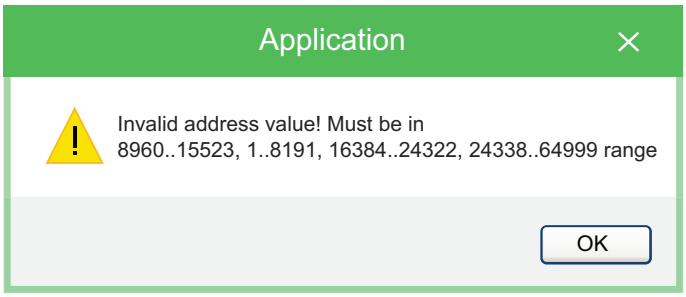
The **Recalc** button allows you to recalculate the addresses of the selected rows.

Address Range

In the targets prior to FREE Optima there is a Modbus address defined for the Status Variables and the Parameters, for FREE Optima however the range of Modbus addresses is extended:

- 1-8191
- 8960-15523
- 16384 - 64999
- Up to 65000 are reserved for the PLC Runtime

When invalid Modbus addresses are set, the warning message appears:



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, page 60 for details about the columns of the editor window.

This table presents the columns of the editor window:

Column	Description
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 FREE Smart target (4-digit 7 segments).
Installer type	Type of data displayed on target and Installer.
IEC 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 **Installer type** and **IEC type**.
Offset	IEC type = scale x Installer type + offset.
Unit	Unit of measurement of Installer type displayed on Installer 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.
Installer Access Level	Here can be set the level of visibility of a parameter in **Installer Software**. There are 4 different level of access: • Admin • Supervisor • Base (no password required) • Never Visible (never visible only in **Installer Software**) Refer to Visibility of Menu Resources, page 63.
Description	Free text.
Note	

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

EEPROM memory is limited, there is a **limitation** in the amount of read and write operations on this block set at 100k 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.

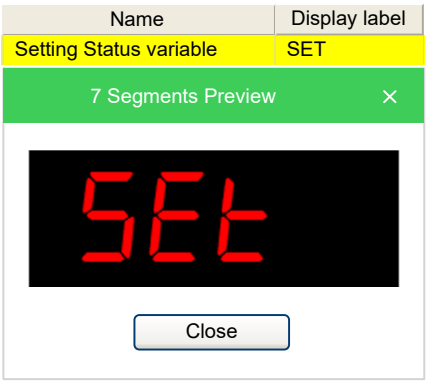
NOTICE
Do not use non-volatile memory registers for cyclic write operations. **Failure to follow these instructions can result in equipment damage.**

Status Variables

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

NOTE: For FREE Smart / FREE Optima, 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 t appears on the display.



This table presents the columns of the editor window:

Column	Description
Address	Resource Modbus address **NOTE:** For FREE Optima there are two different Address Columns: • Address (dec) • Address (hex)
Name	Resource name which may be used by the developer in the controller application.
Display label	Name displayed in the application menu of the FREE Smart target (4-digit 7 segments).
Installer type	Type of data displayed on target and Installer.
IEC 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 Installer type and IEC type.

Column	Description
Offset	IEC type = scale x Installer type + offset.
Unit	Unit of measurement of Installer type displayed on Installer 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.
Display Access Level	User access level required to see and modify related parameter via Display. Refer to Visibility of Menu Resources, page 63.
Installer Access Level	User access level required to see and modify related parameter via Installer. Refer to Visibility of Menu Resources, page 63.
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 **Installer Type** column of the editor window.

For more information about **Enums**, refer to Enumerations, page 131.

Enums, which are generated and managed in **Programming** window can be viewed in **Web Site** pages as all the other parameters.

BIOS Parameters

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

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.
New value	New value of the object.
New min	New minimum value of the object. **NOTE:** Only for FREE Optima.
New max	New maximum value of the object. **NOTE:** Only for FREE Optima.
New um	New unit of measurement of the object. **NOTE:** Only for FREE Optima.
New level	New user access level required to see and modify related parameter. Refer to Visibility of Menu Resources, page 63. **NOTE:** Only for FREE Optima.
Default value	Default value of the object.
Min	Minimum value of the object. **NOTE:** Only for FREE Optima.
Max	Maximum value of the object.

Column	Description
	NOTE: Only for FREE Optima.
Um	Unit of measurement of the object. **NOTE:** Only for FREE Optima.
Level	User access level required to see and modify related parameter. Refer to Visibility of Menu Resources, page 63. **NOTE:** Only for FREE Optima.
Description	Description of the object.

Target Menus

Target Menu FREE Evolution/FREE Advance

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.

Target Menu FREE Smart

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 *sysKeyFunctions[]*).

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:

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

Target Menu FREE Optima

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 *sysKeyFunctions[]*).

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:

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

I/O Mapping

FREE Smart I/O Mapping

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

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

FREE Evolution/FREE Advance/FREE Optima I/O Mapping

You can define the links between variables and physical I/O of FREE Evolution/ FREE Advance/FREE Optima:

- **Local**: local variables of the FREE Evolution/FREE Advance/FREE Optima 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.

Alarms

FREE Smart 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 FREE Smart.

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 FREE Smart/FREE Evolution 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**, SEt appears on the display.

FREE Evolution/FREE Advance Alarms

In the FREE Evolution/FREE Advance target, it is only a Global type USINT declaration.

The alarms for FREE Evolution/FREE Advance are only defined to enable the portability of an FREE Smart project.

FREE Optima 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 FREE Optima.

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 FREE Optima 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**, SEt appears on the display.

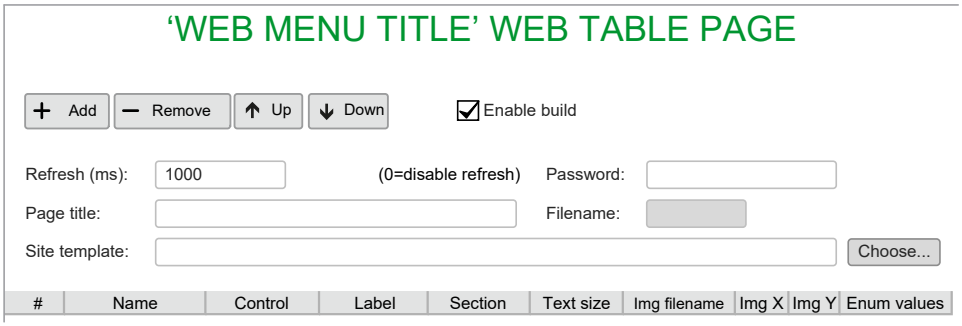
Web Site

Web Functionalities

The FREE Advance 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.



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 and Eliwell adhere 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.

CAN Expansion Bus

CAN Expansion Bus Overview

Description

FREE Evolution/FREE Advance/FREE Optima 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)**.

FREE Panel EVP can be connected using CAN Expansion bus for field mode or for network mode.

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).
- The parameter which represents the maximum number of expansions used by PLC application.

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, page 50.

When you configure the CAN Expansion bus port as **Slave**, the FREE Evolution/Advance 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 FREE Evolution 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:

- **EVD_1_par1** reads from **Free Evolution EVD_2** the *EVD_2_par1* object and puts it in its local *EVD_1_par1* object.
- **EVD_1_par1** reads from **Free Evolution EVD_2** the *EVD_2_par1* object and puts it in its local *EVD_1_par1* 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 FREE Advance Expansion Modules and FREE Advance Controllers

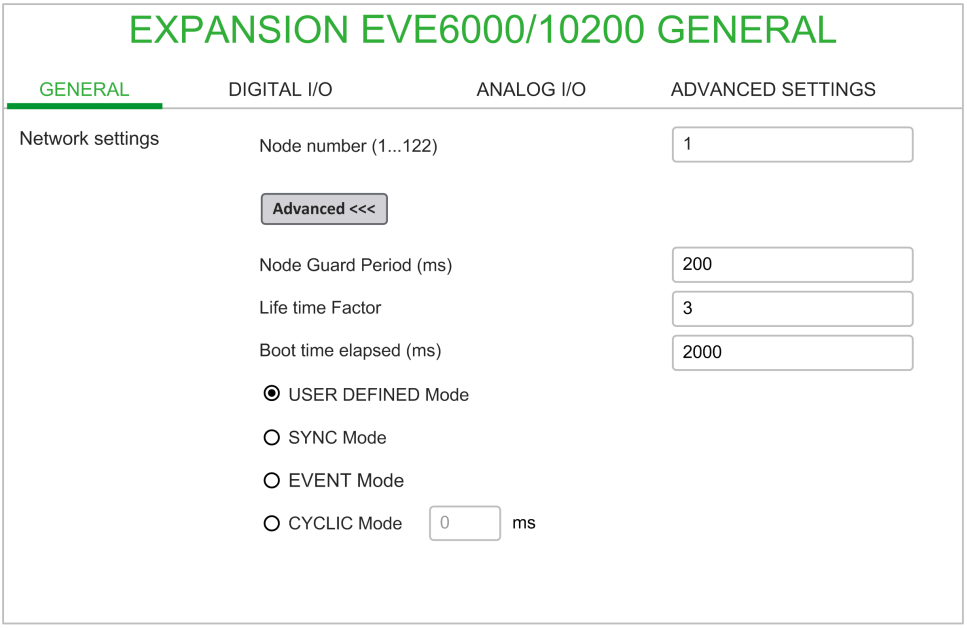
In this configuration example, you want to use FREE Advance Expansion 28 I/Os as expansion of a AV•••••••6•500 device. The same can be done for other logic controllers.

Configure AV•••••••6•500 CAN Expansion Bus in **Master** (for field) mode. From the **Catalog** window, select **Expansion EVE6000/10200** node and drop it on the **CAN Expansion Bus** node.

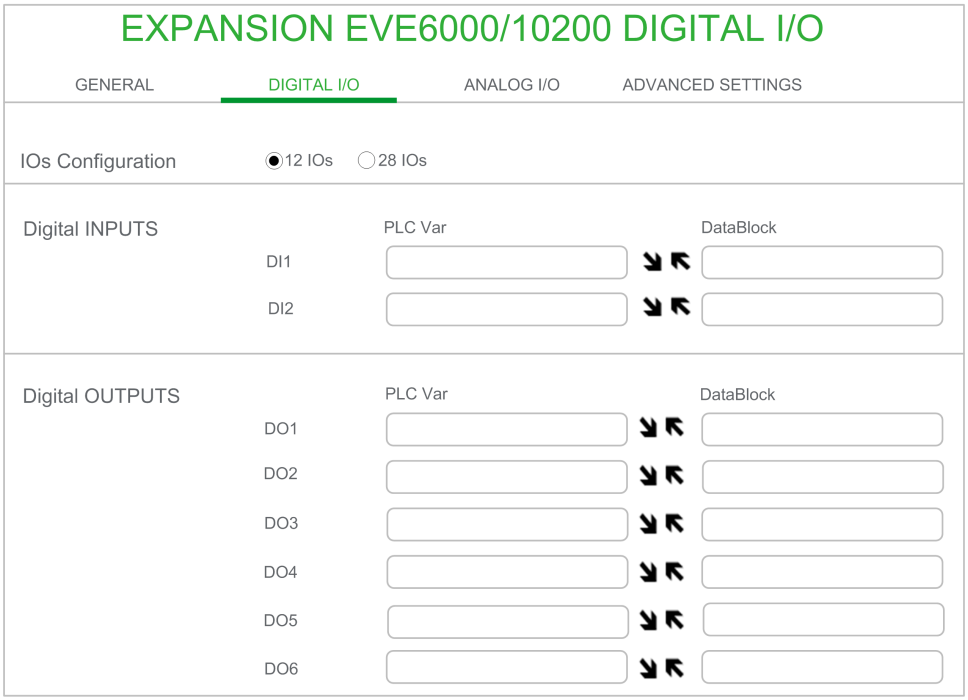
Expansion EVE6000/10200 configuration is divided into four tabs:

- **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 **Expansion EVE6000/10200**:



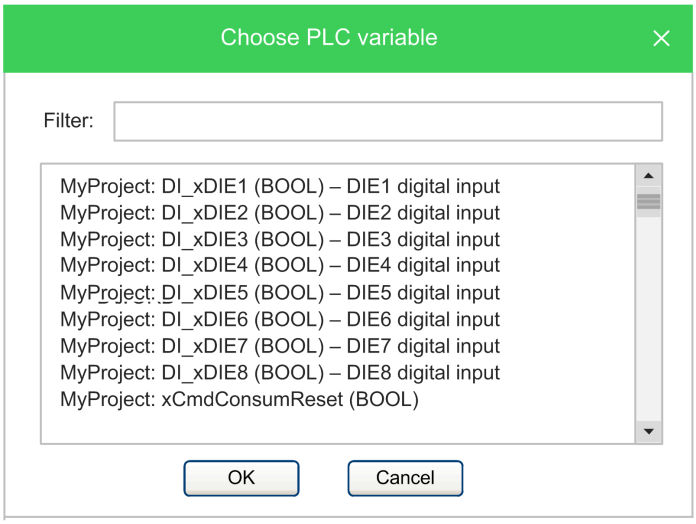
DIGITAL I/O tab of **Expansion EVE6000/10200**:



FREE Studio Plus knows the **Expansion EVE6000/10200** dictionary. Each object can be assigned to a **PLC variable**.

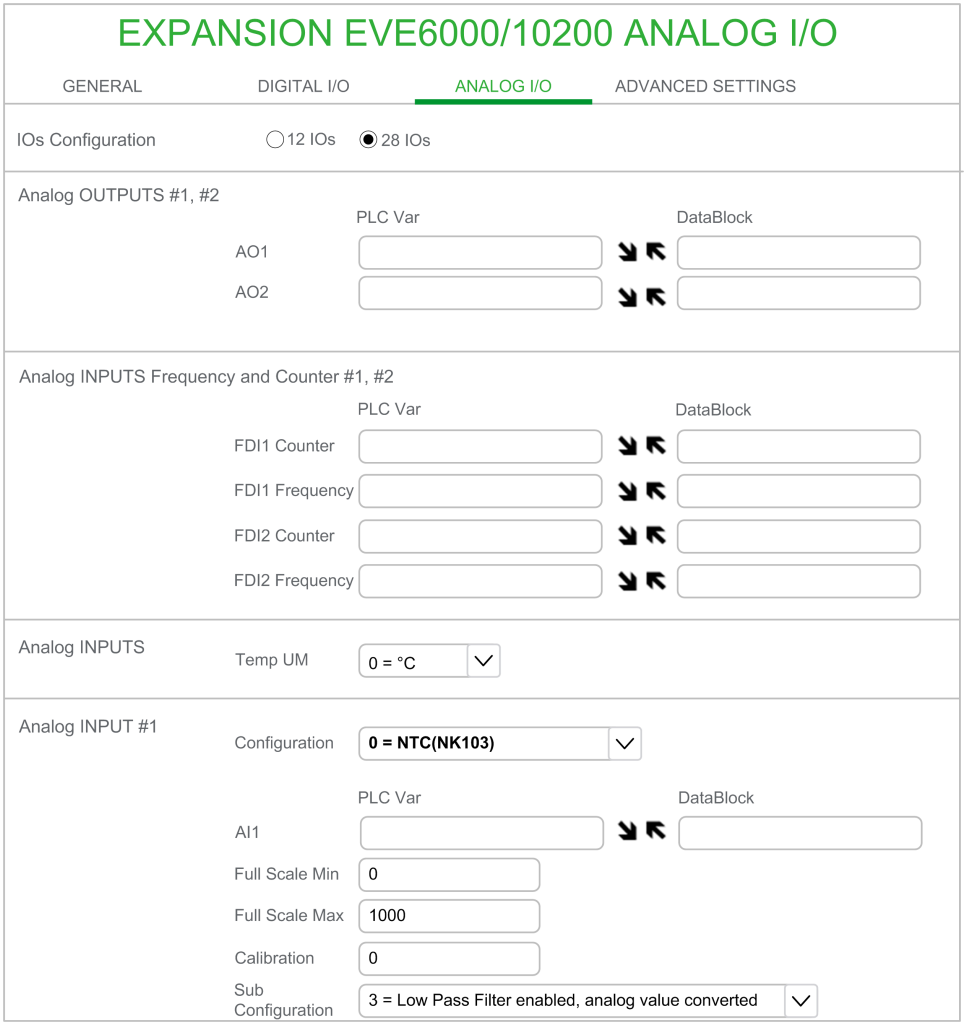
Follow this procedure do assign an object to a **PLC variable**:

Step	Action
1	Click **Assign** button.
2	Select the **PLC variable** that you want to assign to the PLC object:

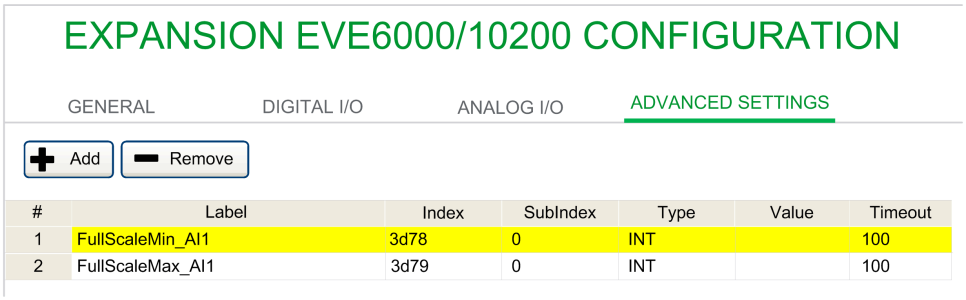
Step	Action
	 **NOTE:** It is possible to apply a filter to the choice list by entering a string of characters.
3	Click **OK** button to validate.
4	The **PLC Var name** is added and its address is displayed in the **DataBlock** field.

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

ANALOG I/O tab of **Expansion EVE6000/10200**:



ADVANCED SETTINGS tab of **Expansion EVE6000/10200**:



CAN Expansion Bus with FREE Evolution Expansion Module and FREE Evolution

In this configuration example, you want to use EVE7500 as expansion of a FREE Evolution device. The same can be done for other logic controllers.

Configure FREE Evolution CAN Expansion Bus in **Master** (for field) mode. From the **Catalog** window, it is possible to select **EXPANSION EVE 7500 (SIC)** node and drop it on the CAN Expansion Bus node.

EXPANSION EVE 7500 (SIC) configuration is similar to CAN Custom configuration (Using a CAN Custom Device, page 74) 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 FREE Panel EVP

Description

FREE Panel EVP can be connected using CAN Expansion bus in field mode or in network mode.

Field Mode

To connect FREE Panel EVP in this mode select FREE Evolution Display or **Free Evolution EVC** CAN Expansion bus node and select the option **Master** (for field) then take FREE Panel EVP device from **Catalog** tab and drop it over CAN Expansion bus node.

Select **Free Evolution EVP_1** child node and configure **Network** settings.

Probes:

Free Evolution EVD_1 can access to **Free Evolution EVP_1** on-board probes. To do so select **Probes - Input** tab then it is possible to map a **Free Evolution EVD_1** parameter to let it obtain the value of an on-board **Free Evolution EVD** probe.

Choose one of the probes and click **Assign** button. Take one of the **Free Evolution EVD_1** INT parameter and click **OK** button.

HMI:

It is possible to associate to a FREE Panel EVP (configured as CAN Expansion bus field slave) an HMI project with local pages. FREE Panel EVP 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, page 50.

In this connection mode, FREE Panel EVP 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 **Free Evolution EVD_1** and **Free Evolution EVC_1** then add as first-level node **Free Evolution EVP_1** to the network taking it from **Catalog** panel.

Click **CAN Exp bus** node of **Free Evolution EVP_1** and select **Master** (for HMI remoting and binding) node, assign unique **Node ID** and select network **CAN Exp bus1**.

Binding of variables between **Free Evolution EVP_1** and **Free Evolution EVC_1** and **Free Evolution EVD_1** is allowed in a network of this type (see chapter CAN Expansion bus - Binding, page 68 for more details).

HMI Remote pages:

In CAN Expansion bus network mode, it is possible to configure FREE Panel EVP 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 **Free Evolution EVP_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

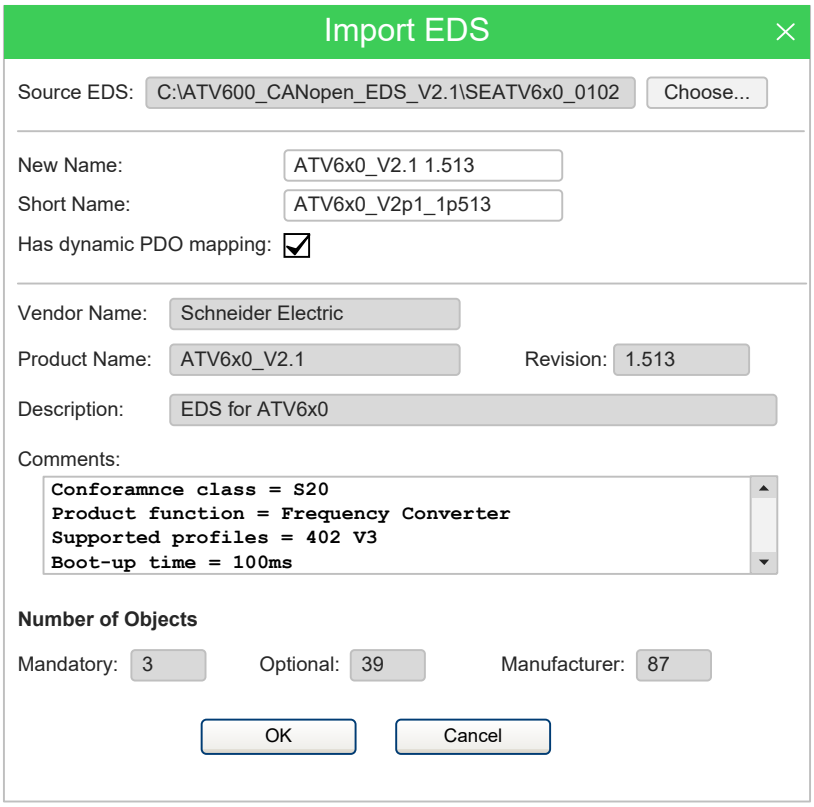
Description

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:



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

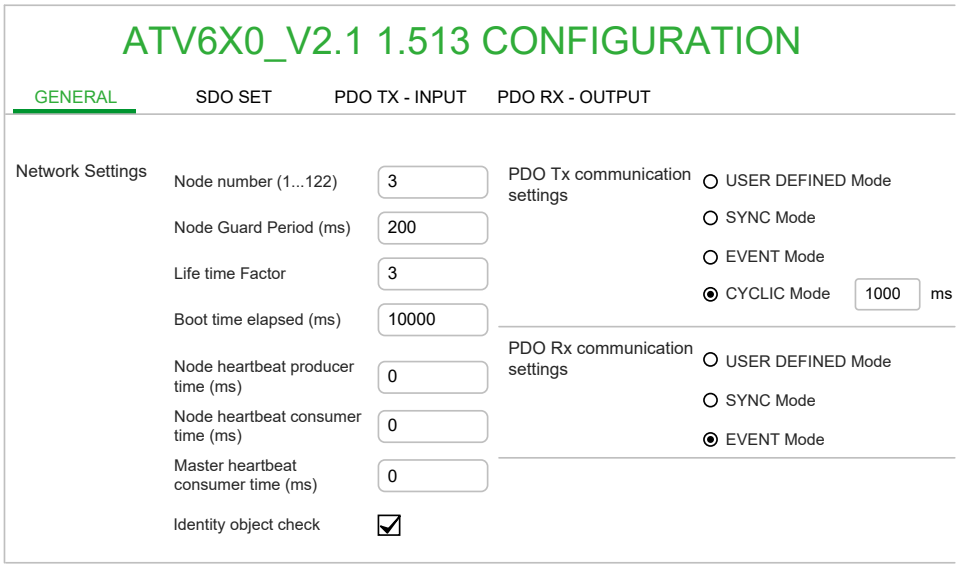
When the device you want to delete is visible in the **Catalog** window (for example when a CAN Expansion bus port is selected and is in **Master** mode), you can right-click on it and choose the **Delete from catalog** command.

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.

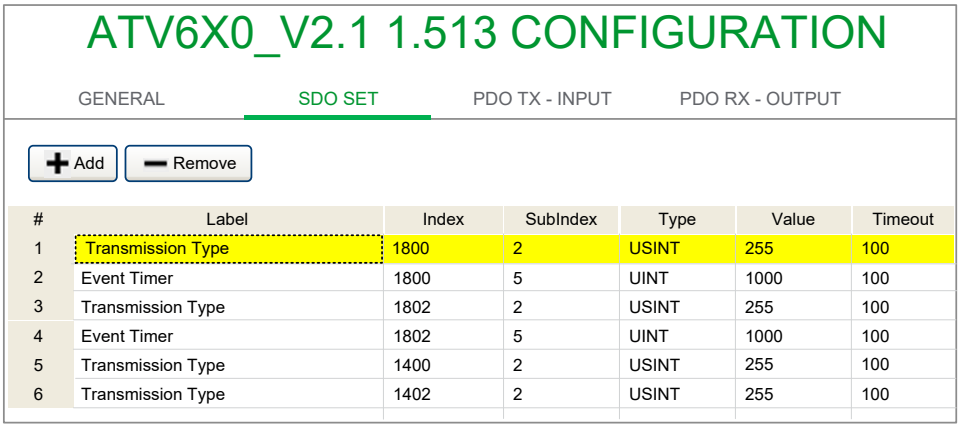
General Tab



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



#	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.

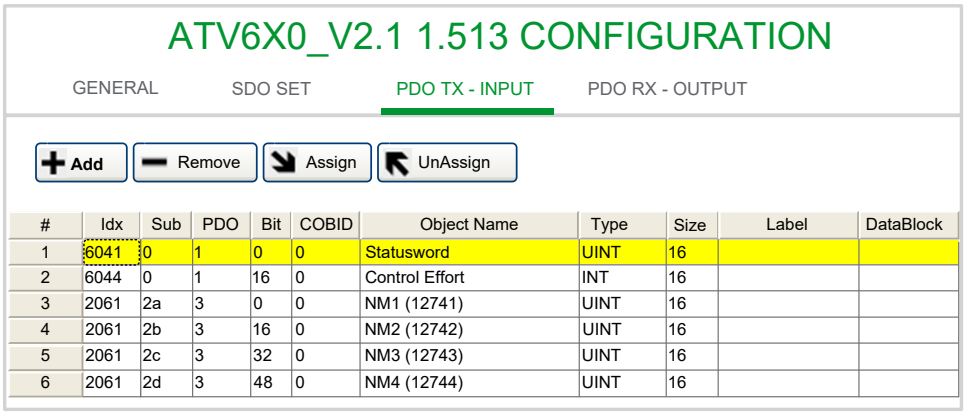
If you check the **Split single bits** option, the object you choose is inserted as single bits to be linked to BOOL variables (that is the default for digital I/O objects in the DS401 standard).

> **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).

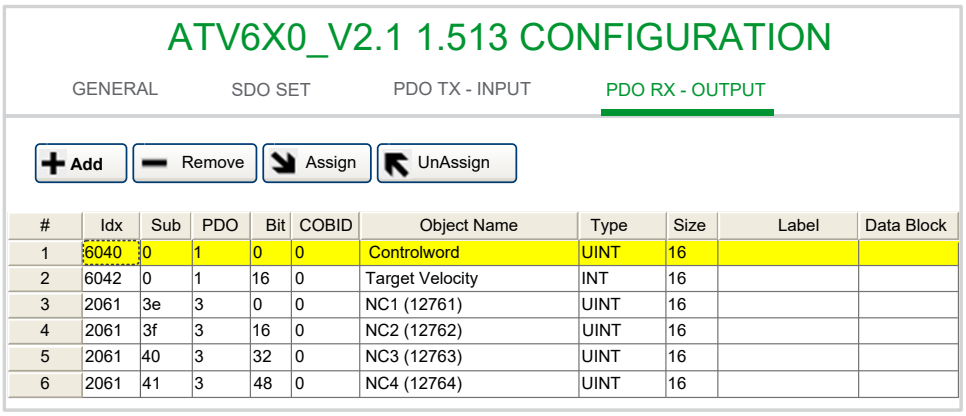
> **NOTE:** It is necessary to rebuild the PLC project with **Programming** to update the list of PLC variables.

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



#	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**:



#	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 FREE Evolution 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: FREE Evolution + 2 x EVK1000 Display Graphic

The two FREE Smart/FREE Evolution Graphic displays are both connected to the CAN expansion bus.

The CAN expansion bus has two default virtual channels that can be connected to a maximum of 2 x EVK1000 Display Graphic.

The default virtual channels are 124 for the first display and 123 for the second EVK1000 Display Graphic.

Click **?** button from the CAN expansion bus to view the values.

The default address of the Display for **EVE_2** display is 127, the default virtual channel 124 and the default CAN baudrate 500 kb/s.

Thus, when physically connecting an EVK1000 Display Graphic to an FREE Evolution with the default settings, upload HMI from the EVK1000 Display Graphic BIOS menu.

In other cases, such as for Display for **EVE_2** (which has the address 126), set the address 126 and the virtual canal 123 from the EVK1000 Display Graphic BIOS menu.

RS-485

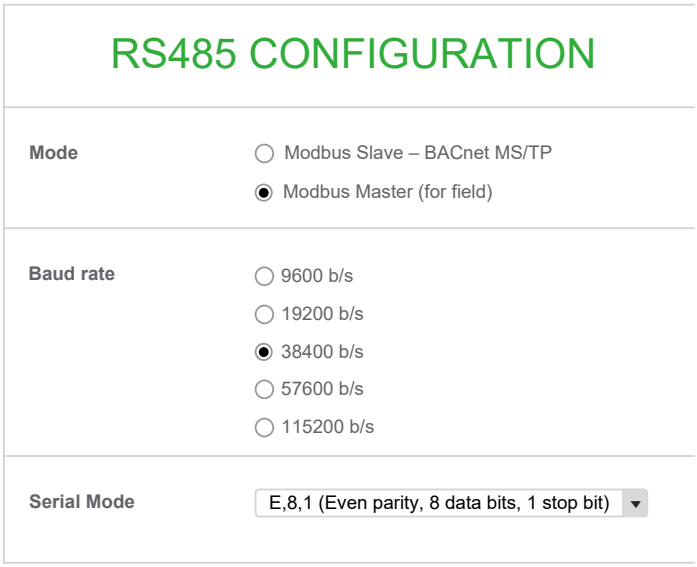
Overview

Description

FREE Evolution Display / FREE Panel EVP has one on-board RS-485 port, plus another one available as an external plugin. Each port can be configured as **Not used** (disabled) or **Modbus Master** (field).

FREE Optima / FREE Advance has two on-board RS-485 ports.

The first port (RS-485-1) is used for Modbus Slave - BACnet MS/TP. The RS-485-2 port can be configured as **Modbus Slave - BACnet MS/TP** or **Modbus Master** (for field).



Field

When you configure the RS-485 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 **RS-485** and select **Add** command. You can also drag a Modbus slave from the **Catalog** window to the **RS-485** item in the **Resources** window.

After you added and configured the Modbus slaves, page 79, 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 EVE7500 27 I/O as RS-485 Slave

Description

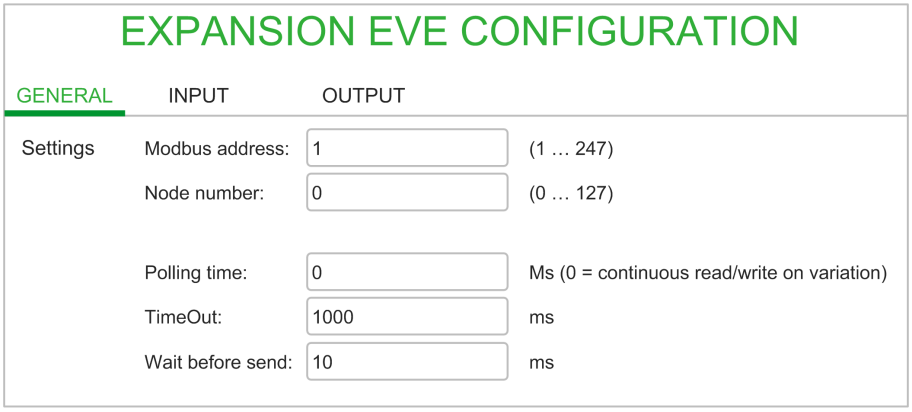
In this configuration example, you want to use **EVE7500 Expansion** as expansion of a FREE Evolution Display device. The same can be done for other logic controllers.

Configure FREE Evolution Display RS-485 in **Modbus Master** (for field) mode. From the **Catalog** window, it is possible to select **Expansion EVE** node and drop it on the **RS-485** node.

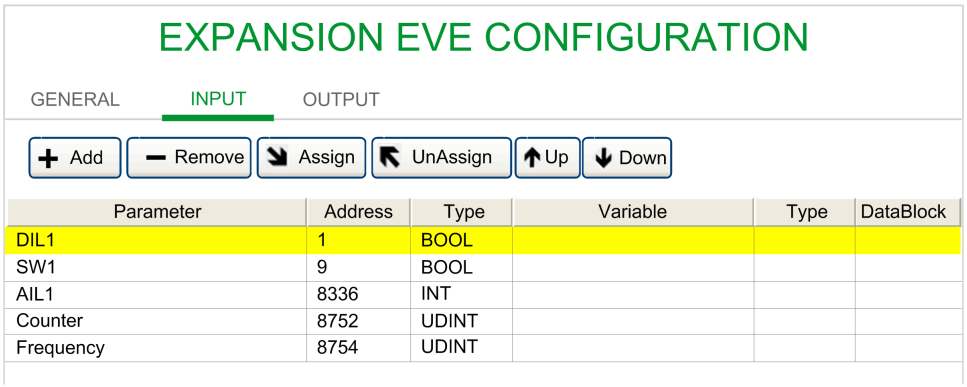
Expansion EVE configuration is similar to a (Modbus Custom device configuration, page 84). It is possible to assign available **Expansion EVE** dictionary I/O objects to **FREE Evolution EVD** PLC variables.

Configuration knows the **Expansion EVE** dictionary. *Input* and *Output* objects can be added, removed, assigned, unassigned, or changed in position. Only assigned objects are requested by **FREE Evolution EVD** device.

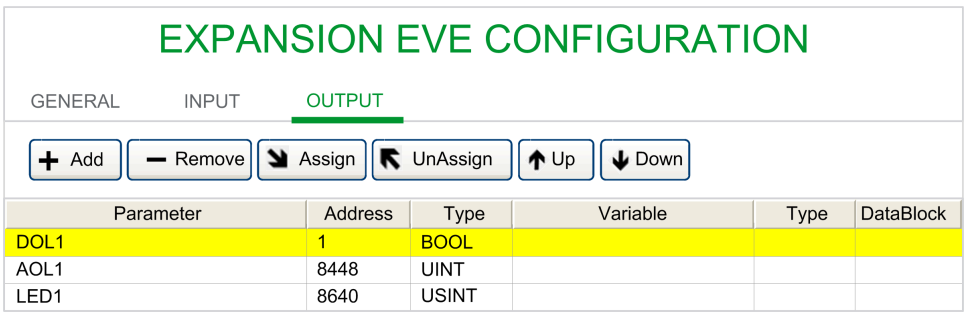
GENERAL tab of **EXPANSION EVE 7500 (SIC) CONFIGURATION**:



INPUT tab of **EXPANSION EVE 7500 (SIC) CONFIGURATION**:



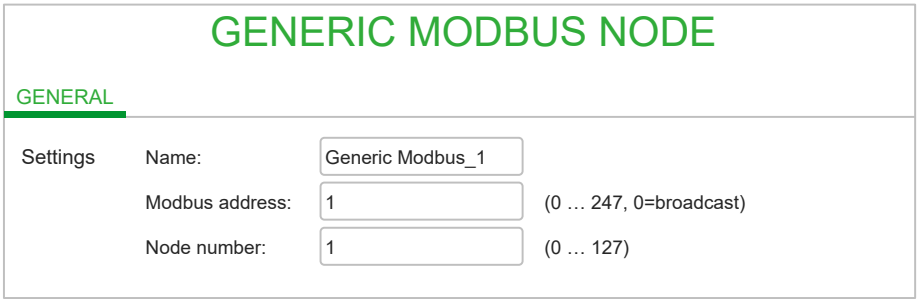
OUTPUT tab of **EXPANSION EVE 7500 (SIC) CONFIGURATION**:



Generic Modbus Object Overview

Description

The **Generic Modbus** object is a generic Modbus slave that can be inserted under the RS-485 port of the logic controller, when configured as **Modbus master**.



To add a **Generic Modbus** object:

Step	Action
1	In the **Resources** window, click **RS-485**.
2	• Drag **Generic Modbus** from the **Catalog** window to **RS-485** item in the **Resources** window. • Or you can right-click **RS-485** and select **Add** command. **Generic Modbus** appears under **RS-485** 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:

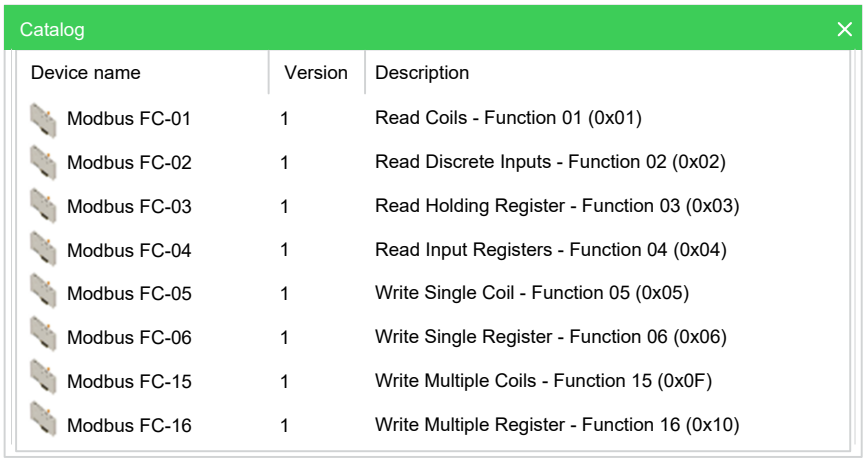
- A name for the object in the project.
- Its Modbus address (in the range 1…247).
- Its Node number (node ID)(in the range 0…127): this value is incremented automatically, and can be used in the PLC program to index the *SysMbMRtuNodeStatus[]* array that contains diagnostic information about each slave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

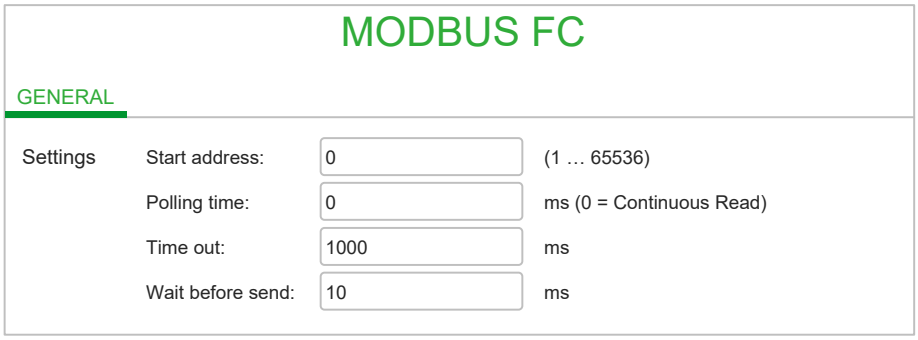


Function		Description	Details	Objects length
1	0x01	Read Coils	Reads one or more read-only discrete output coils	1-bit
2	0x02	Read Discrete Inputs	Reads one or more read-only digital inputs	1-bit
3	0x03	Read Holding Registers	Reads one or more read/write registers	16-bit
4	0x04	Read Input Registers	Reads 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:



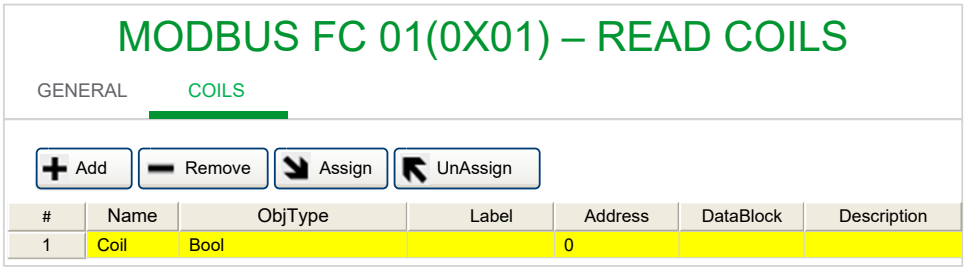
For each message, in its **General** tab you can configure:

- **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.



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

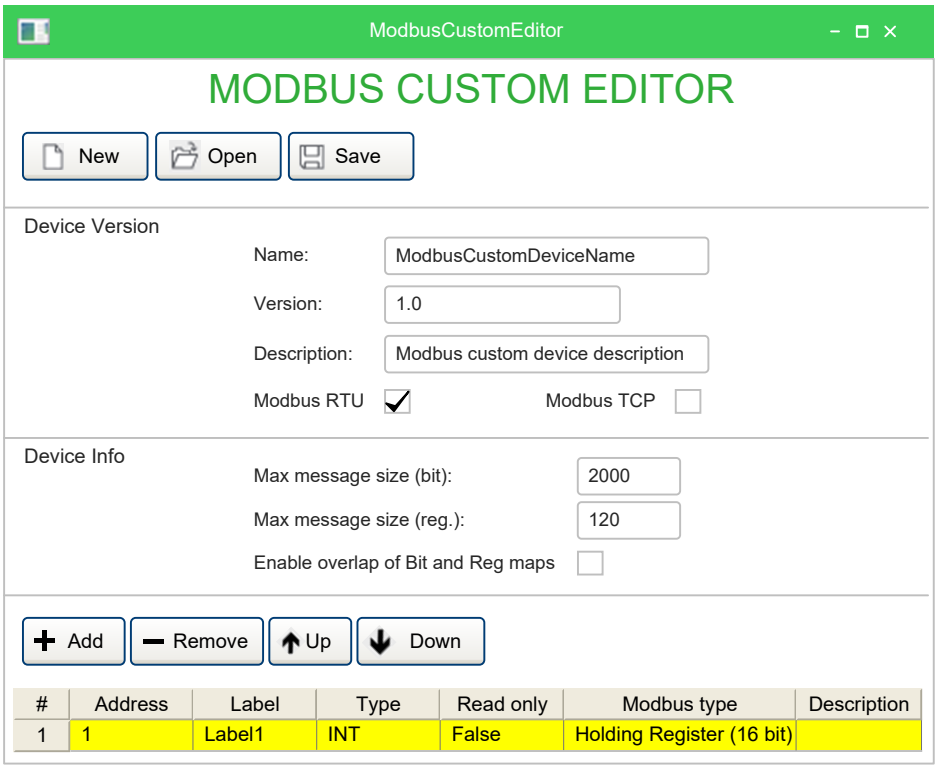
Description

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.



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 RS-485 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

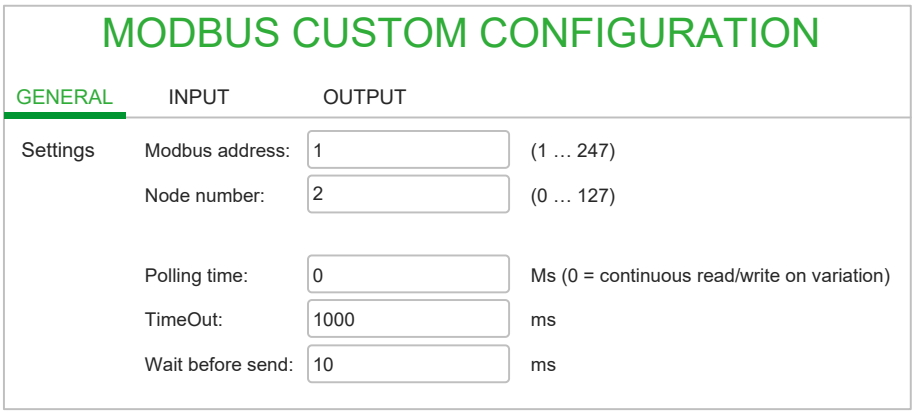
When you insert the **Modbus custom** device as a Modbus slave (for example under an RS-485 port) and click it on the **Resources** window, the Editor window displays three tabs.

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**:

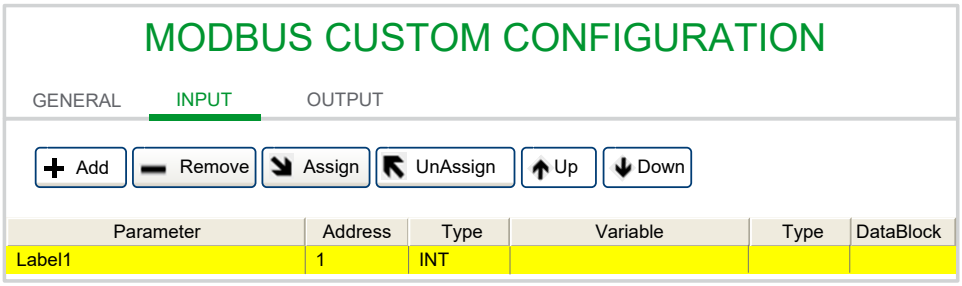


Here you can note that for **Modbus custom** the **Polling time**, **Timeout**, and **Wait before send** are generic for the whole device while for the **Generic Modbus** you can put specific different values for each single message. This is because with the **Modbus custom** the low-level Modbus messages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

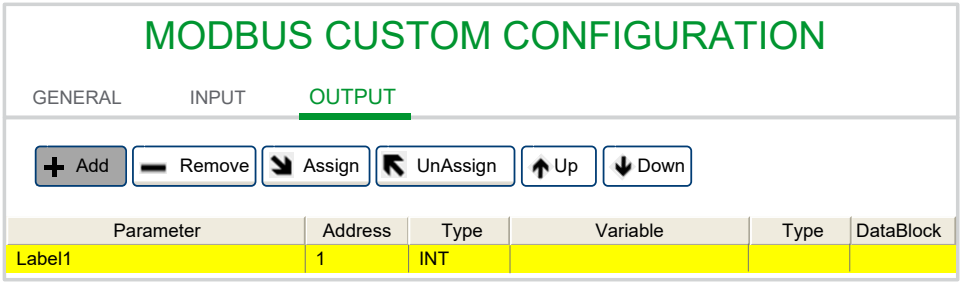
Input and Output Tabs

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**:



OUTPUT tab of **MODBUS CUSTOM CONFIGURATION**:



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 FREE Evolution/FREE Advance).

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

Ethernet

Description

FREE Panel EVP / AV•••••••6•500 are provided with an integrated Ethernet port.

FREE Evolution/AV••••••50500 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 and Eliwell adhere 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.

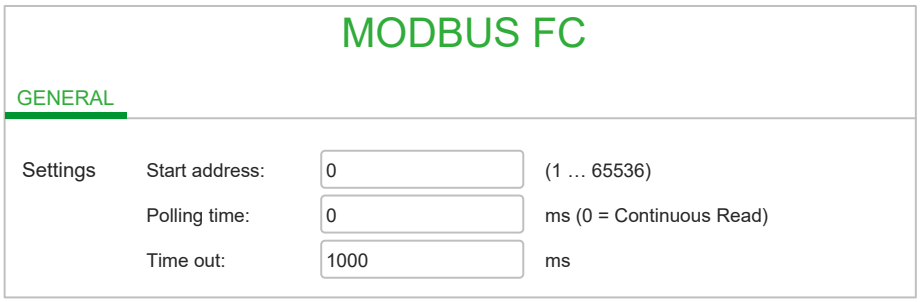
You can add generic Modbus devices, page 80, or custom devices created with the Modbus custom Editor, page 82 in the same way as for RS-485.

After you added and configured the Modbus nodes, you can add Generic Modbus Objects Messages, page 80 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, page 68 for a description and usage of this page.



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.

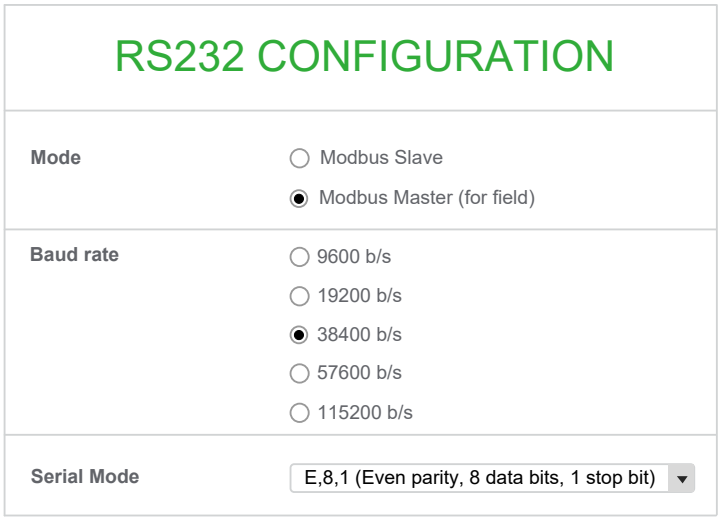
Plugins

Communication Modules Range Overview

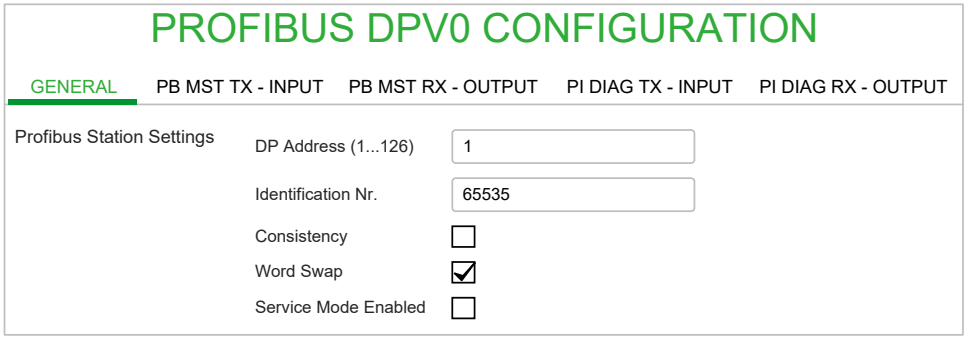
The communication modules (plugins) that can be added are displayed in the **Catalog** window.

Drag the communication module into the **Plugins** element.

Example of RS-232 configuration window:



Example of Profibus DP configuration window:



Communication Modules References

Reference	Description	Terminal type	Compatible controllers
EVS00CA000000	CAN	2 screw terminal blocks	AV•••••••6•500 AV••••••50500 FREE Evolution(1) EWCM 9000 PRO (HF)(2)
EVS0LON000000	LonWorks	1 screw terminal block	
EVS00R4000000	Modbus SL (RS-485)	2 screw terminal blocks	
EVS10R2000000	RS-232 serial link, Relay output	1 SUB-D 9 1 screw terminal block	
EVS00BM000000	Modbus SL, and BACnet MS/TP	2 screw terminal blocks	
EVS00ET000000	Ethernet, Modbus TCP, and BACnet/IP	1 RJ45	AV••••••50500 FREE Evolution(1)
EVS00EB000000	Ethernet, Modbus TCP, BACnet/IP, Modbus SL, and BACnet MS/TP	1 RJ45 2 screw terminal blocks	
EVS00PB000000	PROFIBUS	1 SUB-D 9	FREE Evolution(1)

(1) Not applicable to FREE Panel EVP

(2) Not applicable to EVS00R4000000

For further information about communication modules, refer to the FREE EVS Plugin Instruction Sheet 9IS54405.

Technical Reference

What's in This Chapter

Modbus Protocol

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.

Programming

What's in This Part

The Programming Tab

What's in This Chapter

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.

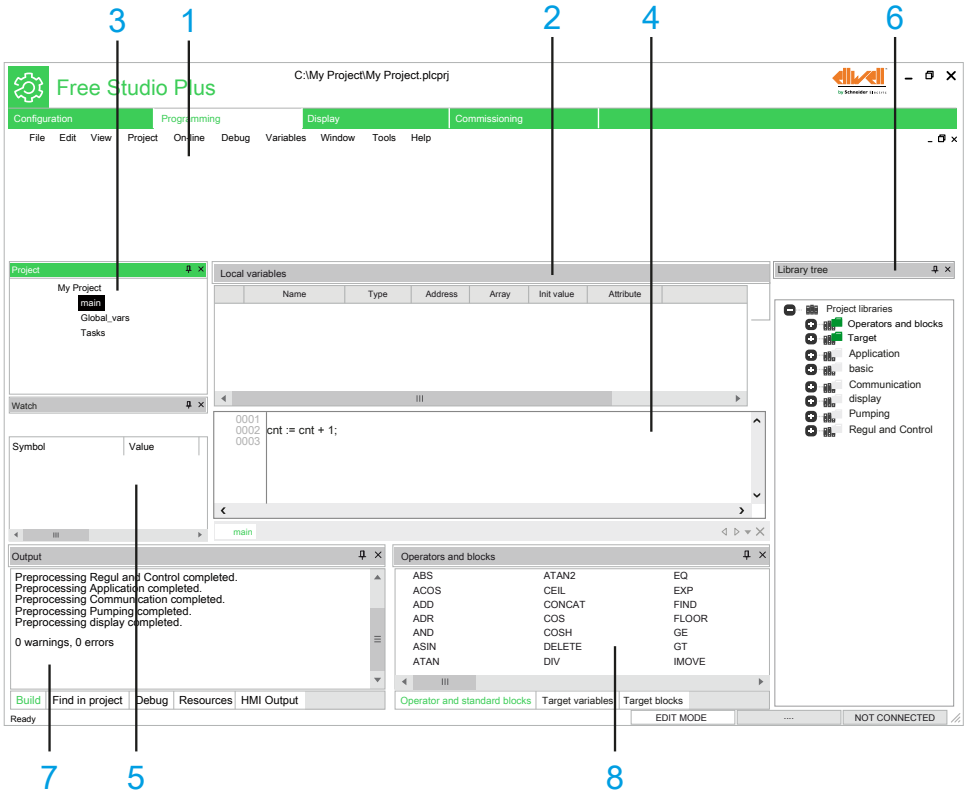
⚠WARNING

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:



Item	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, page 36.
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, page 117.
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, page 114.
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, page 143.
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, page 194.
6	**Library Tree** window	The **Library Tree** window contains a set of different library objects, excepts for two generic folders which are default for any project. These two folders are **Operator and Blocks** folder and **Target** folder which are colored in green. 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, page 109.

Item	Description	
7	**Output** window	This tool window shows the messages relating to the development of the project. For more details, refer to Output window description, page 94.
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, page 35.

Menu Bar

Overview

The menu bar of **Programming** tab is composed of these menus:

- File, page 28
- Edit, page 27
- View, page 33
- Project, page 30
- On-line, page 29
- Debug, page 26
- Scheme, page 31
- Variables, page 32
- Window, page 34
- Tools, page 32
- Help, page 28

NOTE: Programming has a multi-document interface (MDI), so you may find some disabled commands or even some unavailable menus, depending on what kind of document is currently active.

Toolbars

Introduction

The toolbars appear at the top of the FREE Studio Plus 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
	Print preview	-
	Workspace	Ctrl+W
	Output	Ctrl+R
	Operators and blocks	Ctrl+L

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

Icon	Description	Shortcut
	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
	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	-

Icon	Description	Shortcut
	Expression	-
	Return	-
	Jump	-
	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	-
	Remove pin from simultaneous divergent transition	-
	Add pin to simultaneous convergent transition	-
	Remove pin from simultaneous convergent transition	-

Icon	Description	Shortcut
	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
	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	-

Project Options

What's in This Chapter

Project Options

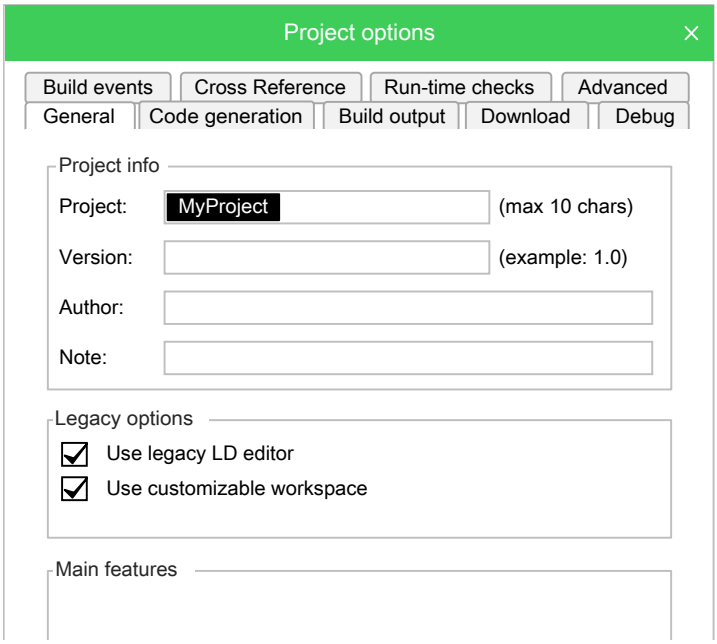
General Information

You can edit significant project options choosing **Project > Options…**.

General Tab

Overview

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



You can set general project information:

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

In addition, you can set compatibility options:

- **Use legacy 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, page 150.
- **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, page 139.

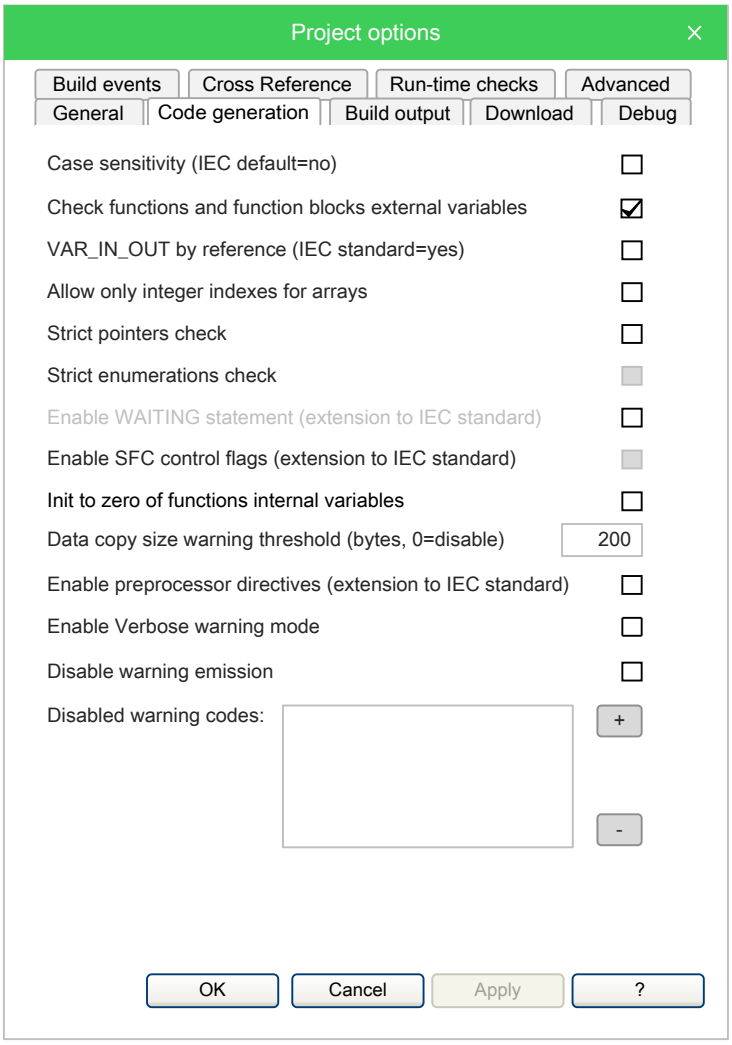
In addition, you can set compatibility options:

- **Use legacy LD editor**: the new Ladder Diagram editor is easier to use, by helping user 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, page 150.
- **Use customizable workspace**: allows user to manage the project tree in order to reach a more efficient workspace. By default, this option is active. For more information, refer to Project Custom Workspace, page 139.
- **Multiple files project**: allows user to save project in *.xplc format.
- **Custom sort of project folders**: enables **Move up** and **Move down** commands in the context menu of the folder, in the **project tree**.

Code Generation Tab

Overview

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



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.

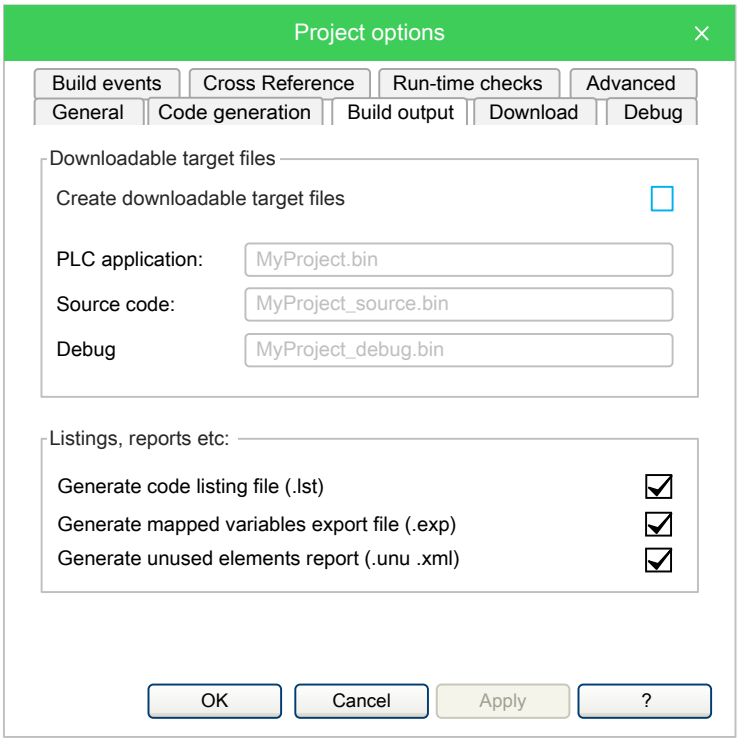
  NOTE: By default, this option is enabled respecting the IEC 61131-3 standard.

- **VAR_IN_OUT by reference**: if this option is checked the variables declared as **VAR_IN_OUT** of a function block will be treated as reference variables, accordingly to IEC standards.
- **Allow only integer indexes for arrays**: if this option is checked you cannot use BYTE, WORD or DWORD as array indexes.
- **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, page 300.
- **Enable SFC control flags (extension to standard)**: if this option is checked, HOLD and RESET flags for SFC POU are enabled.
- **Init to zero of function internal variables**: if this option is checked, the initial value of the internal variables of the functions will be set to zero as default.
- **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.
- **Enable preprocessor directives (extension to IEC standard)**: if this option is checked, IFDEF feature is enabled (user can allow build of portion of code verifying if a certain symbol has been defined).
- **Enable verbose warning mode**: if this option is checked, several minor warning, related to operation between signed and unsigned variables, are emitted (for example, a>=b, where a is INT and b is UINT, will raise a warning if this option is enabled); if this option is disabled those warnings are not 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:



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

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**

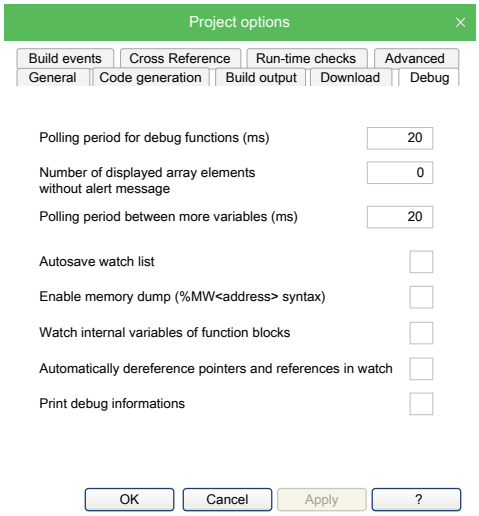
Listing, reports etc section:

- **Generate code listing file (.lst)**: if this option is checked the compiler generates a listing file named as **projectname.lst**.
- **Generate mapped variables export file (.exp)**: if this option is checked the compiler generates an EXP file named as **projectname.exp**.
- **Generate unused elements report (.unu .xml)**: if this option is checked the compiler generates two reports of unused elements named as **projectname.unu** and **projectname.xml**.

Debug Tab

Overview

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



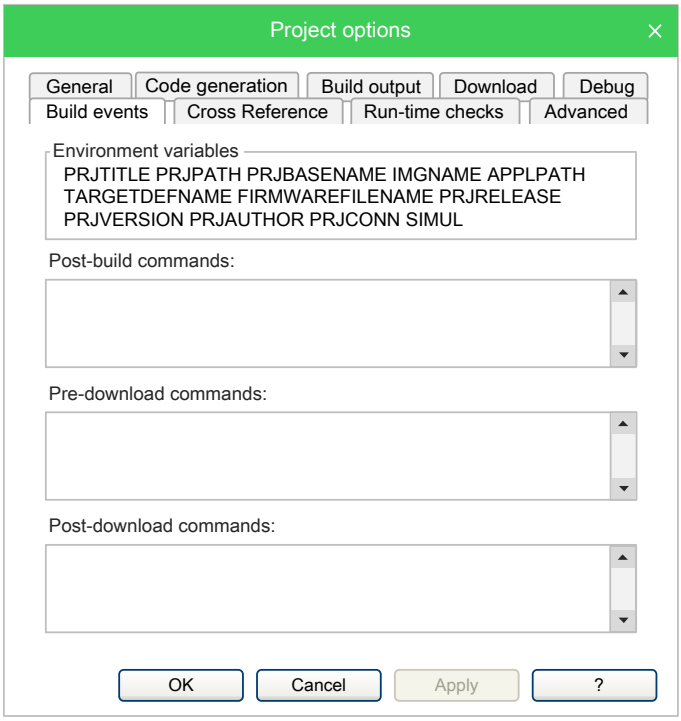
Here user can edit some significant properties of the debug behavior:

- **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 user to view internal variables of “VAR” class.
- **Automatically dereference pointers and references in watch**: if this option is checked, when adding the pointer variable in the watch window, the pointed value will be directly shown; if it is disabled, the watch window will show the content of the pointer which need to be expanded to see the pointed value.
- **Print debug informations**: when compiling, additional information are shown in the output window.

Build Events Tab

Overview

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



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.

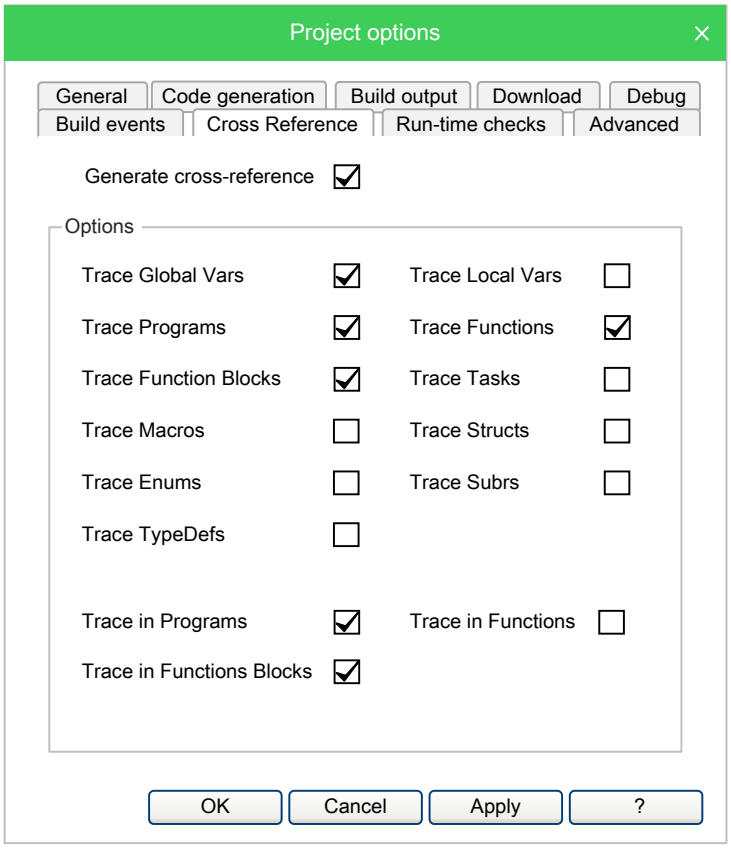
The environment variables are:

- **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:



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.

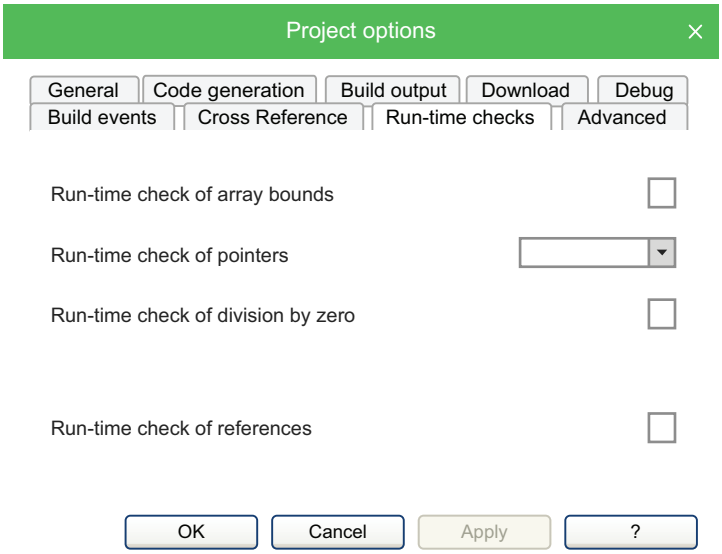
The cross-reference trace options can be set in:

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

Run-time Checks Tab

Overview

Run-time Checks tab of the **Project options** window:



These options allow the user to enable specific controls made at execution time.

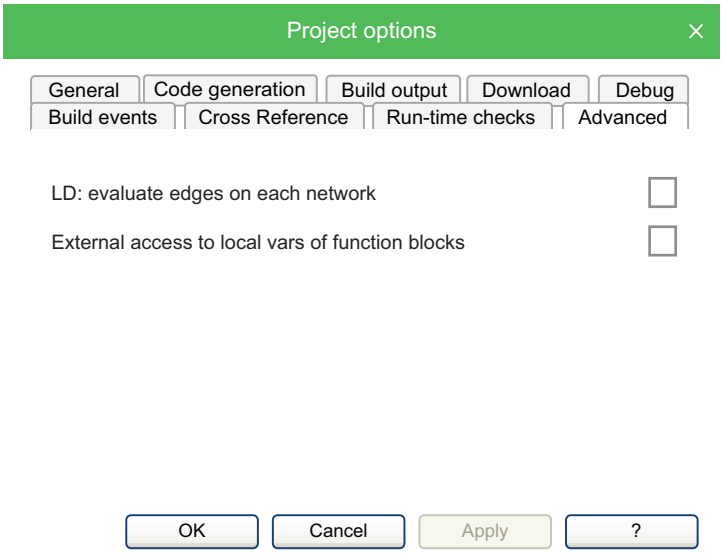
The run-time check options can be set for:

- **Run-time check of array bounds**: if this option is checked some check code is added to verify that array indexes are not out of bounds during run-time. This option can be set depending on target device.
- **Run-time check of pointers**: this combo allows you to choose if and when the pointer will be tested for their validity before their use.
  - Selecting NONE, the check will never be done.
  - Selecting ONLY IF NOT NULL, the check will verify that the pointer value is not NULL; if it is NULL, it will return value zero but will not stop the running application. So the PLC execution will never be interrupted due to a NULL pointer, but you'll never get an error notification.
  - Selecting FULL, the check will verify that the pointer value is not NULL and that the pointed address is within a validity range (this last control requires the user-defined function checkptr on target; if it is not defined, only the first control is executed). If one of this two controls fail, the PLC execution is interrupted and an error message is raised.
- **Run-time check of division by zero**: if this option is checked some check code is added to verify that divisions by zero are not performed on arrays during run-time. This option can be set depending on target device.
- **Run-time check of interfaces**: if this option is checked, allows a references validity check within a method call. This option can be set depending on target device (Object Oriented supported)
- **Run-time check of references**: if this option is checked, allows a references validity check; if a reference is dereferenced to null, a runtime error is generated.

Advanced Tab

Overview

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



These options allow the user to specify specific behaviors, suggested only for expert users.

- **LD evaluate edges on each network**: this option allows the user to change edges evaluation timing (high/low). If it is NOT checked, the edges of an LD2 program are evaluated only once, at the beginning of the program execution. If it is checked, the edges of an LD2 program are evaluated at the beginning of every LD network.
- **External access to local vars of function blocks**: if this option is NOT checked, the local variables of a function block, are considered private and accessible only inside the function block. (standard IEC behavior). If this option is checked, the local variables of a function block are considered as IN/OUT variables; so they are visible and accessible also from the caller of the function block instance.

Working with Libraries

General Information

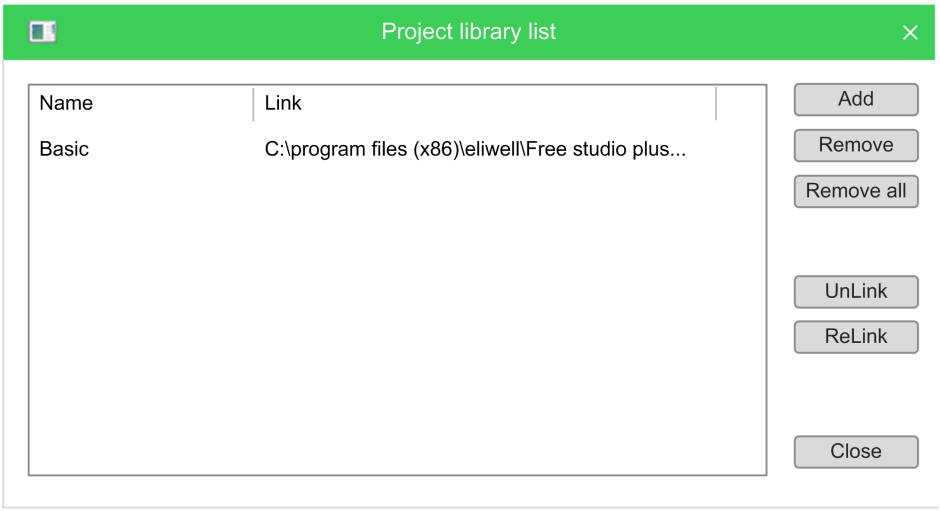
Libraries are a powerful tool for sharing objects between the projects. Libraries are stored in dedicated source file, whose extension is **.pll**.

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:

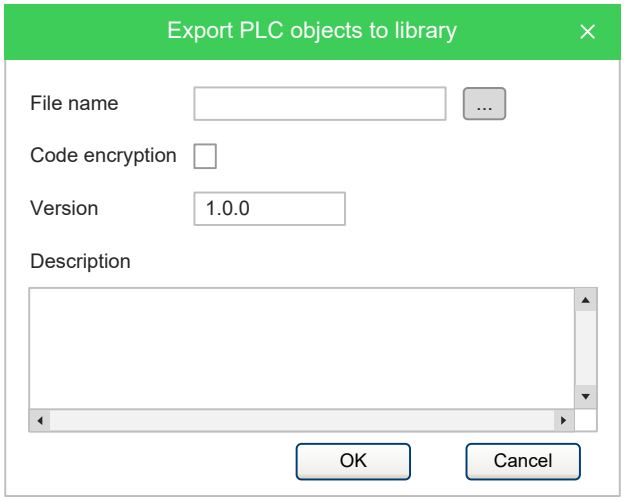
Step	Action
1	Click **Project** > **Library manager** menu of the **Programming** main window, which opens the **Library manager** dialog box.
2	Select the library you wish to remove by clicking its name once. The **Remove** button is now enabled.
3	Click the **Remove** button, which causes the reference 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, FREE Studio Plus 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.

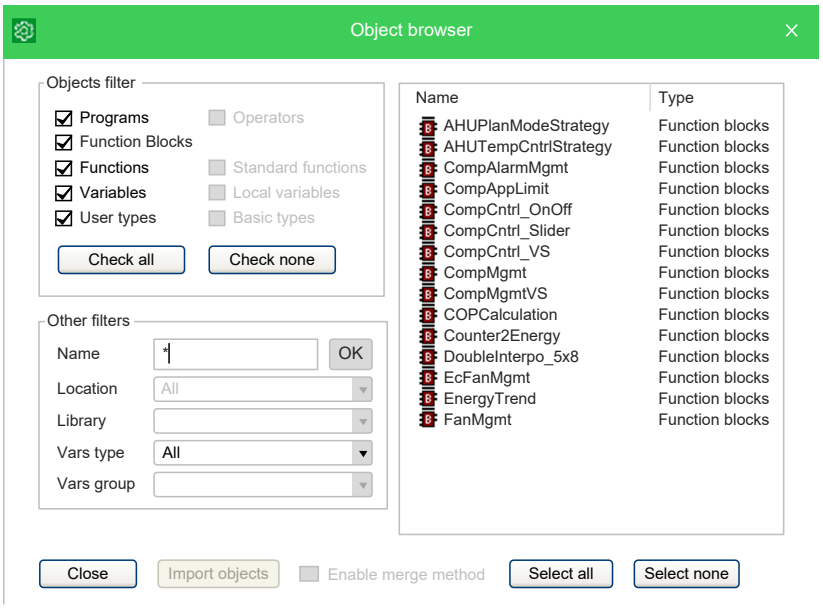
Importing from a Library or Another Source

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: 
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 FREE Studio Plus should keep when encountering those problems by setting the corresponding environment options, page 40.

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	✓	✓	✓	-

Behavior		Ask	Automatic	Take from library	Do nothing
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 FREE Studio Plus 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, FREE Studio Plus 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 FREE Studio Plus:

Step	Action
1	Click **Project > Refresh all libraries**.
2	If the file is correct, FREE Studio Plus updates the linked library content and displays a successful message in the output window, otherwise no modifications are made on the existing linked library.

Managing Project Elements

What's in This Chapter

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, page 115		Initial program.
Var1, page 122	-	Local variable.
Ungrouped_vars, page 118		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, page 125	-	Creates tasks.

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

Program Organization Units

Overview

Description

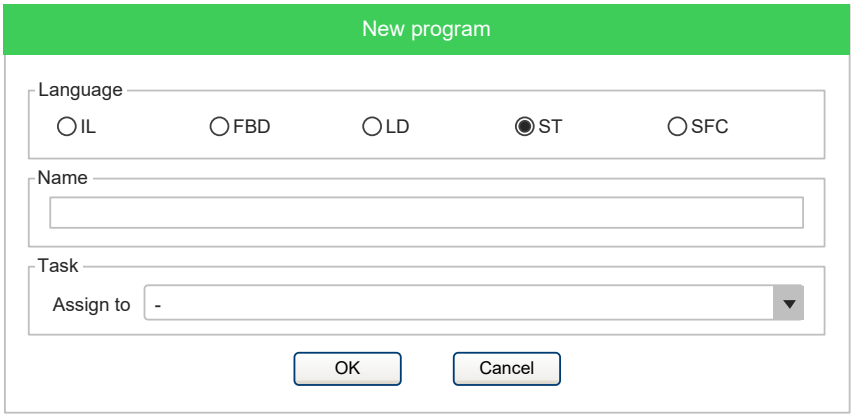
A POU is a Program Organization Unit of type Program, Function, or Function block.

This section presents how to add, edit, and remove POUs in the project.

Creating a Program

Description

To create a program:

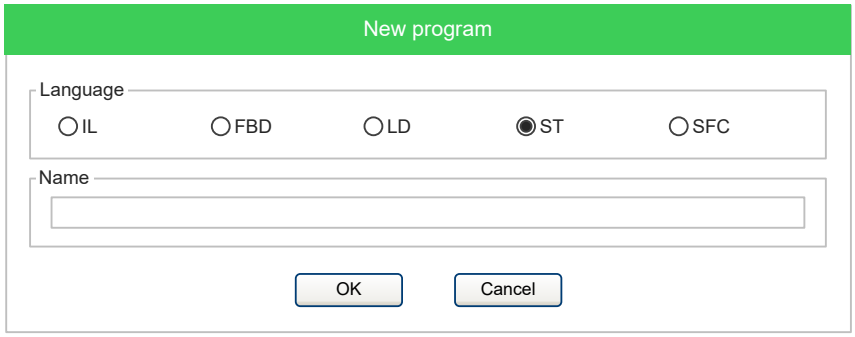
Step	Action
1	Select the target in the **Project Window** tree view. Click **Project** > **New Object** > **New program**.
2	A dialog box is displayed:  Select the specific language.
3	Enter its name.
4	Optionally, select a task in the list to associate the program to the selected task. **NOTE:** If 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, page 125 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, page 141.

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 POUs

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 POUs

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 POUs 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 POUs

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

All POUs are encrypted with the same password.

Decrypt all POUs

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

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.

Global Variables

Description

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

Classes of Global Variables

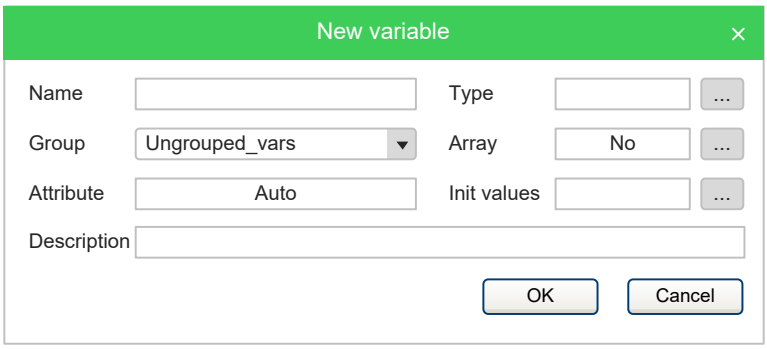
Global variables are organized in special folders of the project tree called **Global variables group**.

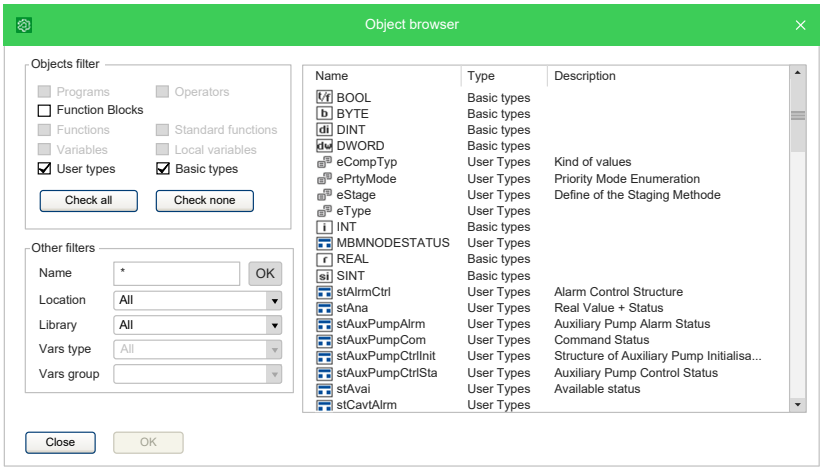
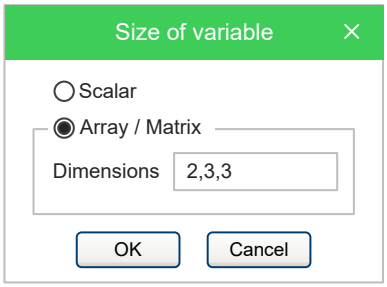
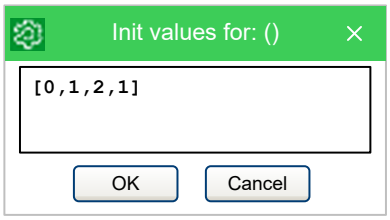
Those variables are classified according to their properties as:

- 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.

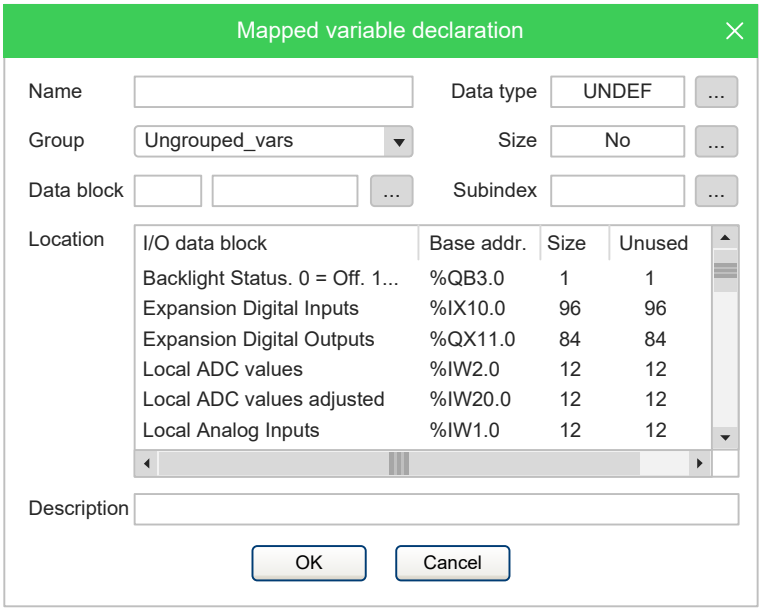
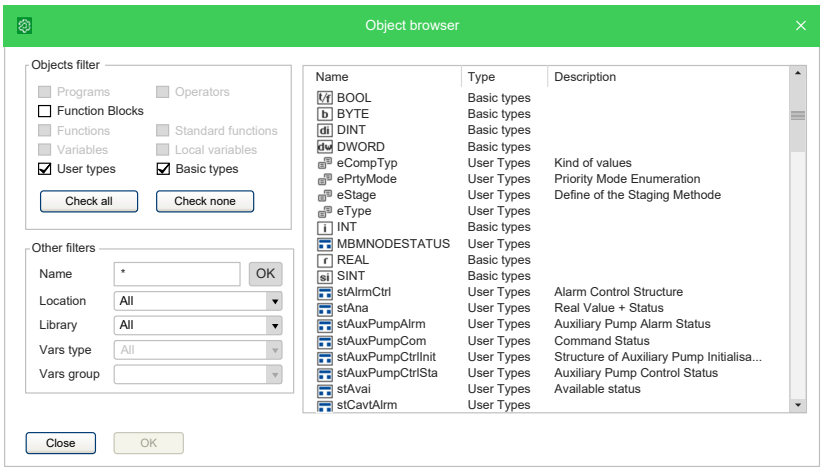
Creating a New Global Variable

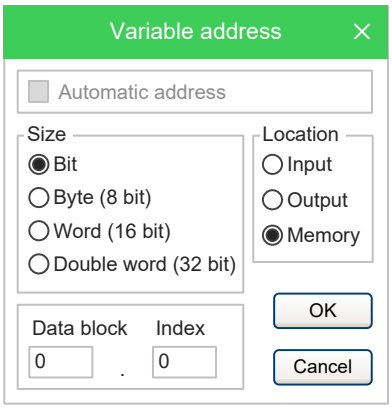
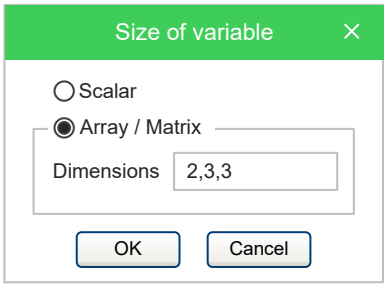
Creating a New Global Variable

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, page 141. **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.
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.

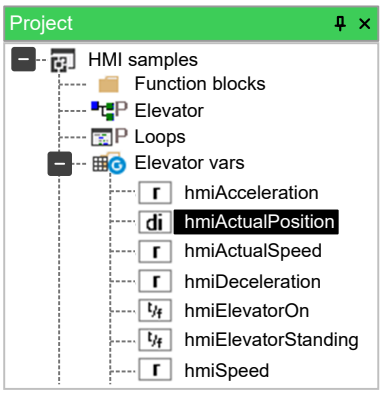
Creating a New Global Mapped Variable:

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: 
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.
5	You are required to specify the address of the variable by doing one of the following operations:

Step	Action
	• Click the **Data block**  browser button to open the editor of the address; then enter the desired value:  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:  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.

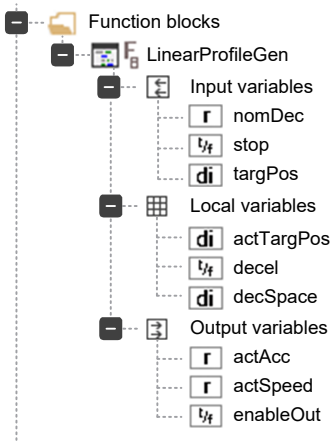
Confirm the operation to delete the variable.

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, page 167.

Creating Local Variables

Click **Variables > Insert** command or click the **Insert record** icon in the **Project** toolbar. You can also create multiple variables, page 124.

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

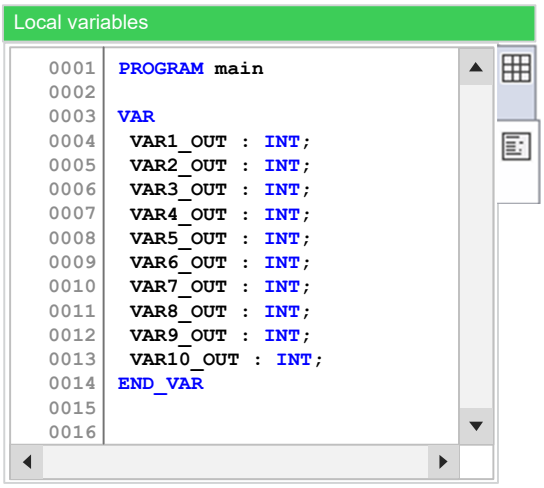
	Name	Type	Address	Array	Init value	Attribute
1	VAR1_OUT	INT	Auto	No		...
2	VAR2_OUT	INT	Auto	No		...
3	VAR3_OUT	INT	Auto	No		...
4	VAR4_OUT	INT	Auto	No		...
5	VAR5_OUT	INT	Auto	No		...
6	VAR6_OUT	INT	Auto	No		...
7	VAR7_OUT	INT	Auto	No		...
8	VAR8_OUT	INT	Auto	No		...
9	VAR9_OUT	INT	Auto	No		...
10	VAR10_OUT	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 local variables appear in the **Project** window, below the program folder, identified by an icon.

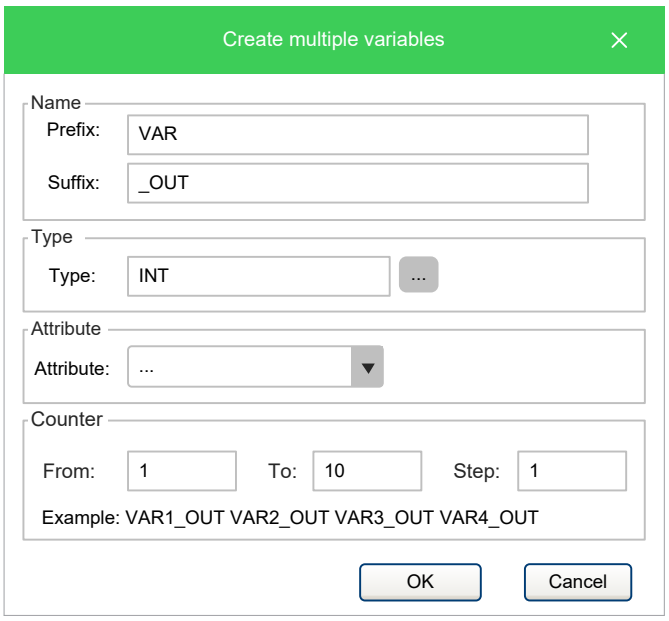
NOTE: The local variables are reinitialized each time the POU is executed.

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 when you click the  **Browse** button.

Step	Action
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.

Tasks

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.

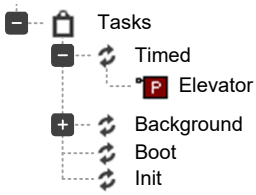
- **Timed** task runs at regular intervals which you can set. The default setting depends on target type.

  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 FREE Smart).

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

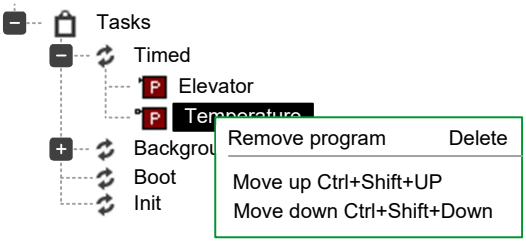
You can assign more than one program to a task.

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:



Task Configuration

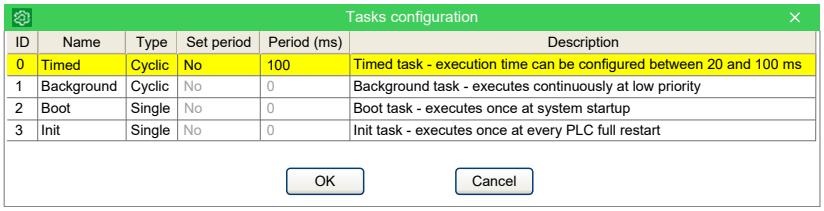
Description

Depending on the target device, it is possible to modify settings of the controller tasks.

Task Configuration

To configure a task:

Step	Action
1	Right-click on the tasks element from the project tree.
2	Select **Task configuration** in the contextual menu.

Step	Action
	Result: The **Tasks configuration** window is displayed: 
3	Select **Yes** in **Set period** list.
4	Enter a new value of period of the task.
5	Click **Ok** to validate

NOTE: For all other targets than FREE Advance, the duration of timed task can be set on **Configuration**, clicking the target name on the tree. There is a checkbox **Set execution time** in the main window.

Derived Data Types

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, page 127
- Structures, page 129
- Enumeration, page 131
- Subranges, page 132

Typedefs

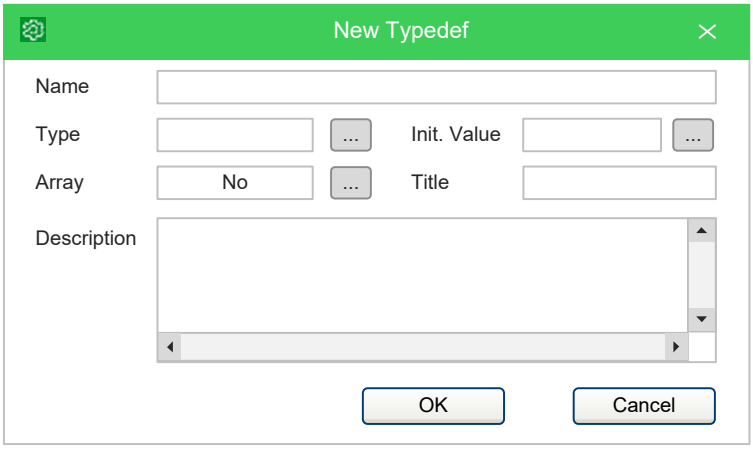
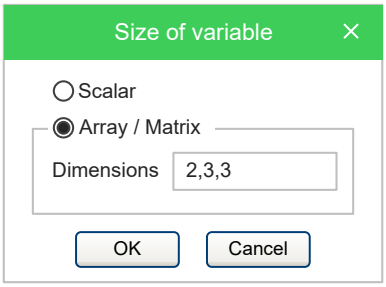
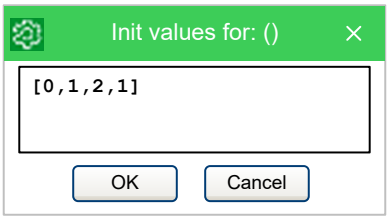
Description

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

For more information about **TypeDefs**, refer to TypeDefs description, page 249.

Creating a New Typedef

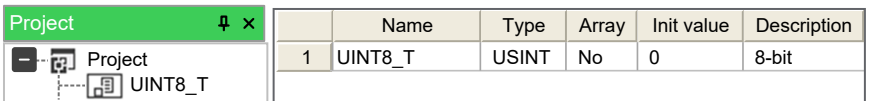
To create a new **TypeDef**:

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.
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 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:  **NOTE:** Initial values must be separated by a comma.

Step	Action
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:



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

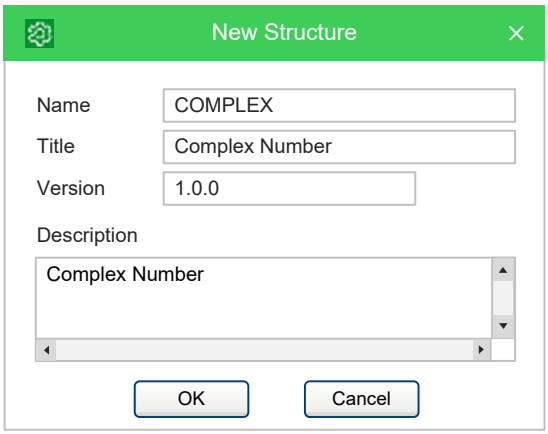
Description

The following paragraphs present how to manage structures.

For more information about **Structure**, refer to Structures description, page 250.

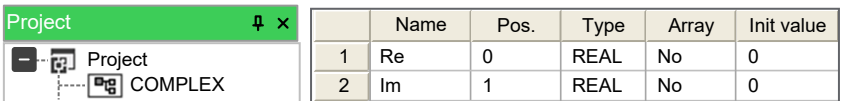
Creating a New Structure

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: 
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.



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

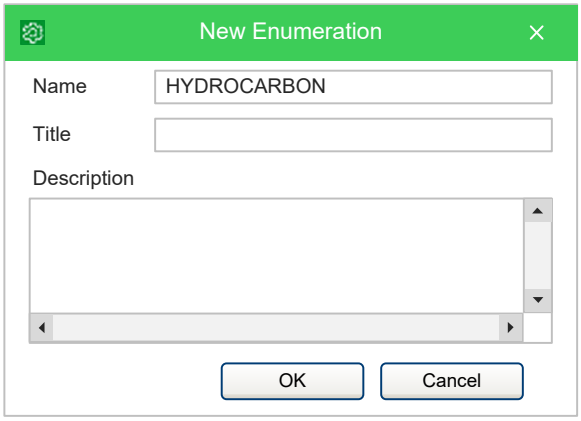
Description

The following paragraphs show you how to manage enumerations.

For more information about **Enumeration**, refer to Enumerated Data Types, page 249.

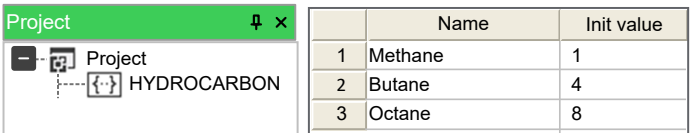
Creating a New Enumeration

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**.
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

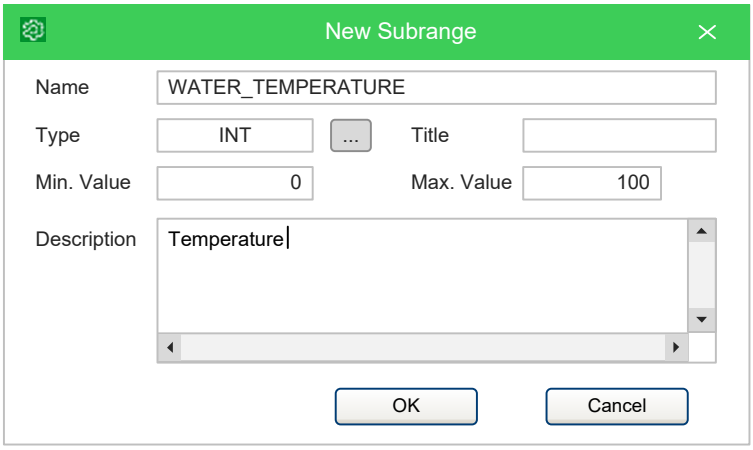
Description

The following paragraphs present how to manage subranges.

For more information about **Subrange**, refer to Subranges description, page 250.

Creating a New Subrange

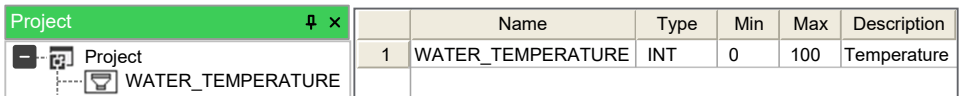
To create a new **Subrange**:

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.

Step	Action
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.



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**.

Browse the Project

Overview

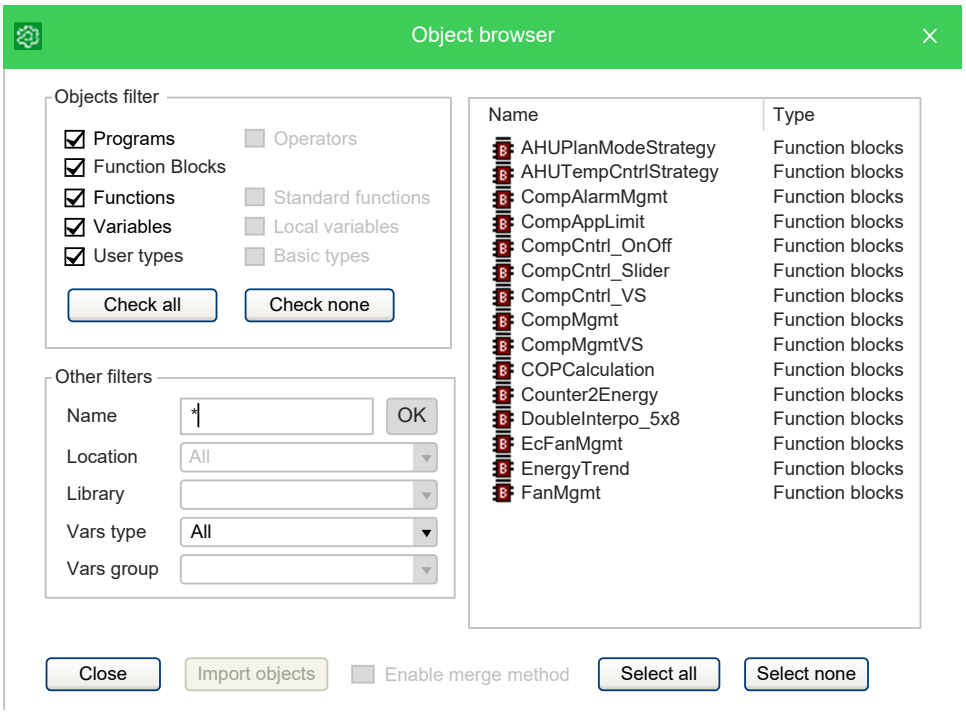
Description

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:

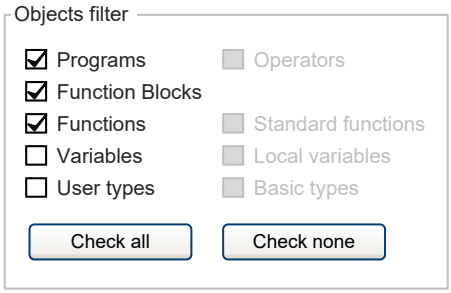
  For example, to add a program to a task, right-click a task item in the **Project** window and select **Add program** command. It opens the **Object browser** window.

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:

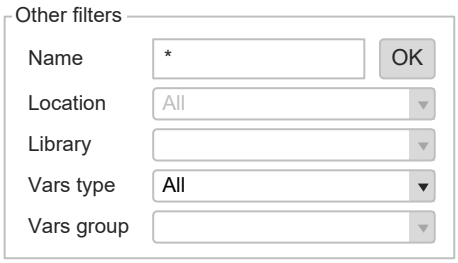


This is the main filter of the **Object browser**. You can select one of the available (enabled) object items.

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:



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, Anlog_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.

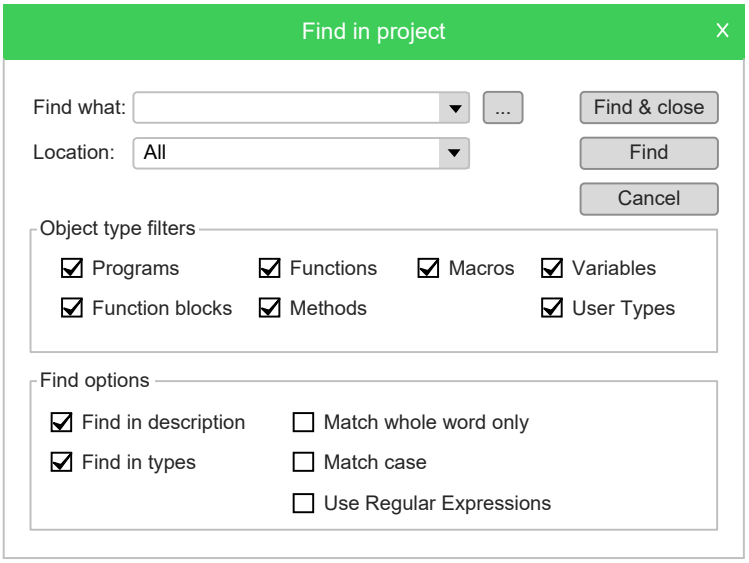
Search with the Find in Project Command

Description

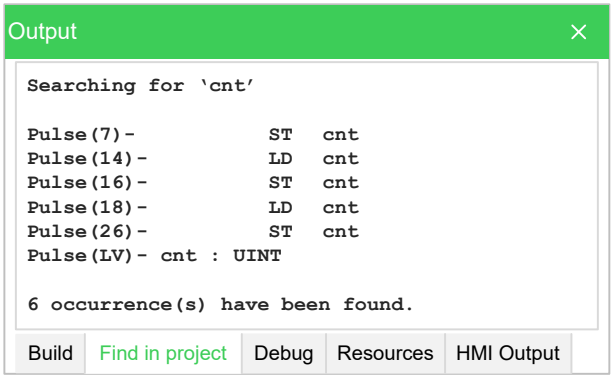
The **Find in project** command retrieves all the instances of a specified character string in the project.

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.

Step	Action
4	Tick the relevant options check boxes in the **Find options** frame.
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. 

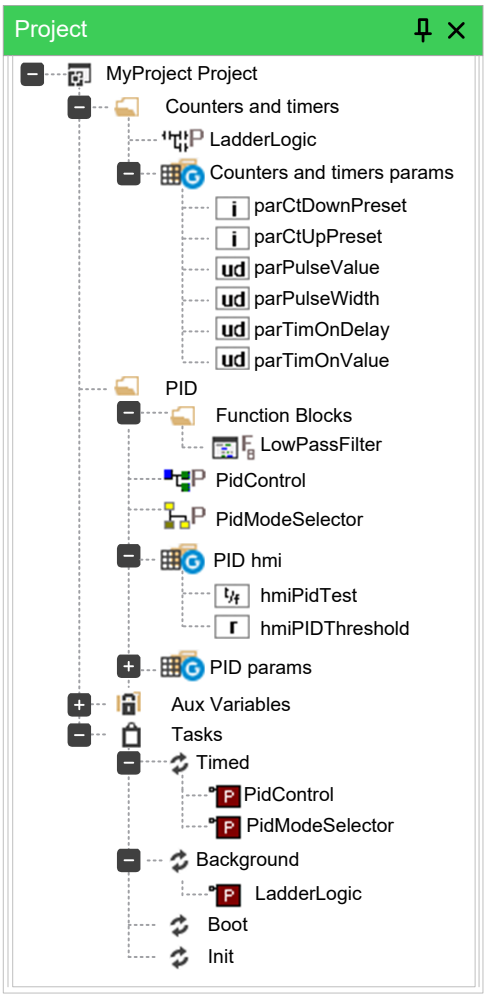
Project Custom Workspace

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, page 101.

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 POUs 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 POUs, 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, page 111.

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, page 166.

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, page 115.

- 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, page 161.

- **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, page 125.

Editing the Source Code

What's in This Chapter

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), page 143.
- Function Block Diagram (FBD), page 145.
- Ladder Diagram (LD), page 150.
- Structured Text (ST), page 159.
- Sequential Function Chart (SFC), page 161.

The editors, both graphical and text one, support tooltips. By enabling them, page 39, **Programming** shows some information about symbols on mouse-over.

Instruction List (IL) Editor

Overview

Description

The IL editor allows you to write code and modify POUs 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

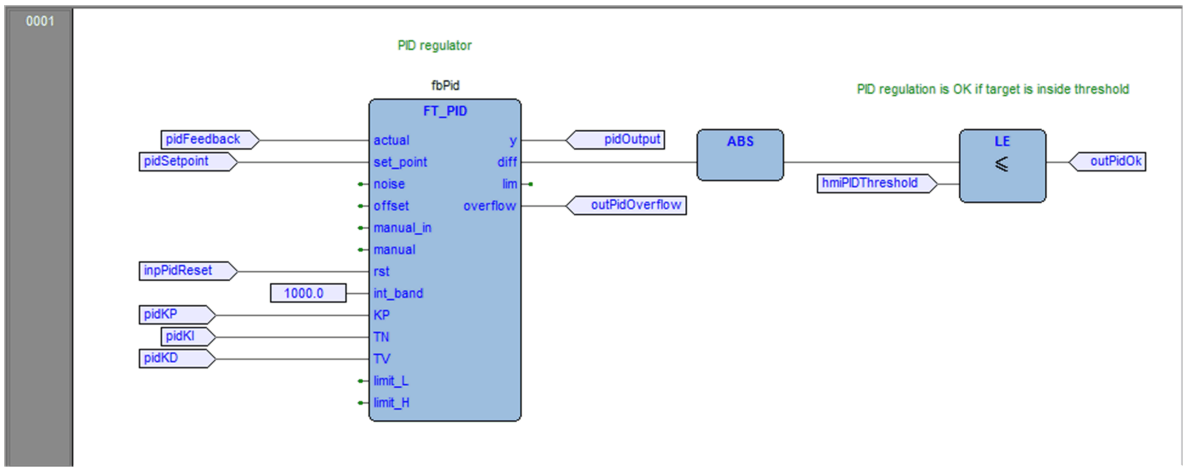
Move the cursor to anywhere on the line containing the bookmark, then press **Ctrl +F2**.

Function Block Diagram (FBD) Editor

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, page 274.

Creating a New FBD Document

Description

For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit, page 115.
- Creating a Function Block/Function, page 115.
- Editing POUs, page 116.

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**.

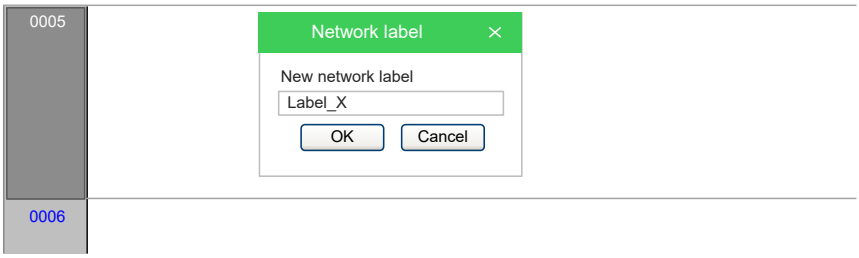
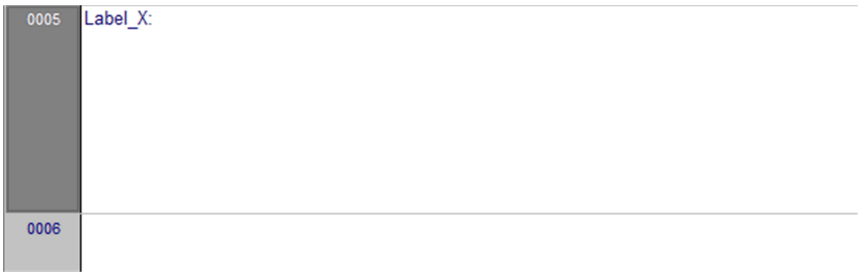
Labeling Networks

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.

Step	Action
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, page 97

  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.

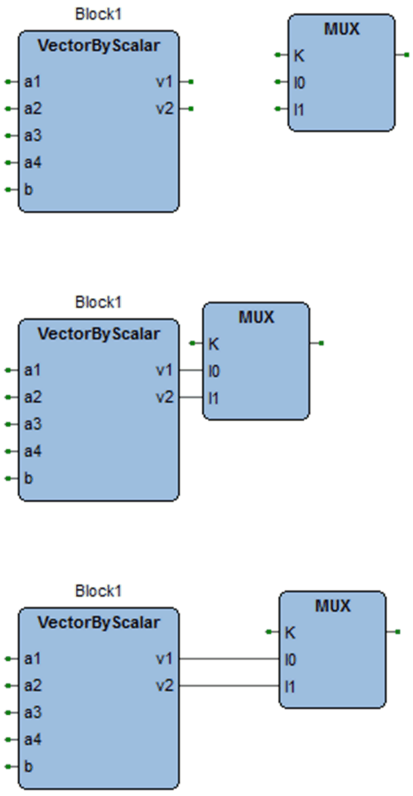
Connecting Blocks

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, page 155 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

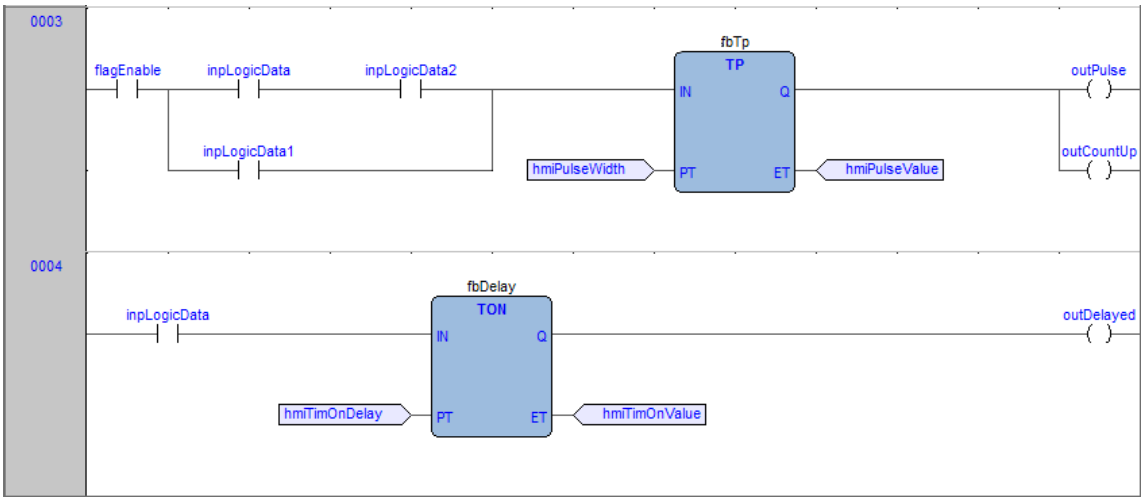
The FBD editor also automatically displays the location of compiler errors. To reach the block where a compiler error occurred, double-click the corresponding error line in the **Output** bar.

Ladder Diagram (LD) Editor

Overview

Description

The LD editor allows you to code and modify POUs using Ladder Diagram (LD):



For more details, refer to Ladder Diagram language reference, page 278.

Creating a New LD Document

Description

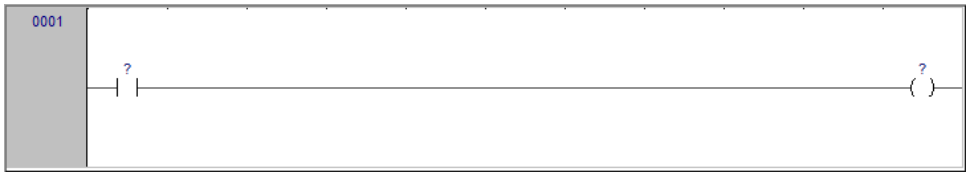
For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit, page 115.
- Creating a Function Block/Function, page 115.
- Editing POUs, page 116.

Adding/Removing Networks

Description

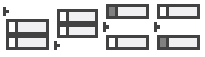
Each POU coded in LD consists of a sequence of networks. A network is defined as the set of interconnected graphic elements. The upper and lower bounds of every network are fixed by two straight lines while each network is delimited on the left by a gray area containing the network number.



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:

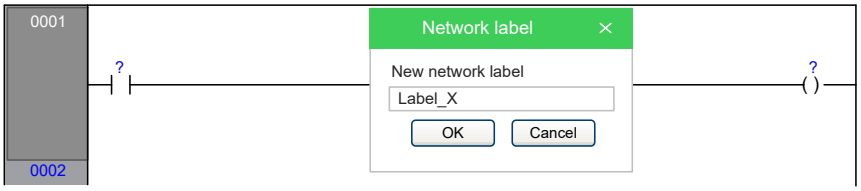
- To add a new blank network, click **Scheme > Network > New**, or click one of the equivalent buttons  in the **Network** toolbar.
- To display a background grid which helps you to align objects, click **View > Grid**  .
- To add a comment, click **Scheme > Object > New Comment**  or press **Shift+M**.

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

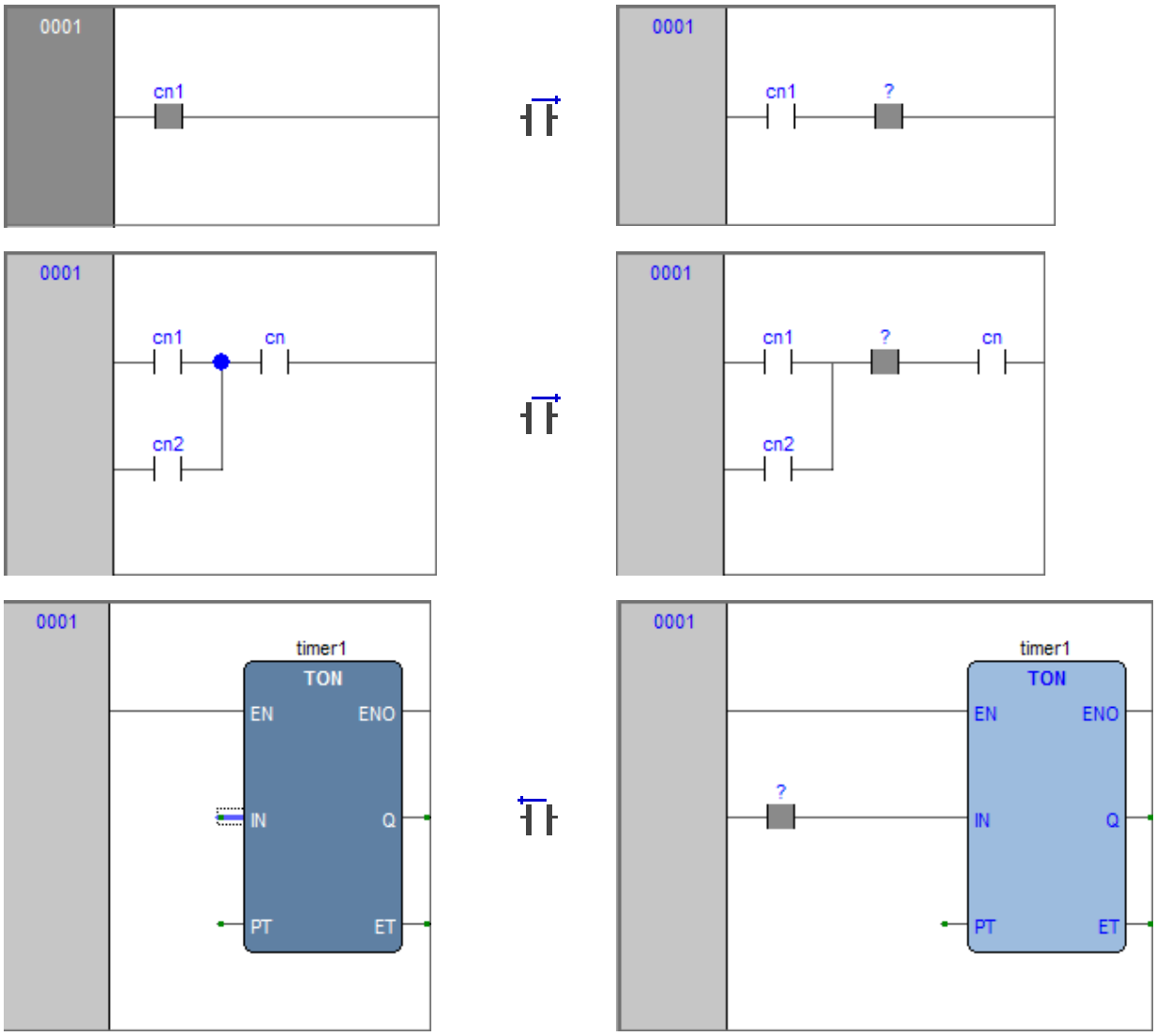
Description

To insert new contacts on the network, apply one of the following procedures:

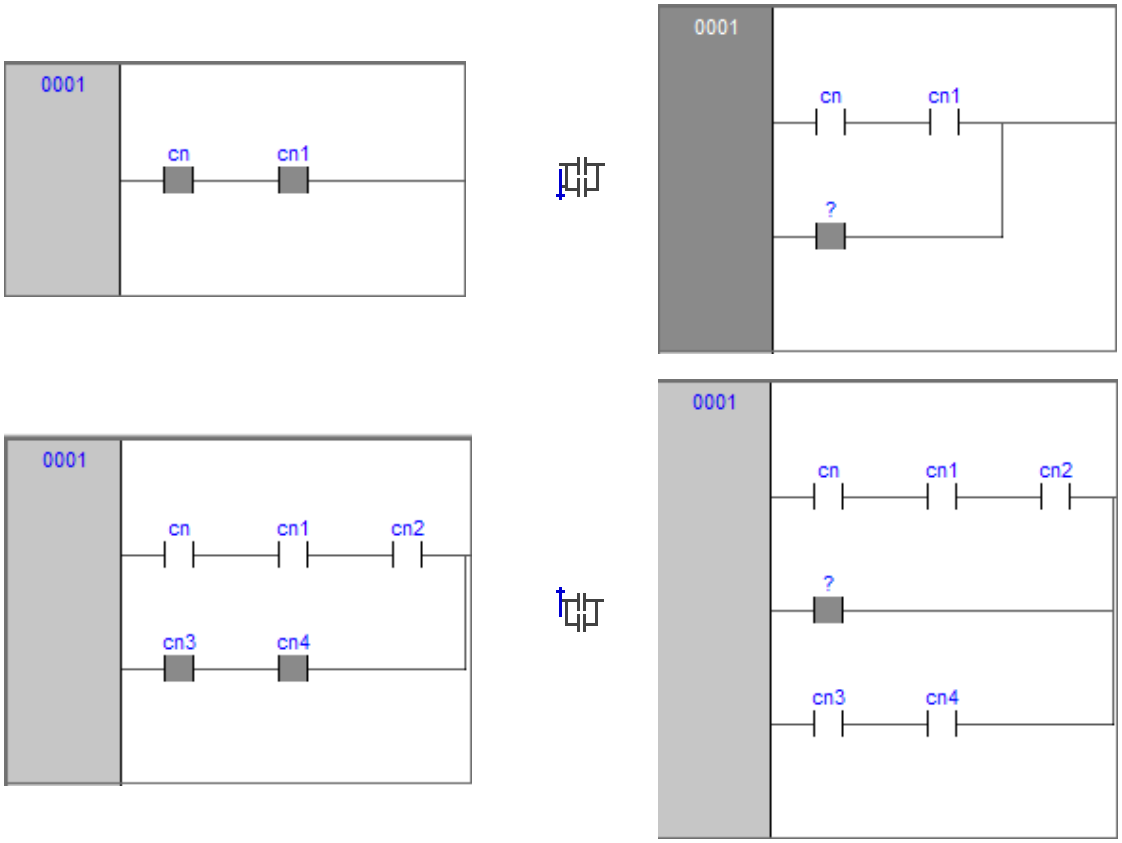
- Drag a boolean variable to the desired place over an object. For example, global variables can be taken from the **Workspace** window, whereas local variables can be selected from the local variables editor. Contacts inserted with drag and drop will always be inserted in series after the destination 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



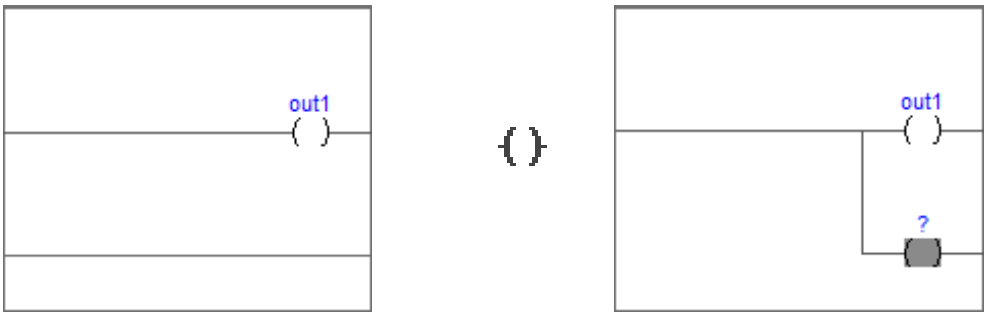
Inserting Coils

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.

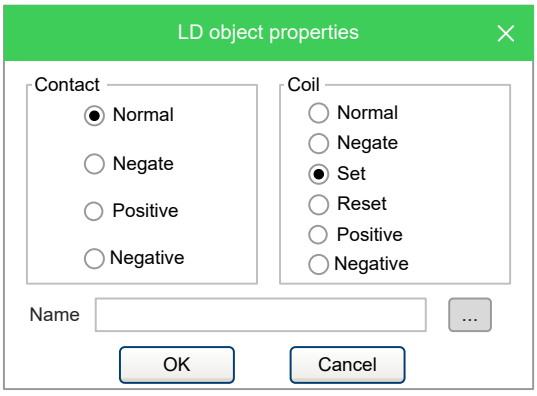
Editing Coils and Contacts Properties

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.

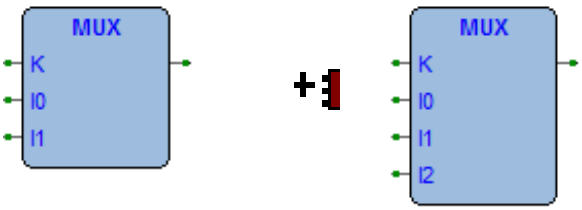
Adding, moving, deleting, or copy/pasting objects will automatically recalculate the layout of the network objects; because of this, it is not possible to manually “draw” connection lines or freely placing objects without connecting them to the network.

Modifying Properties of Blocks

Description

- Click **Scheme > Increment pins** to increment the number of input pins of some operators and embedded functions.

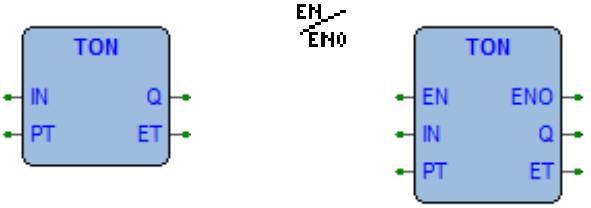
  NOTE: You can also remove pins by clicking **Decrease pins** .



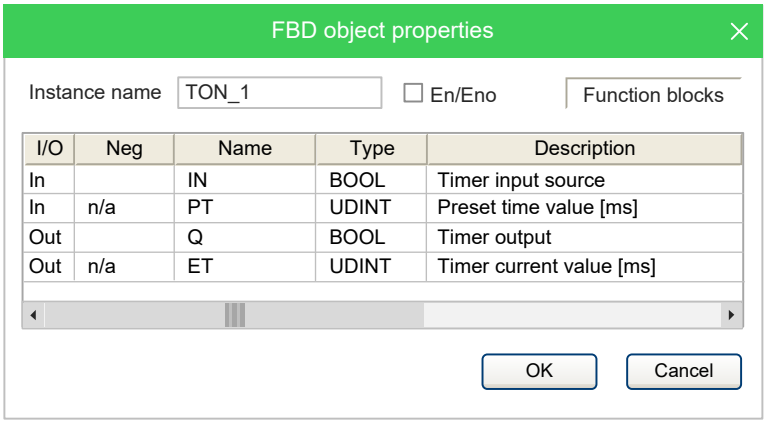
- Click **Scheme > Enable EN/ENO pins** to display the enable input and output pins.

  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.



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.

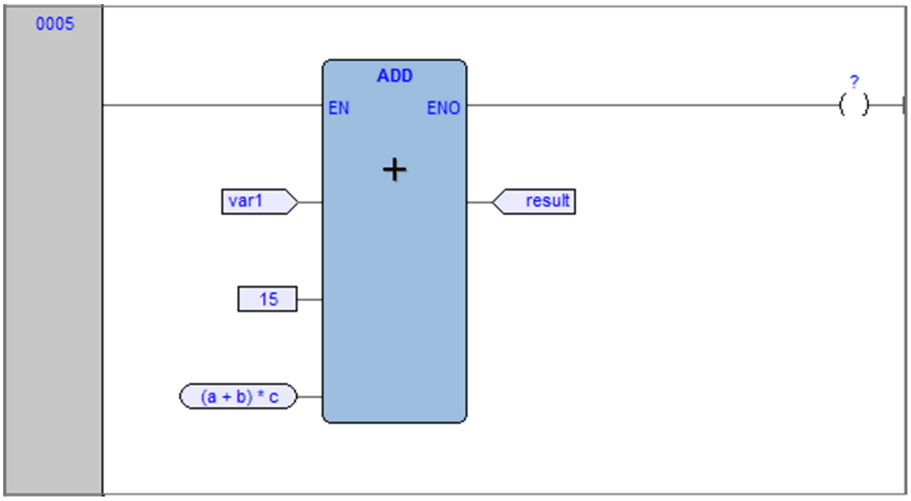
Inserting Expression

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)
```

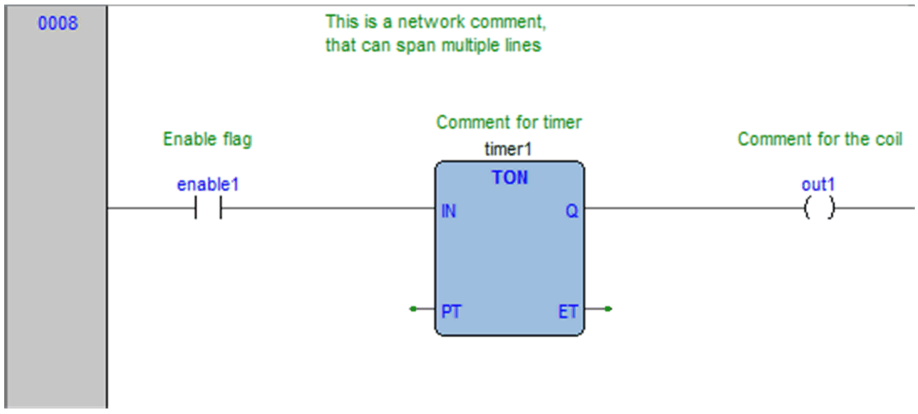


Comments

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.



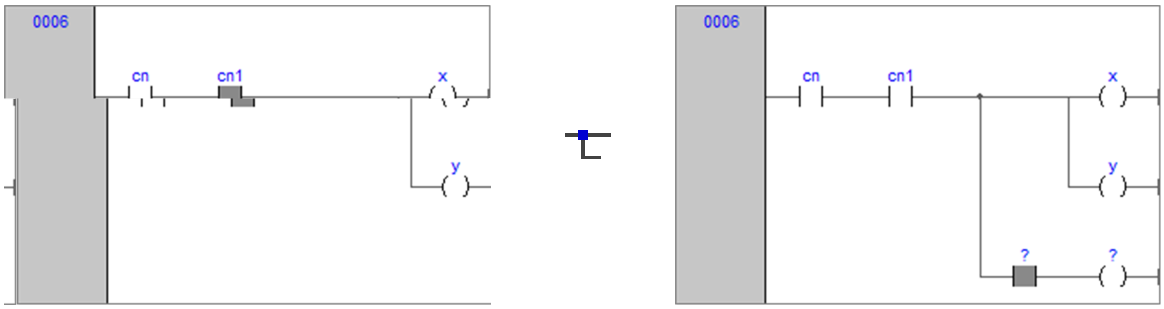
Branches

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.

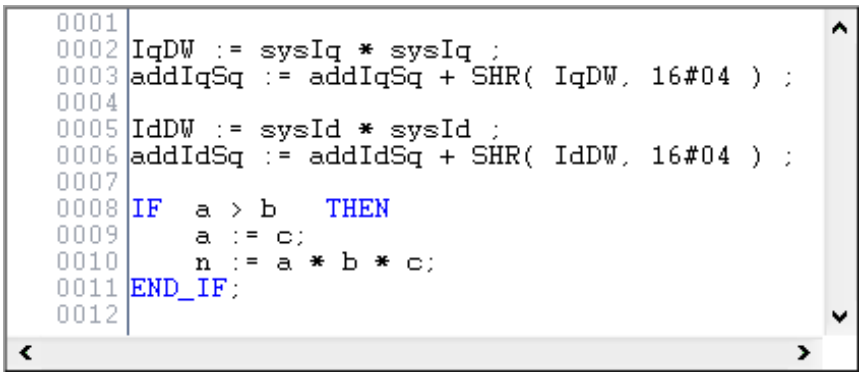


Structured Text (ST) Editor

Overview

Description

The ST editor allows you to code and modify POUs using Structured Text (ST):



For more details, refer to Structured Text language reference, page 280.

Creating and Editing ST Objects

Description

For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit, page 115.
- Creating a Function Block/Function, page 115.
- Editing POUs, page 116.

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**.

Sequential Function Chart (SFC) Editor

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, page 98.

For more details, refer to Sequential Function Chart language reference, page 289.

Creating a New SFC Document

Description

For creation and modification of FBD documents, refer to:

- Creating a Program Organization Unit, page 115.
- Creating a Function Block/Function, page 115.
- Editing POUs, page 116.

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

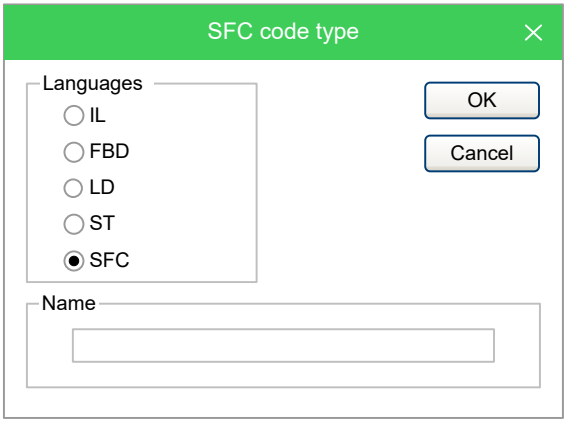
This paragraph explains how to implement an action and how to assign it to a step.

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:



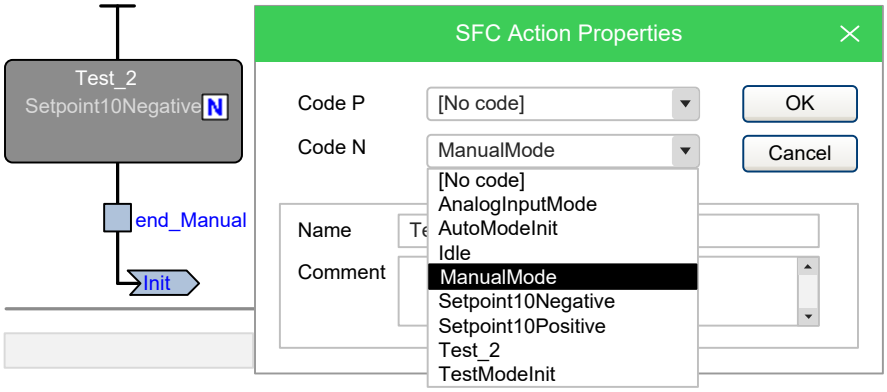
Select one of the languages and type the name of the new action in the text box at the bottom of the dialog box. Then either confirm by clicking **OK**, or quit by clicking **Cancel**.

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

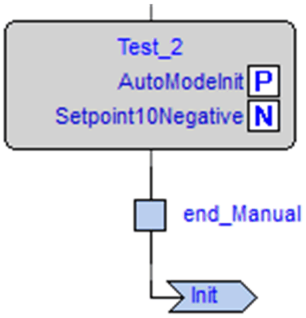
When you have finished writing the code, double-click the step you want to assign the new action to. This causes the following dialog box to appear.



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.



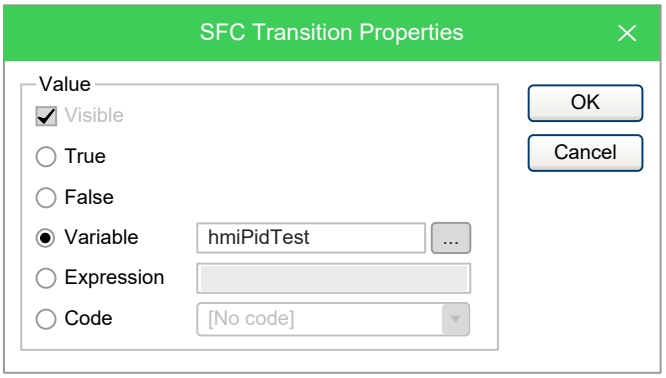
If later you need to edit the source code of the action, you can double-click these letters. Alternatively, you can double-click the name of the action in the **Actions** folder of the **Workspace** window.

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:



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

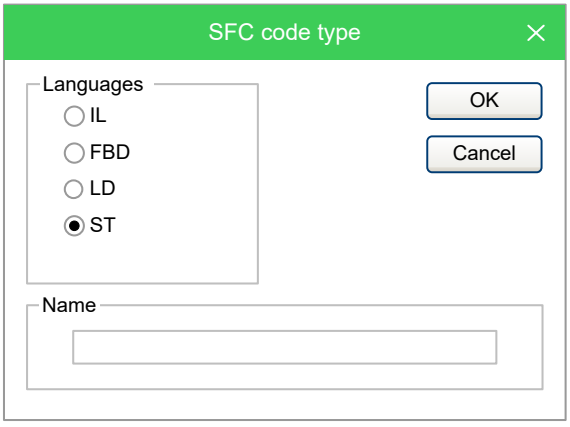
This paragraph explains how to specify a condition through a piece of code, and how to assign it to a transition.

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:



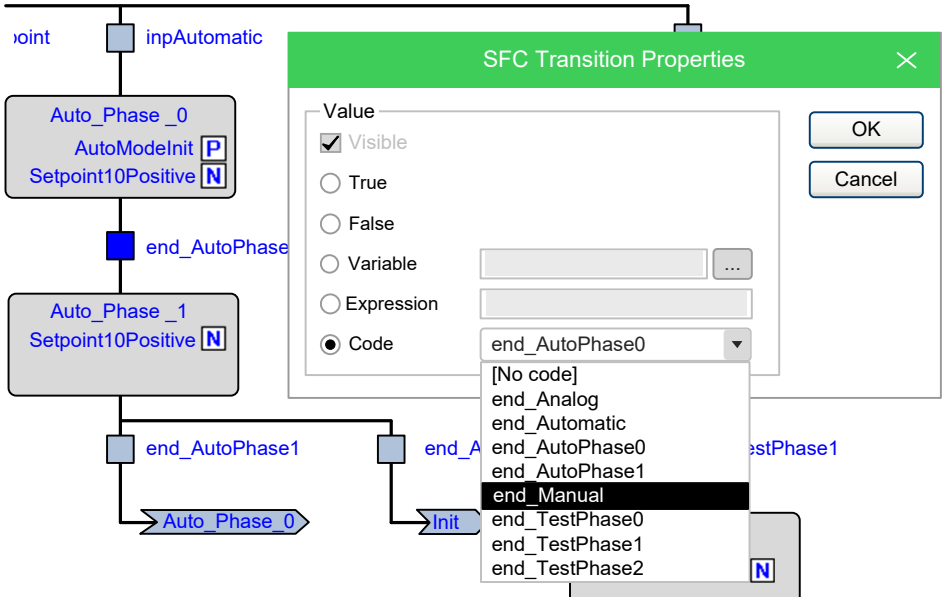
Select one of the languages and type the name of the new condition in the text box at the bottom of the dialog box. Then either confirm by clicking **OK**, or quit by clicking **Cancel**.

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:



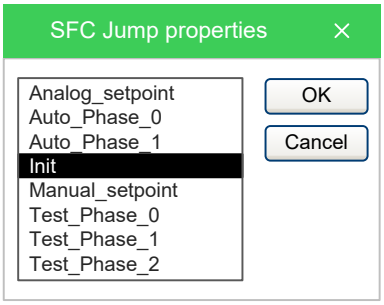
Select the name of the condition you want to assign to this step. Then confirm by clicking **OK**.

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.

Variables Editor

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

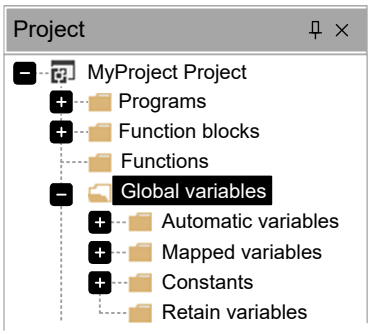
The corresponding source code is represented like this:

```
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:

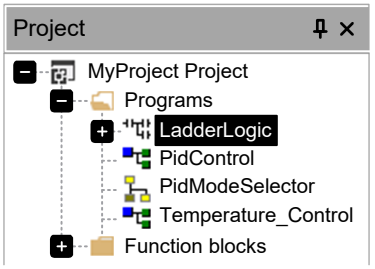


	Name	Type	Address
1	parCtDownPreset	UDINT	Auto
2	parCtUpPreset	UDINT	Auto
3	parPulseValue	UDINT	Auto
4	parPulseWidth	UDINT	Auto
5	parTimOnDelay	UDINT	Auto
6	parTimOnValue	UDINT	Auto
7	pidKD	REAL	Auto
8	pidKI	REAL	%MD1

NOTE: If you use the customizable workspace, page 101, 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:



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, page 101, 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

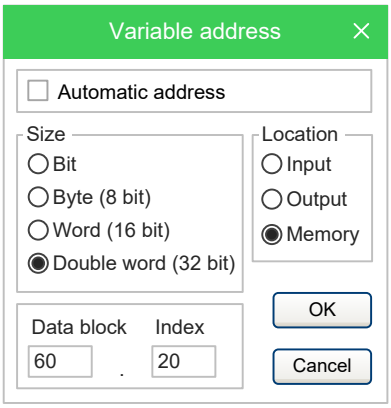
Follow this procedure to edit the declaration of a variable in a variables editor (the following steps are optional and you will skip most of them when editing a variable):

Step	Action
1	Edit the name of the variable by entering the new name in the corresponding cell:
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:
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:
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:
5	Choose whether a variable is an array or not. If it is, edit the size of the variable:

	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

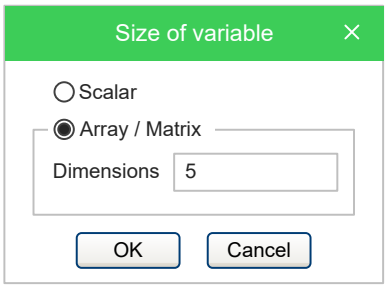
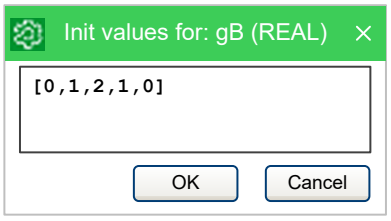
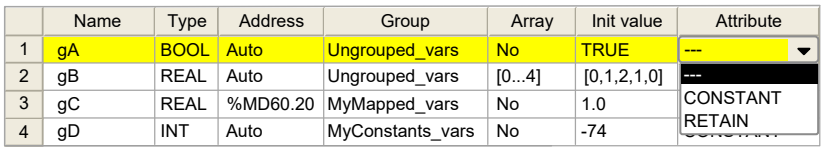
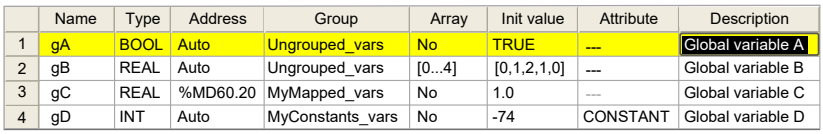
	Name	Type	Address	Group
1	gA	BOOL	Auto	Ungrouped_vars
2	gB	REAL	Auto	Ungrouped_vars
3	gC	REAL	%MD60.20	MyMapped_vars
4	gD	INT	Auto	MyConstants_vars

	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



	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	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

Step	Action
	
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: 
7	Assign an attribute to the variable (for example, *CONSTANT* or *RETAIN*), by selecting it from the list which opens when you click the corresponding cell: 
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: 
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_iAI	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, page 39 and Merge Function, page 112 for further details.

Creating an Error Variable

Description

Error variables are implemented only in the FREE Optima controllers.

Create an error variable by choosing from the error messages available:

sysErrMsg	**Error ID**	**Error Message**	**Blink Mode**
sysErrMsg0	-1000	OUT OF RANGE (1)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg1	-32768	PROBE ERROR (1)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg2	-32767	REMOTE PROBE ERROR (1)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg3	-32766	PROBE CONFIG ERROR (1)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg4	-32765	AI PAIRS ERROR (1)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg5	-32764	- (2)	0 = Blink OFF (3) 1 = Blink ON

sysErrMsg	Error ID	Error Message	Blink Mode
			2 = Blink Fast
sysErrMsg6	-32763	- (2)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast
sysErrMsg7	-32762	- (2)	0 = Blink OFF (3) 1 = Blink ON 2 = Blink Fast

(1) Editable default message (max 20 characters). The meaning of the error does not change but the text message displayed can be changed.

(2) Default empty string, to be modified with the error you want to display (max 20 characters)

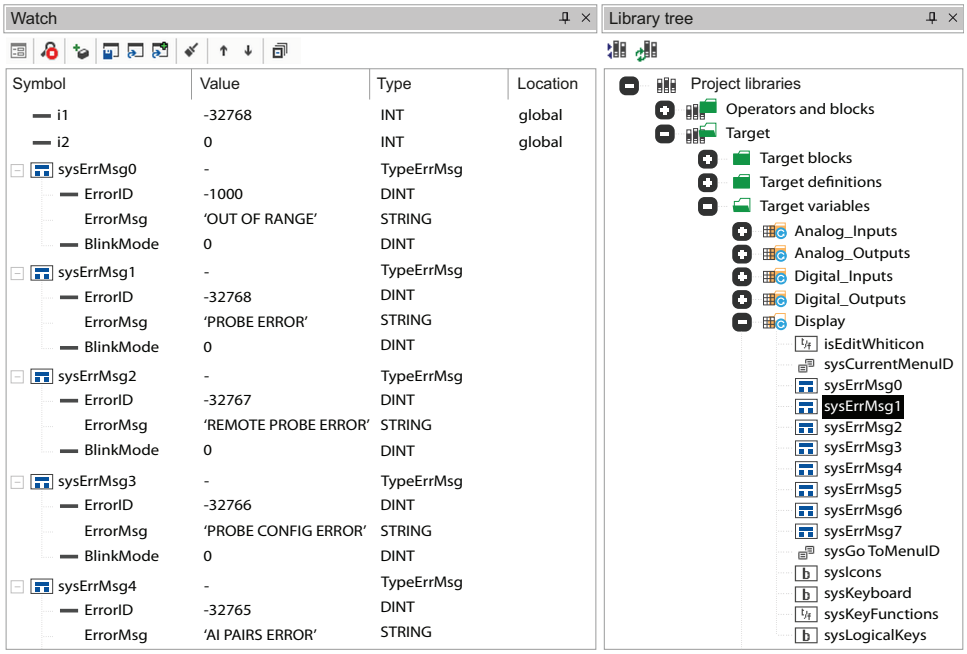
(3) Default Blink Mode

The Error Message is displayed when the variable to be shown has a value equal to the Error ID.

NOTICE

Do not modify the Error ID value. Error ID represents the value associated with the sysErrMsg.

Failure to follow these instructions can result in equipment damage.



To insert a new error message:

- In the Library Tree click **Target > Target Variables > Display** and choose the appropriate *sysErrMsg*.
- Load *sysErrMsg* on the Watch Window (Refer to Watch Window, page 194 for details) and modify, if necessary, the description of the error you want to display with double-click on the ErrorMsg in the Value column.
- Modify the parameter with the value of the Error ID associated with the chosen *sysErrMsg*.

Compiling

What's in This Chapter

Overview

Description

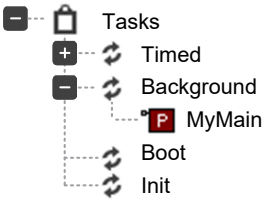
Compilation consists of transforming the PLC source code into another programming language (the target language) such as binary code, which can be executed by the processor on the target device.

Compiling the Project

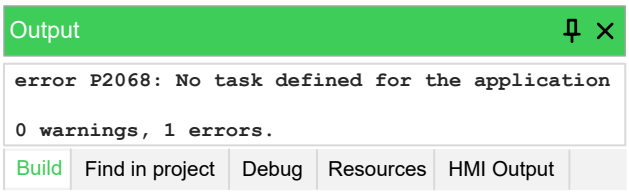
Overview

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:

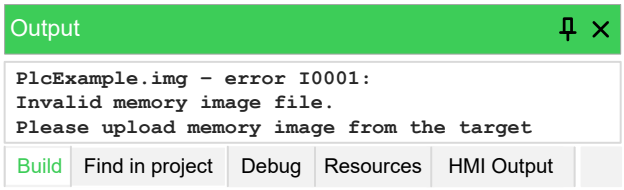
- In the menu, click **Project > Compile**.
- In the Project toolbar, click .
- Press the **F7** key.

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.



Compiler Output

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:

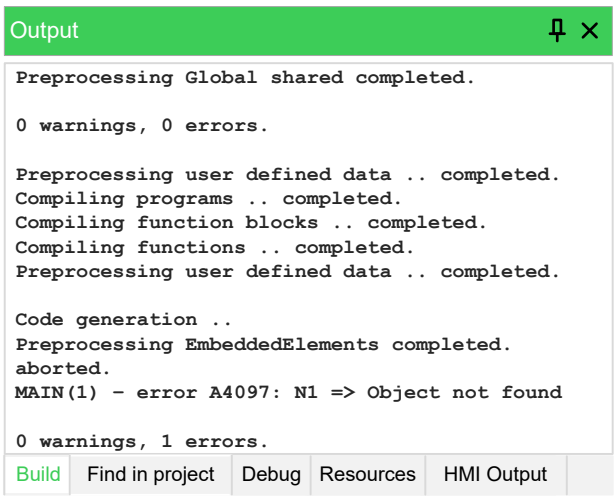
```
m warnings, n errors
```

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.

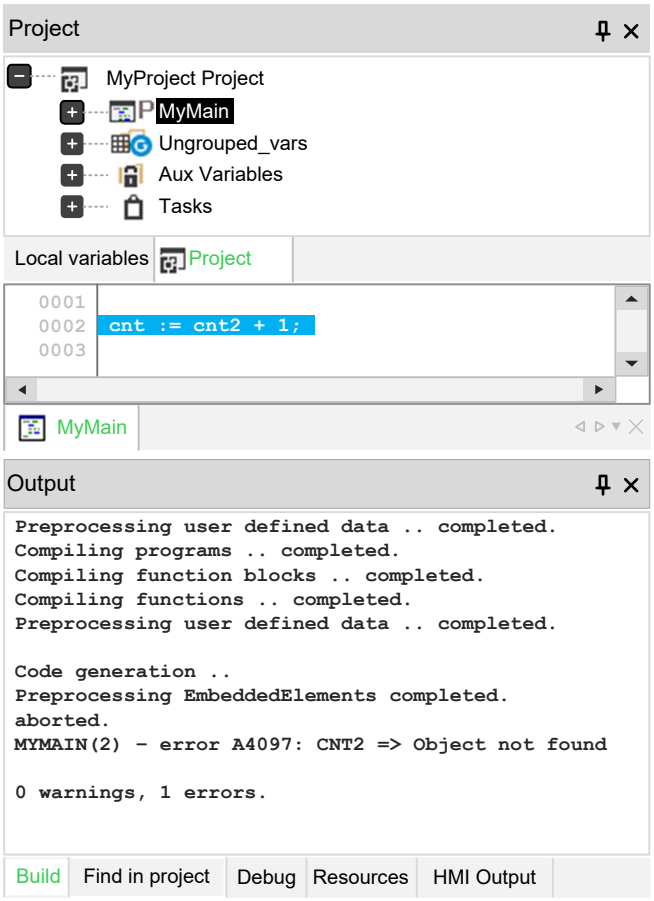


For each detected error, the information includes:

- 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, page 301.

If you double-click the error message in the **Output** window, **Programming** opens the source code and highlights the line containing the detected error:



Command-Line Compiler

Description

The compiler can be used independently from the integrated software environment: in the directory of FREE Studio Plus, 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.

Launching the Application

What's in This Chapter

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, page 194.

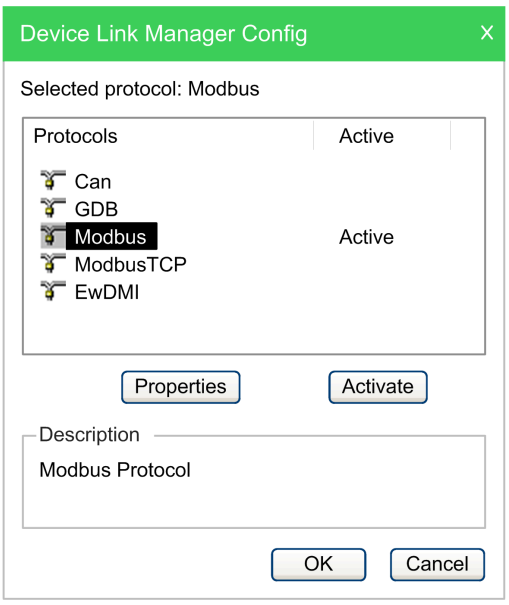
Setting Up the Communication

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, page 177

Step	Action
	• GDB, page 177 • Modbus, page 177 • ModbusTCP, page 178 • DMI, page 180
3	Click **Activate** button to active 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 with the target, page 181.

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 FREE Smart/FREE Evolution/FREE Advance/FREE Optima controllers.

The GDB protocol is reserved for internal software use.

Modbus

Select **Modbus** in the case of USB/RS-485 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 RS-485 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 RS-485 protocol selection	2 = uNET 3 = Modbus/RTU 4 = BACnet MS/TP	3	-

Parameter	Description	Values	Default	Note
Databit_RS485_OB	RS-485 data bit number On-board	8	8	Fixed at 8
Stopbit_RS485_OB	On-board RS-485 stop bit number	1 = 1 stop bit 2 = 2 stop bit	1	-
Parity_RS485_OB	On-board RS-485 protocol parity	0 = NULL 1 = ODD 2 = EVEN	2	-

Modbus TCP

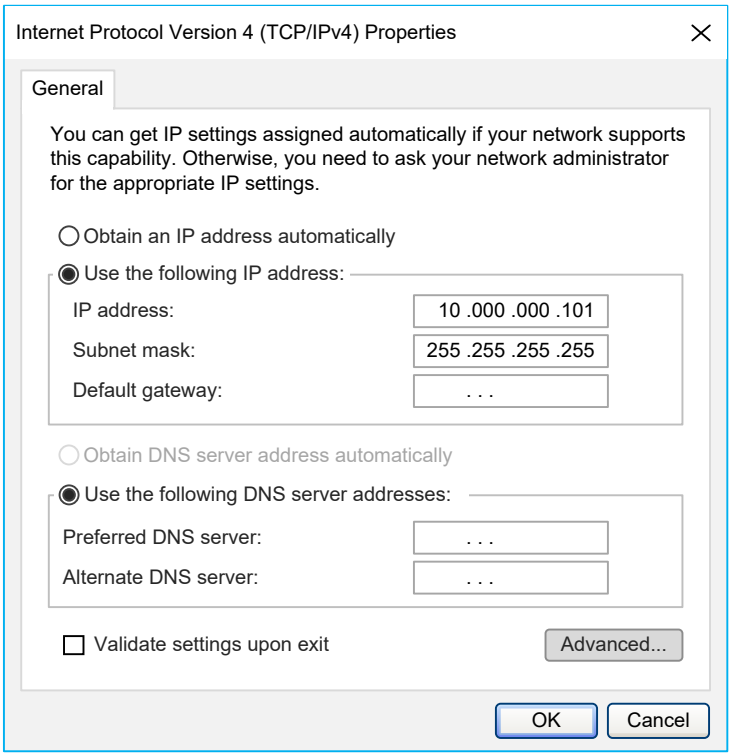
Select the **ModbusTCP** protocol in the case of Ethernet connection, using the relevant communication module if necessary.

In the protocol properties window:

- The **IP or hostname** box is for entering either an IP address (the default setting for FREE Evolution/Advance 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 FREE Evolution/Advance.

Configure the TCP/IPv4 connection in the Ethernet port properties of your PC with the address (10.0.0.101):



NOTE: The default FREE Evolution/Advance 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 FREE Evolution/Advance via the Ethernet port.

FREE Evolution/Advance has a number of BIOS parameters for managing the connection between the target and FREE Studio Plus but, unlike FREE Smart, 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

FREE Panel EVP **Specific HMI Management:**

In addition to the BIOS parameters, FREE Panel EVP 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 (RS-485 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 RS-485: 1.38400.P81
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	

Parameter	Description	Values	Default	Note
HmiList_Addr_4	Remote HMI 1 navigation resource address for TCP (IP part 4)	0…255	0	Modbus TCP: 010.000.000.100
HmiList_File_1	Remote HMI navigation file 1 (DOS 8.3 uppercase format)	Alphanumeric string, 8 characters	********	The default name is HMIREM.KBD

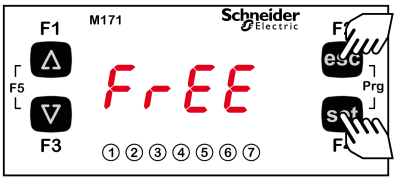
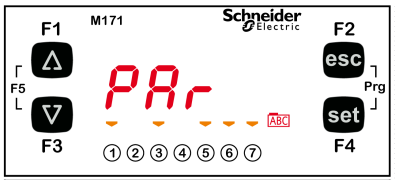
DMI

Select DMI in the case of connection with the FREE Smart with the DMI programming cable.

FREE Smart has parameters in the **CF** folder in the controller for managing the connection between the target and FREE Studio Plus.

If the target is “empty”, for example there is no controller application on the device, FREE Smart displays the message FrEE. Otherwise, a controller application exists on FREE Smart 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 FREE Smart target and FREE Studio Plus:

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 RS-485 port cannot be used simultaneously.					
(2) CF31	5 = 38400 baud (RS-485: not supported) 6 = 57600 baud (RS-485: not supported) 7 = 115200 baud (RS-485: not supported)				

For other parameters and to manage parameter visibility levels, refer to *FREE Smart 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.

Connect to a Device

Prerequisite

You have to set up the communication, page 176.

Procedure

To launch the connection to the physical device, perform one of the following operations:

- In the dedicated toolbar, click 

- In the menu, click **On-Line > Connect**.

Information Messages

Before establishing the connection, the software could display an information message:

Message	Cause	Risk	Solution
“Target Device ID indicated for the project is different from the value from the target. Proceed Anyway ?”	The firmware of the device in the application is different from the firmware of the physically connected device.	Unintended Equipment Operation	• Update the BIOS of the physically connected device. • Change the device in the application • Change the physical device.
“Could not correctly identify the connected device: found a FREEAdvance 668.11 instead. Connect anyway?”			

Click **OK** to continue the connection process.

On-Line Status

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, page 176.

- 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.

Downloading the Application

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 FREE Studio Plus 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.

Control the PLC Execution

Overview

The PLC application execution can be controlled using the related functions.

These functions are accessible in the Project toolbar, page 96 and in the On-line menu, page 29.

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 FREE Studio Plus 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, 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 FREE Studio Plus.

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**.

Simulation

What's in This Chapter

Simulation Function

Overview

Main Purpose

The main purpose of the simulation function is to execute PLC applications and HMI pages simultaneously in a simulated environment.

Simulation can simulate execution of:

- PLC applications, IEC 61131-3 (made in **Programming** tab).
- HMI pages (made in **Display** tab).

The execution can thus take place on the same PC used for the development process with the advantage of a faster and simpler testing and debugging phase because the real final hardware is not necessary.

NOTE: The simulation is not intended as a substitute for real, empirical testing during commissioning. It is a means for the programmer to submit its application, or parts of application, to unit testing and verification. Only empirical testing with live equipment in the complete application can be considered a valid mechanism for validation.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

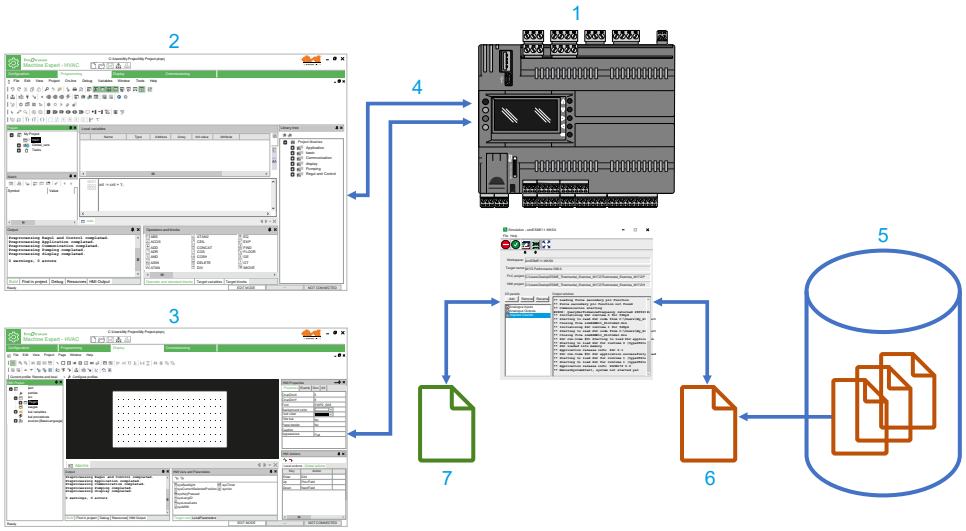
Always empirically test your application during commissioning before placing your application and associated equipment into service.

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

Simulation Environment Components

Description

The following diagram shows the main components of the simulated environment:



Item	Description	
1	Simulation	Simulation function allows you to execute PLC applications and HMI pages simultaneously in a simulated environment.
2	**Programming**	PLC development environment.
3	**Display**	HMI development environment.
4	TCP/IP Server	Assignment to the simulated controller of a local IP address for communication with development environments.
5	Catalog	Repository of the target definitions, used by the software components.
6	Target file (TGSX)	Catalog components that define the targets to simulate. These files have TGSX extension.
7	Workspace file (WKSX)	User file with WKSX extension that contains the elements of a working session of the simulator (I/O panels, source PLC, HMI project, and so on). The project can have multiple simulation workspace files, and you can manage them.

Simulation Operating Modes

Overview

Simulation is activated when the target simulation file (TGSX) is available in the catalog.

The correct TGSX file is selected automatically by the calling program, depending on the current active target in the PLC or HMI project.

Simulation has the following features:

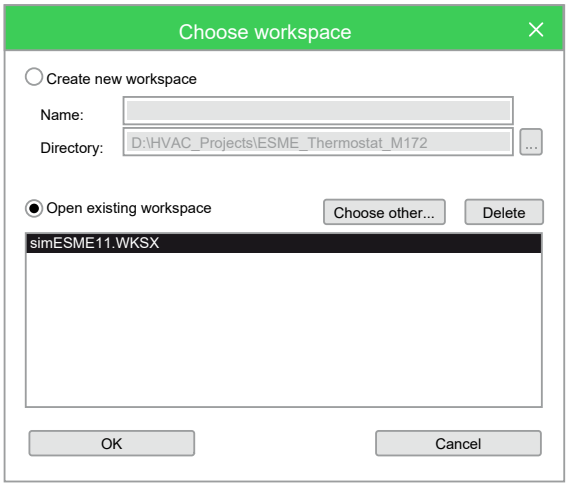
- Simultaneous simulation of both PLC application and HMI pages.
- Availability of the target panel to have a visual and realistic representation of the target to run and interact with HMI pages.
- Execution of the simulated application tasks handled by a scheduler that can reproduce the real target scheduler policy.

- The simulated target can have some parts implemented in C and/or IEC to implement the real target behavior and characteristics to react to PLC application as the real target would do.
- Use of the I/O panels, that you can configure to view and/or modify the simulated status and I/O variables of the target.

Simulation with FREE Studio Plus

Start the Simulation

To carry out a simulation session:

Step	Action
1	Write your PLC code in **Programming** or your HMI pages in **Display**.
2	Click  to compile the project and to check the correctness of the code.
3	According to the tab: • In **Programming** tab, click **Debug** > **Simulation mode** to activate the simulation. • In **Display** tab, click  **Simulation mode** icon.
4	You can choose to open a recently used simulator workspace (WKSX) or create a new one if it is the first simulation session with this project. The last used workspace is then proposed as the default choice. The list of the used workspaces is saved inside the project itself.  Click **OK**. **Result:** The simulation control panel is displayed, page 190.
5	Compile and download the code inside the simulated target.

NOTE: FREE Studio Plus can activate the simulation status that is similar to the normal connection to a physical target device, with a different connection status indicator. While in simulation status, the project will be built for the x86 processor and the connection will take place using the GDB protocol over TCP/IP on the local host (`127.0.0.1`).

Use the Simulation

The simulation mode makes it possible to verify that the behavior of the controller conforms to your expectations.

If not, you must correct any anomalies present in the project before its actual commissioning.

To test your project, you can:

- Simulate local I/O and work on pages (with mouse and keyboard) in the target panel (if there is one).
- Modify the values of the application parameters with the I/O panels.

For more details, refer to Simulator Interface, page 189.

NOTE: You can debug with the **Programming** debugging features, page 194, independently of the real target.

Stop the Simulation

The simulation session is terminated when you deactivate the simulation mode inside FREE Studio Plus (and the simulator is automatically closed).

You can also manually close simulation. In this case, the communication or the next downloads in FREE Studio Plus go in the timeout state, as in the real situation when the physical target is powered off or disconnected.

When the simulation is stopped, everything is saved inside the current workspace (I/O panels, window positions, and so on).

FREE Studio Plus saves the list of recently used workspaces inside the project for further use.

Simulation Interface

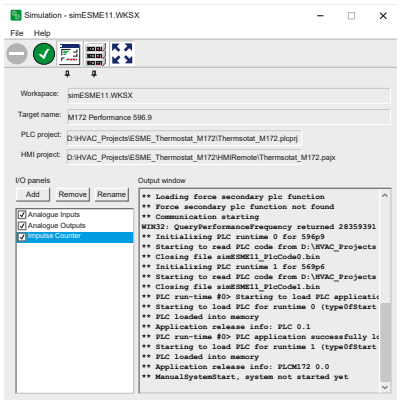
Simulation Interface Overview

Overview

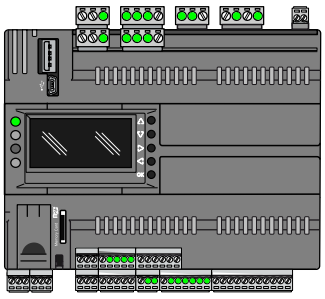
Simulation is dialog-based Windows program that is one or more independent windows that can be moved and placed on the screen.

The following pictures show the main windows:

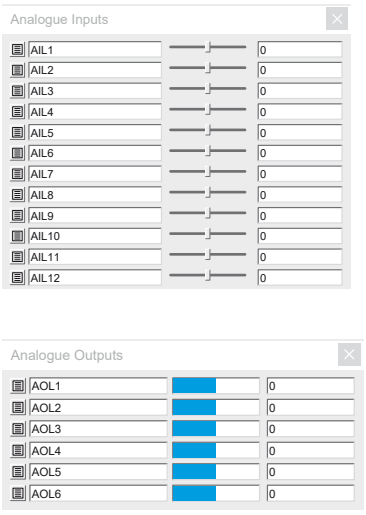
Control Panel

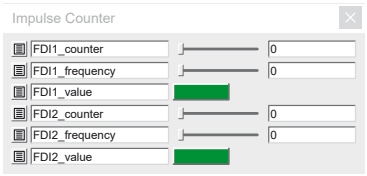


Target Panel



I/O panels





Control Panel

Overview

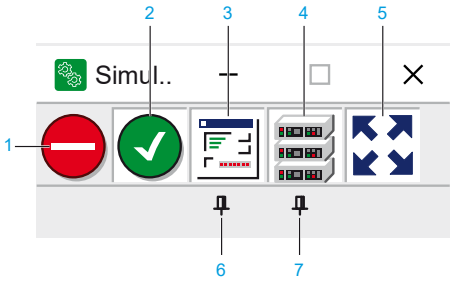
This is the main window of the simulator. When you launch the simulator, the control panel is shown in a compacted form, with 5 main buttons and no menu bar.

When you click the **Expand** button, it is expanded to show:

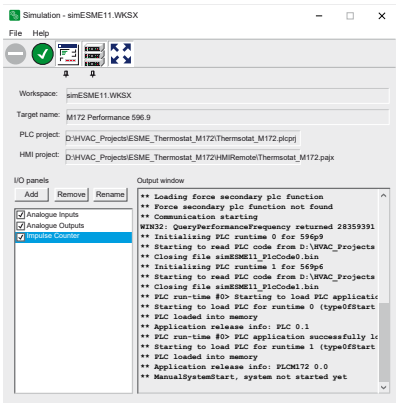
- The menu bar with the standard new/load/save/exit commands.
- A central panel showing the main characteristics of the current workspace.
- An output window showing execution logs.
- The I/O panels list.

With the control panel, you can control and monitor the state of the simulated PLC runtime, choose which other windows to show or hide (and their topmost behavior), and manage I/O panels.

Compacted Control Panel



Expanded Control Panel



1 Stop PLC code

2 Run PLC code

3 Show Target Panel, page 190

4 Show I/O Panels, page 191

5 Expand Control Panel

6 Target Panel topmost (foreground)

7 I/O Panel topmost (foreground)

Target Panel

Overview

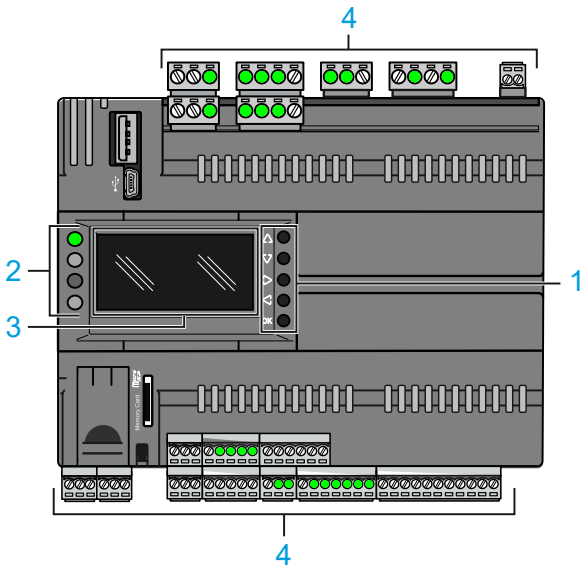
This is a floating window that shows a visual representation of the simulated physical target. Its presence and layout is defined inside the target definition file (TGSX).

This panel has an image of the real target, with some sensible areas that show simulated inputs or outputs (for example LEDs for digital outputs) and a simulated LCD graphic display where the HMI pages are drawn.

You can interact with this panel with the mouse or with the PC keyboard that emulates the real device keys.

You can right-click on it and select:

- **Topmost**: place the panel in foreground (it stays always above any other window).
- **Locked**: you can no more move the panel on your screen.
- **Close**: close the panel.



1 Simulated buttons

2 Simulated LEDs

3 Simulated LCD screen

4 Simulated I/Os

I/O Panels

Overview

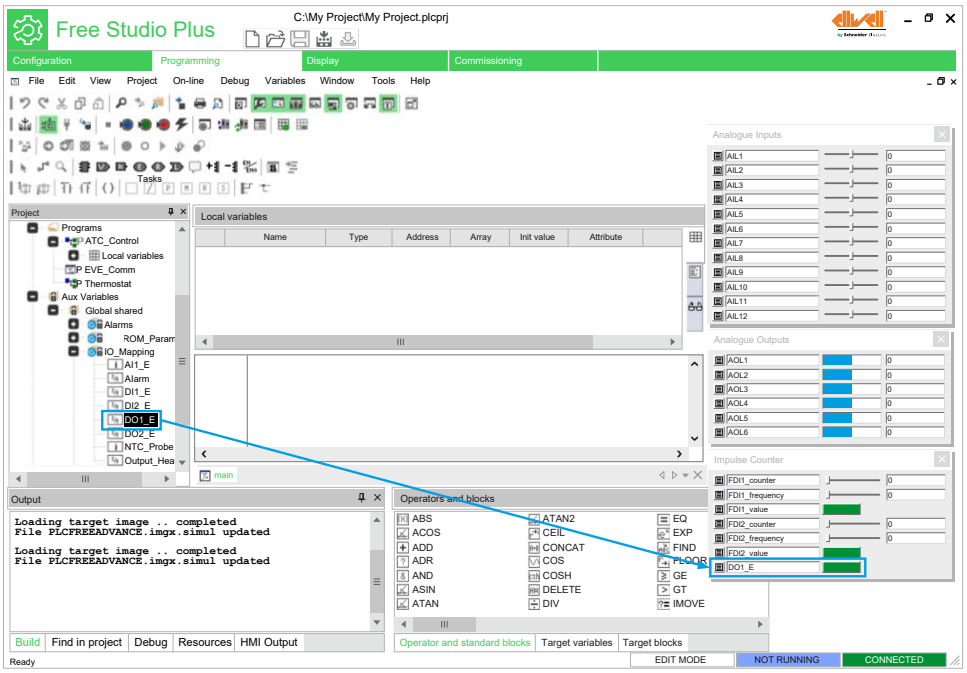
These small floating windows lets you monitor and modify the values of the various I/O and status variables of the simulated target; the only requirement is that the object to watch is allocated on data-block.

You can create as many I/O Panels as you want. You can also decide which objects to put on each panel freely. They are complementary to the target panel because with them you can watch and edit the I/O modules that are not already visible there.

The I/O panels can be put in Topmost mode (always above the visible windows). This is useful for example while debugging with **Programming** at full screen. The configuration is then saved inside the workspace file.

Adding Elements to Panels

To add an element (or “signal”) to an existing I/O panel to watch or edit its value, you can drag it from **Programming** inside the panel itself. You can drag it from the target variables panel, from the workspace tree, or from a variables grid inside an editor.



Depending on the type of the source variables, an analog (slider or progress bar) or digital (LED or button) control is generated, and associated with the original signal.

It is possible to add only PLC variables that reside on a DataBlock, with an explicit address (for example %MW1.0); you cannot add to an I/O panel automatic, local, or global variables.

Editing I/O Panel Elements

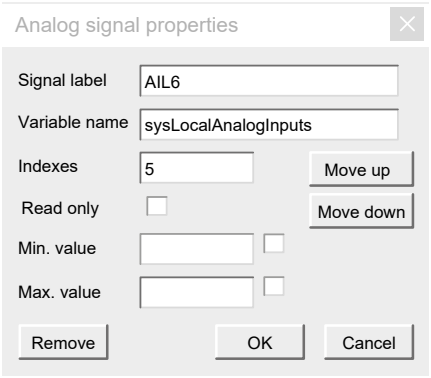
You can edit the advanced options of each signal by clicking the small icon on the left of each name.

For a digital I/O, the options are:

- Label to be viewed on the panel;
- Name of the associated source variable, and its index if it is an array;
- Read-only attribute: the control is an LED (output, read-only) or a button (input,read/write);
- Selector attribute: it is valid only for read/write variables (buttons), if active the button keeps its value (pressed or not pressed), otherwise it keeps the new value only as long as the mouse button is pressed, then it goes back to its previous value.

For an analog I/O, the options are:

- Label to be viewed on the panel;
- Name of the associated source variable, and its index if it is an array;
- Read-only attribute: the control is a progress-bar (output, read-only) or a slider (input, read/write);
- Minimum and maximum limits: if not set, absolute minimum and maximum limits of the original data type is used. The progress and slider uses these limits; they can be individually activated or not.



Removing Elements from I/O Panels

To remove a signal, edit the element, page 192 and click **Remove** button.

I/O Panels List

Overview

When the control panel is expanded, you can manage (add/remove/rename) the I/O panels.

Adding a New I/O Panel

To add a new empty panel, click the relevant **Add** button in the control panel.

A new empty panel without name is created and opened.

Editing an I/O Panel

In order to rename the panel, select it in the list and click the **Rename** button. You are asked for the name to give to the window that is shown in its title bar.

Any panel can be drag around the screen in any place. To temporary hide it you can toggle the check next to its name in the list.

You can also toggle the topmost button to bring the panels above the other windows (this is a global setting that applies to the panels).

Removing an I/O Panel

In order to remove a panel, select it in the list and click the remove button; the panel and its signals and settings are permanently removed.

Debugging

What's in This Chapter

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.

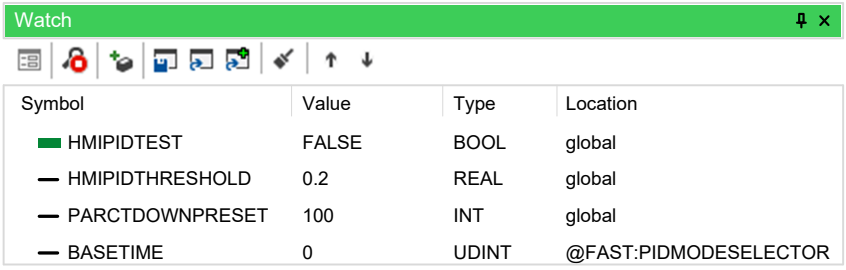
Watch Window

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.



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:

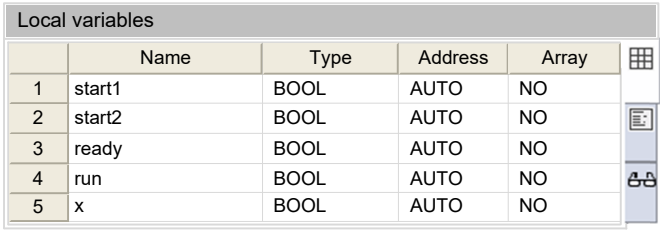
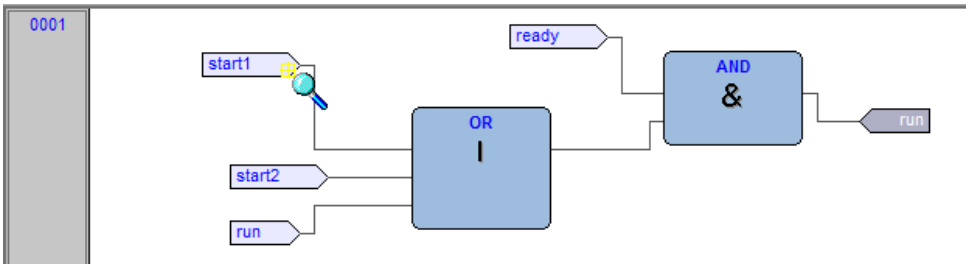
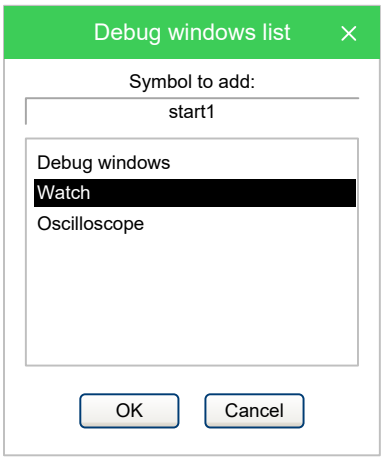
Step	Action

1	Double-click the variable you wish to display in the **Watch** window.
	`0001 x := x + hmiFrequency;`
	`0002`
	`0003 hmiSinVal := SIN( x ) * hmiAmplitude;`
	`0004 hmiCosVal := COS( x ) * hmiAmplitude;`
	`0005`
	`0006 hmiStep := hmiStep + 1;`
	`0007`
	`0008`

2	Drag the selected variable in the **Watch** window: Watch Symbol / Value / Type / Location HMISINVAL / 0.982414 / REAL / @FAST:LOOPS

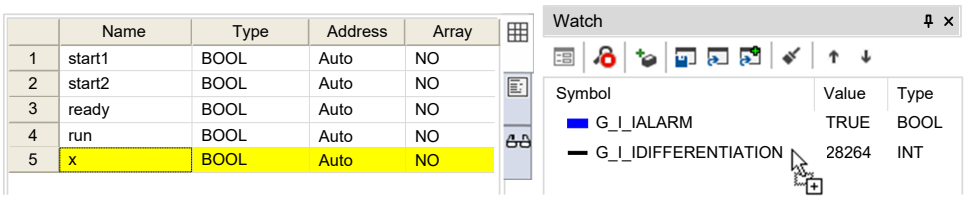
Adding a Variable from a Graphical Source Code Editor

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.

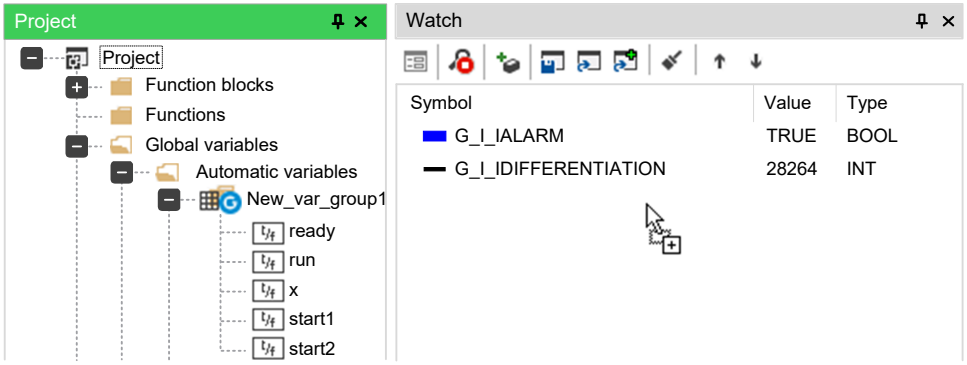
Adding a Variable from a Variables Editor

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

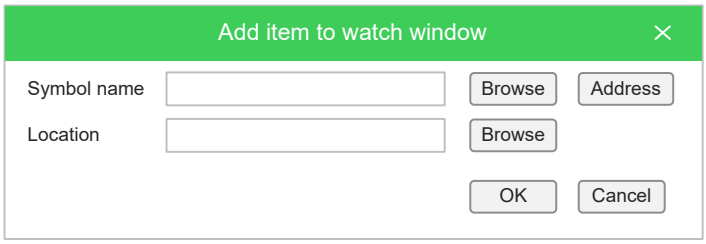
To add a variable to the **Watch** window, select it in the project tree and drag it in the **Watch** window:



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).



Removing a Variable

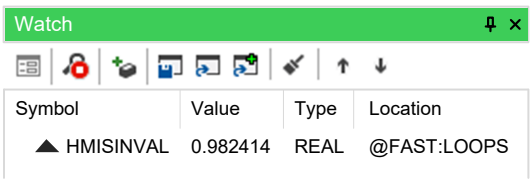
Description

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

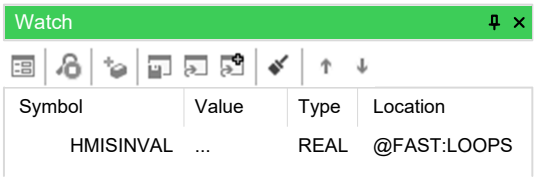
The watch window manager reads periodically from memory the value of the variables.



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

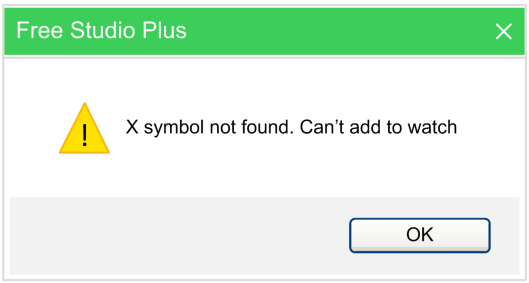
If the target device is disconnected, the **Value** column contains three dots.



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

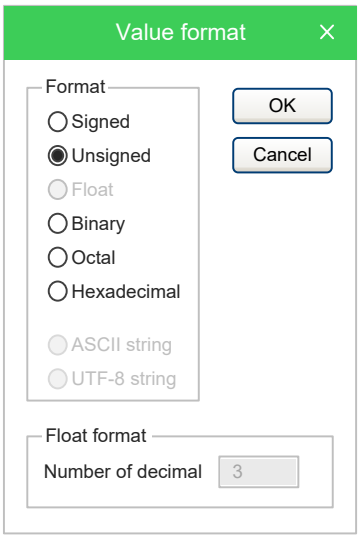
Description

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, page 105 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.

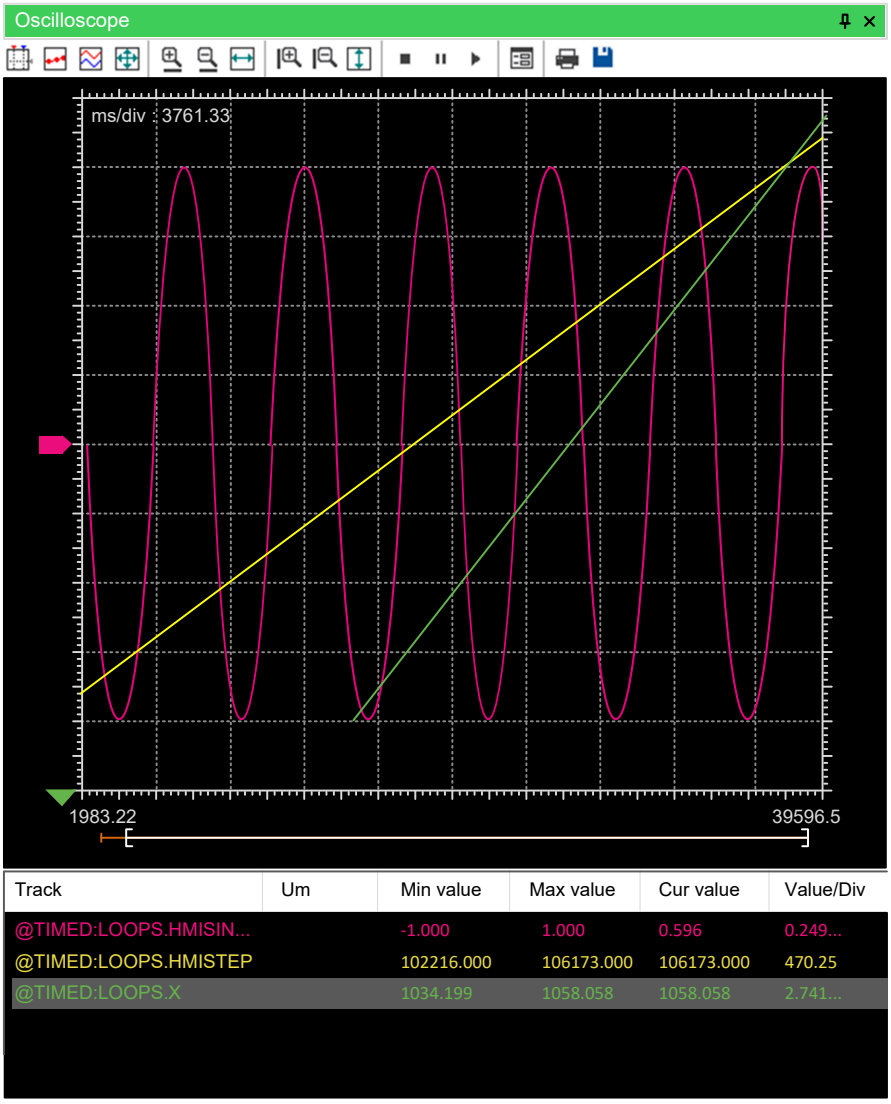
Oscilloscope

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

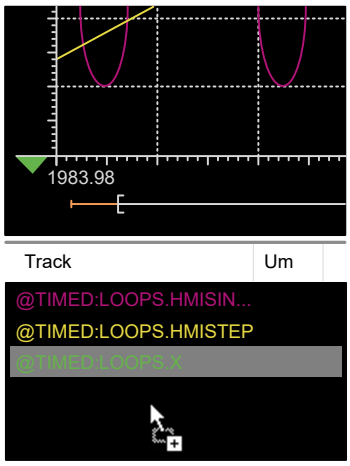
In order to plot the evolution of the value of a variable, you need to add it to the Oscilloscope.

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.

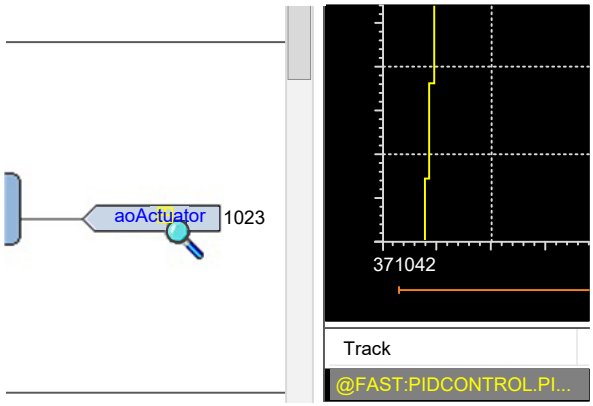
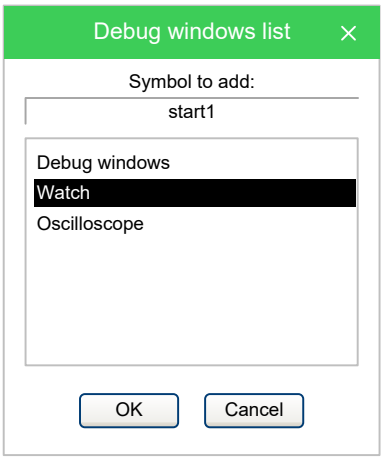
```
0001
0002
0003  x := x + hmiFrequency;
0004
0005  hmiSinVal := SIN( x ) * hmiAmplitude;
0006  hmiCosVal := COS( x ) * hmiAmplitude;
0007
0008  hmiStep := hmiStep + 1;
0009
```



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:

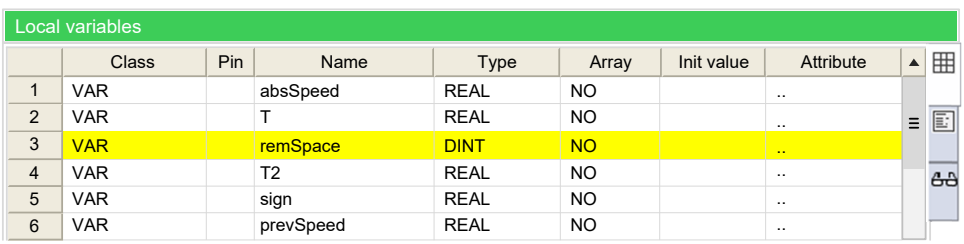
Step	Action
1	Click  **Edit > Watch mode**.
2	Click the block representing the variable you wish to trace in the Oscilloscope: 
3	A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked:  Select **Oscilloscope**, 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.

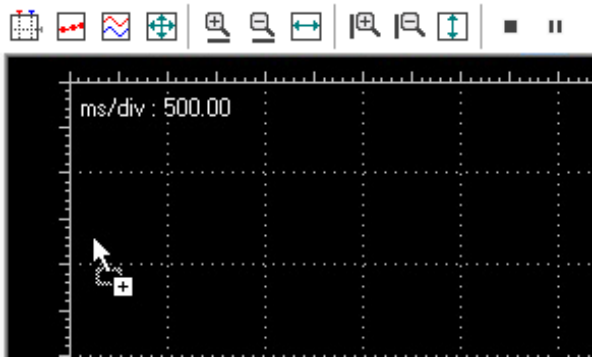
Once you have added to the Oscilloscope all the variables you want to observe, you should click  **Edit > Insert/Move mode**: the mouse cursor turns to its original shape.

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:



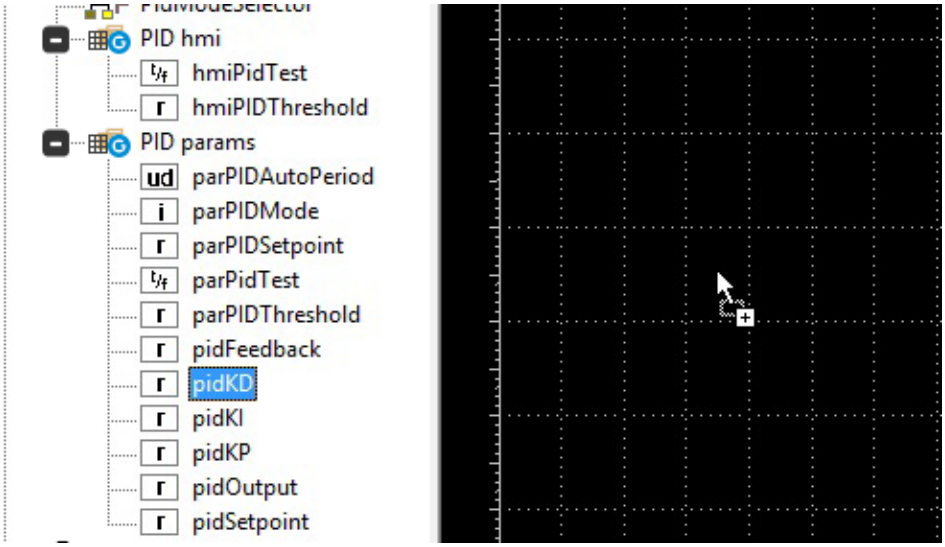
	Class	Pin	Name	Type	Array	Init value	Attribute
1	VAR		absSpeed	REAL	NO		..
2	VAR		T	REAL	NO		..
3	VAR		remSpace	DINT	NO		..
4	VAR		T2	REAL	NO		..
5	VAR		sign	REAL	NO		..
6	VAR		prevSpeed	REAL	NO		..



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**.

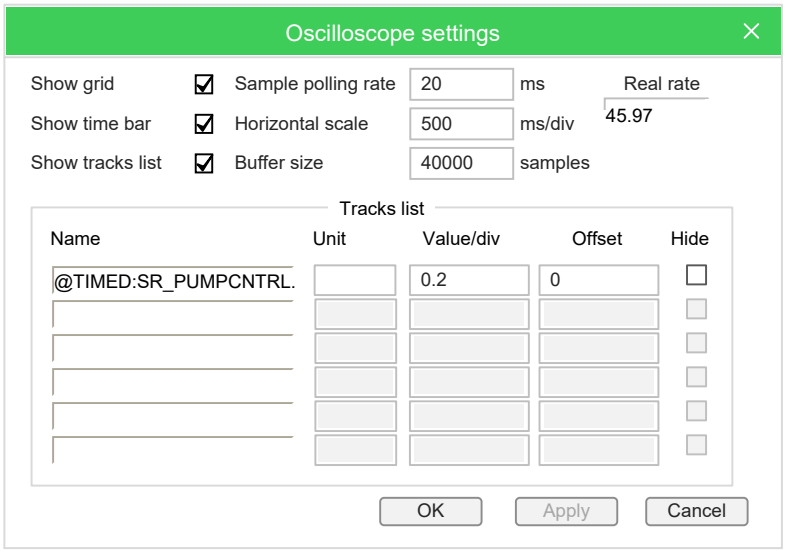
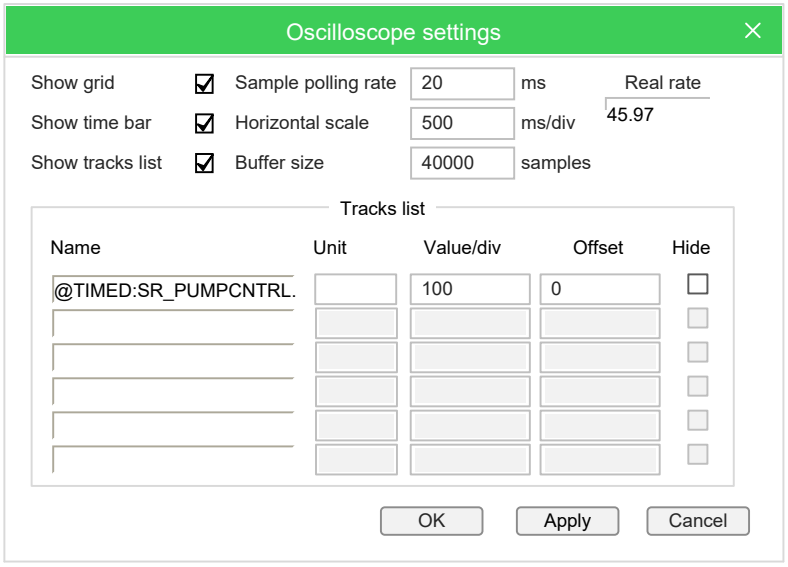
In order to stop the acquisition, you may click **Stop acquisition**.

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:

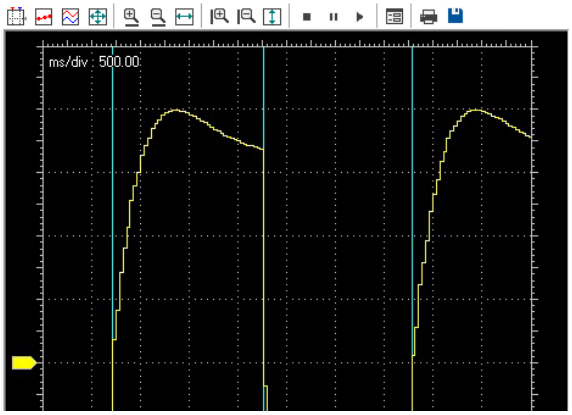
Step	Action
1	Open the **Oscilloscope settings** by clicking the  **Graph properties** button in the toolbar.
2	Set the scale of the horizontal axis, which is common to all the tracks: 
3	For each variable, you may specify a distinct scale for the vertical axis: 
4	Confirm your settings. The graph adapts to reflect the new scale.

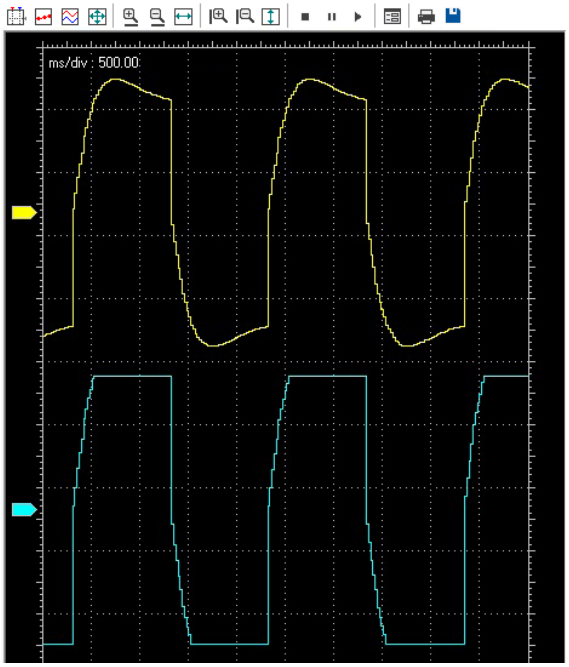
You can also zoom in and out with respect to both the horizontal and the vertical axes:

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

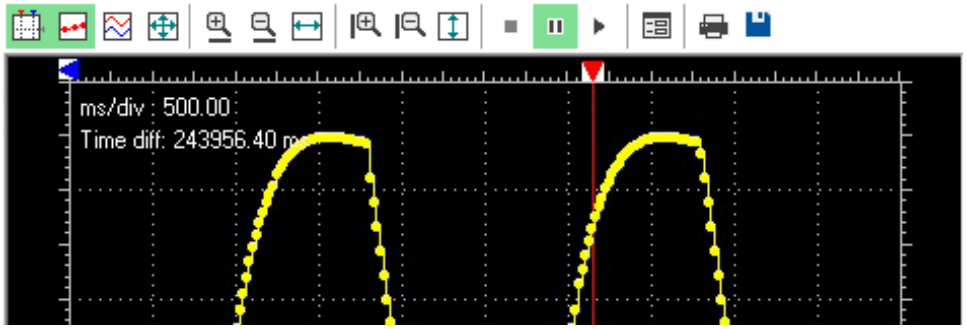
If you click the  **Show samples** item in the **Oscilloscope** toolbar, the tool highlights the single values detected during data acquisition:

You can click the same item again in order to go back to the default view mode.

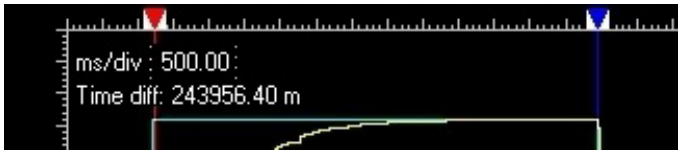
Taking Measures

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.

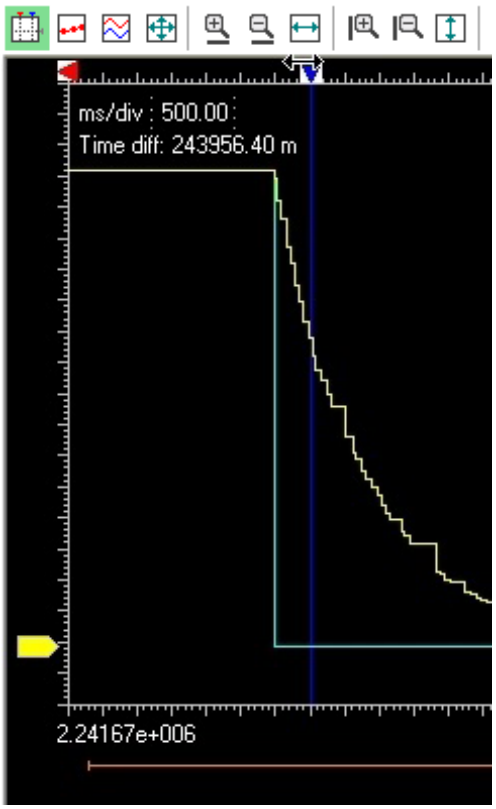
If you want to measure a time interval between two events, you have to move one bar to the point in the graph that corresponds to the first event and the other to the point that corresponds to the second one:



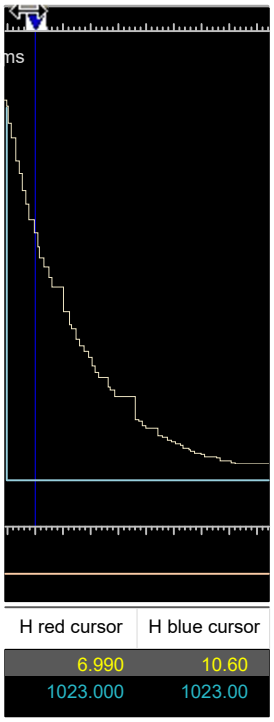
The time interval between the two bars is displayed in the top left corner of the chart:



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:



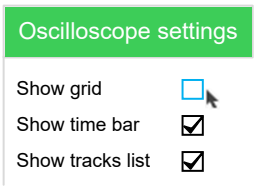
In the following table (below the graphic), you can now read the values of all the variables at that particular moment:



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**:

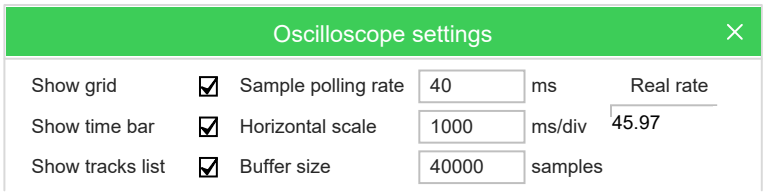


Changing the Polling Rate

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.

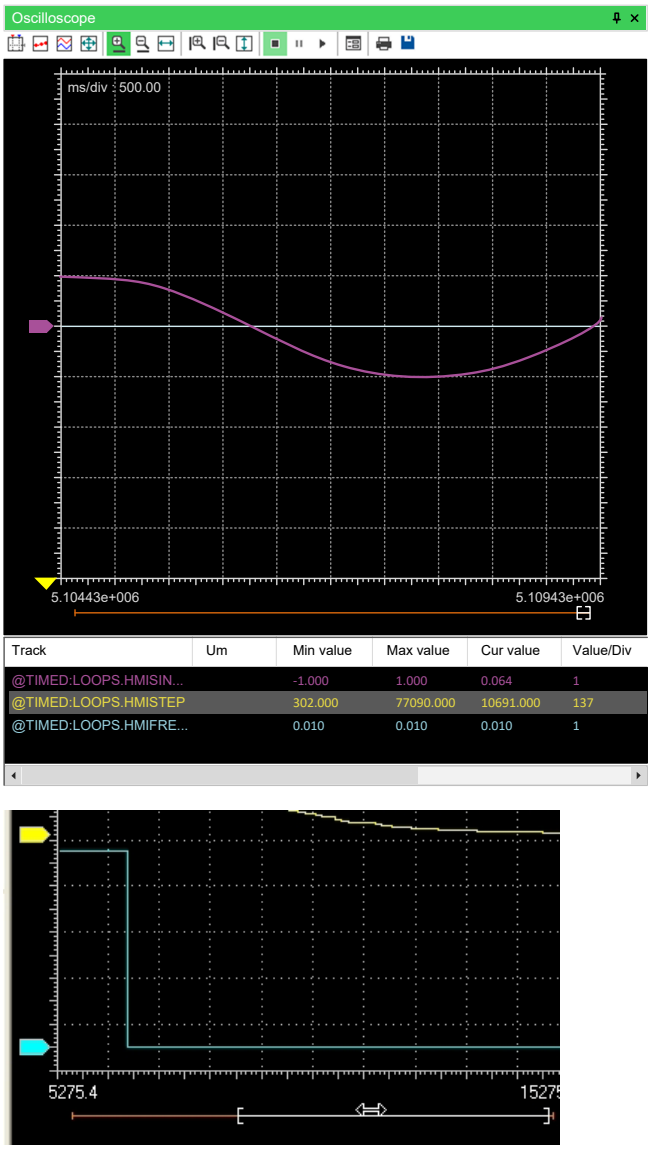
Saving Data to a File

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: 
3	Click the  **Print graph** item.

Edit and Debug Mode

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:

DEBUG MODE	SOURCE OK	CONNECTED

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**.

Live Debug

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 FREE Studio Plus.

⚠WARNING

UNINTENDED EQUIPMENT OPERATION

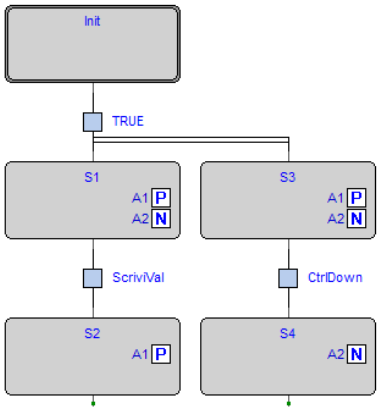
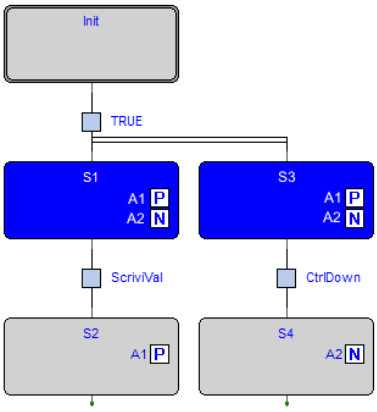
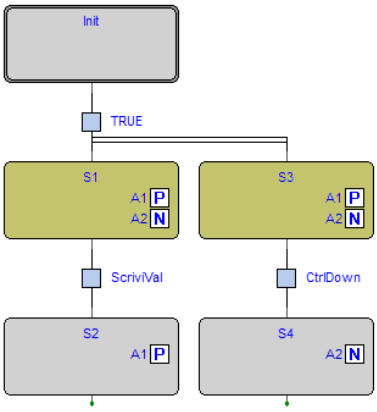
- You must have a thorough understanding of how forcing will affect the outputs relative to the tasks being executed.
- Do not attempt to force I/O that is contained in tasks that you are not certain will be executed in a timely manner, unless your intent is for the forcing to take affect at the next 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 FREE Studio Plus 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.

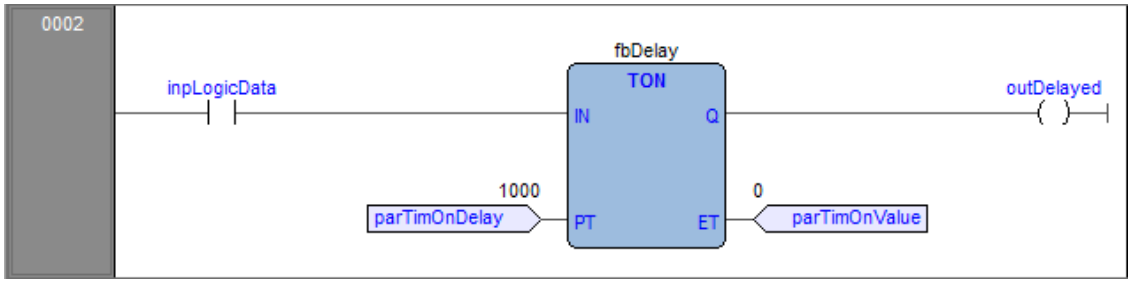
Debugging Actions and Conditions

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):

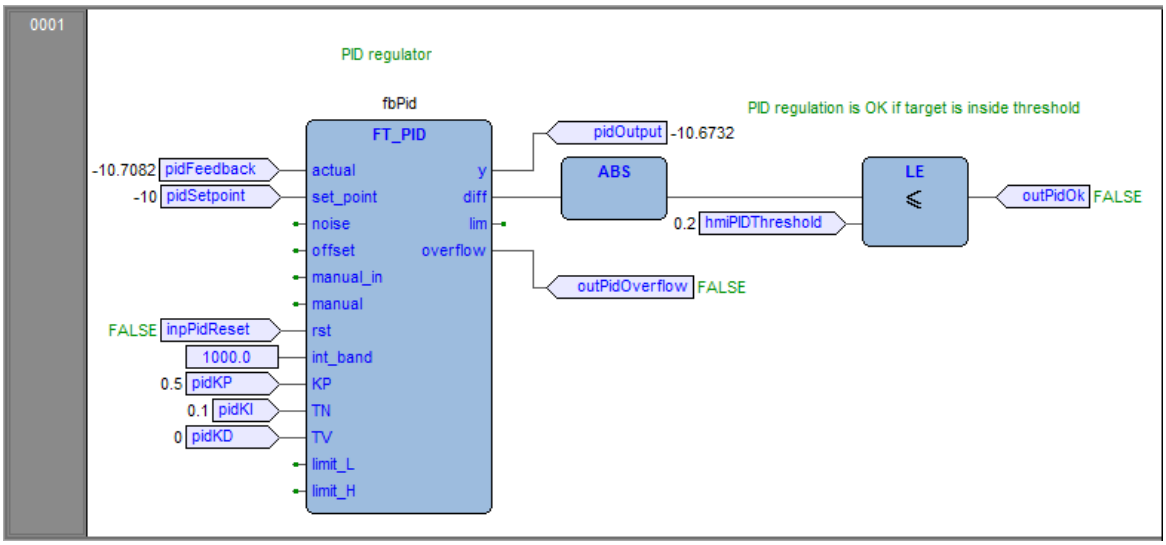


The LD animation manager tests periodically the state of all the elements. It may happen that an element remains true too briefly to be displayed. The fact that an element is never highlighted does not imply that its value never becomes true (the sampling rate may be too slow).

FBD Simulation

Description

In live debug mode, **Programming** displays the values of all the visible variables directly in the graphical source code editor:



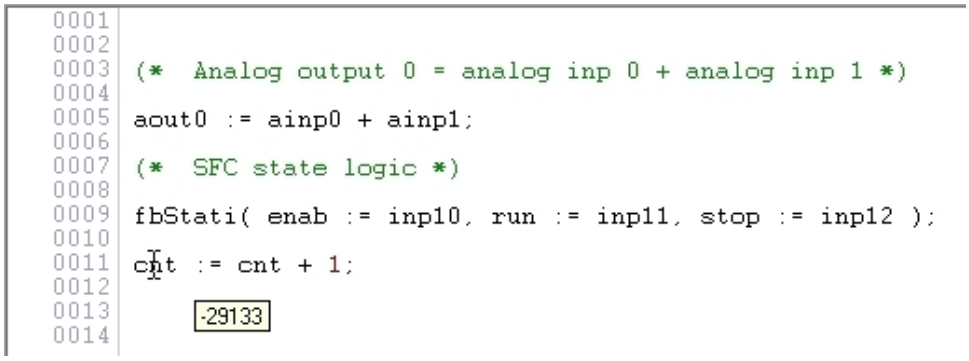
This works for both FBD and LD programming language.

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:



Triggers

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

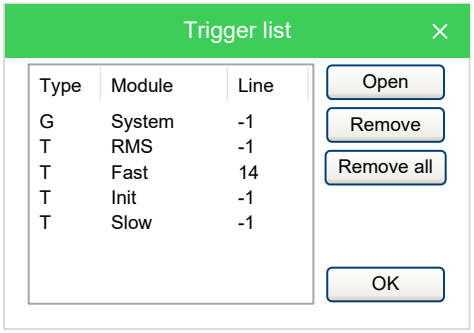
A graphic trigger window takes the whole free space of the application code sector. Once such a debugging tool has been started, it is not possible to add any trigger window, and an error message appears if you attempt to start a new window. Once the graphic trigger window is closed, trigger windows 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**.

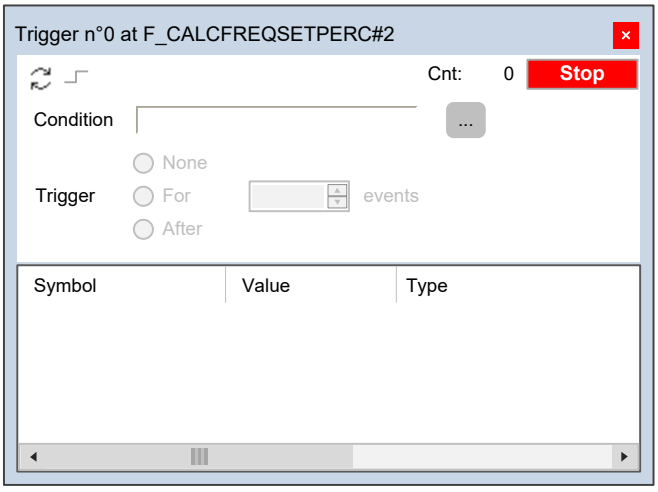
Icon	Command	Description
	**Remove all triggers**	Clicking this button causes all the existing trigger windows and the graphic trigger window to be removed simultaneously. Shortcuts key: **Ctrl+Shift+F9**.
	**Trigger list**	This button opens a dialog listing all the existing trigger windows:  Shortcut key: **Ctrl+I**.

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:BID* where: *N1* = instruction line number *N2* = network number *BT* = block type (operand, function, function block, and so on) *BID* = block identifier

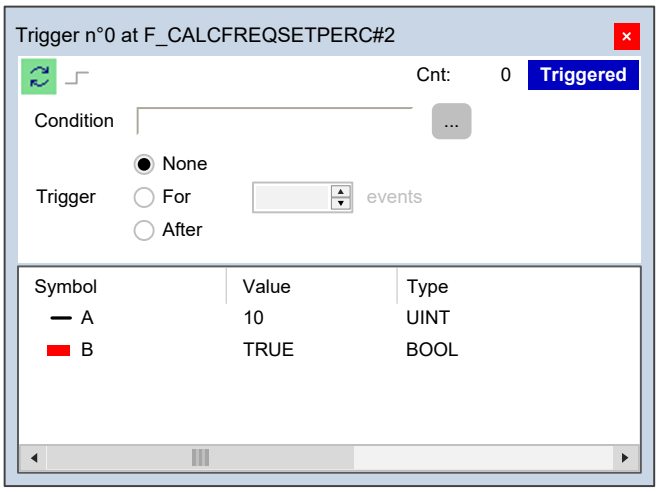
Controls section

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, page 227.

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 Window: Drag and Drop Information

To watch a variable, you need to copy it to the lower section of the **Debug** window.

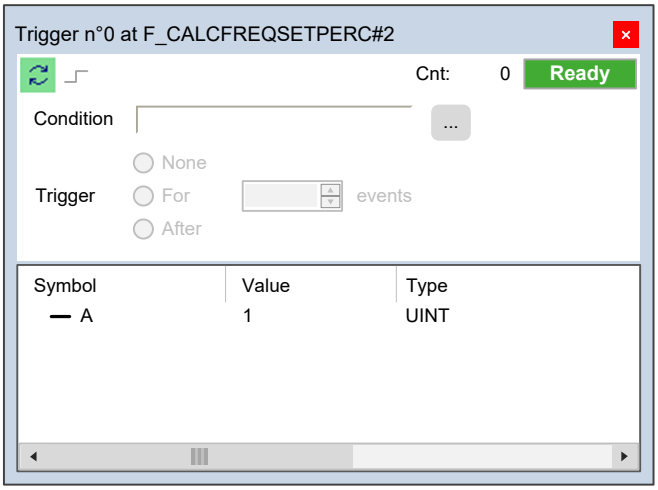
This section is a table consisting of a row for each variable you dragged in. You can drag into the trigger window only variables local to the module where you placed the relative trigger, or global variables, or parameters. You cannot drag variables declared in another program, or function, or function 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

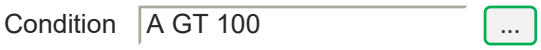
This read-only control displays the state of the **Debug** window. It can assume the following values.

Ready	The trigger has not occurred during the task execution.
Triggered	The trigger has occurred during the task execution.
Stop	System is not triggering. It has been stopped by you or a halt condition has been reached.
Error	Communication with target interrupted, the state of the trigger window cannot be determined.

User-defined condition

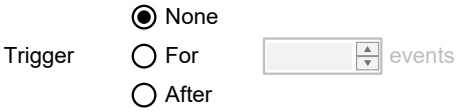
If you define a condition by using this control, the values in the **Debug** window are refreshed every time the window manager is triggered and the user-defined condition is true.

After you have entered a condition, the control displays its simplified expression:



Counters

These controls allow you to define conditions on the trigger counter:



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.

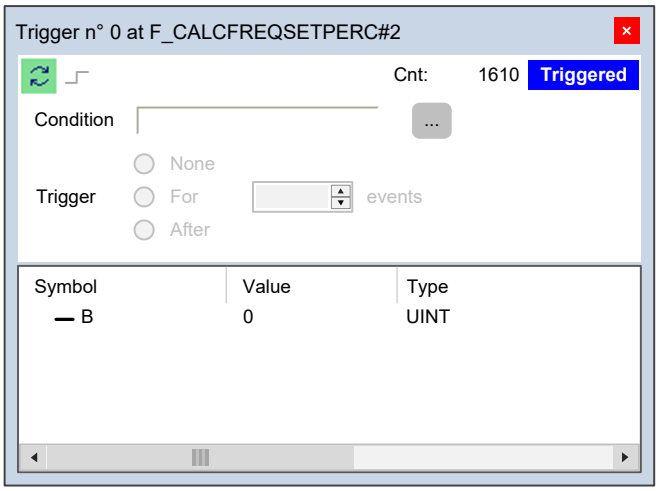
Adding a Variable to a Trigger Window from an IL Module

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
```

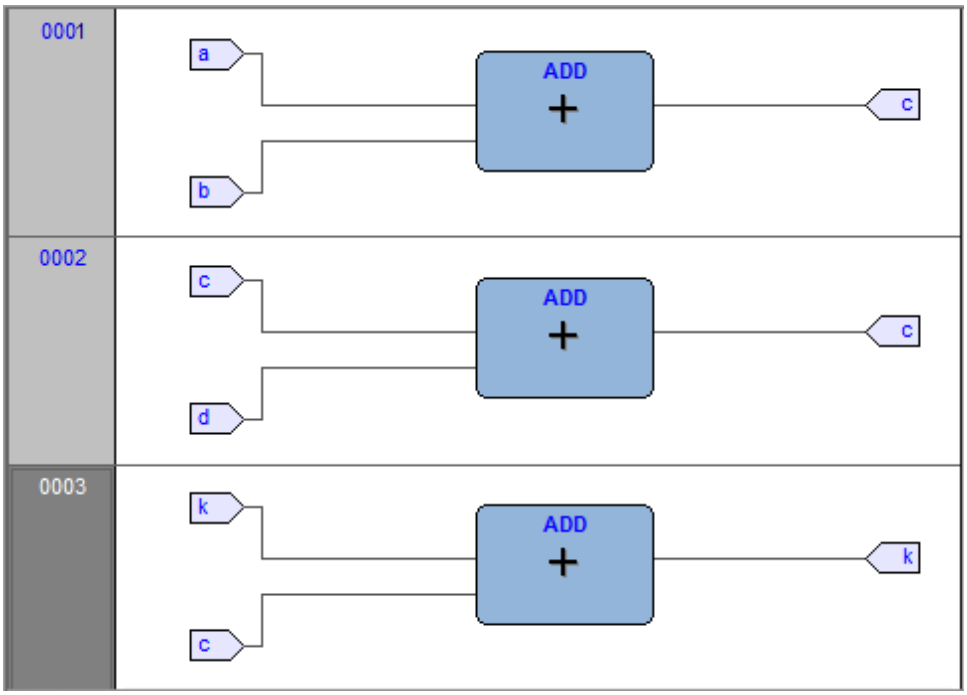
The name of the variable is now displayed in the **Symbol** column:



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:



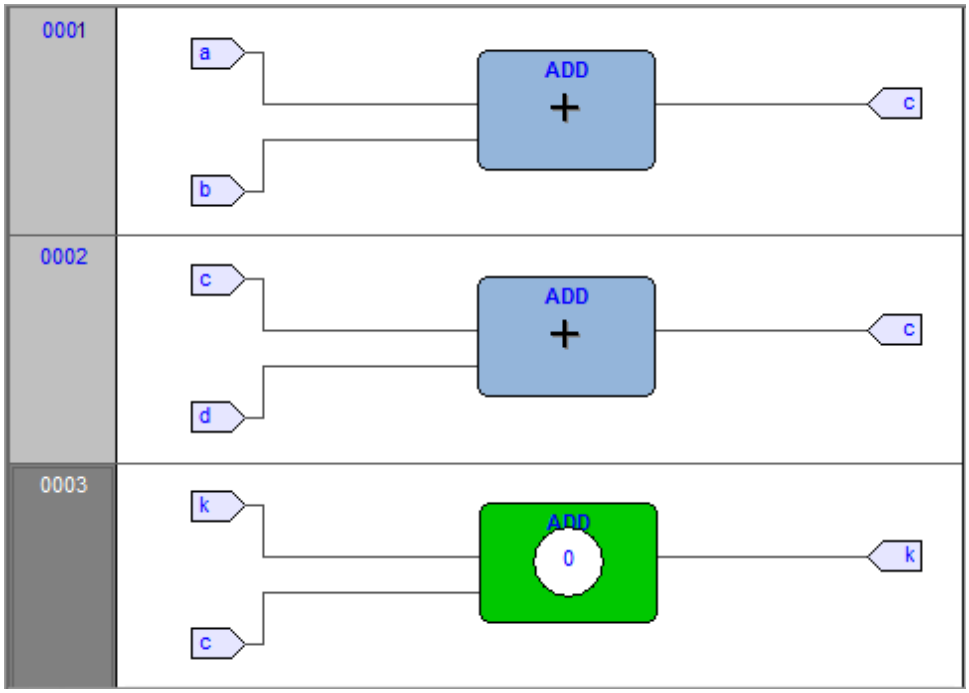
You want to know the values of *C*, *D*, and *K*, just before the *ST k* instruction is executed.

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:

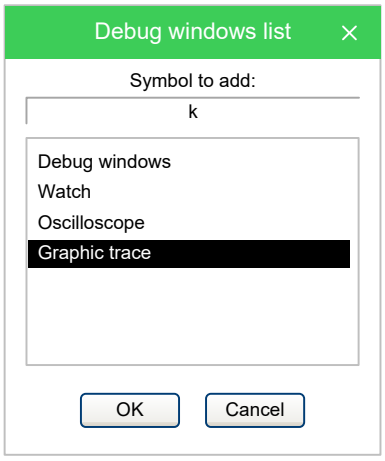
Click  **Edit > Watch mode**.

The cursor will become as follows:  .

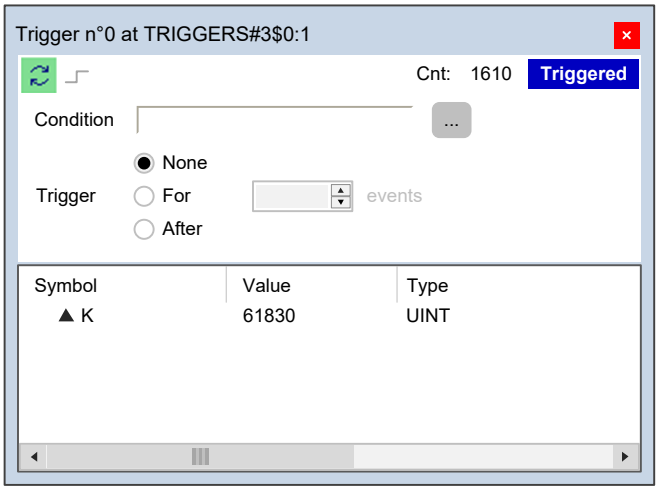
Now you can click the block representing the variable you wish to be displayed in the trigger window.

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:

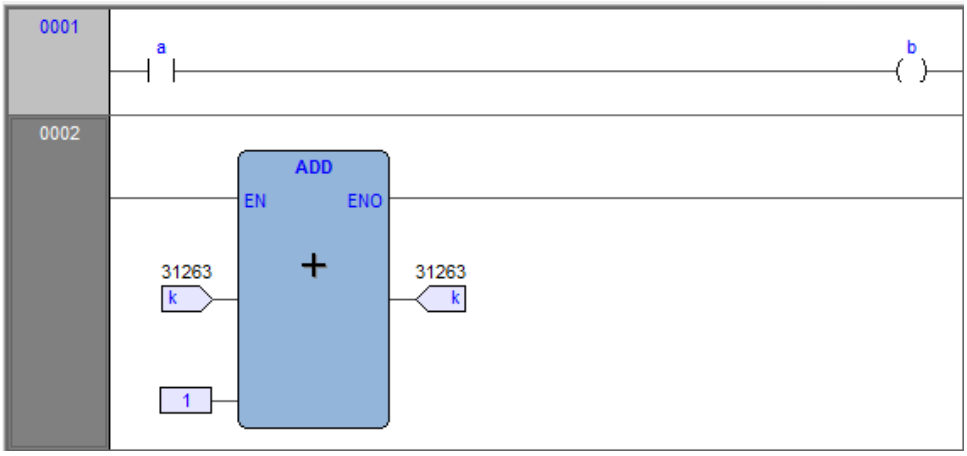


The same procedure applies to all the variables you wish to monitor:

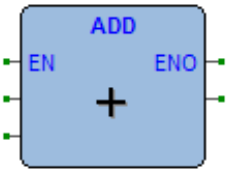
Once you have added to the **Graphic watch** window all the variables you want to observe, you can click **Edit > Insert/Move mode**, to let the cursor take back its original shape.

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:

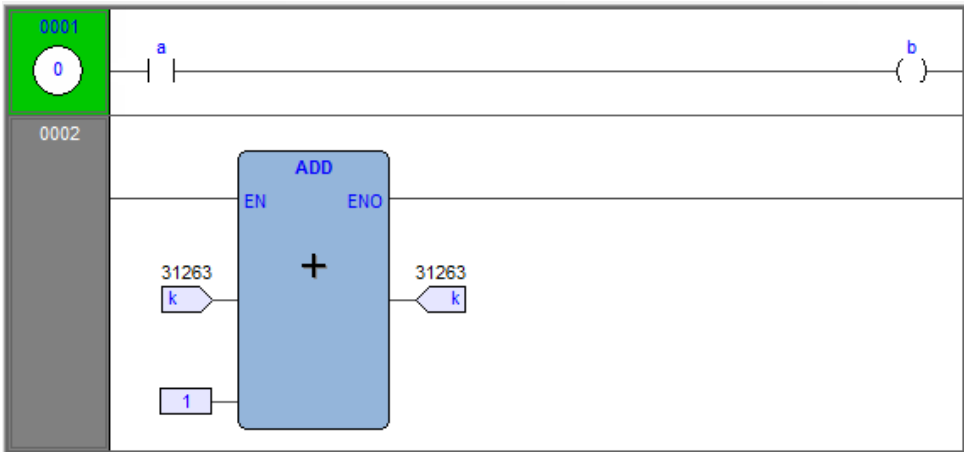


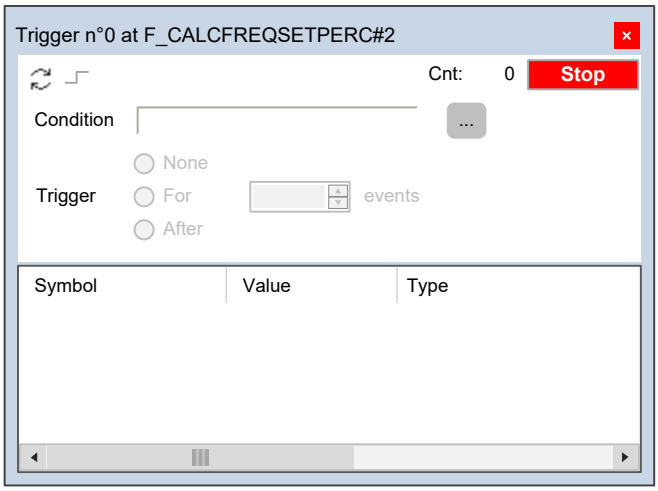
In this case, the same rules apply as to insert a trigger in an FBD module on a contact or a coil.

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

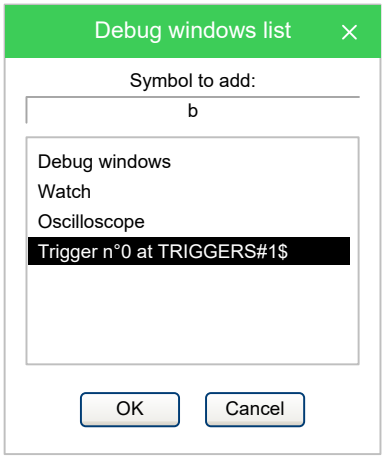
To watch the value of a variable, you need to add it to the trigger window. For example, you want to monitor the value of variable *b* in the LD code:

Click  **Edit > Watch mode**.

The cursor becomes as follows: 

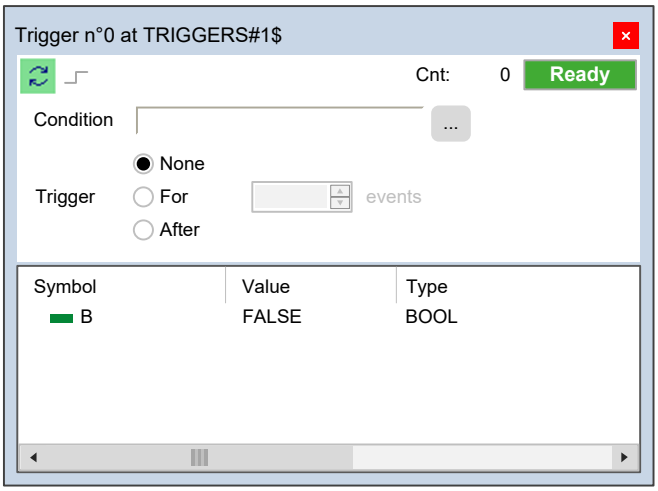
Now you can click the item representing the variable you wish to be displayed in the trigger window.

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 )`

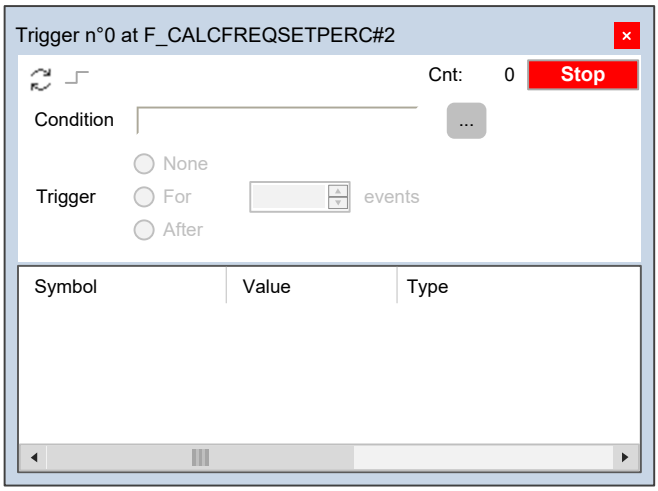
is executed. To do so, move the cursor to line 6.

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
```

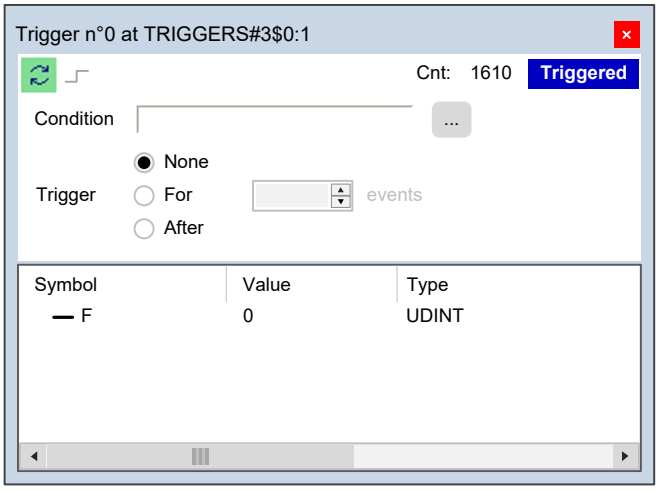
and the related trigger window pops up:



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

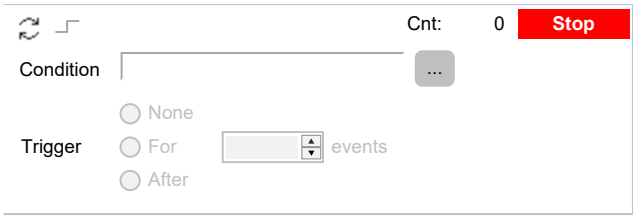
If you want to remove a variable from the trigger window, select it by clicking its name once, then press the **Delete** key.

Using Controls

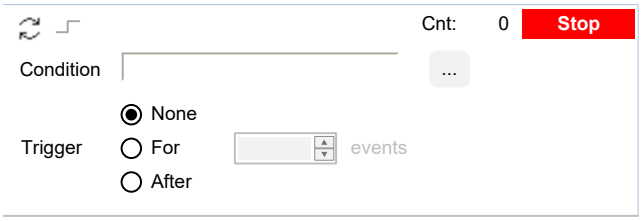
Trigger windows controls allow you to supervise the working of this debugging tool. The main purpose of trigger window controls is to let you define more limiting conditions so that variables in the **Variables** window are refreshed when the processor reaches the trigger location and these conditions 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**.

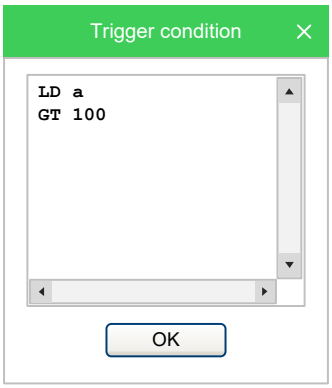
Trigger and condition settings become the settings when the triggering is (re) started.

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:

To modify it, click the button  again.

The Trigger condition window appears, containing the IL code you originally wrote, which you can now edit.

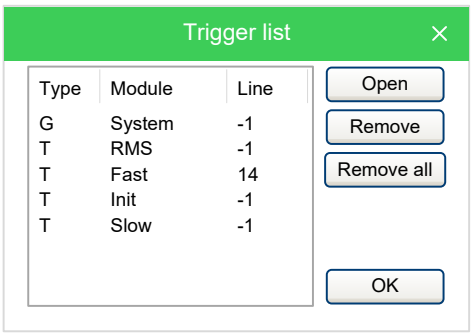
To completely remove a condition, delete all of the IL code in the window, then click **OK**.

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.

Graphic Triggers

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.

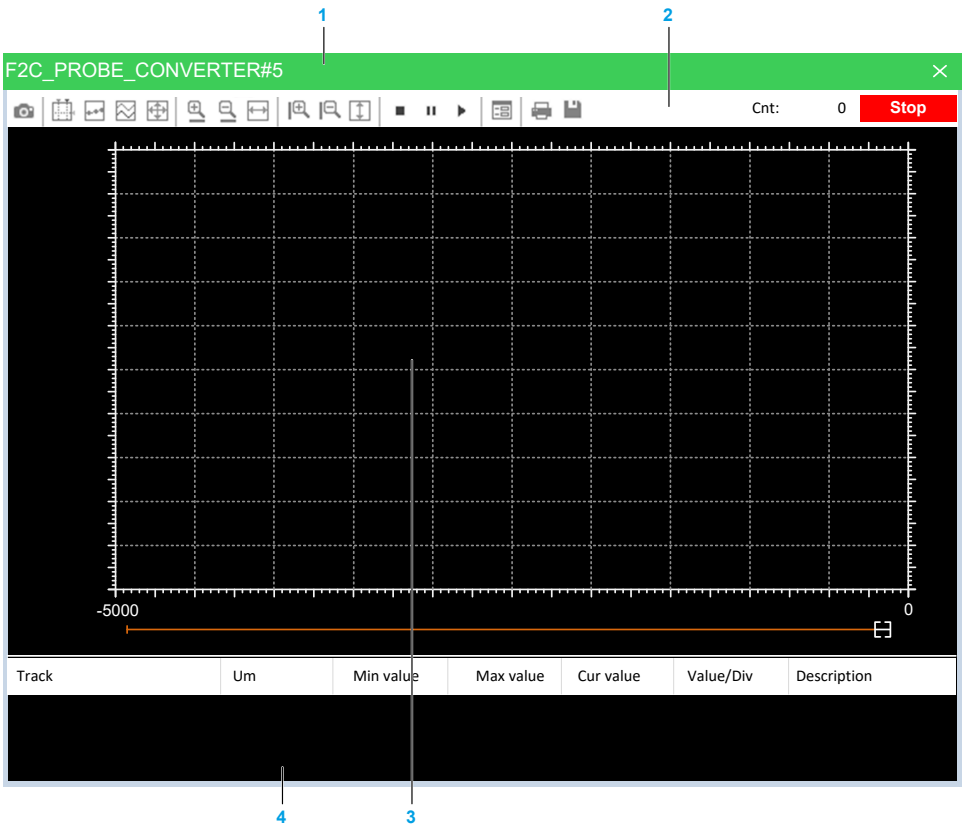
Pre-Conditions to Open a Graphic Trigger Window

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 *ModuleName*. If *ModuleName* is in IL, ST, *Location* has the format: *N1* Otherwise, if *ModuleName* is in FBD, LD, it becomes: *N2$BT:BID* *N1* = instruction line number *N2* = network number *BT* = block type (operand, function, function block, and so on) *BID* = 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, page 232.

The Chart area

The **Chart** area includes six items:

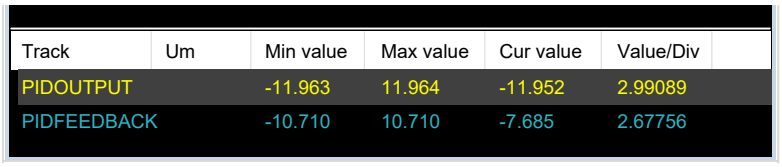
- Plot: area containing the plot of the curve of the dragged-in variables.
- Samples to acquire: number of samples to be collected by the graphic trigger window manager.
- Horizontal cursor: cursor identifying a horizontal line. The value of each variable at the intersection with this line is reported 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.
 and 	**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.
 and 	**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.
	**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, page 234.
	**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:

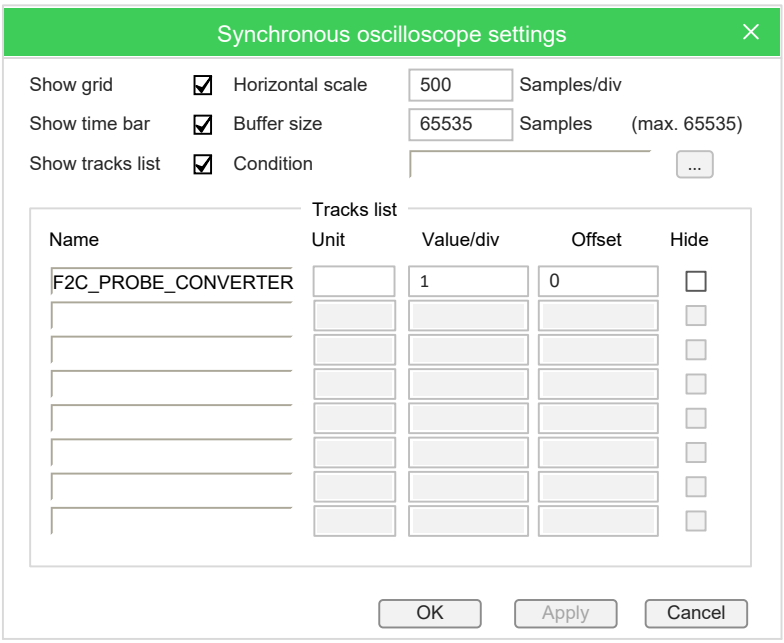
Ready	No sample(s) taken, as the trigger has not occurred during the current task execution.
Triggered	Sample(s) collected, as the trigger has occurred during the current task execution.

Stop	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.
Error	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.

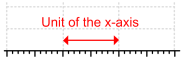
General



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 	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:

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:

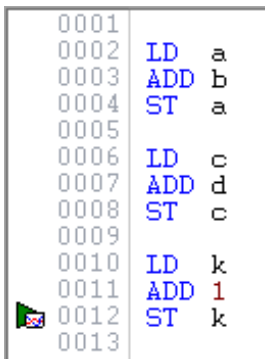
```
0001
0002 LD  a
0003 ADD b
0004 ST  a
0005
0006 LD  c
0007 ADD d
0008 ST  c
0009
0010 LD  k
0011 ADD 1
0012 ST  k
0013
```

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**.

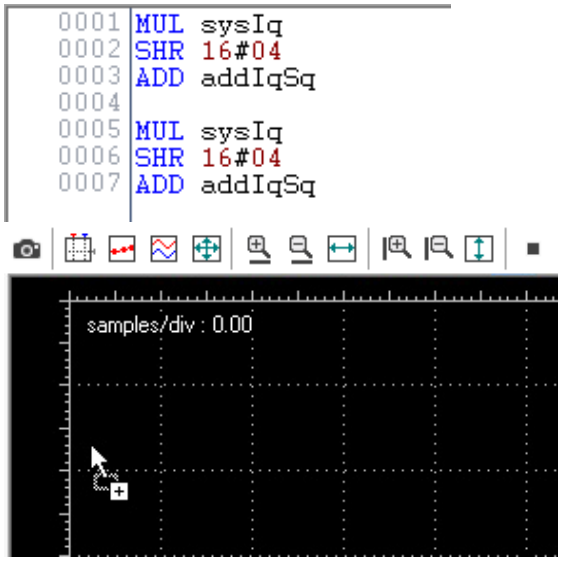
A green arrowhead appears next to the line number, and the graphic trigger window pops up:



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:

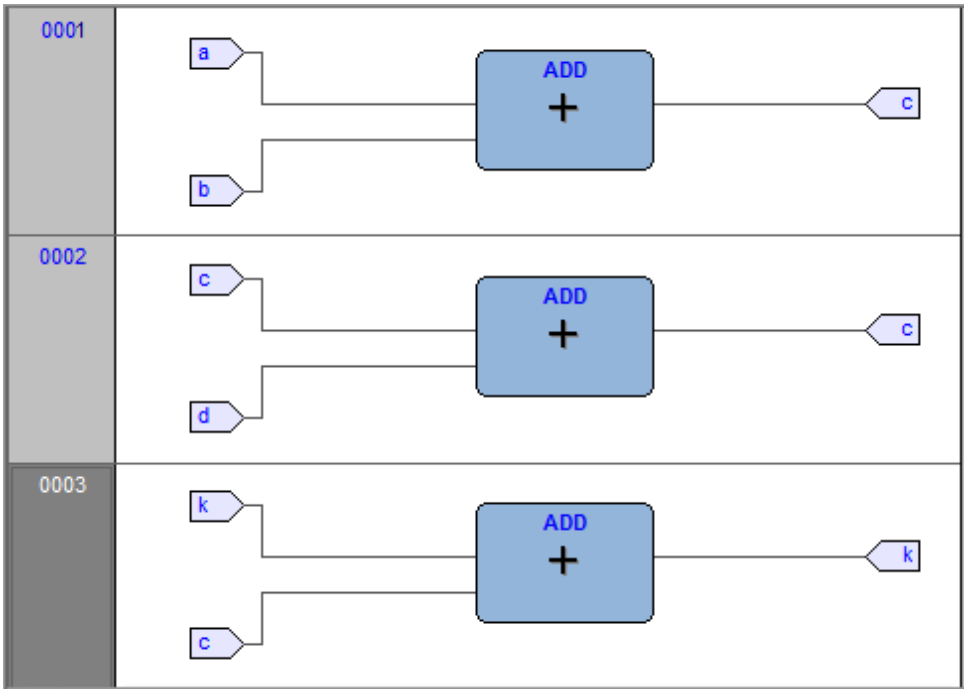


The same procedure applies to all the variables you wish to monitor.

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:



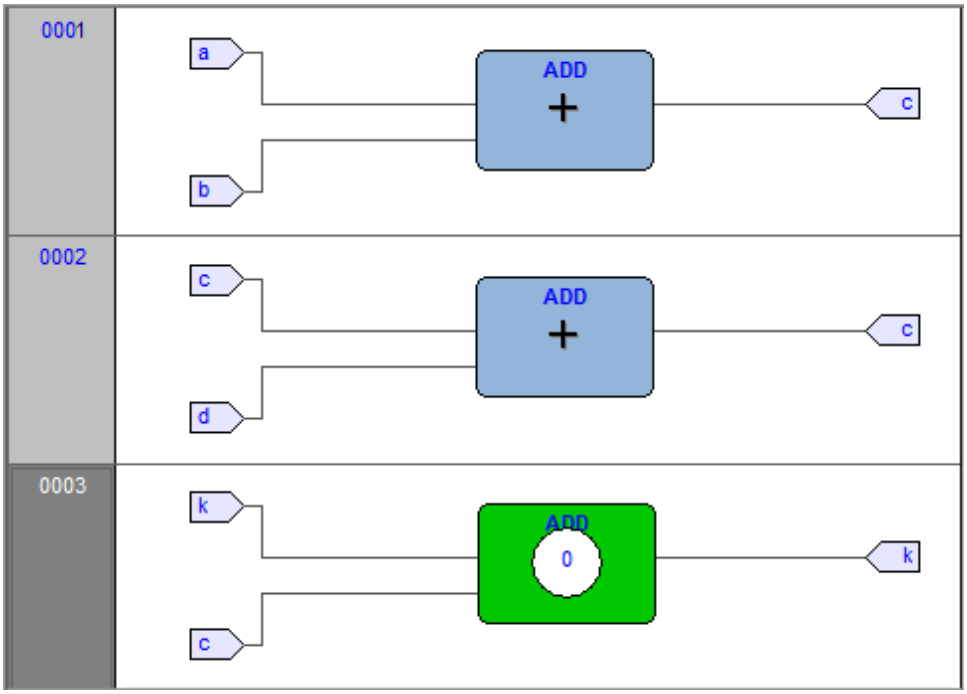
Further, assume that you want to know the values of *c*, *d*, and *k*, just before the *ST k* instruction is executed.

You can never place a trigger in a block representing a variable such as 

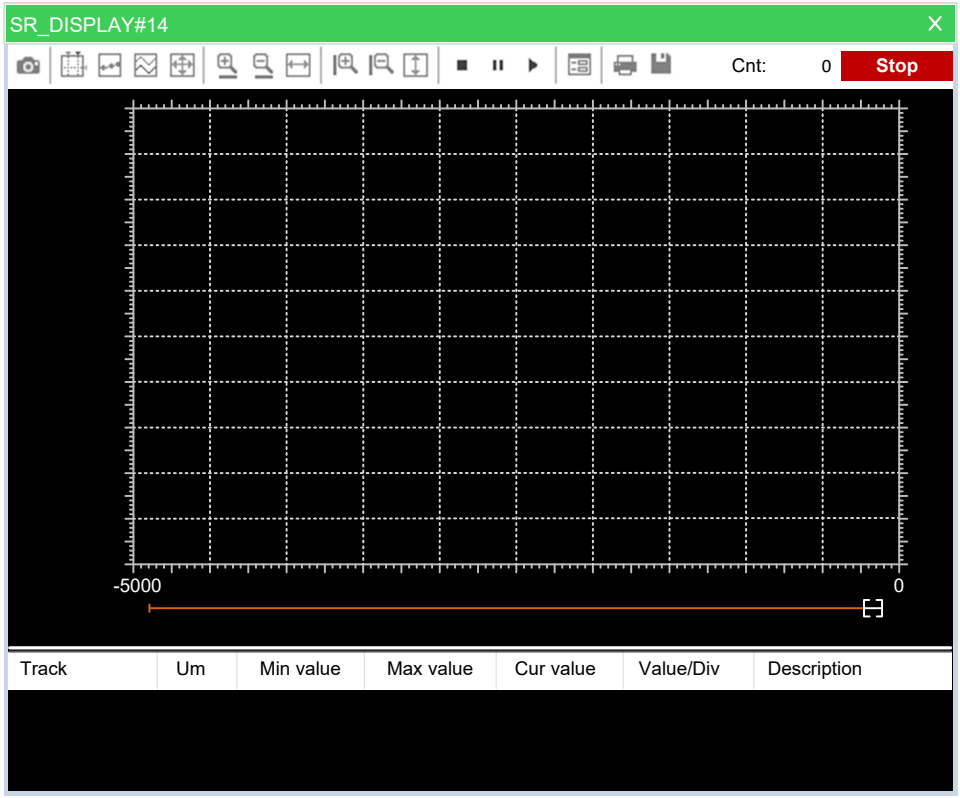
. You must select the first available block preceding the selected variable. In the example of the previous figure, you must move the cursor to network 3, and click the *ADD* block.

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.

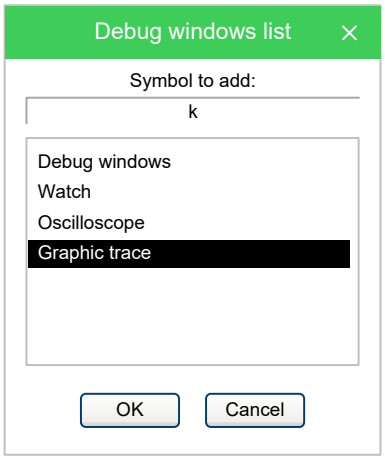
Click  **Edit > Watch mode**.

The cursor becomes as follows: 

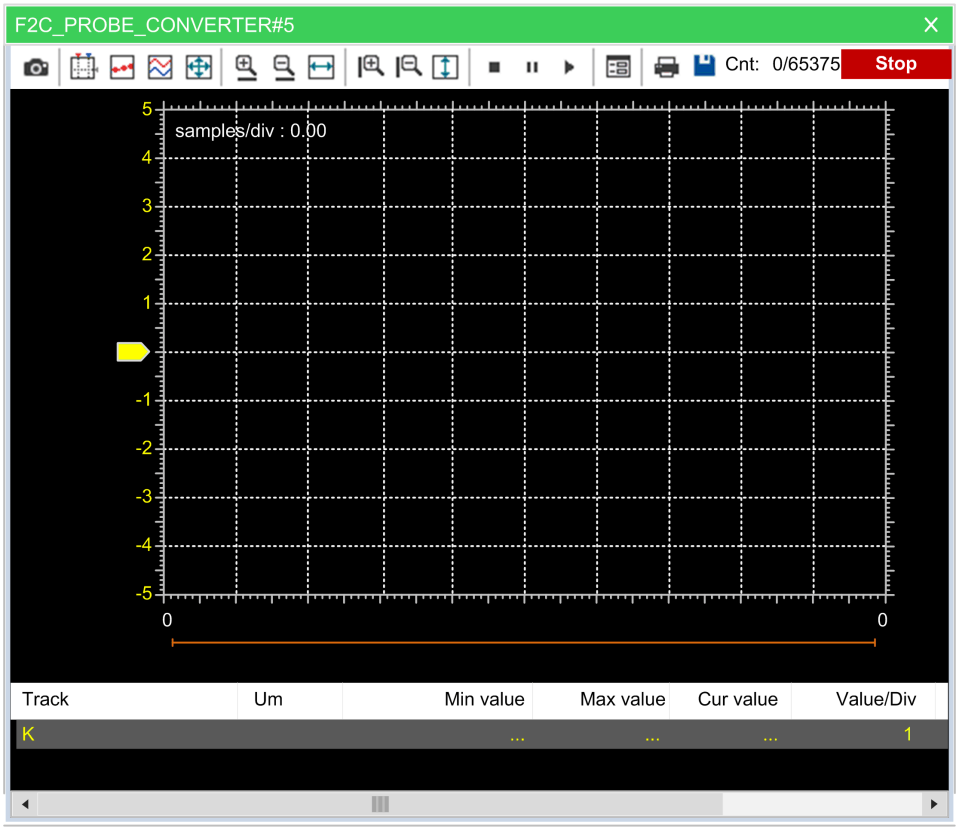
Now you can click the block representing the variable you wish to be displayed in the graphic trigger window.

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:



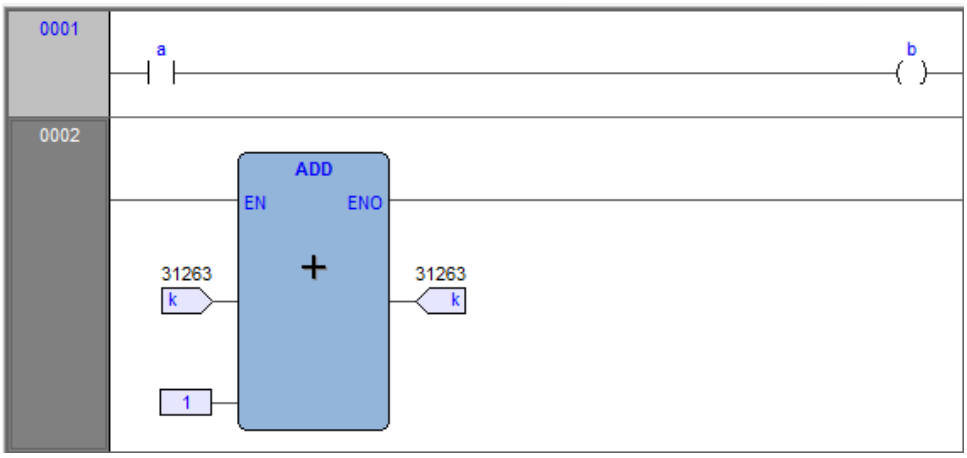
The same procedure applies to all the variables you wish to monitor.

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.

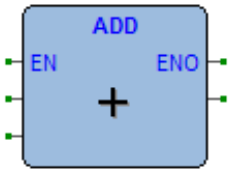
Once the first variable is dropped into a graphic trace, the **Graphic properties** window is automatically displayed and allows you to setup sampling and visualization properties.

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:

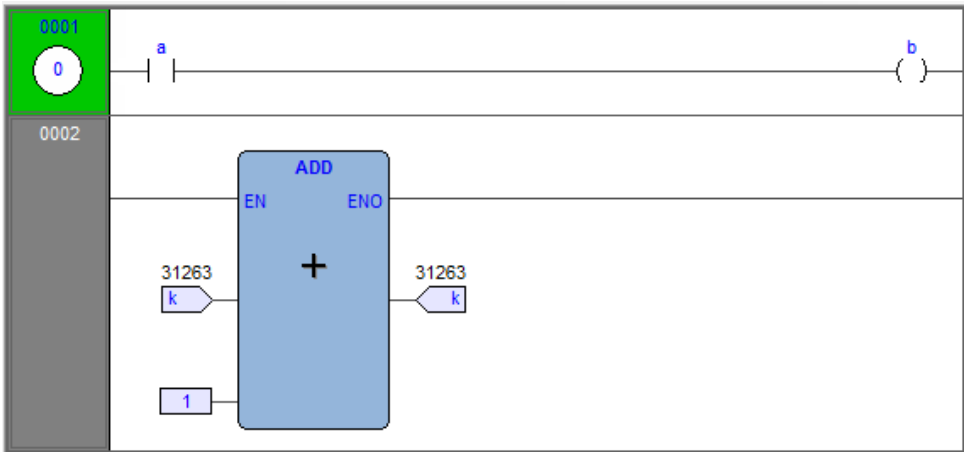


In this case, the same rules apply as to insert the graphic trigger in an FBD module.

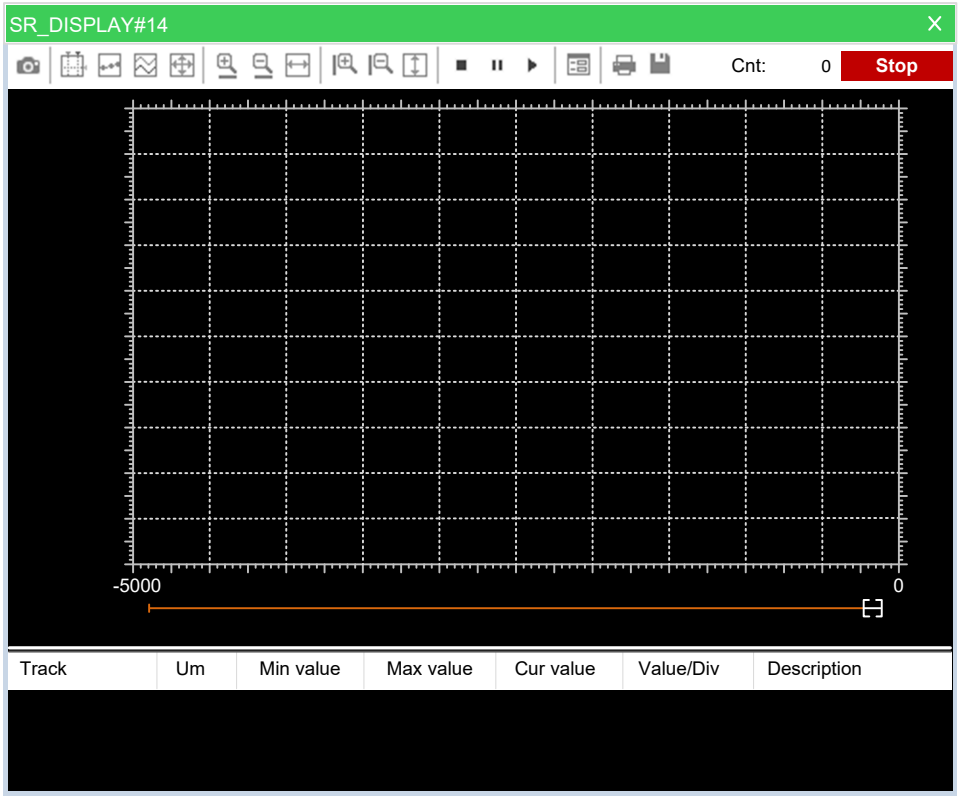
Further, assume that you want to know the value of some variables every time the processor reaches network number 1.

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.

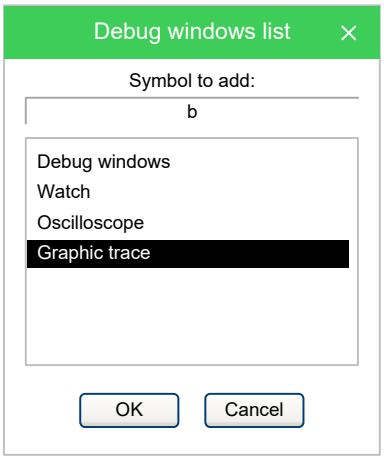
Click  **Edit > Watch mode**.

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

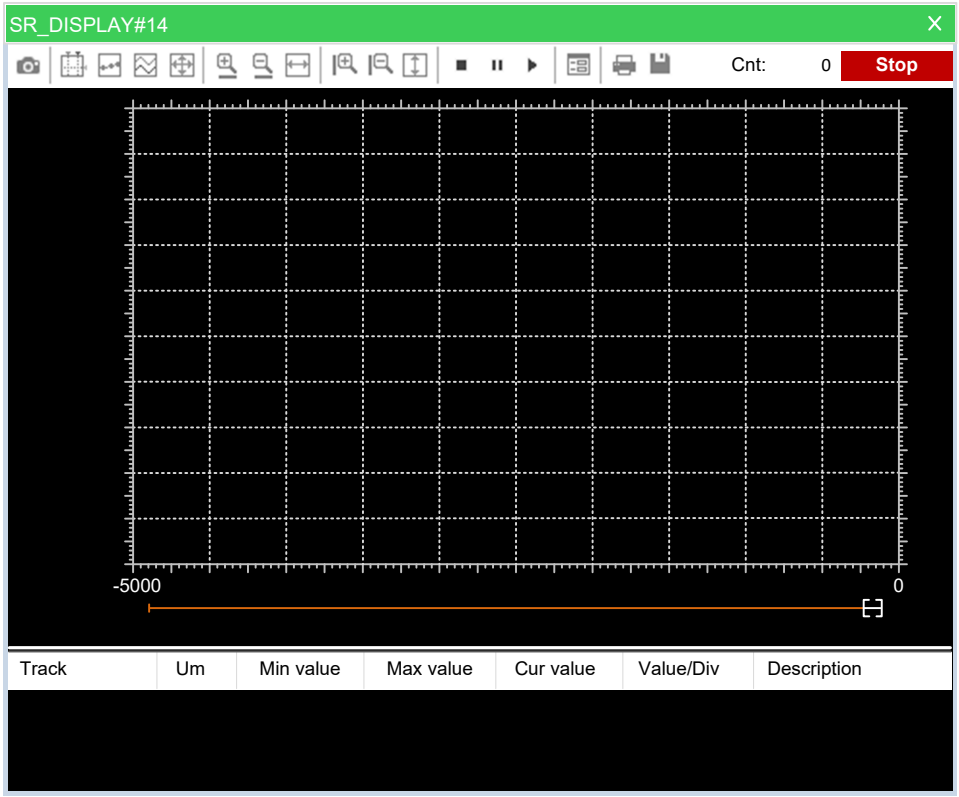
```
f := f+ SHR( d, 16#04 )
```

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
```

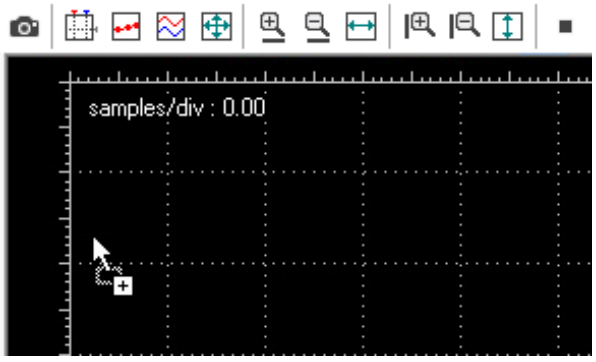


Not all the ST instructions support triggers. For example, it is not possible to place a trigger on a line containing a terminator such as *END_IF*, *END_FOR*, *END_WHILE*, and so on.

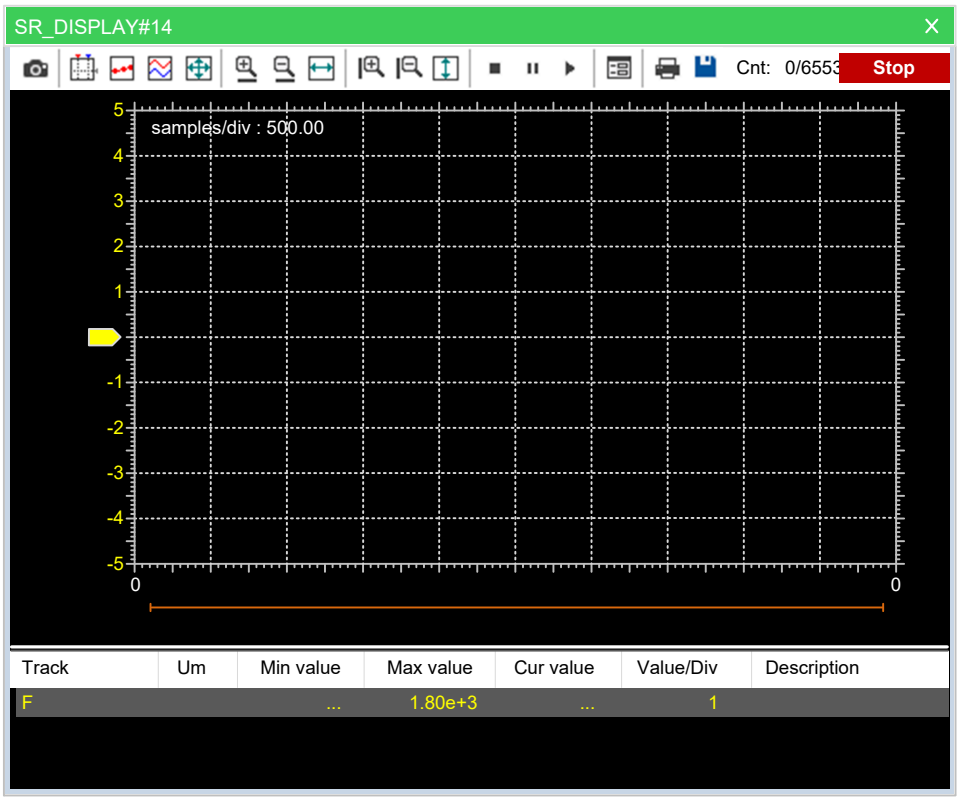
Adding a Variable to the Graphic Trigger Window from an ST Module

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:

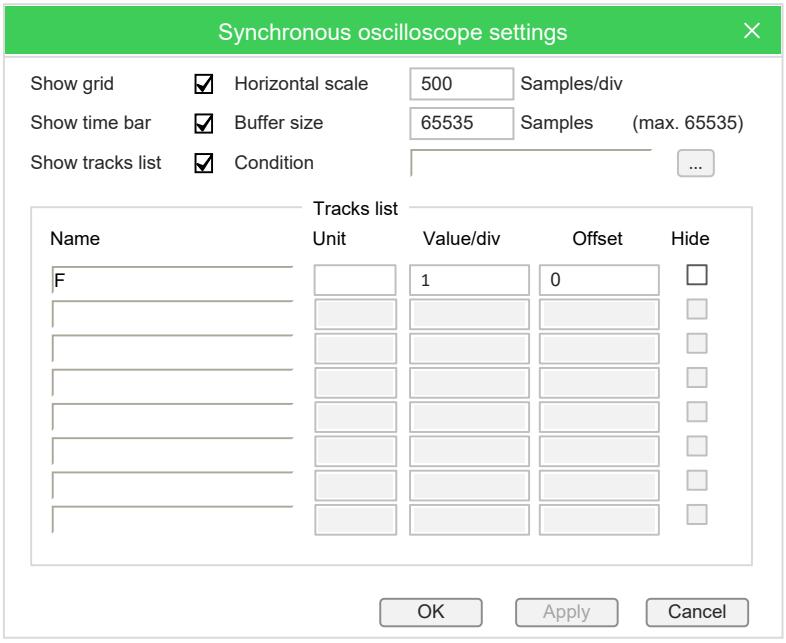


The variable now appears in the **Track** column:



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:



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.

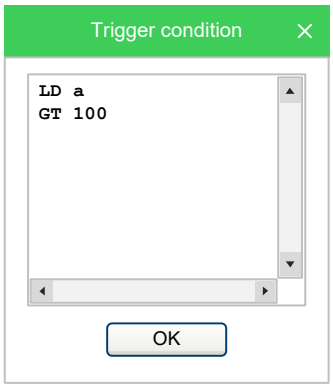
If you release the **Start graphic trace** button before all the required samples have been acquired, the acquisition process stops and all the collected data get lost.

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:

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.

Language Reference

What's in This Chapter

Description

FREE Studio Plus 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, FREE Studio Plus implements some extensions:

- Pointers
- Macros

Common Elements

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 FREE Studio Plus through specific editors, forms, and tables. FREE Studio Plus 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 FREE Studio Plus;

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	$-2^{31}…2^{31}-1$
UDINT	Unsigned long integer	32	$0…2^{32}$
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
LREAL	Long real number	64	-1.7E+308…+1.7E+308
STRING	String of characters encoded with UTF-8	-	Characters are delimited by single quotes (‘abc’)
WSTRING	String of characters encoded with UTF-16	-	Characters are delimited by double quotes (“abc”)
DATE	Date expressed in seconds represented with format YYYY-MM-DD	32	1970–01–01…2038–01-19
LDATE	Date expressed in nanoseconds represented with format YYYY-MM-DD	64	1970–01–01…2262–04-11
TIME	Time expressed in milliseconds represented with format dd_hh_mm_ss_ms	32	-24d_20h_31m_23s_648ms… +24d_20h_31m_23s_647ms
LTIME	Time expressed in nanoseconds represented with format dd_hh_mm_ss_ms_us_ns	64	-106751d_23h_47m_16s_854ms_775us_808ns…+106751d_23h_47m_16s_854ms_775us_807ns
DATE_AND_TIME	Date expressed in seconds represented with format YYYY-MM-DD-hh:mm:-ss	32	1970–01–01–00:00:00…2038–01-19–03:14:07
LDATE_AND_TIME	Date expressed in nanoseconds represented with format YYYY-MM-DD-hh:mm:ss.us	64	1970–01–01–00:00:00…2262–04-11–23:47:16.854

Keyword	Data type	Bits	Range
TIME_OF_DAY	Time of day expressed in milliseconds represented with format hh:mm:ss.ms	32	00:00:00…23:59:59.999
ITIME_OF_DAY	Time of day expressed in nanoseconds represented with format hh:mm:ss.ns	64	00:00:00…23:59:59.999999999
@ANY_TYPE	Pointer to a variable of any type (NOT IEC standard)	32/64	Refer to Pointers, page 299.
PVOID	Pointer to a generic variable, without type specified (NOT IEC standard)	32/64	Refer to Pointers, page 299.
(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 *FALSE* and *TRUE*	*FALSE* *TRUE*
For more details, refer to Two-character strings, page 252.	

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
'RL'	String of length two containing *CR* and *LF* characters (1)
'$0A'	String of length one containing the *LF* character (2)
(1) For more details, refer to Two-character strings, page 252. **(2)** The three-character combination of the dollar sign ($) 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 *$1*	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
VAR	Internal to organization unit.
VAR_INPUT	Externally supplied.

Keyword	Variable usage
VAR_OUTPUT	Supplied by organization unit to external entities.
VAR_IN_OUT	Supplied by external entities, can be modified within organization unit.
VAR_EXTERNAL	Supplied by configuration via *VAR_GLOBAL*, can be modified within organization unit.
VAR_GLOBAL	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, page 102.

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
CONST	The attribute *CONST* 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.
RETAIN	The attribute *RETAIN* 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, page 254.
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
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 FREE Studio Plus 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 FREE Studio Plus

Whatever the PLC language you are using, FREE Studio Plus 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_EXTERNAL
      declaration of external variables
   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.
`VAR_EXTERNAL . . END_VAR`	A function 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 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 FREE Studio Plus.

Declaring functions

Whatever the PLC language you are using, FREE Studio Plus 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 FREE Studio Plus.

Declaring functions

Whatever the PLC language you are using, FREE Studio Plus 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 FREE Studio Plus.

Writing programs

Whatever the PLC language you are using, FREE Studio Plus 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

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	`OUT := ABS( -5 );(* OUT = 5 *)`
	`OUT := ABS( -1.618 );(* OUT = 1.618 *)`
	`OUT := ABS( 3.141592 );(* OUT = 3.141592 *)`

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	`OUT := ACOS( 1.0 ); (* OUT = 0.0 *)` `OUT := ACOS( -1.0 ); (* OUT = PI *)`

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	`OUT := ADD( 20, 40 ); (* OUT = 60 *)`

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

ASIN	
Output data type	LREAL where available, REAL otherwise
Examples	`OUT := ASIN( 0.0 ); (* OUT = 0.0 *)` `OUT := ASIN( 1.0 ); (* OUT = PI / 2 *)`

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	`OUT := ATAN( 0.0 ); (* OUT = 0.0 *)` `OUT := ATAN( 1.0 ); (* OUT = PI / 4 *)`

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	`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 *)`

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	`OUT := CEIL( 1.95 ); (* OUT = 2.0 *)` `OUT := CEIL( -1.27 ); (* OUT = -1.0 *)`

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	`OUT := COS( 0.0 ); (* OUT = 1.0 *)` `OUT := COS( -3.141592 ); (* OUT ~ -1.0 *)`

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
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

MUL	
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	`OUT := SIN( 0.0 ); (* OUT = 0.0 *)` `OUT := SIN( 2.5 * 3.141592 ); (* OUT ~ 1.0 *)`

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	`OUT := SINH( 0.0 ); (* OUT = 0.0 *)`

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	`OUT := SQRT( 4.0 ); (* OUT = 2.0 *)`

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	`OUT := SUB( 10, 3 ); (* OUT = 7 *)`

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	`OUT := TAN( 0.0 ); (* OUT = 0.0 *)` `OUT := TAN( 3.141592 / 4.0 ); (* OUT ~ 1.0 *)`

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	`OUT := TANH( 0.0 ); (* OUT = 0.0 *)`

* 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	`LD x`
	`R y`
	`ST z`

S	
Description	Boolean set.
Number of operands	1
Input data type	BOOL
Output data type	BOOL

Examples	`LD x`
	`S y`
	`ST z`

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	`OUT := ROL( IN := 16#1000CAFE, 4 );` `(* OUT = 16#000CAFE1 *)`

ROR	
Description	Input #0 right-shifted of Input #1 bits, circular.
Number of operands	2
Input data type	Any numerical type

ROR	
Output data type	Same as Input #0
Examples	`OUT := ROR( IN := 16#1000CAFE, 16 );` `(* OUT = 16#CAFE1000 *)`

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	`OUT := SHL( IN := 16#1000CAFE, 16 );` `(* OUT = 16#CAFE0000 *)`

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	`OUT := SHR( IN := 16#1000CAFE, 24 );` `(* OUT = 16#00000010 *)`

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	`OUT := EQ( TRUE, FALSE );(* OUT = FALSE *)` `OUT := EQ( 'AZ', 'ABC' ); (* OUT = FALSE *)`

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	`OUT := GE( 20, 20 ); (* OUT = TRUE *)` `OUT := GE( 'AZ', 'ABC' ); (* OUT = FALSE *)`

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	`OUT := GT( 0, 20 ); (* OUT = FALSE *)` `OUT := GT( 'AZ', 'ABC' ); (* OUT = TRUE *)`

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	`OUT := LE( 20, 20 ); (* OUT = TRUE *)` `OUT := LE( 'AZ', 'ABC' ); (* OUT = FALSE *)`

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	`OUT := LT( 0, 20 ); (* OUT = TRUE *)` `OUT := LT( 'AZ', 'ABC' ); (* OUT = FALSE *)`

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	`OUT := NE( TRUE, FALSE ); (* OUT = TRUE *)` `OUT := NE( 'AZ', 'ABC' ); (* OUT = TRUE *)`

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*). FREE Studio Plus 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	`out := TO_BOOL( 0 ); (* out = FALSE *)`
	`out := TO_BOOL( 1 ); (* out = TRUE *)`
	`out := TO_BOOL( 1000 ); (* out = TRUE *)`

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	`out := TO_BYTE( -1 ); (* out = 16#FF *)` `out := TO_BYTE( 16#100 ); (* out = 16#00 *)`

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	`out := TO_DINT( 10.0 ); (* out = 10 *)` `out := TO_DINT( 16#FFFFFFFF ); (* out = -1 *)`

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	`out := TO_DWORD( 10.0 ); (* out = 16#0000000A *)` `out := TO_DWORD( -1 ); (* out = 16#FFFFFFFF *)`

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	`out := TO_INT( -1000.0 ); (* out = -1000 *)` `out := TO_INT( 16#8000 ); (* out = -32768 *)`

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	`out := TO_LREAL( -1000 ); (* out = -1000.0 *)` `out := TO_LREAL( 16#8000 ); (* out = -32768.0 *)`

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	`out := TO_REAL( -1000 ); (* out = -1000.0 *)` `out := TO_REAL( 16#8000 ); (* out = -32768.0 *)`

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	`out := TO_SINT( -1 ); (* out = -1 *)` `out := TO_SINT( 16#100 ); (* out = 0 *)`

TO_STRING	
Description	Conversion to STRING
Number of operands	1
Input data type	Any numerical type
Output data type	STRING
Examples	`str := TO_STRING( 10.0 ); (* str = '10,0' *)` `str := TO_STRING( -1 ); (* str = '-1' *)`

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	`str := TO_STRINGFORMAT(10, '%04d'); (* str = '0010' *)`

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	`out := TO_UDINT( 10.0 ); (* out = 10 *)` `out := TO_UDINT( 16#FFFFFFFF ); (* out = 4294967295 *)`

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	`out := TO_UINT( 1000.0 ); (* out = 1000 *)` `out := TO_UINT( 16#8000 ); (* out = 32768 *)`

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	`out := TO_USINT( -1 ); (* out = 255 *)` `out := TO_USINT( 16#100 ); (* out = 0 *)`

TO_WORD	
Description	Conversion to WORD (16-bit string)
Number of operands	1
Input data type	Any numerical type or STRING
Output data type	WORD
Examples	`out := TO_WORD( 1000.0 ); (* out = 16#03E8 *)` `out := TO_WORD( -32768 ); (* out = 16#8000 *)`

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	`OUT := TRUE AND FALSE;(* OUT = FALSE *)` `OUT := 16#1234 AND 16#5678;(* OUT = 16#1230 *)`

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	`OUT := NOT FALSE; (* OUT = TRUE *)` `OUT := NOT 16#1234;(* OUT = 16#EDCB *)`

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	`OUT := TRUE OR FALSE; (* OUT = FALSE *)` `OUT := 16#1234 OR 16#5678;(* OUT = 16#567C *)`

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
Output data type	Same as Inputs
Examples	`OUT := TRUE OR FALSE; (* OUT = TRUE *)` `OUT := 16#1234 OR 16#5678; (* OUT = 16#444C *)`

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	`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 *)`

MAX	
Description	Maximum value selection
Number of operands	2…30
Input data type	Any numerical type
Output data type	Same as max Input
Examples	`OUT := MAX( -8, 120, -1000 ); (* OUT = 120 *)`

MIN	
Description	Minimum value selection
Number of operands	2…30
Input data type	Any numerical type
Output data type	Same as min Input
Examples	`OUT := MIN( -8, 120, -1000 ); (* OUT = -1000 *)`

MUX	
Description	Multiplexer. Selects one of N inputs depending on input K
Number of operands	3…30
Input data type	Any numerical type
Output data type	Same as selected Input
Examples	`OUT := MUX( 0, A, B, C ); (* OUT = A *)`

SEL	
Description	Binary selection
Number of operands	3
Input data type	BOOL, Any, Any
Output data type	Same as selected Input
Examples	`OUT := SEL( G := FALSE, IN0 := X, IN1 := 5 ); (* OUT = X *)`

Standard Operators

ADR	
Description	Address of
Number of operands	1
Input data type	Any type
Output data type	Pointer to type
Examples	`ptr_x := ADR(x)`

IMOVE	
Description	Query interface
Number of operands	1
Input data type	Any interface type
Output data type	Any interface type
Examples	`intf1 ?= obj1;`

JMP	
Description	Jump (conditioned/negated)
Number of operands	0
Input data type	Input
Output data type	Label name
Examples	`JMP mylabel;`

MOVE	
Description	Move
Number of operands	1
Input data type	Any type
Output data type	-
Examples	`MOVE x, y;`

REF	
Description	Reference to
Number of operands	0
Input data type	Any type
Output data type	Reference to type
Examples	`ref_x = REF(x);`

RET	
Description	Return (conditioned/negated)
Number of operands	0
Input data type	-
Output data type	-
Examples	`RET;`

SIZEOF	
Description	Size of
Number of operands	1
Input data type	Any type
Output data type	Numeric output
Examples	`mysize := SIZEOF(myvar);`

String Functions

CONCAT	
Description	Character string concatenation
Number of operands	2
Input data type	STRING
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' *)`

Instruction List (IL)

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

Operator	Modifiers	Supported operand types: Acc_type, Op_type	Semantics
MOD	*(*	Any but BOOL	Modulo-division
GT	*(*	Any but BOOL	Comparison: greater than
GE	*(*	Any but BOOL	Comparison: greater or equal
EQ	*(*	Any but BOOL	Comparison: equal
NE	*(*	Any but BOOL	Comparison: not equal
LE	*(*	Any but BOOL	Comparison: equal or less
LT	*(*	Any but BOOL	Comparison: less than
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.

Function Block Diagram (FBD)

Overview

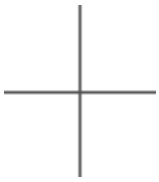
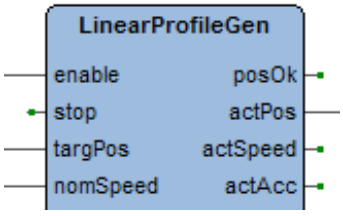
Description

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	
Blocks with connecting lines and unconnected pins	

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

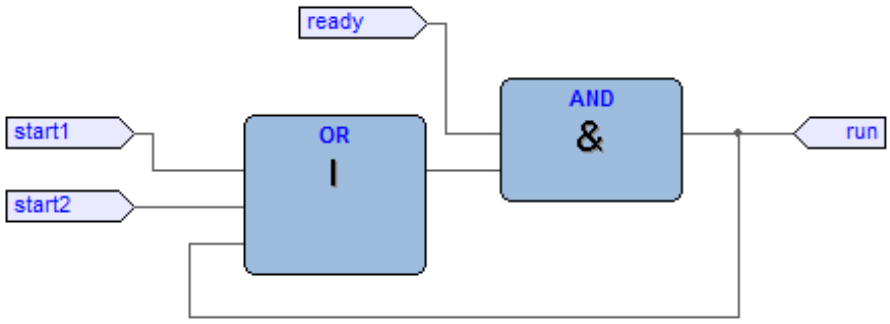
A feedback path is said to exist in a network when the output of a function or function block is used as the input to a function or function block which precedes it in the network; the associated variable is called a feedback variable.

Feedback paths can be utilized subject to the following rules:

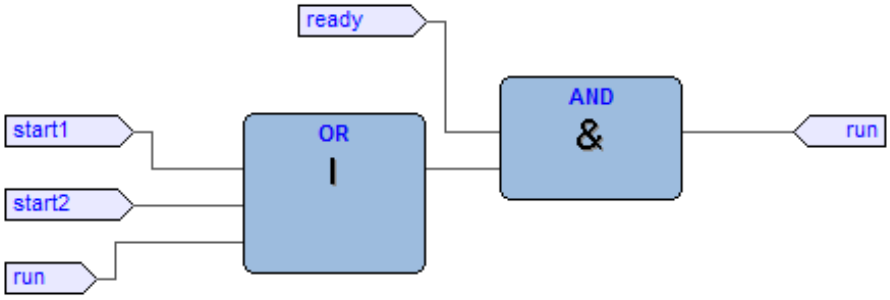
- Feedback variables must be initialized, and the initial value is used during the first evaluation of the network. Look at the global variables editor, the local variables editor, or the parameters editor to know how to initialize the respective item.
- Once the element with a feedback variable as output 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

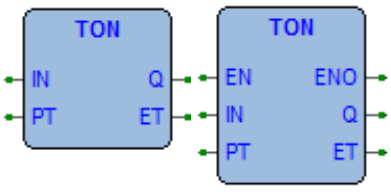
EN/ENO Signals

Additional boolean *EN* (Enable) input and *ENO* (Enable Out) characterize FREE Studio Plus 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, page 149 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

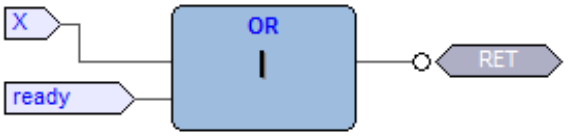
Jumps are represented by a boolean signal line terminated in a double arrowhead. The signal line for a jump condition originates at a boolean variable, or at a boolean output of a function or function block. A transfer of program control to the designated network label occurs when the boolean value of 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

Ladder Diagram (LD)

Overview

Description

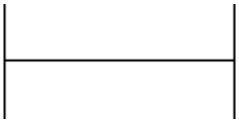
This section defines the semantics of the LD (Ladder Diagram) language.

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 FREE Studio Plus 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

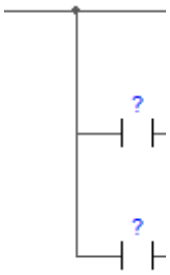
Description

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.

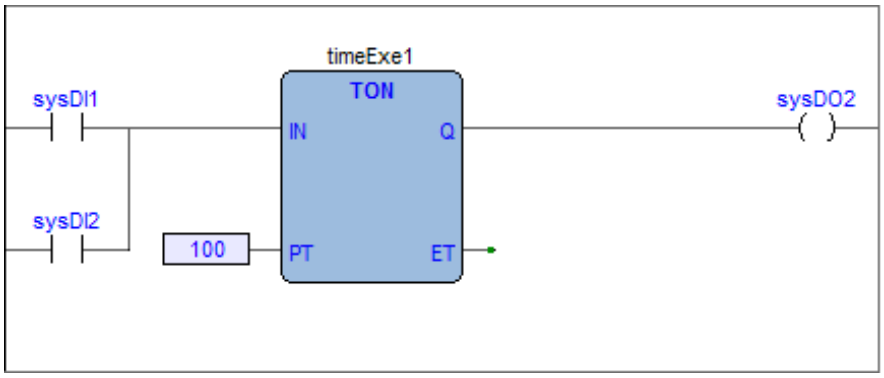
Standard coil symbols are presented in the following table:

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:



Structured Text (ST)

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, page 248. FREE Studio Plus 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	.
Boolean Exclusive *OR*	XOR	.
Boolean *OR*	OR	. . LOWEST

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;
- FREE Studio Plus 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.

FREE Studio Plus 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 ;
```

IFDEF Statement to Exclude a Portion of Code

LogicLab allows you to exclude from the compilation only a specific portion of code and checking if a certain symbol is defined, using the *IFDEF* feature.

First of all, the *IFDEF* feature needs to be enabled, since it's not an IEC standard feature. You can do that by selecting **Project > Project options > Code generation** and then checking the *Enable preprocessor directives* checkbox.

Refer to Code Generation Tab, page 102.

IMPORTANT: This feature is available only in ST, LD and FBD languages.

This feature will exclude from compilation the selected code only if a specified symbol has not been defined; the symbol to be specified can be any symbol or POU (program, function, function block, global variable...) but ***it must be GLOBALLY VISIBLE***.

Using IFDEF in ST Languages

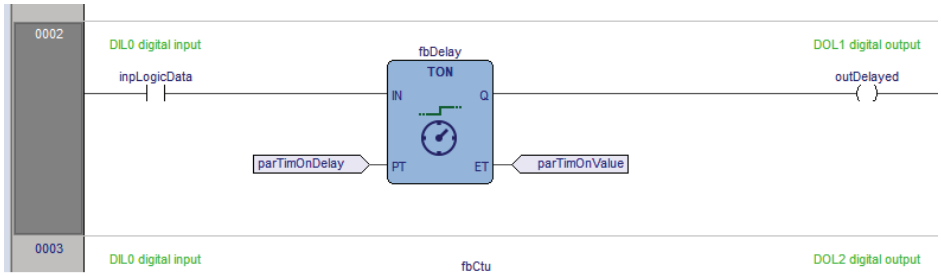
Inside an ST program, you can disable a portion of code including it inside the IFDEF syntax.

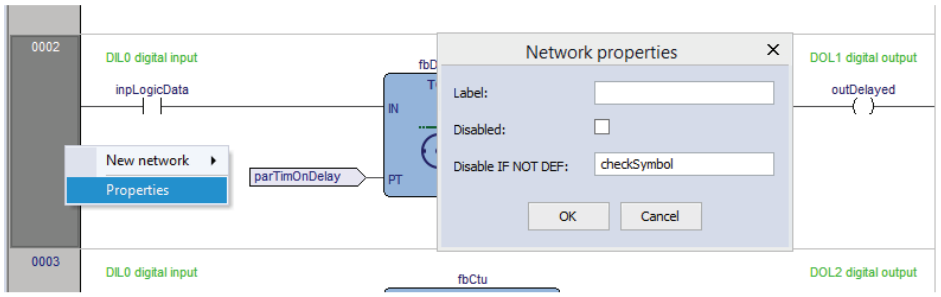
```
{ IFDEF : checkSymbol }
        loopsValue := 0;
        for I := 0 to 15 do
                bit := (y + 0.9) > (0.125*TO_REAL(i));
                if bit then
                        loopsValue := loopsValue or RotateBit(i);
                end_if;
        end_for;
{ ENDIF }
```

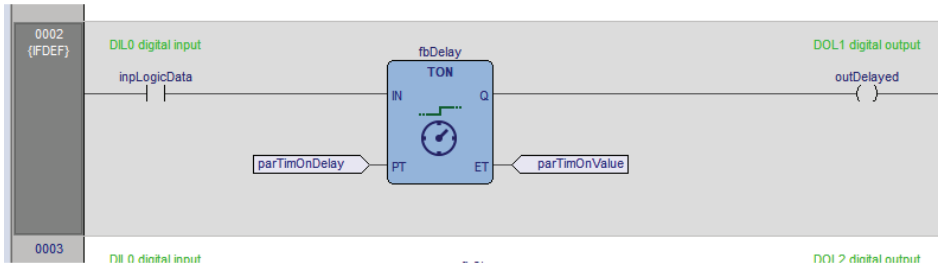
Using IFDEF in LD Languages

Inside an LD program, user can put under IFDEF condition every single network, but not just a portion of a network.

Must be opened the network properties window, and inserted the specific symbol to be checked.



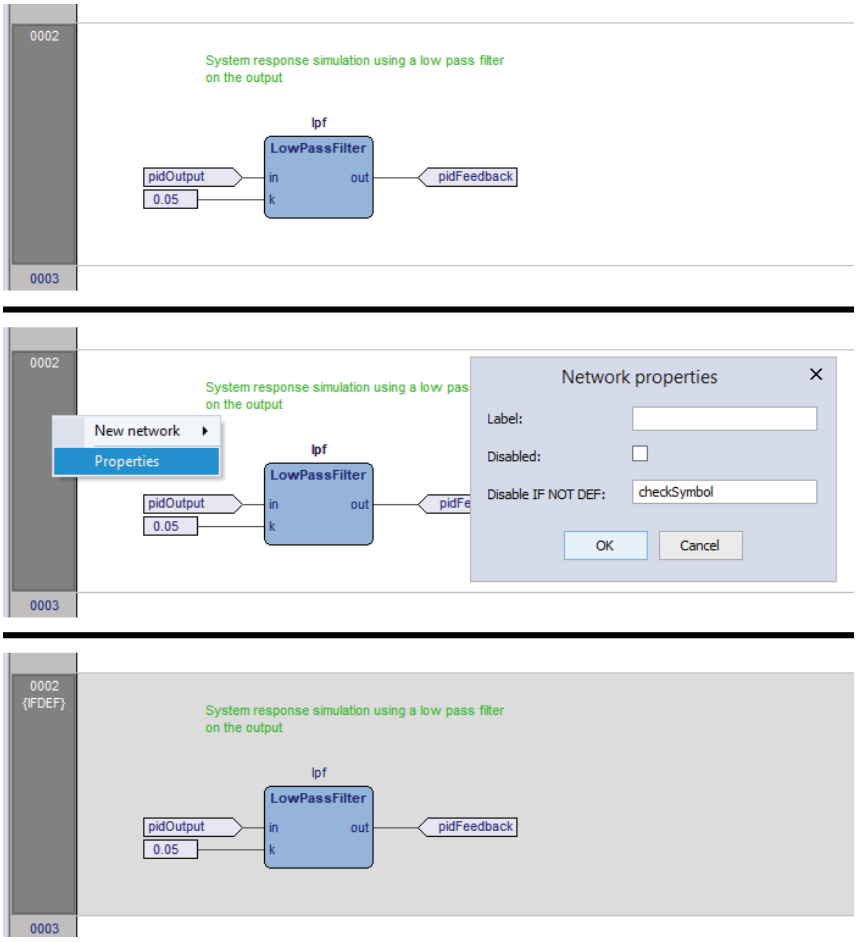




Using IFDEF in FBD Languages

Like LD language, also with FBD is possible to put under IFDEF condition every single network, but not just a portion of a network.

Must be opened the network properties window, and inserted the specific symbol to be checked.



IFDEF Supported Format

The condition of a valid IFDEF syntax can be more complex thag just a globally visible symbol; here's some example of valid IFDEF syntax:

- {IFDEF: symbol_1}
- {IFDEF: symbol_1 AND symbol_2}
- {IFDEF: symbol_1 OR symbol_2}
- {IFDEF: symbol_1 AND (symbol_2 OR symbol_3)}
- {IFDEF: symbol_1 AND NOT symbol_2}
- {IFDEF: symbol_1 OR NOT symbol_2}

Be aware that currently, due to an implementation limit, the negation of an expression is not supported; that means that the following syntax is NOT supported:

- {IFDEF: symbol_1 AND NOT (symbol_2 AND symbol_3)}

The NOT statement must be used with a symbol, not with an expression.

Sequential Function Chart (SFC)

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

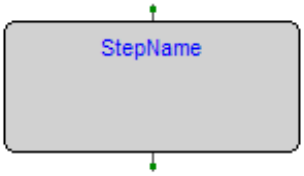
The SFC elements provide a means of partitioning a PLC program organization unit into a set of steps and transitions interconnected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition.

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.

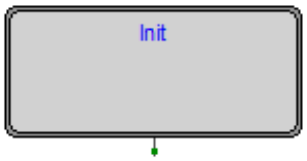
Representation	Description
	Step (graphical representation with direct links)

FREE Studio Plus does not set any constraint on the maximum number of steps per SFC within the bounds of the available memory.

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.

FREE Studio Plus 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 **Isc** 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.

FREE Studio Plus 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.

Representation	Description
	Jump (logical link to the destination step)

Transitions

Definition

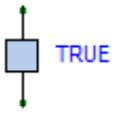
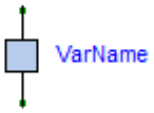
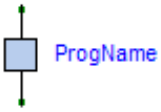
A transition represents the condition whereby control passes from one or more steps preceding the transition to one or more successor steps along the corresponding directed link. The transition is represented by a small gray square across the vertical directed link.

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 {*TRUE, FALSE*} 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 FREE Studio Plus, 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:

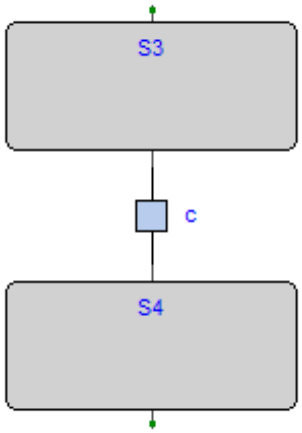
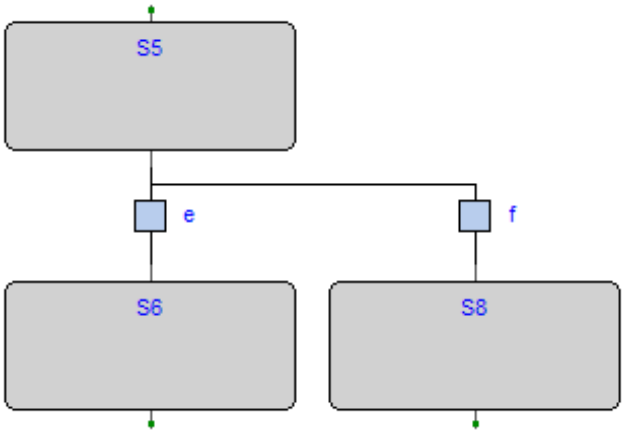
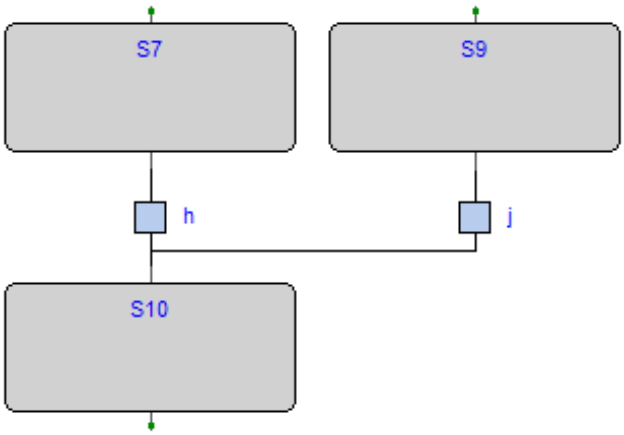
- Two steps are never directly linked; they are always separated by a transition;
- Two transitions are never directly linked; they are always separated by a step.

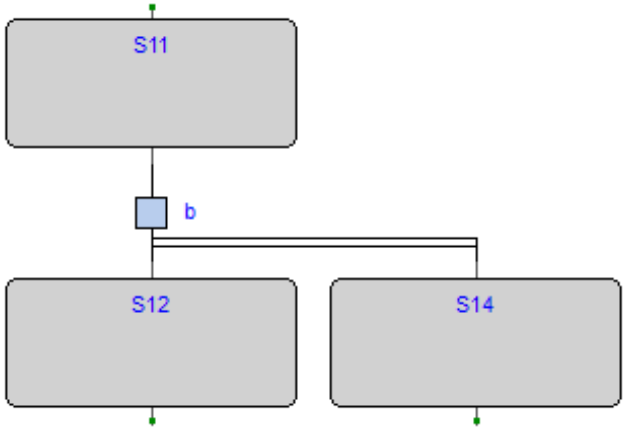
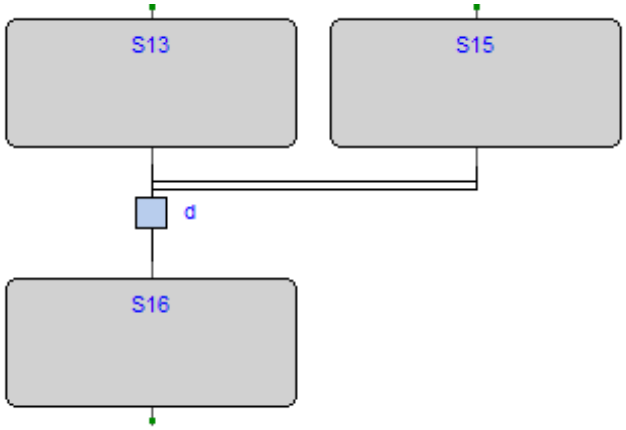
When the clearing of a transition leads to the activation of several steps at the same time, the sequences which these steps belong to are called simultaneous sequences. After their simultaneous activation, the evolution of each of these sequences becomes independent. In order to emphasize the special 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.

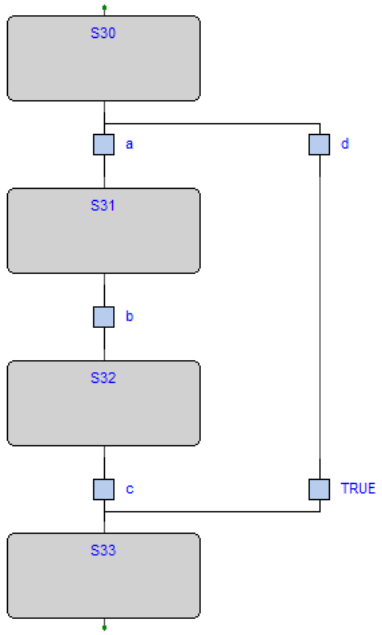
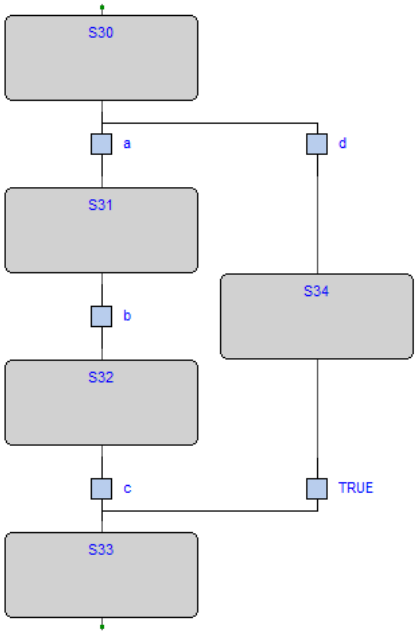
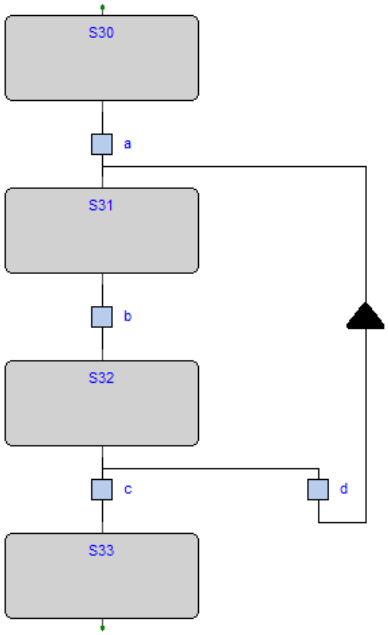
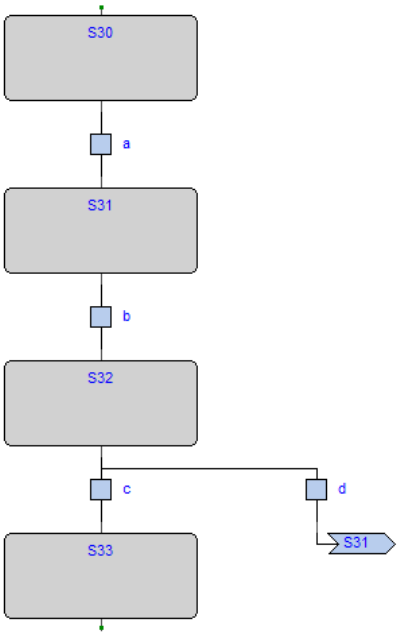
Sequence Evolution Table

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*.
	**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*.
	**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

FREE Studio Plus provides some control flags for SFC program or function blocks.

To enable this feature, refer to paragraph Code generation, page 102.

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, trough <*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.

FREE Studio Plus 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, FREE Studio Plus 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, FREE Studio Plus 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

FREE Studio Plus makes available a library called **SFCControl.pll** to allow you to manage the SFC states trough 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 form 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 *).`

FREE Studio Plus Language Extensions

Overview

Description

FREE Studio Plus 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

FREE Studio Plus 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.

Use of PVOID type

Beware that the pointer type and the pointed variable type must be of the same type; else an error message is raised when compiling. To avoid type mismatching you can use PVOID type as pointer type, this way the pointed type will be always accepted.

Waiting Statement

Description

FREE Studio Plus 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, page 102.

Errors Reference

Compile Time Error Messages

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 index 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.
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	-
A4122	Invalid bit mapped type	Bit mapped variables must be of type BOOL
A4123	Invalid access to private member	-
A4124	Invalid datablock type for bit mapping	-
A4126	Invalid label specification	-
A4127	Not a function	Invalid function specification
A4128	Invalid bit mapping index	-
A4129	Not a structured type	-

Error code	Short description	Explanation
A4130	Not a function block instance	-
A4131	Incompatible external declaration	-
A4132	Label not found	-
A4133	Not a variable	-
A4134	Index exceeds array size	Index value is out of the array range
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	-
A4142	Invalid symbol specification	-
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	-
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	-
A4157	Invalid init element	-
A4158	Invalid target function table entry	-
A4159	Invalid bit access syntax	-
A4160	Invalid bit string type	Bit access allowed only on bit string data types (BYTE, WORD, DWORD)
A4161	Invalid bit index	-
A4162	Object is not a method	-
A4163	Method not found	-
A4164	Invalid usage of THIS/SUPER	-
A4165	Parent function block not found	-
A4166	Variable name already used into a parent	-
A4167	Erroneus method override	Return value or input variables mismatch
A4168	Erroneus local method override	Override of a parent method belonging to a locally implemented interface
A4169	Not an interface instance	-

Error code	Short description	Explanation
A4170	Not a reference	-
A4171	Error dereferencing interfaces	Interfaces can not be dereferenced from references/ pointers
A4172	Relocation table generation failure	-
A4173	Bit mapped variables cannot be arrays	-
A4174	64-bit variables must be aligned to a 64-bit boundary	On some architecture is required that 64-bit variables are mapped on memory address alinged to 64-bit
A4175	Enum base type must be DINT	-
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	-
C0035	IFDEF directives are not enabled	-
C0036	Syntax error in IFDEF condition	-
C0037	Recursive IFDEF condition	-
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.
C4099	Invalid location specification	The address string of the indicated variable is not correct, the data access type indication is not 'I', 'Q' or 'M'.
C4100	Invalid location type	The address string of the indicated variable is not correct, the data type indication is not '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	-
C4110	Invalid binary file for initialization	-
C4112	Duplicate type name	-
C4353	Duplicate label	The indicated label has already been defined in the current POU (program, function or function block).

Error code	Short description	Explanation
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 an 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	-
C4618	Invalid constant wstring	-
C4619	Time constant exceedes maximum value	-
C4620	LTime constant exceedes maximum value	-
C4621	A non-most significant time unit exceede its range	-
C4622	A non-least significant time unit has a decimal part	-
C4623	Invalid date constant	-
C4624	Invalid Date and Time typed constant	-
C4626	Invalid Time of Day typed constant	-
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	Empty init value	-
C16387	Invalid structure init value	Invalid element name in structure init value
C16388	Unexpected token	-
C16389	Syntax error	-
C16390	Invalid function declaration	Function declaration must begin at line one
C16391	Invalid variable init value	Initial value must begin on the same line as the variable name
C16392	Invalid description	Description exceeded 1024 characters
C16393	Invalid POU name declaration	Declared POU name does not match actual POU name
C16394	Missing POU header	Missing POU header (e.g.: PROGRAM main)

Error code	Short description	Explanation
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 an 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.
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)	-
F1035	Undefined function block	-
F1036	Missing VAR_IN_OUT assignment	-
F1037	Unknown function	-
F1038	Unavailable default value for function parameter	-
F1039	Invalid pin	-
F1040	Only variables with physical storage can be assigned to VAR_ IN_OUT	-
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 or function, 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 aren't 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	-
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	-

Error code	Short description	Explanation
G0059	Operand extension	-
G0060	Division by zero	-
G0061	Comparison between different types	-
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	-
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 real	-
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	-
G0094	Init value type mismatch	-

Error code	Short description	Explanation
G0095	Improper init value	-
G0096	Init value object not found	-
G0097	Invalid assignment to pointer	-
G0098	Unsupported data type	-
G0099	Variable bit access not supported	-
G0100	Symbolic initialization of constants not supported	-
G0101	Type mismatch in assignment	-
G0102	Array size mismatch in assignment	-
G0103	Copy of array or structures not supported	-
G0104	Data size mismatch in assignment	-
G0105	Copy of data having a large size (see threshold in project options)	-
G0106	Object oriented features not supported	-
G0107	Recursive usage of function	-
G0108	Recursive usage of method	-
G0109	Recursive usage of function block	-
G0110	Parent function block not found (with EXTENDS)	-
G0111	Recursive inheritance (with EXTENDS)	-
G0112	Object oriented programming not supported by target system	-
G0113	Undefined interface (with IMPLEMENTS)	-
G0114	Incomplete interface implementation (with IMPLEMENTS)	-
G0115	Method prototype differs from interface definition	-
G0116	Redundant interface implementation	-
G0117	Function block does not implements interface	-
G0118	Copy between different interfaces	-
G0119	Parent interface not found	-
G0120	Recursive interface hierarchy (EXTENDS)	-
G0121	Method redefinition in interface hierarchy (EXTENDS)	-
G0122	Invalid operands for query interface operator ?=	-
G0123	Invalid assignment to reference	-
G0124	Can not load reference/address of an interface	-
G0125	Invalid operation on reference	-
G0126	Improper assignment to a reference, different type	-
G0127	Usage of deprecated pointer initialization, use NULL instead	-
G0128	Comparison between pointer and non-pointer	-
G0129	Comparison between reference and non-reference	-
G0130	Operation between pointer and non-pointer	-
G0131	Check for division by zero unsupported for LREAL type	-

Error code	Short description	Explanation
G0132	Mismatch in ENUM data types	-
G0133	Operation between ENUM and generic constant	-
G0134	Operation requires explicit type cast	-
G0135	Operation required an implicit type cast	-
G0136	Type cast is not allowed	-
G0137	Initialization of constants with addresses is not allowed	-
G0138	Illegal conversion to pointer	-
G0139	Array dimension constant not found	-
G0140	Invalid constant for array size	-
G0141	Invalid pointer arithmetic operation	-
G0142	VAR_IN_OUT cannot be a reference	-
G0143	VAR_IN_OUT can be assigned to other VAR_IN_OUT only	-
G0144	Only variables can be assigned to VAR_IN_OUT	-
G0145	Invalid MOVE operation	-
G0146	Found invalid instruction in patch code, could not set breakpoint/ trigger	-
G0147	Variable bit access with variable index not supported	-
G0148	Invalid array size indication	-
G0149	Invalid operand on function call	-
G0150	Argument types mismatch on function call	-
G0151	Operand types mismatch on function invocation	-
G0152	Time parameter not allowed	-
G0153	Converting a time into a number	-
G0154	Converting a time into a string	-
G0155	Converting a time into a bool	-
G0156	Converting a number into a time	-
G0157	Converting a string into a time	-
G0158	Implicit conversion of Time to LTime	-
G0159	Cannot convert an LTime into a Time implicitly	-
G0160	Invalid operation with a time typed operand	-
G0161	Operation not allowed on Time operand	-
G0162	Destination type not supported for Time type	-
G0163	Destination type not supported for LTime type	-
G0164	Implicit conversion of DATE to LDATE	-
G0165	Destination type not supported for DATE type	-
G0166	Destination type not supported for LDATE type	-
G0167	Cannot convert an LDATE into a DATE implicitly	-
G0168	Converting a date into a number	-
G0169	Converting a date into a string	-
G0170	Converting a date into a bool	-

Error code	Short description	Explanation
G0171	Converting a number into a date type	-
G0172	Operation not allowed on date operand	-
G0173	Operation between a date type operand and a non date type operand is not allowed	-
G0174	Operation between different date type operands (Date and LDate) is not allowed	-
G0175	Operation not allowed on TIME or LTIME	-
G0176	Operation not allowed on DATE or LDATE	-
G0177	Cannot convert a DATE_AND_ TIME into a DATE implicitly	-
G0178	Cannot convert a DATE_AND_ TIME into an LDATE implicitly	-
G0179	Cannot convert an LDATE_AND_ TIME into a DATE implicitly	-
G0180	Cannot convert an LDATE_AND_ TIME into an LDATE implicitly	-
G0181	Implicit conversion of DATE_AND_ TIME to LDATE_AND_TIME	-
G0182	Cannot convert an LDATE_AND_ TIME into a DATE_AND_TIME implicitly	-
G0183	Operation between a date and time type operand and a non date and time type is not allowed	-
G0184	Operation between different date and time type operands (DT and LDT) is not allowed	-
G0185	Operation not allowed on date and time type operand	-
G0186	Operation not allowed on DATE_ AND_TIME or LDATE_AND_TIME	-
G0187	Converting a date and time type into a string	-
G0188	Converting a DATE into a DATE_ AND_TIME	-
G0189	Converting a LDATE into a DATE_ AND_ TIME	-
G0190	Converting a DATE into a LDATE_ AND_ TIME	-
G0191	Converting a LDATE into a LDATE_ AND_ TIME	-
G0192	Destination type not supported for DATE_ AND_TIME type	-
G0193	Destination type not supported for LDATE_ AND_TIME type	-
G0194	Date typed parameter not allowed	-
G0195	Date and time typed parameter not allowed	-
G0196	Converting a floating point into a time	-
G0197	Converting a bool into a time type	-
G0198	Converting a bool into a date and time type	-
G0199	Converting a bool into a date and time type	-
G0200	Converting a string into a date and time type	-
G0201	Converting a number into a date and time type	-
G0202	Converting a date type into a time type	-
G0203	Converting a date and time type into a time type	-

Error code	Short description	Explanation
G0204	Converting a floating point into a date type	-
G0205	Converting a string into a date type	-
G0206	Converting a bool into a date type	-
G0207	Converting a time type into a date type	-
G0208	Converting a time type into a date and time type	-
G0209	Converting a date type into a floating point	-
G0210	Converting a date and time type into a floating point	-
G0211	Converting a date and time type into a number	-
G0212	Converting a date and time type into a bool	-
G0213	Converting a String into WString	-
G0214	Converting a WString into String	-
G0215	Converting a string into a bool	-
G0216	Operation not allowed on TIME_ OF_DAY or LTIME_OF_DAY	-
G0217	Cannot convert an DATE_AND_ TIME into a TIME_OF_DAY implicitly	-
G0218	Cannot convert an DATE_AND_ TIME into an LTIME_OF_DAY implicitly	-
G0219	Cannot convert an LDATE_AND_ TIME into an TIME_OF_DAY implicitly	-
G0220	Cannot convert an LDATE_AND_ TIME into an LTIME_OF_DAY implicitly	-
G0221	Implicit conversion of TIME_OF_ DAY to LTIME_OF_DAY	-
G0222	Destination type not supported for TIME_OF_ DAY type	-
G0223	Destination type not supported for LTIME_ OF_DAY type	-
G0224	Cannot convert an LTIME_OF_DAY into a TIME_OF_DAY implicitly	-
G0225	Operation between a time of day operand and a non time of day operand is not allowed	-
G0226	Operation between different time of day type operands (TOD and LTOD) is not allowed	-
G0227	Operation not allowed on time of day type operand	-
G0228	Time of day typed parameter not allowed	-
G0229	Converting a time of day type into a bool	-
G0230	Converting a time of day type into a floating point	-
G0231	Converting a time of day type into a number	-
G0232	Converting a time of day type into a string	-
G0233	Converting a time of day type into a time type	-
G0234	Converting a time of day type into a date type	-
G0235	Converting a time of day type into a date and time type	-
G0236	Converting a bool into a time of day type	-
G0237	Converting a number into a time of day type	-

Error code	Short description	Explanation
G0238	Converting a floating point into a time of day type	-
G0239	Converting a string into a time of day type	-
G0240	Converting a time type into a time of day type	-
G0241	Converting a date type into a time of day type	-
G0242	Zero length string not allowed	-
G0243	Operation not allowed on references	-
G0244	Different array/string size	-
G0245	Complex parameter not supported	-
G0246	Does not support bool accumulator	-
G0247	Does not support float accumulator	-
G0248	Time typed accumulator not supported	-
G0249	Date typed accumulator not supported	-
G0250	Date and time typed accumulator not supported	-
G0251	Time of day typed accumulator not supported	-
G0252	String typed accumulator not supported	-
G0253	Converting a number into a String	-
G0254	Converting a String into a number	-
G0255	Operation between floating point and integer	-
G0256	Operation between signed and unsigned variables	-
G0257	Converting %s to REAL	-
G0258	Converting %s to LREAL	-
G0259	Converting a number into a WString	-
G0260	Converting a WString into a number	-
G0261	Converting a WString into a bool	-
G0262	Does not support a non boolean condition	-
G0263	Operation between a string type and a non string type is not allowed	-
G0264	Operation between different time typed operands (TIME and LTIME) is not allowed	-
G0265	Branch higher of 1 MB	-
G0266	Branch higher of 16 MB	-
G0267	Branch higher of 32 MB	-
G0268	Converting %s to the type of the second operand	-
G0269	Converting %s to the type of the first operand	-
G0270	Operation between boolean and integer	-
G0271	Invalid operation on different pointed types	-
G0272	Operation on different pointed types	-
G0273	Implicit conversion of type \'%s\' to type \'%s\' is not admitted	-
G0274	Invalid conversion of type \'%s\' to type \'%s\'	-
G0275	Loss of precision while converting a %s into %s	-

Error code	Short description	Explanation
G0276	Invalid operation on operands with different granularity	-
G0277	Invalid pointer operation	-
G0278	Invalid string length indication	-
G0279	String length constant not found	-
G0280	Invalid constant for string length	-
G0281	Operation between array and scalar variables	-
G0282	Extended Unicode characters are not enabled	-
G0283	Invalid number of inout parameters	-
G0284	A variable cannot be assigned to a VAR_IN_OUT with a different type	-
G0285	All inputs shall be of the same type	-
G0286	VAR_IN_OUT on functions not implemented	-
G8193	Type definition of unknown data type	-
G8194	Type definition has exceeding array dimensions	-
G8195	Cyclic definition of data type	-
G8196	Double pointers are not supported	-
G8197	No enumerative elements	-
G8199	Invalid or undefined initialization constant	-
G8200	Global variable and ENUM field with the same name	-
G10241	Too many initializers for variable	-
G10242	Too less initializers for variable	-
G10243	Constant without init values	-
L1153	Unconnected pin	-
L1154	Jump to non existing label	-
L1155	Invalid operand	-
L1156	Undefined contact	-
L1157	Undefined variable	-
L1158	Undefined constant	-
L1159	Undefined coil	-
L1160	Undefined jump destination	-
L1161	Undefined expression	-
L1162	Assignment not admitted in expressions	-
L1163	Comments not admitted in expressions	-
L1164	Undefined function block	-
L1165	VAR_IN_OUT must be assigned in function block invocation	-
L1166	Unknown function	-
L1167	Unavailable default value for function parameter	-
L1168	VAR_IN_OUT parameters must be assigned	-
L1169	Only variables can be assigned to VAR_IN_OUT parameters	-
P2048	PostBuild error	-
P2049	Symbol table file not created	-

Error code	Short description	Explanation
P2051	Cannot create directory	-
P2052	Cannot open source project	-
P2053	Save project error	-
P2054	Generic file error	-
P2055	Cannot copy file	-
P2056	Cannot save file	-
P2057	Object already exist in project	-
P2058	Cannot open library file	-
P2059	Listing file not created	-
P2060	Cannot create PLC application binary file	-
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 actual 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	-
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	-
P2085	Missing external aux source	-
P2086	Object is already defined in the project and will be unloaded	-
S1281	Generic ST error	-
S1282	Too many expressions nested	-
S1283	No iteration to exit from	-
S1284	Missing END_IF	-

Error code	Short description	Explanation
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	-
S1313	Invalid range in CASE definition	-
S1314	Value overlap in CASE definition	-
S1315	Exceeding number of parameters	-
S1316	Wrong number of function parameters	-
S1317	Duplicated function parameter	-
S1318	Improper use of THIS/SUPER	-
S1319	Improper usage of query interface operator ? =	-
S1320	Invalid reference to expression	-
S1321	Missing IL block end marker ({IL})	-
S1322	Function in/out variable doesn’t exist	-
S1323	VAR_IN_OUT must be assigned in function block invocation	-
S1324	Complex type parameters cannot have default value	-
S1325	Invocation of an unexisting function block	-

Error code	Short description	Explanation
S1326	Missing inout parameter	-
S1537	Generic SFC error	-
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 jump destination	-
S1548	Duplicates action. Same SFC action cannot be used in more than one step	-
S1549	Unconnected block in SFC schema	-
T8193	Communication timeout	The communication with the target system failed because there is no answer from the system itself. More common causes of this problem are wrong cable connection, invalid target address in communication settings, invalid settings of communication parameters (such as baud rate), target system failure.
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 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.
T8200	Communication device unavailable	The compiler tried to communicate with the target system but the communication channel is not available. If the problem persist 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	-
T8207	Not implemented	-
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	-

Error code	Short description	Explanation
T8212	Memory read error	-
T8213	Memory write error	-
T8214	Not enough space available on the target device for the PLC application binary	-
T8215	Generic communication failure	-
X4097	Recursive POU	-
X4098	Recursive data type	-

Display

What’s in This Part

The Display Tab

What's in This Chapter

Overview of the Display Window

Purpose

Display is used to create user interfaces for embedded systems based on HMI runtime.

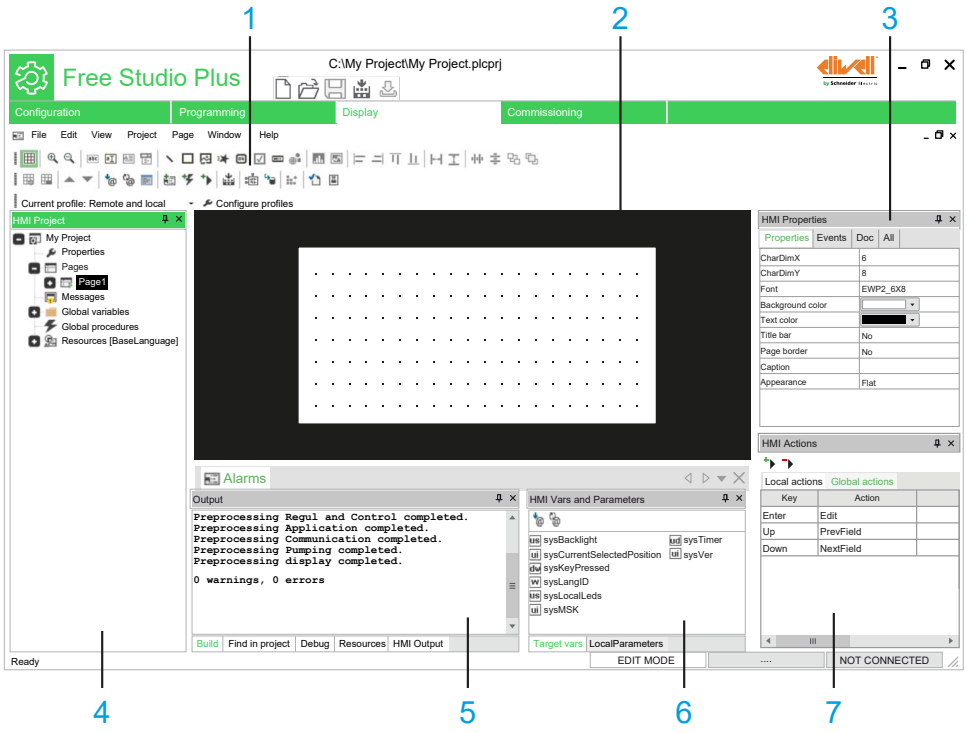
It allows you to implement graphical interfaces in a visual way. The created pages are viewed in **Display** as they appear on the final target device.

Thanks to their multi-page structure, **Display** can support HMI (human machine interface) applications with an arbitrary number of pages.

It is equipped with several tools to realize complex applications and it interfaces directly to the PLC IEC1131 **Programming** compiler for managing the variables which are defined in the target device PLC application.

Display Layout

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



Item	Description	
1	Toolbars	This toolbar shows the tools in form of icons. For more information, refer to Toolbars, page 323.
2	Editor window	This window allows you to edit the content of the current selection in **HMI Project** window.
3	**HMI Properties** window	Shows the properties and events of the selected object:

Item	Description	
		• **Properties**: the properties of the selected page or control are displayed in the form of a table and can be modified in the right column. • **Events**: the events of the currently selected object are displayed in the form of a table and each event can be associated with local or variable procedures in the right column. • **Doc**: the description of the currently selected object is displayed in the form of a table and it can be modified (or created if it does not exist) in the right column.
4	**HMI Project** window	This window includes: • **Project**: shows the project tree and the objects that compose it. Each page contains the list of: ◦ Local variables (visible and usable only in the page where they are declared), ◦ Local procedures (which can be started only from the page where they are implemented). Moreover there are the nodes of: ◦ Asynchronous messages, ◦ Global variables (visible and usable from whatever page), ◦ Global procedures which you can start from whatever page. • **Resources**: shows the project resources (fonts, bitmaps, string tables, enumerated data types, image lists, and sets).
5	**Output** window	This tool window shows the messages relating to the development of the project.
6	**HMI Vars and Parameters** window	This window is composed of two tabs. One tab contains the list of the available variables, and the other tab contains the list of the local parameters.
7	**HMI Actions** window	This window shows in tabular form the actions associated with the buttons of the target device (either according to the page currently displayed or according to the pages). Actions can be changed in the right columns.

Set of Controls

Each page may contain an arbitrary number of defined graphic controls.

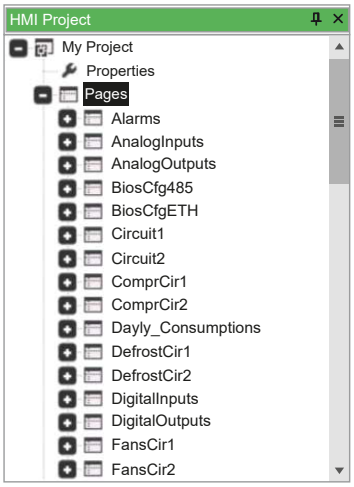
There are two classes of graphic controls:

- Static controls: drawing tools such as lines, rectangles, and figures.
- Dynamic controls: multi-layered objects, which enable data, image display, and user interaction (strings, boxes, buttons, progress, custom controls, charts, and trends).

Display is an open system allowing the implementation of custom controls which may be included in the target device system.

Multi-pages Structure

Display supports the definition of an arbitrary number of pages. Each page may contain links to other pages so that the whole project takes a tree structure:

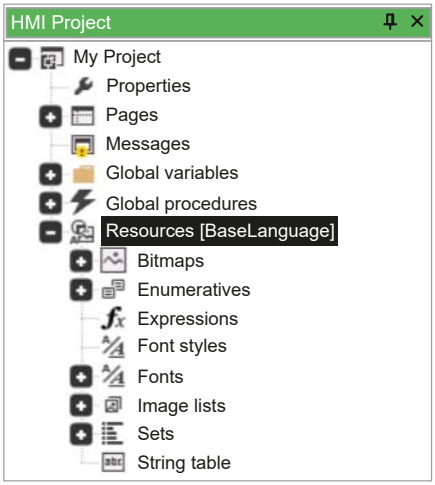


Resources Management

The properties of the controls in the page are not statically defined in the project code, but they can be managed separately as resources.

Resources include for example:

- Bitmaps, page 376
- Enumeratives, page 378
- Fonts, page 376
- Sets, page 378
- String table, page 377



Display allows you to import bitmap files directly from the Windows-formatted file (*.bmp, *.gif, *.emf, *.jpg, *.ico, and so on).

For more information about **Resources**, refer to Resources, page 376.

Variables and Procedures

Display enables the implementation of several procedures in the ST language. Through these procedures, you can customize the behavior of the logical system interface.

Example of variables:

	Name	Type	Array	Init value	Description
1	InsertedPassword	WORD	No		
2	ReqLevVis	BOOL	No		
3	AccessLevelPasswor	UINT	[0..3]		
4	i	USINT	No		
5	dummy	USINT	No		
6	ADDR_LEV0_PASSW	UINT	No	16966	
7	PASSWORD_DISABL	UINT	No	0	
8	WiredLev3Password	UINT	No	0	
9	WIRED_ELIWELL_PA	UINT	No	1603	

Local variables - [PasswordInsert] OnLoadScript - [PasswordInsert]

Example of procedures:

```
0001
0002 InsertedPassword := 0;
0003
0004 IF (PageAccessHandshake = PAGE_ACCESS_REQUESTED) THEN (* Calling page is requesting a permission access
0005 *)  IF (PermLevelCurr >= PermLevelReq) THEN
0006         PageAccessHandshake := PAGE_ACCESS_GAINED; (* Permission gained *)
0007     END_IF;
0008 ELSE (* No permission request; user manually selected login page *)
0009     PageAccessHandshake := PAGE_ACCESS_DENIED;
0010 END_IF;
0011
0012 retUINT := Video_GetParam( 0, ADDR_PSW1, 0, ?AccessLevelPasswords[1], tyUINT);
0013 retUINT := Video_GetParam( 0, ADDR_PSW2, 0, ?AccessLevelPasswords[1], tyUINT);
0014 retUINT := Video_GetParam( 0, ADDR_PSW3, 0, ?AccessLevelPasswords[1], tyUINT);
0015
0016
0017
0018
0019
```

Local variables - [PasswordInsert] OnLoadScript - [PasswordInsert]

For more information about variables and procedures, refer to Declaration of Variables, page 370.

Run-time Functionalities

Overview of the run-time functionalities:

- Managing asynchronous messages: **Display** supports the issue of asynchronous messages whatever their complexity. You can customize management of issue messages by typing a ST procedure.

  For more information about managing asynchronous messages, refer to Asynchronous Messages, page 327.

- Multilingual support: **Display** allows you to modify strings, resources, and enumerations language without recompiling nor reloading the application.

  For more information about Multilingual support, refer to Language Selection, page 338.

- Events management: **Display** applications are structured in events; you may seize the available events and manage them through ST-coded procedures.

  For more information about events management, Events, page 373.

Menu Bar

Overview

The menu bar of **Display** tab is composed of these menus:

- File, page 28
- Edit, page 27

- View, page 33
- Project, page 30
- Page, page 29
- Variables, page 32
- Window, page 34
- Help, page 28

Toolbar

Introduction

The toolbar appears at the top of the FREE Studio Plus window to provide access to frequently used functions.

For generalities of toolbars, refer to Toolbars description, page 34.

HMI Page Toolbar

The **HMI Page** toolbar has the following buttons:

Icon	Description
	**Grid** Show or Hide the grid
	**Zoom In** Zoom in the current page
	**Zoom Out** Zoom out the current page
	**Insert static** Insert a static object, page 343
	**Insert new edit** Insert a new edit box, page 348
	**Insert new text box** Reserved
	**Insert new combo box** Reserved
	**Insert line** Insert a new line, page 343
	**Insert rectangle** Insert a new rectangle, page 344
	**Insert image** Insert a new image object, page 345
	**Insert animation** Insert a new animation object, page 347

Icon	Description
	**Insert new button** Insert a new button object, page 353
	**Insert new check box** Insert a new check box, page 344
	**Insert new progress** Insert a new progress bar, page 357
	**New custom control** Reserved
	**Insert new chart** Reserved
	**Insert new trend** Reserved
	**Align left** Align the selected objects to the left
	**Align right** Align the selected objects to the right
	**Align top** Align the selected objects to the top
	**Align bottom** Align the selected objects to the bottom
	**Make same width** Make objects of the same width
	**Make same height** Make objects of the same height
	**Space across** Evenly space the selected objects horizontally
	**Space down** Evenly space the selected objects vertically
	**Send to back** Send to back
	**Bring to front** Bring to front

HMI Project Toolbar

The **HMI Project** toolbar has the following buttons:

Icon	Description
	**New record** Insert a new record
	**Remove record** Remove the current record
	**Move up** Moves the current record one position up
	**Move down** Moves the current record one position down
	**Parameters management** Parameters management
	**Refresh parameters** Refresh parameters
	**Template management** Template management
	**New page** Insert a new page object
	**New event** Insert a new event object
	**New action** Insert a new action in list
	**Compile HMI project** Compile HMI project and generate IEC code
	**Simulation mode** Simulation mode
	**Download HMI code** Download the compiled HMI project
	**Generate RSM** Generate redistributable source module
	**Generate Doc.** Generate auto documentation
	**Open compiled project to resolve compile errors** Open compiled project to resolve compile errors

HMI Profiles Toolbar

The **HMI Profiles** toolbar has the following buttons:

Icon	Description
-	**Current profile** Select the active project profile (must be activated in **Configure profiles**)
	**Configure profiles** Choose active project profiles (**Remote and local** or **Local only**)

Managing Display Elements

What's in This Chapter

Managing Pages

Pages Overview

Navigating between Pages

Display manages the creation of pages for a specific application.

It is composed of several pages where you can arbitrarily arrange the controls.

You have to specify the start page, page 333 which is displayed at the start of the target device. Other pages have at least a parent page from which they are invoked and may have a child page to invoke. The invoking/invoked relations implicitly give to the whole project a tree structure.

A child page may be invoked in two ways:

- Through an action associated to a key: associate an **OpenPage** action with a physical key (if there is a keyboard) or with a virtual key (whose pressure is an event raised by software).
- Through an action associated to a key: associate an **OpenPage** action with a physical key (if there is a keyboard) or with a virtual key (whose pressure is an event raised by software).

Pages

There are two main types of pages in **Display**:

- Child pages, page 328 (which are called **Frame** in FREE Studio Plus).
- Pop-up pages, page 330.

Asynchronous Messages

Asynchronous messages are similar to standard pages, except the following features:

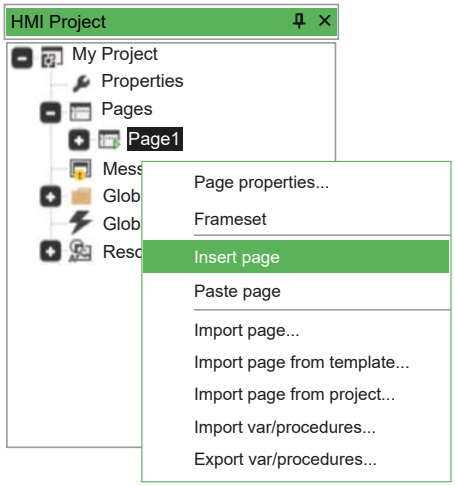
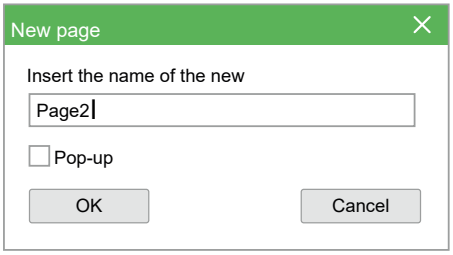
- They have an additional property, that is the identifier of the associated message (`Msg ID`).
- They can not contain invocations to child pages.
- They have no defined parent page nor a tree structure, but they can be invoked from any other standard page.

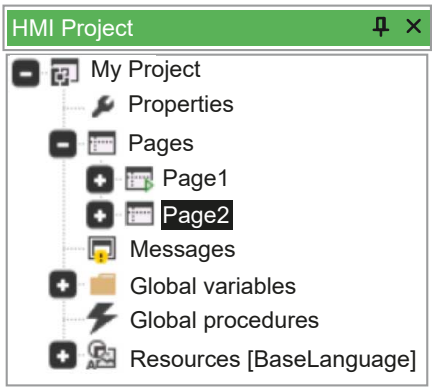
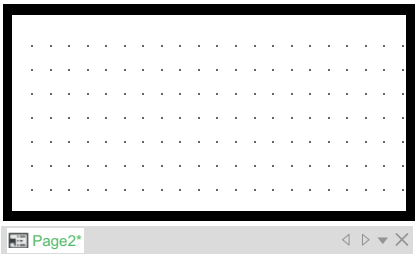
An asynchronous message can not be explicitly invoked. The system displays it whatever the active page when it intercepts a message containing the corresponding `Msg ID`. This message may be launched either by the firmware or by a procedure through the `Video_SendMessage` function by using the following syntax: `Video_SendEvent` (`kWM_MSG`, `Msg ID`).

When an asynchronous message is active, the controls of the **Frameset** (see Frameset, page 339) are automatically disabled.

Child Pages

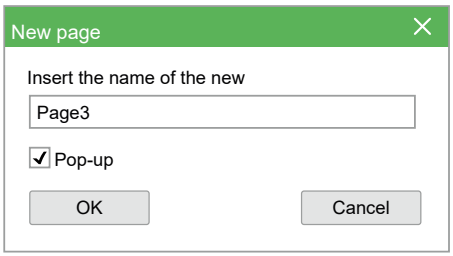
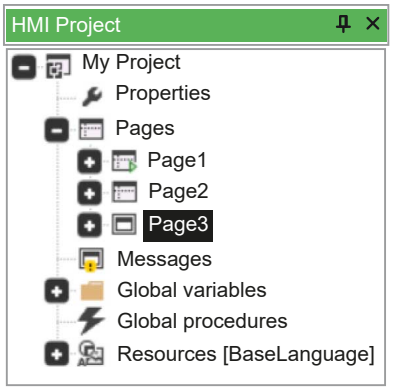
Create a Child Page

Step	Action
1	Right-click on the **Pages** item of the project tree and click **Insert page**: 
2	The **New page** dialog box appears and you have to specify the name of the new page and whether the page is a pop-up one or not.  **NOTE:** If you do not select the **Pop-up** check box when creating the new page, the page is called “Child Page”. Its main feature is that it fits the whole area. So you cannot define position and size of a child page because they are automatically set depending on the area and on an eventual frame set.

Step	Action
3	Choose to create a child page and if you want to call it “Page2” for example: type the name **Page2** in the apposite field and press **OK** to confirm your choice. A new node appears in the page folder of the project tree: 
4	Double-click the **Page2** item to open the document with this page preview, which is blank: 

Pop-up Pages

Create a Pop-up Page

Step	Action
1	Right-click on the **Pages** item of the project tree.
2	Click **Insert page** command from the contextual menu.
3	Write the name of this page (for example **Page3**) in the dialog box which appears and select the pop-up property. 
4	A new item appears in the **Pages** folder of the project tree: 

Dimensioning and Setting the Pop-Up Page

Child and pop-up pages have different icons in **HMI Project**.

Pop-up pages are not subjected to any restriction from the frameset.

When a pop-up page is active, the controls of the **Frameset** (see Frameset, page 339) are automatically disabled.

You can choose their dimensions and positions in **HMI Properties** window:

- Sets the dimensions in **XDim** and **YDim** properties.
- Sets the position in **XPos** and **YPos** properties.

Modal Property

When a pop-up page opens:

- If **Modal** property is set to **Yes**, the controls are inactive. It means that the parent page objects are disabled.
- If **Modal** property is set to **No**, the controls are disabled. It means that the parent page objects are enabled if they are completely visible.

Basic Page Settings

Viewing the Title Bar and the System Button

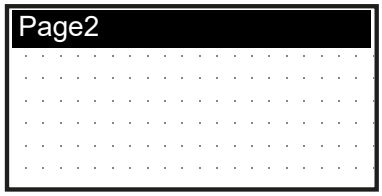
Display allows you to create a title bar for each page by selecting **Yes** in the **Title bar** property.

You can name this title bar by writing a name in **Caption** text box.

If you want to activate the title bar and the close button, and to display the **Page2** string as title, set as follow:

Title bar	Yes
Page border	Yes
Caption	Page 2
Appearance	Flat

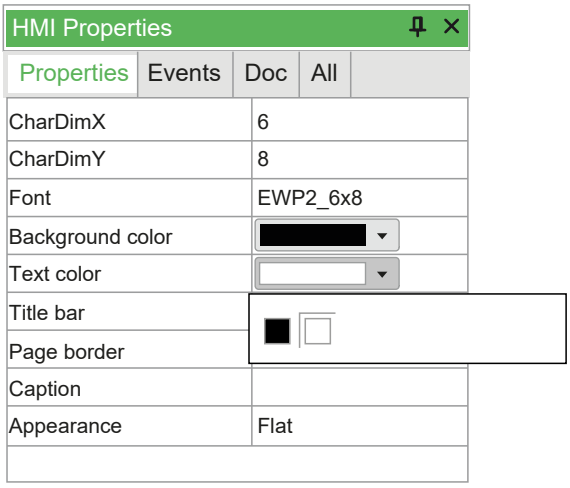
Then the page looks like the following picture:



Editing the Colors of the Page

You can edit the background color of the page and the text color through the **HMI Properties** window: click the drop-down menu in the right-column of **Background color** or **Text color** field.

The palette of colors appears, select the desired one:

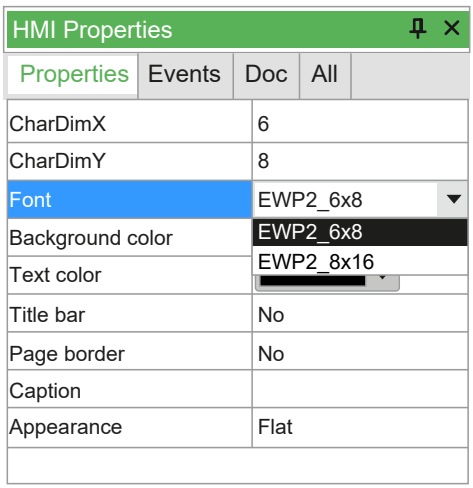


NOTE: For monochrome screens, it is recommended to choose white as background color and black as default text color.

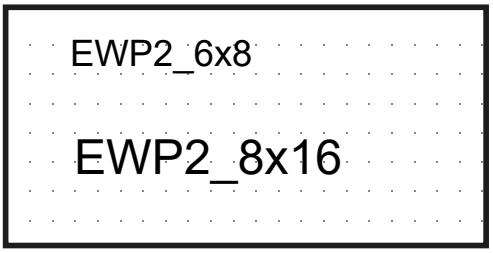
Changing the Font

You can change the font of textual elements on the page through the **HMI Properties** window: click the drop-down menu in the right-column of **Font**.

The different fonts available appear. Choose the one which suits you the best:



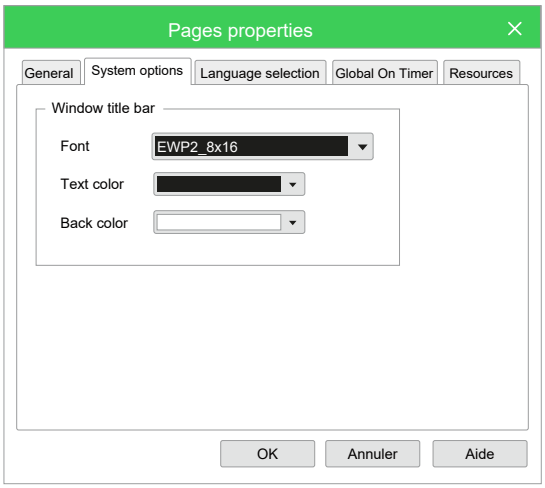
NOTE: EWP2_6x8 and **EWP2_8x16** are the same font but with a different size (the second font is larger).



Window Title Bar Parameters

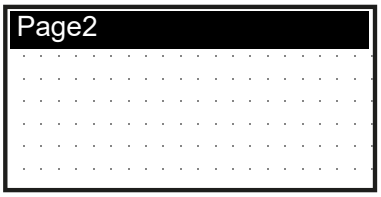
The text, the background color and the used font are the same for all the pages of the project, so you do not find them in this specific page properties. In order to customize the window title bar of the pages only, double-click on the **Properties** item of the project tree.

The window **Pages properties** opens. In **System options** tab, assign the font (in this case **EWP2_6x8**), the text color, and the background color (in this case respectively black and white).

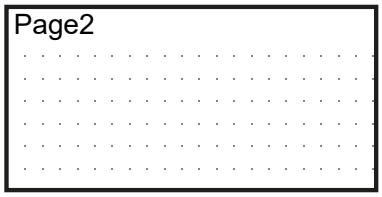


For example, the page looks like the following figures:

Before the procedure described above



After the procedure described above



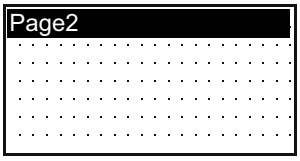
Assigning a Style to the Page

Display supports three styles for the pages, which you can select through the **Appearance** property:

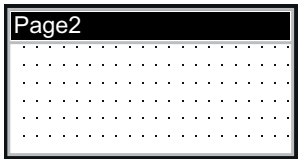
- **Flat** (the default style when you create a page),
- **Raised**,
- **Sunken**.

Overview of the three styles:

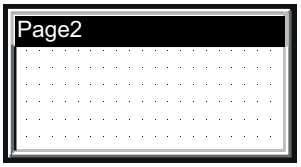
Flat



Raised



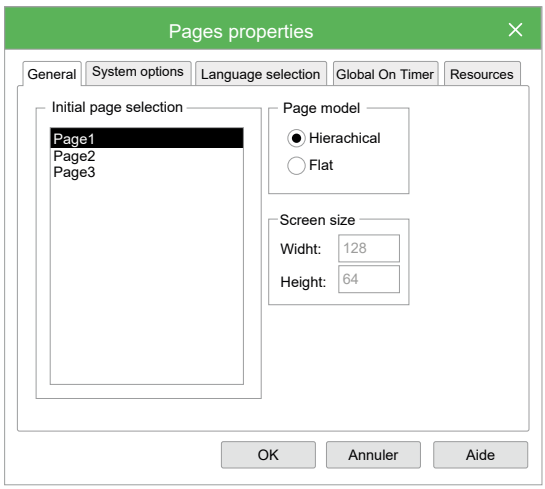
Sunken



Choosing the Start Page

You have to indicate the start page of the whole HMI project. The start page opens at the HMI application start. If the project consists in one single page, the system takes this one as start page.

To indicate the start page, double-click the **Properties** item of the project tree. In the **General** tab, select the window to define as the start page and click **OK**:



You can also indicate the start page by right-clicking a page in the project tree and clicking **Set as start page**.

The start page is marked in the project tree by a green triangle:

NOTE: The pop-up pages can not be set as starting page.

Basic Page Operations

Overview

Basic operations such as export/import, copy/paste, and page-based template management can be done with **Display**. Next paragraphs show these arguments in detail.

Export of Pages to Files

Each page can be saved to be used later in other projects.

To export a page, right-click on the page in the **HMI Project** window and click **Export page…** command.

Then, the application asks you to write the name of the file in which the page is saved. This file has a ***.pex** extension. The exported file contains page information and local procedures.

Import of Pages from Files

Each page can be saved to be used later in other projects.

To import a page, right-click **Pages** node and click **Import page…** command. Then, you can select the file of the page to import. The imported page takes the same name that it had when it was exported.

Export/Import Procedures and Variables

It is possible to export/import local or global variables and procedures using the menu commands **Export var/procedures…** or **Import var/procedures…**.

Copy/Paste of Pages in the Project

It is possible to copy and to paste a page inside the project.

To copy a page, right-click on the desired page and click **Copy page** command.

Then, you can paste the copied page by right-clicking **Pages** node and clicking **Paste page** command.

Rename Pages

To rename a page, right-click on the page to rename and click **Rename** command. This allows you to modify the name of the page.

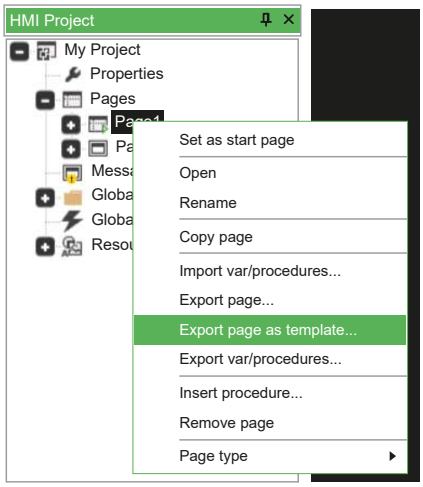
NOTE: This operation modify only the name of the page, project references to the renamed page are not automatically updated.

Templates of Page Management

Templates allow you to save only the structure of the page and not the whole page. Templates can be described as pages without references to external variables. Templates can be grouped in library files (***.petx**) and can be linked into the project.

Export Pages into a Template File

To export a page into a library of templates, right-click on the page in the **HMI Project** window and click **Export page as template…** command:



A library file with ***.petx** extension (new or already existing) is indicated. Template is appended to the existing templates and a name for the library is requested. If the template is already available in the library, a message asks you if you want to rewrite the existing template or not.

Page is exported as template into the specified library with its element but without any referenced variable.

Scripts and local variables are exported without modifications. References to variables contained in the scripts are not modified.

Child pages, popup, and asynchronous messages can be treated as templates.

Usage of the Template Library in a Project

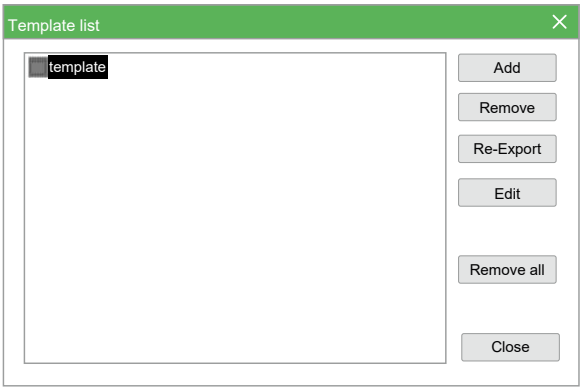
It is possible to include a template library in a project in order to use templates when desired.

Click **Template Management…** command from **Project** menu. The following window appears:

Available operations are listed here:

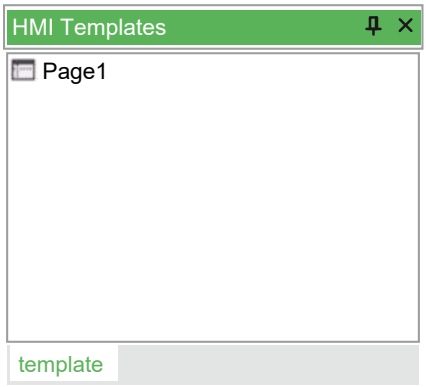
- **Add**: add a template library to the project. Including a library means that a reference to the library of the ***.petx** file is added to the current project, and that a local copy of the library is made.
- **Remove**: Remove template library from current project.
- **Edit**: To modify local copy of the template library removing no more used templates.
- **Re-Export**: Export local copy of the template library into a new ***.petx** library file.
- **Remove all**: Remove the template libraries from current project.

To add a template library to the project, click the **Add** button. Once chosen one of the available libraries, **Template list** window appears as shown here:



Template library has been included to the project. Then, click the **Close** button.

HMI Templates window shows a tab for each library imported in current project. Each tab shows the list of templates of the corresponding library:



NOTE: If **HMI Templates** window is not displayed, click **View > Tool windows > HMI Templates**.

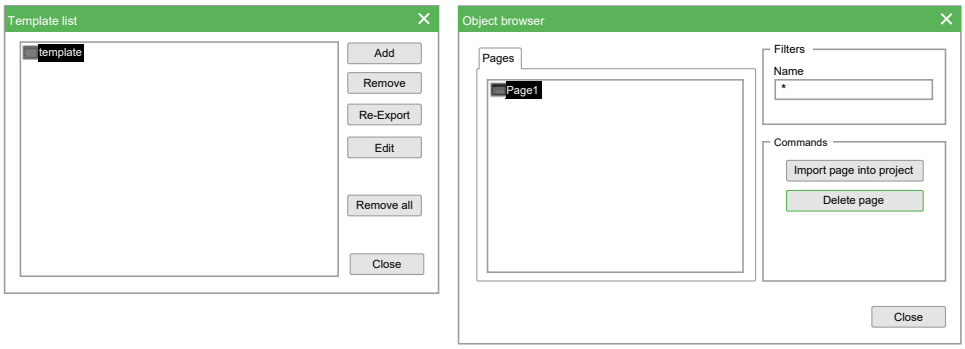
Using a Template

Once a template library has been added, you can use its elements by dragging the chosen one from the **HMI Templates** window and by dropping it on the project tree in the **HMI Project** window.

Once the item has been dropped, application asks you to write the name of the new page created (based on the template).

Project Template Update

You can delete templates from the (local) template library using **Edit** button in the **Template list** window:



Organization of Created Pages

HMI Actions Window

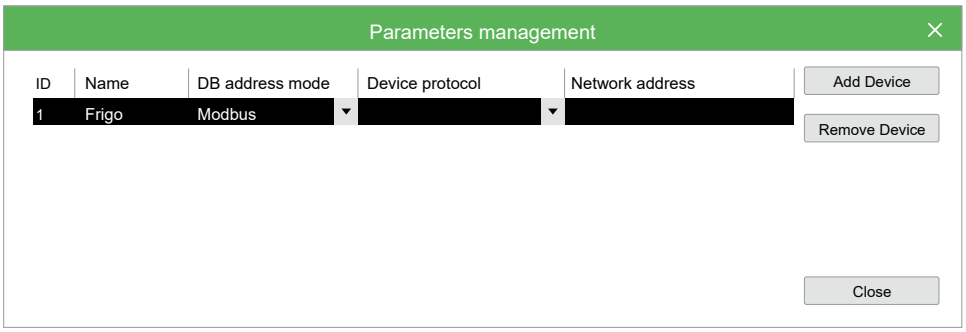
Overview

The **HMI Actions** window allows you to assign an action to a key of the target device:

- **Local actions**: assign to a key of the target device an action which depends on the selected page,
- **Global actions**: assign to a key of the target device an action which is always the same whatever the selected page.

The **HMI Actions** window is made up of three boxes:

- **Key**: key of the target device,
- **Action**: action to be performed when the key is pressed,
- **Link**: name of the page to open (to be used when the selected action is **Call** or **OpenPage**).



Project Properties

Overview

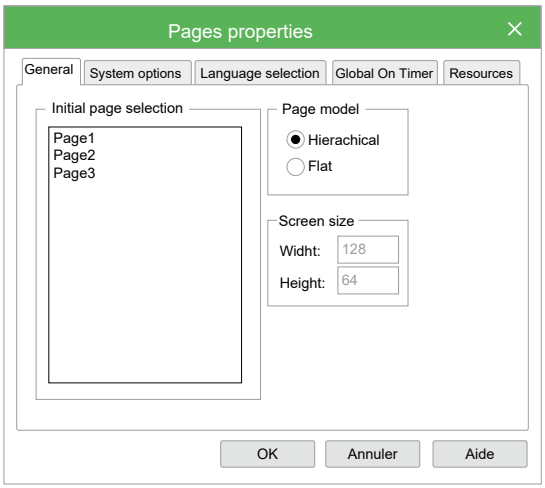
Display manages the creation of pages for a specific application.

It is composed of several pages where you can arbitrarily arrange the controls.

You have to specify the start page which is displayed at the start of the target device. Other pages have at least a parent page from which they are invoked and may have a child page to invoke. The invoking/invoked relations implicitly give to the whole project a tree structure.

Project Properties Window

To open the **Pages properties** window, double click **Properties** item in **HMI Project** window. This window is composed of five tabs:



- **General**,
- **System options**,
- **Language selection**,
- **Global On Timer**,
- **Resources**.

General

It allows to select the start page among the implemented pages.

The **Page model** area allows you to select the type of page model in case of a page calls another page. If the model is **Hierarchical**, then a child page can not recall a parent page. If the model is **Flat**, all the pages can call the others without limitations.

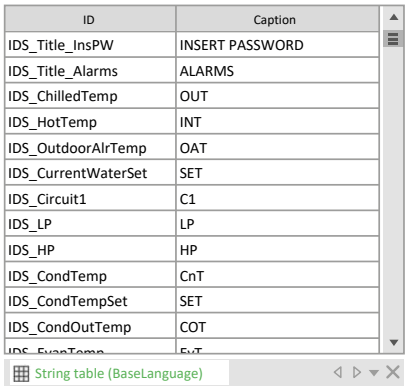
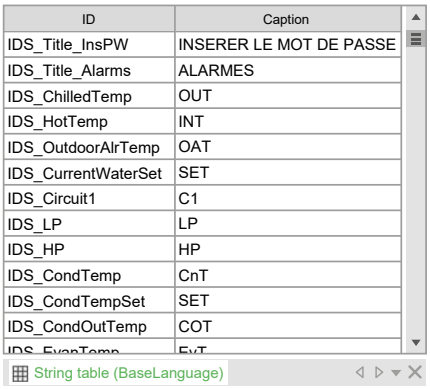
System Options

It allows you to customize the title bar features of the window: the font, the text color and the background color.

Language Selection

Strings and enumerated data types are structured as to ease the multilingual device. Moreover **Display** provides a function to export/import the above

mentioned elements to/from a text file in order to simplify the translation from a language to another.

English version	French version
	

It allows you to add, remove, export, import, and select the resources languages (for more information, refer to Strings Table, page 377 and Enumeratives, page 378). The label: **sysLangID Value** indicates the value which the **sysLangID** target variables must take to display the pages in the selected language.

To add a language (for example Chinese), follow the procedure below:

Step	Action
1	Export the language supported by the translator, select **BaseLanguage** (or any other valid translation).
2	Click **Export...** button.
3	It opens a window requiring the destination folder for the selected language file.
4	At the end of the exportation, the file is composed of the resources of the project which have to be translated: stings and enumeratives.
5	Translate the exported file and replace the text under the **LANGUAGE** tag with the one of the new language (for example, in this case change it into “Chinese”).
6	Then, in the **Language selection** area, click **Import ...** button and select the translated file.
7	The new language appears in the list.

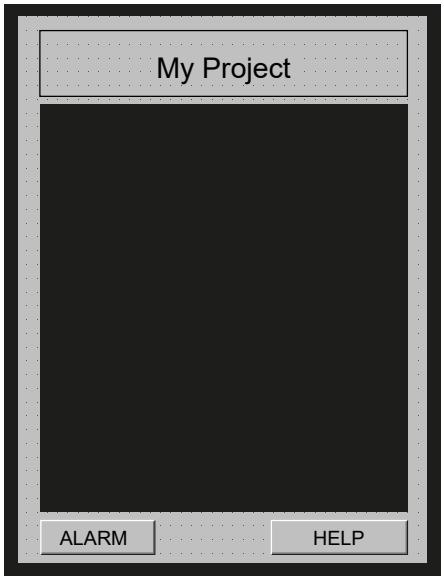
Global Periodic Procedure

GlobalOnTimer allows you to specify the name of a global procedure to be periodically and independently executed on the active page. Such a procedure may be effectively used to constantly test one or more PLC variables and to emit alarm messages, for example through asynchronous messages, page 327.

Frameset

Overview

Display allows you to define areas which are called frames and are placed on the sides of the screen and are always active.



You can set these dimensions of the frames and insert some controls which are active whatever the currently loaded page. Consequently, frames are useful to host the objects which have to appear in the whole project. In this way you do not need to duplicate them in each page.

There are two exceptions:

- The pop-up pages, page 330 when the **Modal** property is set to **Yes** in **HMI Properties** window.
- All the asynchronous messages, page 327.

When these pages are active, the controls of the frameset are automatically disabled.

Activate and Deactivate the Frameset

To activate the frameset, right-click **Pages** in the **HMI Project** window and select **Frameset**.

To deactivate the frameset, right-click **Pages** in the **HMI Project** window and select **Frameset**. In the window that appears, click **OK** (the current frameset will be lost).

For more information about frameset properties, refer to Frame Set, page 360.

Multiple Pages Management

Overview

These functions allow you to construct pages with data of different kind that must be represented on distinct pages for space reasons.

Sets, page 378 can be used with edit boxes or progress bars. Sets are ensemble of variables even of different type. Set definition can be done from the **Resource** in **HMI Project** window, they are implemented using a table with a series of variables that are dynamically associated to the control basing on the current index assigned to the page.

Association of Elements of a Set

Elements of a set can be associated to a control using the following syntax: character # first of all, then the name of the set followed by the index of the position of the element in the page, between round brackets.

Position index is used to indicate the order in which elements are shown in case of more than one element in the same page.

A page contains one or more controls based on one (or more) set. At runtime, the page is replied in order to show all the elements contained in the set. In the last page, if any control can not be filled with element value, that control is hidden.

We created before a set of five elements named `BIOSParameters`, now we can associate `#BIOSParemeters(0)` to the first edit box and `#BIOSParemeters(1)` to the second. So there are three pages: first page with the first two elements of the set, second page with elements 2 and 3 and third page showing the last element of the set. In the last page, the second edit box is not visible.

Navigation of the Elements of a Set

Navigation of pages that represent a set of elements is automatically done using:

- `NextEdit` event of the last selectable control of the page,
- `PrevEdit` event of the first selectable control of the page.

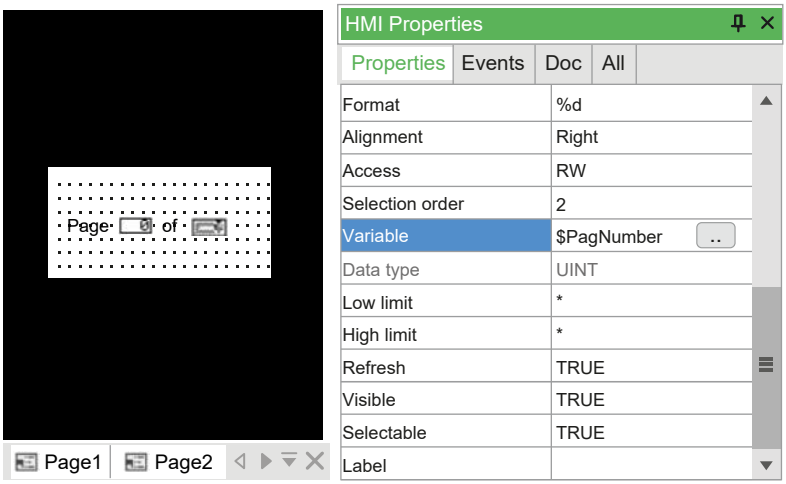
It is also possible to send special events to force the change of the page in this way: `Video_SendEvent` (`kEV_WM_CHANGESETPAGE`, `numpage`). Where `numpage` is the number of the page of the set.

Pages Numbering

Display defines two variables related to pages numbering:

- `$PagIndex`: current index of the page containing controls based on a set,
- `$PagNumber`: number of pages that complete the visualization of the whole elements of the set.

These variables can be used in the page to show the numeration of the pages. They can be used as variables associated to edit box controls in this way:



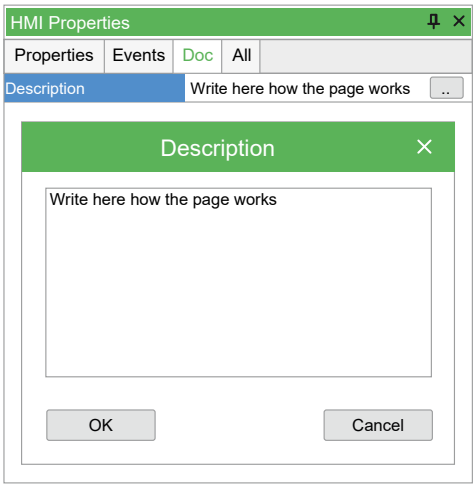
Automatic Documentation

Overview

During project development, it is usually necessary to write comments for each page in order to explain how the page works.

Display integrates into its development environment the automatic documentation feature. It consists in the generation of a graphical report with all the previously inserted comments followed by the pages they refer to.

Comments related to controls and pages should be inserted in the **Doc** tab of the **HMI Properties** window.



Documentation is generated when the **Generate Doc.** button is pressed.

At the end of the process, a dialogue box is shown. Click **Open documentation** link to view the generated report using the browser.

It is also possible to manually open the *.html file generated. This file is created in the project folder and is named “project name.html”.

> **NOTE:** Documentation generation process requires the file “Documentation. xsl” to be in the project folder. You can personalize this file to redefine report style.

Insertion of Controls

Controls

Overview

A control is a display element (textual or graphical) contained in a HMI page. It can be used to present information. Control content can be fix or can modifiable with variables or parameters through HMI application. **Display** supports the following controls:

- Static, page 343
- Edit Box, page 348
- Text Box, page 356
- Combo Box, page 345
- Graphic, page 343
- Image, page 345
- Animation, page 347
- Button, page 353
- Check Box, page 344
- Progress Bar, page 357

To insert a control on a page, you can do the following:

- In the **HMI Page** toolbar, click the icon of the control to be inserted. Then, click at the place on the page where this control should be displayed.
- In the **Page** menu, click the icon of the control to inserted. Then, click at the place on the page where this control should be displayed.

- From the **HMI Vars and Parameters** window, drag and drop a variable or a parameter into a page. The **Insert object** window appears and lets you choose which control to use to display this variable or parameter.

Static

Static controls display a fixed string whose contents cannot be edited when executing. You must specify the text of the string directly or by the association of the ID of a string defined as resource to support multi-language management. For project resources and multi-language support, refer to Resources, page 376.

To insert a static control, click **Insert static** icon in **HMI Page** toolbar. Then, click in the editor window the point where you want to insert the control.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Static control** and click **OK** button.

For more information about properties and events of the static control, refer to Static Properties, page 363.

Graphic

Overview

Graphic controls are objects which are drawn when you open the page. They do not change until the page is active.

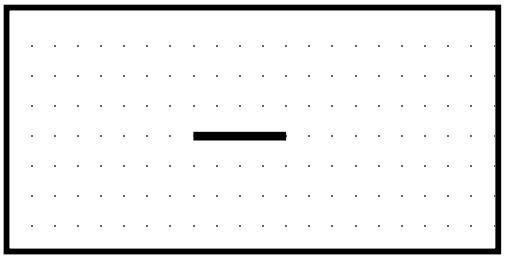
They display a static line or rectangle. Their properties cannot be edited when executing.

Inserting a Line

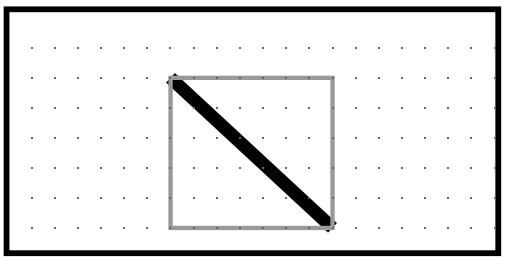
To insert a line, click **Insert line** icon in **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

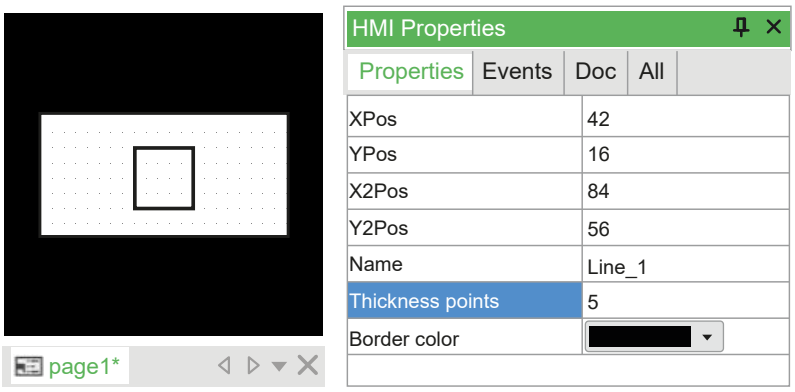
The inserted line has a default size and horizontal alignment:



You can resize it by dragging one of the two ends of the line:



You can edit the color and the thickness of the line in the **HMI Properties** window.

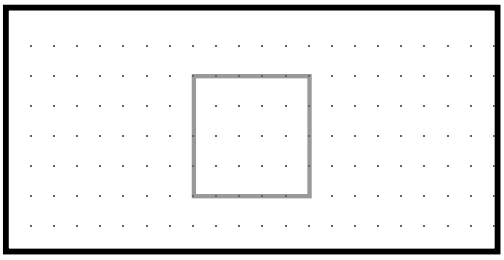


Inserting a Rectangle

To insert a line, click **Insert rectangle** icon in **HMI Page** toolbar.

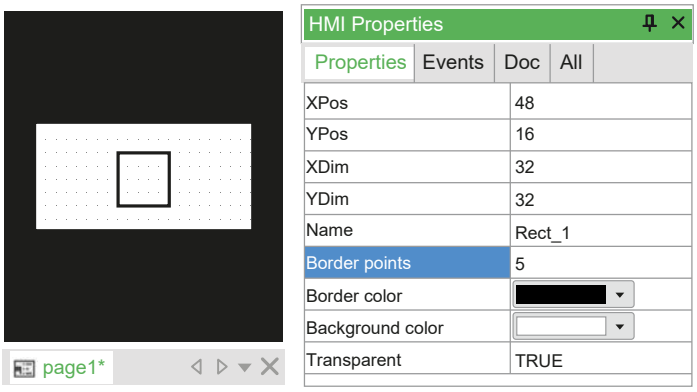
Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

The inserted rectangle has a default size:



You can customize the inserted rectangle by modifying its parameters in the **HMI Properties** window.

For example, you can modify the thickness of the borders, the color of the borders or the background or make the background transparent.



Check Box

Check box control displays a check box that allows you to select a true or false condition identified by a variable.

To insert a check box control, click **Insert new combo box** icon in **HMI Page** toobar. Then, either click in the editor window the point where you want to insert the control or drag a variable from the project tree or from the library window.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Checkbox control** and click **OK** button.

Combo Box

Combo box control shows a list of strings connected to a variable with an enumerator element.

To insert a combo box control, click **Insert new combo box** icon in **HMI Page** toobar. Then, either click in the editor window the point where you want to insert the control or drag a variable from the project tree or from the library window.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Combobox control** and click **OK** button.

Image

Overview

Image control displays a bitmap image.

Static images are different from:

- Animations, page 347: which are images which may change dynamically, even though they have fixed position and dimensions.
- Floating images: which are images which move in the page.

Importing a Bitmap in the Project

The following image formats are supported by FREE Studio Plus:

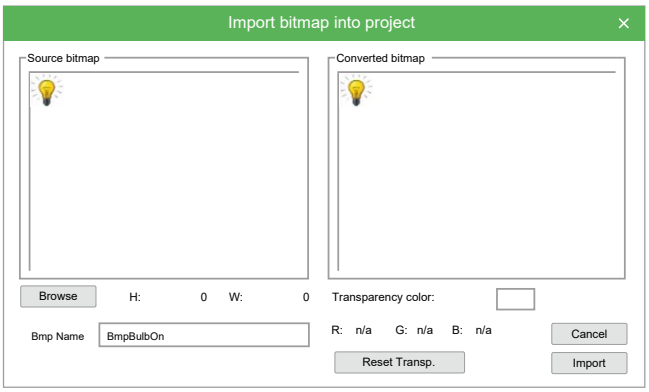
- BMP (*.BMP;*.DIB;*.RLE)
- JPEG (*.JPG;*.JPEG;*.JPE;*.JFIF)
- GIF (*.GIF)
- EMF (*.EMF)
- WMF (*.WMF)
- TIFF (*.TIF;*.TIFF)
- PNG (*.PNG)
- ICO (*.ICO)

  NOTE:

  FREE Studio Plus displays the images on the target device without resizing them. So make sure that the size of the imported images is not too large for the target device on which they are displayed.

In **Resources [BaseLanguage]**, right-click **Bitmaps** item and click the **Import bitmap…** command.

In the **Import bitmap into project** window, click **Browse** button. You can navigate in the computer resources and select the desired image. In this case, the image file is **BulbOn.jpg**, which represents a lighted bulb:



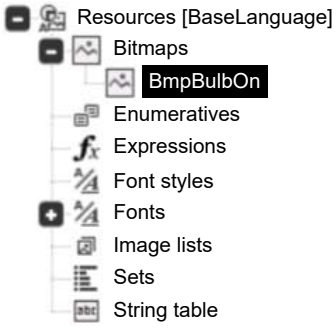
In the **Bmp Name** field, you can assign the bitmap name which appears in the **Resources [BaseLanguage]** tree structure. The default name is the file name without extension and preceded by the **Bmp** prefix.

The **Transparency color** field allows you to specify a transparency color. This color is not drawn but lets the elements appear through the bitmap background.

You can customize the transparency color by taking the desired one with the mouse from the **Converted bitmap** box.

RGB indicates the transparency color components. If the values are **n/a**, it means that no transparency color has been selected. The **Reset Transp.** button allows you to cancel the last selected transparency color.

At last, you can confirm the operation by clicking the **Import** button. The imported bitmap appears as a new item in the **Resources [BaseLanguage]** tree structure:



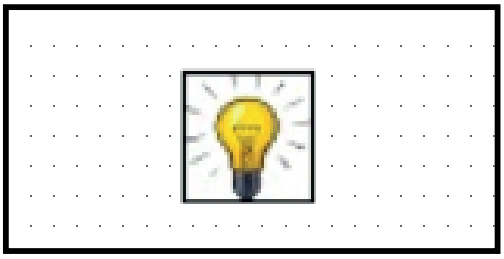
Associating an Imported Bitmap with an Image Control

The control which is aimed to display the static images is called **Image**.

To insert an image, click **Insert image** icon in **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

A new blank frame appears in the page. Drag the desired image from the **Bitmaps** list and drop it to the blank frame. The image control does not modify its size to be compatible with the assigned bitmap measures.



Animation

Overview

An animation control allows you to associate each variation in the value of a variable with the display of a different image.

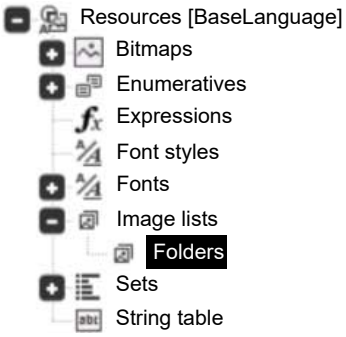
It displays a bitmap image which you select from a list of images depending on the value of an associated selection variable.

Inserting an Animation

In order to create an animation, you must first create a list of images. This list of images includes the images that the variable can display. Each image is associated with a value or a range of values that the variable can take. To know how to import an image, refer to Importing a Bitmap in the Project.

To create a list of images, right-click **Image lists** in**Resources [BaseLanguage]** item and click **Add new** command.

Then, right-click on the newly created list and click **Rename** command. Write the name of your choice and press the **Enter** key.



To add images to the list, click the **New Record** icon in the **HMI Project** toolbar. A line appears in the editor window and must be completed like this:

- **Init value**: indicate the start of range value,
- **End Value**: indicate the end of range value,
- **Bitmap**: select the image to display for the specified value range from the list.

Repeat the operation until the list is complete:

Init value	End Value	Bitmap
1	1	BmpSettings32x32
2	2	BmpMonitor32x32
3	3	BmpOnOff32x32
4	4	Bmplo32x32
5	5	BmpPassword32x32

Image lists - [Folders]

Once the list of images is established, you have to create an animation. To do so, select the desired page and click **Insert new animation** icon in **HMI Page** toolbar.

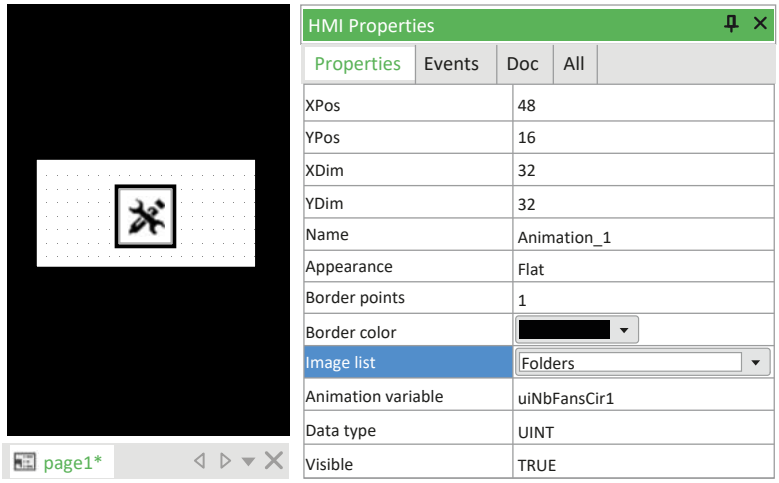
Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Animation control** and click **OK** button.

A new blank frame appears in the page. Select it and fill in the following parameters in **HMI Properties** window as follows:

- **Image list**: indicate the list of images to use with the variable,
- **Animation variable**: indicate the animation variable,
- **Data type**: indicate the type of the variable (filled in automatically after selecting the variable).

The creation of the animation is finished. If you want, you can also edit the other parameters present in the **HMI Properties** window.



Edit Box

Overview

An edit box control is a text frame which lets you display and edit an associated variable or parameter.

It displays the contents of an associated variable.

Inserting an Edit Box in the Page

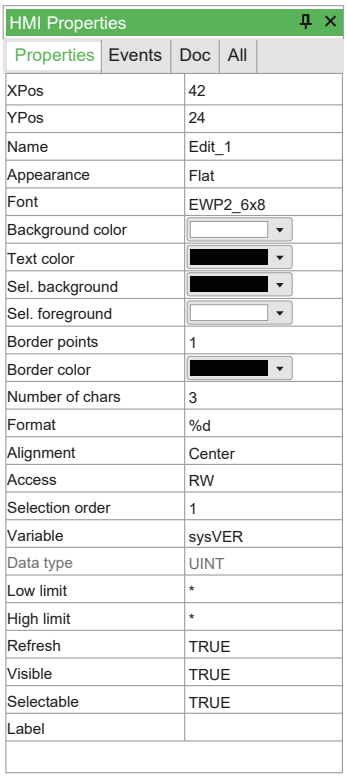
To insert an edit box, click **Insert new edit** icon in **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Edit control** and click **OK** button.

A new text frame appears. It consists by default in some characters and its font is specified in the **Font** property of the page.

In **HMI Properties**, you can modify several parameters including these:



- **Appearance**: appearance of the edit box (**Flat** by default).
- **Font**: font used by the text in the edit box (**EWP2_6x8** by default).
- **Sel. background** and **Sel. foreground**: respectively text and background colors when the edit box is selected.
- **Number of chars**: maximum number of characters which can be displayed.
- **Access**: to set the read-only mode, replace **RW** (read/write) by **RO** (read-only).
- **Refresh**: to update constantly the contents of the edit box, select the **TRUE** option.

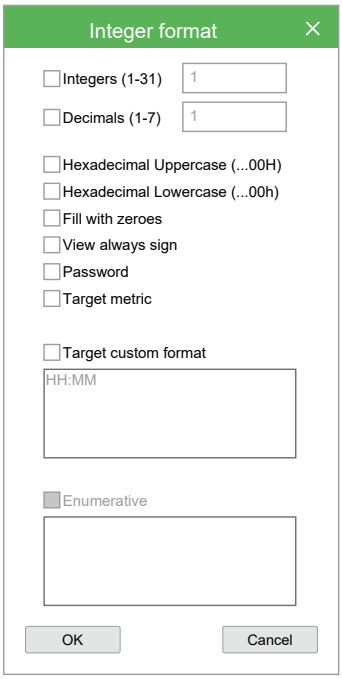
  Otherwise, the contents are refreshed just when drawing the page for the first time.
- **Label**: if the target has a touchscreen display, shows keyboard and has this feature enabled.

  It is possible to add this text/'string resource' as header of keyboard.
- **Format**: it represents the display format of the associated value of the variable. The format value can be inserted only if a variable is available. It opens a dialog window with these settings according to the type of variable (integer, real, string).

For more control properties details, refer to Edit Box, page 364.

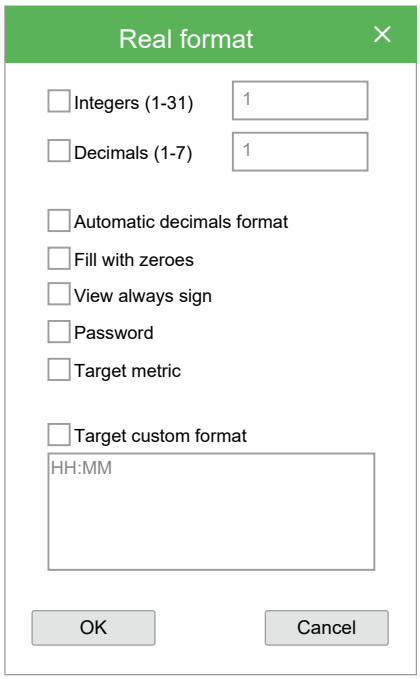
Format Property Details

Integer format window:



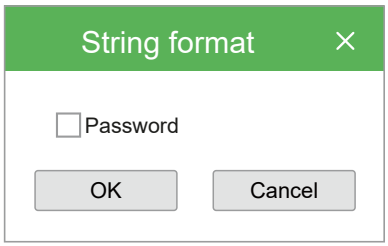
- **Integers (1-31)**: number of digits before decimal point.
- **Decimals (1-7)**: number of digits after decimal point.
- **Hexadecimal Uppercase (…00H)**: the number is shown as **…00H** representation with uppercase H letter.
- **Hexadecimal Lowercase (…00h)**: the number is shown as **…00h** representation with lowercase h letter.
- **Fill with zeroes**: fill the entire edit box controls with 0 where there are not numbers.
- **View always sign**: show the **+** or **-** symbol in edit box.
- **Password**: show only ***** symbols.
- **Target metric**: reserved.
- **Target custom format**: the target can define custom format to show the data in a particular way. In that case, there is a variable on the target with the value of the corresponding user mode.
- **Enumerative**: this representation allows you to select a string value corresponding to numeric value defined in **Resources**, under **Enumeratives**.

Real format window:



- **Integers (1-31)**: number of digits before decimal point.
- **Decimals (1-7)**: number of digits after decimal point.
- **Automatic decimals format**: hides unnecessary zeros in decimals. If checked, **Decimals (1-7)** is disabled.
- **Fill with zeros**: fill the entire edit box controls with zero where there are not numbers.
- **View always sign**: show the **+** or **-** symbol in edit box.
- **Password**: show only ***** symbols
- **Target metric**: reserved.
- **Target custom format**: the target can define custom format to show the data in a particular way. In that case, there is a variable on the target with the value of the corresponding user mode.

String format window:



- **Password**: show only ***** symbols.

The **Target custom format** is a special feature which enables a particular custom format implemented on the target.

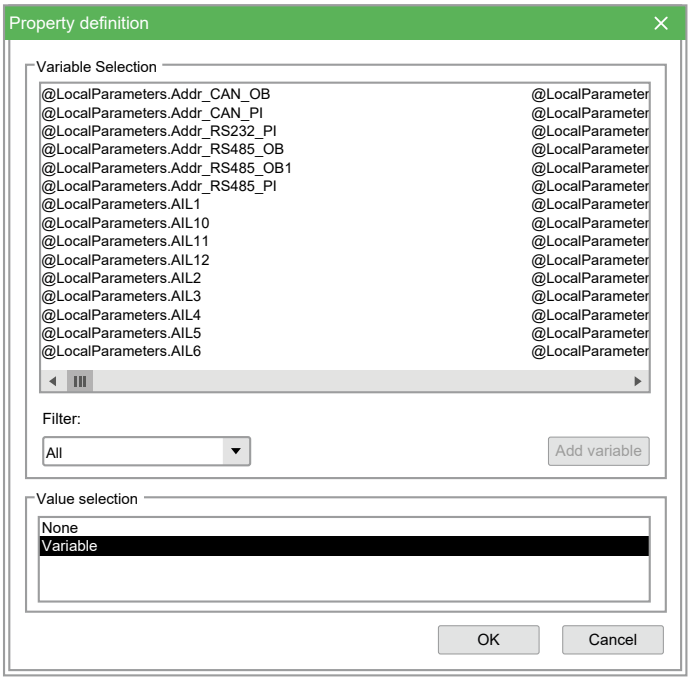
The format is specified according with language **printf** syntax, refer to Format Specification - Printf, page 366.

Edit Box and Display Variable Association

In order to display values, edit boxes have to be associated with variables.

To link an edit box to a variable, select the edit box by clicking it once and select the **Variable** in **HMI Properties** window.

Then, enter the name of the desired variable. If you do not know its name, click the **..** button to open the **Property definition** window.



The **Property definition** window allows you to find the desired variable.

You can use the **Filter** field to search for a variable according to its type:

- **Page locals**
- **Global procedures**
- **Target** (variables which allow the interaction between user interface and system)
- **PLC application**
- **Parameters** (for more information about how to name a parameter, refer to Edit Box to a Parameter, page 352)
- **PLC libraries**

Once found, select the desired variable and click **OK**.

Now the edit box control shows the value of the selected variable constantly refreshed.

Linking an Edit Box to a Parameter

The name of parameters is composed like this: **@device.variable name**. This is different from the name of the variables which show just their name.

The parameter may be inserted in the apposite controls property in the following forms:

- Explicit form: **@d.oi.os:type** (example: **@1.2010.0:UINT**),
  - **d**: numerical ID of the device,

    This field of the device is a numerical or symbolic identifier (to be defined at project creation). It refers to a specific device which may be local (the device which executes the pages itself) or on the fieldbus.
  - **oi**: object index,
  - **os**: object subindex,
  - **type**: PLC type.

- Implicit form: **@dev.name** (example: **Frigo.AIL1**).
  - **dev**: symbolic identifier of the device,

    This field is a symbolic identifier of a device whose numerical ID can be retrieved by **Display**.
  - **name**: symbolic name of the parameter.

Linking an Edit Box to a Variable by Dragging and Dropping

You can add variables and parameters to the **HMI Vars and Parameters** window by dragging and dropping them in the page. **Display** requests to define the type of control to insert to associate it with the variable.

Button

Overview

Buttons are controls which allow you to interact with the system, particularly in case of touchscreen systems without keyboard.

There are four kinds of button control:

- LED button: to view an associated state of a boolean variable.
- Status button: to modify the state of a boolean variable.
- Opening button: to open an other page.
- Activation button: to start the execution of a customized procedure.

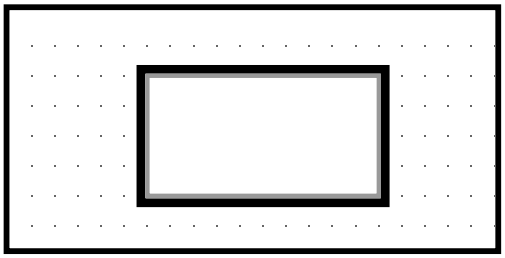
Inserting a Button

To insert a new button, click **Insert new button** icon in **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Button control** and click **OK** button.

A new button control appears. It has a default size. You can change the dimensions of the button by dragging one side with the mouse.

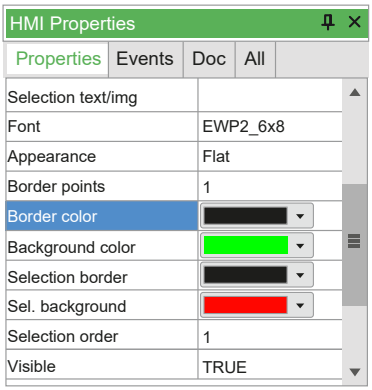


Creating a LED Button

Once the button is inserted, you must define the colors associated with the different states of the boolean variable:

- **Border color**: defines the border color when the button is inactive.
- **Background color**: defines the background color when the button is inactive.
- **Selection border**: defines the border color when the button is selected.
- **Sel. background**: defines the background color when the button is selected.

You can also customize the button appearance through the **Appearance** property.



Then, you must associate the LED button with a boolean variable through the **Variable** field. This property defines the state of the button and can be associated with:

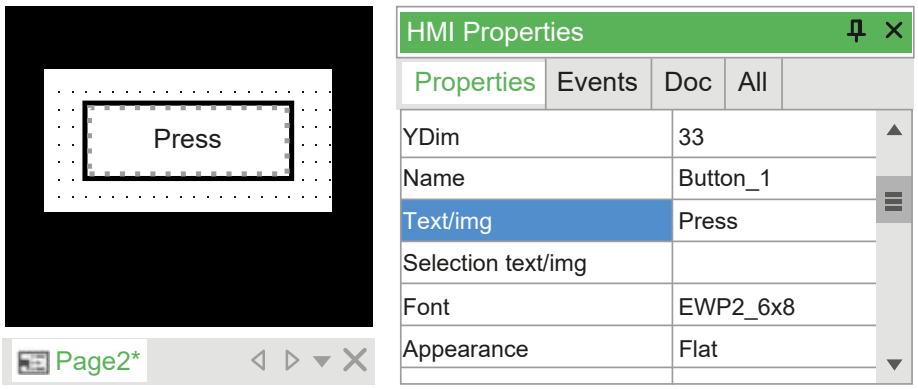
- A constant value:
  - **FALSE**: the control is always inactive.
  - **TRUE**: the control is always selected.
- A boolean variable whose value defines dynamically the selection state.

  To declare a boolean variable and associate it with the button control, write its name in **Variable** field or select it through the ... button.

Creating a Status Button

Once the LED button is created, you must insert a new button (which corresponds to the status button) and set it next to the LED button.

In **HMI Properties** window, write in **Text** field a name to display inside the new inserted button.



Then, you must associate the status button with a boolean variable through the **Press variable** field. In this case, the value of the boolean variable corresponds to the pressure state of the status button.

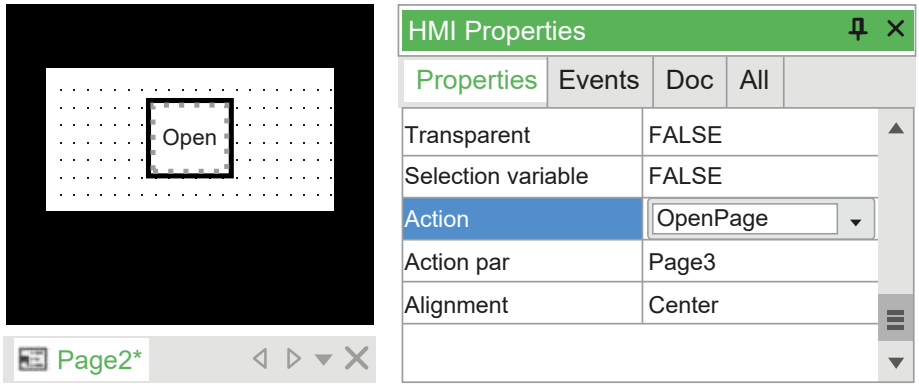
This allows you for example to display the LED button in green at runtime and to display it in red as soon as you press the **Press** button (and therefore as soon as you modify the value of the boolean variable).

Creating an Opening Button

Once the button is inserted, it can be used to open an other page when you press it.

In **HMI Properties** window, select **Action** field and click **OpenPage**. Then, in **Action par** field, write the name of the page to open when the opening button is pressed.

In the following example, when you press the **Open** button of the **Page2**, it opens the **Page3** page.



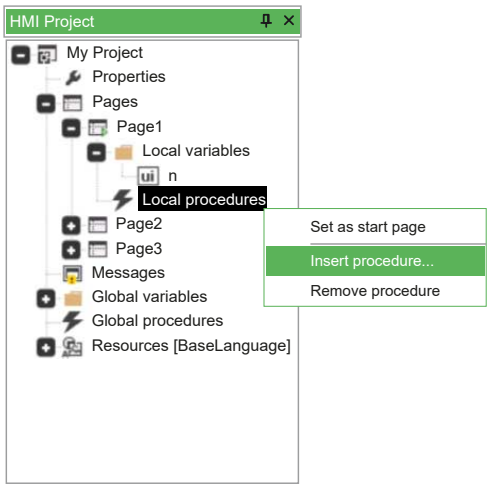
Creating an Activation Button

Display enables you to implement some procedures (refer to Procedures that Can Be Associated to Events, page 374) through which it is possible to customize the HMI behavior.

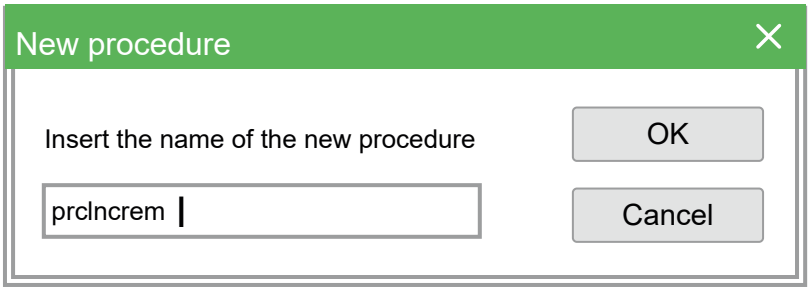
Take the example of the creation of a procedure to increment the local variable “n” of the **Page1** page.

As this procedure applies on a local variable, it is local in the **Page1** page too.

To create a procedure on a page, expand its structure tree. Then, right-click on the **Local procedures** item and click **Insert procedure…** command.

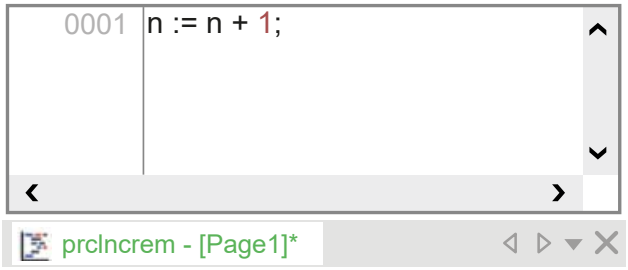


In the **New procedure** window, write the name of the procedure and click **OK** button.

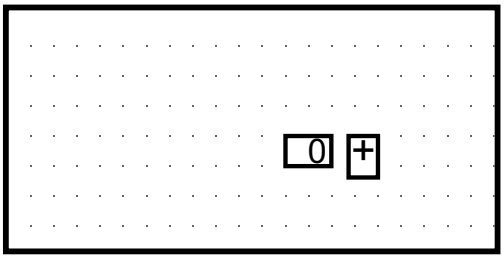


Double-click the local variable which has been declared and appears under **Local procedures** item. The ST language editor opens and lets you either implement or edit the selected code of the procedure.

Write a procedure that applies a unit increment to the “n” variable. Then, close the document.



Insert a new button control (which corresponds to the activation button) beside the edit box associated with the “n” variable and write the character “+” in the **Text** property.



Take the example of the execution of the **prclncrem** procedure by clicking the **+** button. In **HMI Properties** window, select **Call** action in the **Action** field. Then, write the name of the procedure in the **Action par** field.

Every time you press the **+** button when executing the HMI, **n** increases by one and the edit box shows the up-to-date value.

Text Box

Overview

Text boxes are not part of static controls because they have some properties which let them modify themselves in a page through time. Visibility, selection, and refresh may be assigned to variables, which may modify their value at any time.

Text box displays the contents of an associated string variable. It supports the formatting on several lines of the text which is contained in the string.

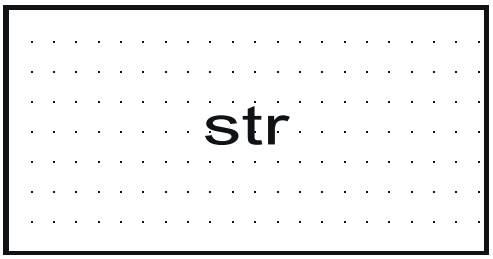
Inserting a Text String

To insert a new text string, click **Insert static** icon in the **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Textbox control** and click **OK** button.

A new text string control appears. It has a default text which can me modified through the **HMI Properties** window by editing the **Text** field.



This is the basic use of a text string. You can also assign text strings by taking them from the resources (refer to Stings Table, page 376).

Progress Bar

Overview

A progress bar control allows you to display the variations in the value of a variable in the form of a progress bar.

It represents the progress of an operation by showing a stained bar in a horizontal or vertical rectangle. The length of the bar shows the percentage of the completed operation.

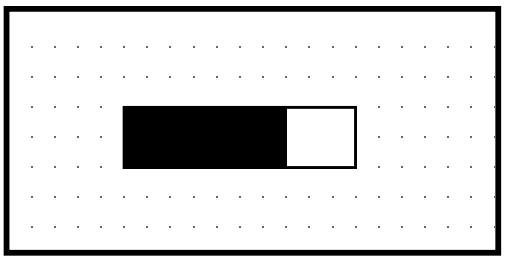
Inserting a Progress Bar

To insert a progress bar, click **Insert new progress** icon in **HMI Page** toolbar.

Then, move the mouse to the active area of the page. A **+** cross shows the insertion point of the object. Click-left to insert the object in the grid.

You can also drag a variable from the **HMI Project** window and drop it into the editor window. In the **Insert object** window, select **Progress control** and click **OK** button.

The inserted progress bar has a default size and horizontal orientation:



The orientation as well as the parameters of the progress bar can be modified through the **HMI Page** window.

You must associate a variable with the progress bar. To do so, select the desired variable in the **Progress variable** field of the **HMI Properties** window. The **Data type** field is automatically updated.

Then, define the minimum value that the variable can reach in the **Low limit** field and the maximum value that the variable can reach in the **High limit** field. When

the value of the variable is at the minimum, the progress bar is empty. When the value of the variable has reached its maximum, the progress bar is full.

Editing Control Properties

Visibility and Updating of Controls

Overview

Each control has its own properties which you can customize through the **HMI Properties** window. The content of this window depends on the selected control.

Two properties that are common to most controls: **Refresh** and **Visible**.

Visibility Property

Most controls have the **Visibility** property, which determines whether the object is visible or not.

This property can be associated either with:

- A constant value:
  - **TRUE**: the field is always displayed,
  - **FALSE**: the field is always hidden.
- A boolean variable: whose value dynamically establishes the visibility state.

It is possible to condition the visibility of a variable. For example, in case you want to display a variable when its value is even and hide it when its value is odd.

For the purposes of the demonstration, take the detailed case in Inserting a Button to Launch a Procedure, page 353. Suppose that you want to display the text string when the value of the variable n is even and hide it when this value is odd.

For this purpose, it is necessary to declare a new boolean local variable which indicates whether currently **n** is even.

	Name	Type	Array	Init value	Description
1	n	UINT	No	100	Counter variable
2	even	BOOL	No	TRUE	Local variable n is even

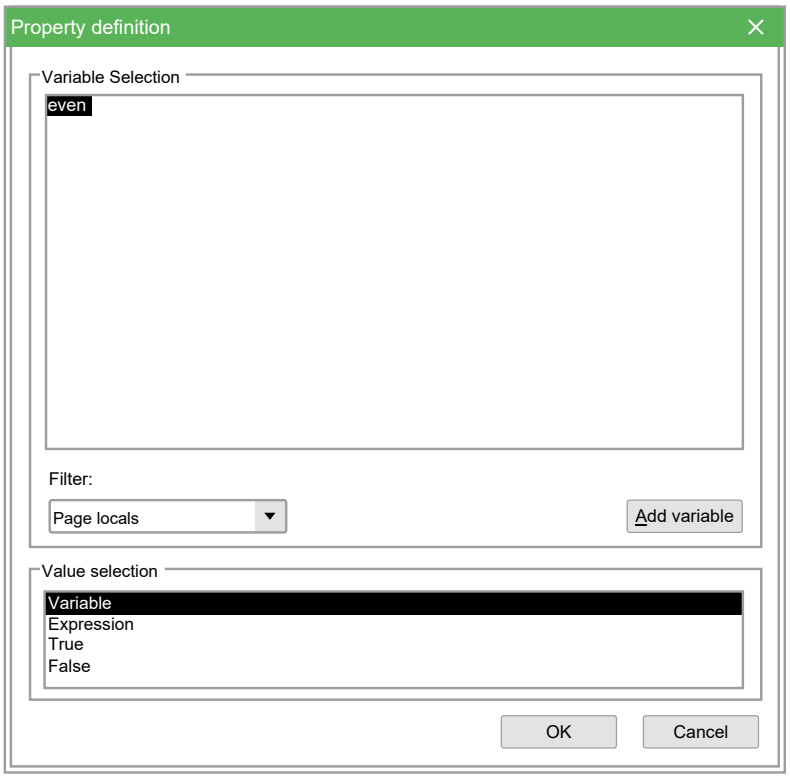
It is necessary to edit the **prcIncrem** procedure so that, when it refreshes the **n** value, it evaluates again whether it is even or odd. In order to access the **prcIncrem** source code, select the corresponding item in the project tree by right-clicking it. Then, click **Open** command.

The ST language editor opens and the code of the procedure may be extended as follows:

```
0001 n := n + 1
0002
0003 even := (n MOD 2) = 0;
0004
0005
```

In order to associate the visibility of the string state with the **even** boolean variable, select the text string and click ... button in the **Visibility** property of **HMI Properties** window.

In the **Value selection**area, click **Variable** and select **Page locals** in the **Filter** list. Then, click the local boolean variable **even**.



Click **OK** button. The text string is only visible when the variable **n** is even.

Refresh Property

When available, the **Refresh** property determines if the associated object has to be drawn once (when opening the page or coming back from a child page) or if it needs to be constantly refreshed.

For example, this property distinguishes the edit box and the text box.

With regard to the edit box, the refresh property has to be set when compiling and it can not be edited at runtime. If you assign **Refresh** = **TRUE**, the associated value of the variable value is constantly read and refreshed. Otherwise (**Refresh** = **FALSE**), the value is read and refreshed only when you open the page or when you come back from a child page.

There is another option about text boxes: you can associate a boolean variable that is used as trigger for refresh. When the trigger variable becomes **TRUE**, the contents of the controls are refreshed. Then, it is automatically reset by **Display** to **FALSE**.

Using Expression to Set Visibility and Selectable Properties

Expressions can be used to dynamically change visibility and selectable properties of a control.

From the **Resources** item in **HMI Project** window, double-click **Expressions** item. The definition table of expressions appears.

To add a new expression definition row, click **New record** icon in **HMI Project** toolbar. Then, you can fill the fields:

Expressions ID	Expression	Result type
The univoque ID associated to the expressions, you can not use special characters. This is the ID that would be indicated in the page controls visibility/ selectable field.	• You can use PLC variables, target variables, global variables. You can use simple variables, elements of array or structure field. • Define your expression in ST code, do not enter "=" at the beginning or ";" at the end of the expression. Expression code should be something like this: • TRUE • `sysTimer` > 10000 • `sysAlarm` = TRUE • (`COND[0]` OR `COND[1]`) AND `S.RUN`	Result type of all expressions should be always `BOOL`.

You can use expressions to set the visibility property of static, edit box, image, animation, button, check box, and progress bar controls.

You can use expressions to set the **Selectable** property of a static, edit box, and check box controls.

In **HMI Properties** window, click **Visible** or **Selectable** field of the control. Then, open the **Property definition** window by clicking ... button. In **Value selection** area, select **Expression** box and the list of the defined expressions appears.

From the **Expression Selection** area, double-click the defined expression you want to associate to the control. Property field assume the value: `EXPR:yourExpressionID`.

You can also use expression for visibility and selectable properties of a set. From a set definition grid, click the set item you want to modify. Then, in the **Visible** field, click the ... button and apply the method previously described.

The defined expressions are evaluated:

- After the execution of the **GlobalOnTimer** page event (if specified),
- Before the execution of the **OnTimer** page event (if specified),
- Before the refresh of the controls of the page.

Page and Object Properties

Frame Set

Properties

Properties	Available values	Description
TopDim	≥ 0	Top-height of the frame (#pixel).
BottomDim	≥ 0	Bottom-height of the frame (#pixel).
LeftDim	≥ 0	Left-width of the frame (#pixel).
RightDim	≥ 0	Right-width of the frame (#pixel).
CharDimX	≥ 0	Horizontal space among grid points (#pixel).
CharDimY	≥ 0	Vertical space among grid points (#pixel).
Font	Name found in **Resources**	Default font used when inserting new objects in page.
Background Color	...	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.

Properties	Available values	Description
Text Color	...	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
Title bar	**Yes, No**	Title bar, settings can be found in **System options** dialog: • **Yes**: page has title, • **No**: page has not title.
Page Border	**Yes, No**	• **Yes**: page with outer border, • **No**: page without outer border.
Caption	Text otherwise **Resource ID**	Text on title bar or **Resource ID**. This property is not sensible if **Title Bar** field is set to **No**.
System menu	**Yes, No**	If **Yes** denotes that there is a button with 'X' image on it and the behaviour is similar to **Windows Dialog**: • **Yes**: page has close button, • **No**: page has not close button.
Appearance	**Flat, Raised, Sunken**	• Flat • Raised • Sunken

Child Page

Properties

Properties	Available values	Description
CharDimX	≥ 0	Horizontal space among grid points (#pixel).
CharDimY	≥ 0	Vertical space among grid points (#pixel).
Font	Name found in**Resources**	Default font used when inserting new objects in page.
Background Color	…	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.
Text Color	…	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
Title bar	**Yes, No**	Title bar, settings can be found in **System options** dialog: • **Yes**: page has title, • **No**: page has not title.
Page Border	**Yes, No**	• **Yes**: page with outer border, • **No**: page without outer border.
Caption	Text otherwise **Resource ID**	Text on title bar or **Resource ID**. This property is not sensible if **Title Bar** field is set to **No**.
System menu	**Yes, No**	If **Yes** denotes that there is a button with 'X' image on it and the behavior is similar to **Windows Dialog**: • **Yes**: page has close button, • **No**: page has not close button.
Appearance	**Flat, Raised, Sunken**	• Flat • Raised • Sunken

Events

Events	Description
OnLoad	On loading this page, i.e. when calling from parent page.
OnUnload	On closing this page, when the page returns and the parent page is restored.
OnDeactivate	On calling a child page and the current page is no more active. This event does not exist in main page.

Events	Description
OnActivate	When the previous opened child page is closed. This event does not appear in leaf page, that is, in the pages which do not call child pages.
OnDraw	When the page starts drawing all the objects. The page has just drawn border, background, and title.
OnTimer	Asynchronous event. You can link a procedure which is executed cyclically.

Pop-Up Page

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge of full page.
YPos	≥ 0	Top-left 'y coordinate' edge of full page.
XDim	> 0	Width of the page (#pixel).
YDim	> 0	Height of the page (#pixel).
CharDimX	> 0	Horizontal space among grid points (#pixel).
CharDimY	> 0	Vertical space among grid points (#pixel).
Modal	**Yes, No**	• **Yes**: the parent page objects are disabled, • **No**: all the parent page objects are enabled if they are completely visible.
Font	Name found in **Resources**	Default font used when inserting new objects in page.
Background Color	...	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.
Text Color	...	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
Title bar	**Yes, No**	Title bar, the settings can be found in **System options** dialog: • **Yes**: page has title, • **No**: page has not title.
Page Border	**Yes, No**	• **Yes**: page with outer border, • **No**: page without outer border.
Caption	Text otherwise **Resource ID**	Text on title bar or **Resource ID**. This property is not sensible if the **Title Bar** field is set to **No**.
System menu	**Yes, No**	If **Yes** denotes that there is a button with **X** image on it and the behaviour is similar to **Windows Dialog**: • **Yes**: page has close button; • **No**: page has not close button.
Appearance	**Yes, No**	• Flat • Raised • Sunken

Events

Events	Description
OnLoad	On loading this page, i.e. when calling from parent page.
OnUnload	On closing this page, when the page returns and the parent page is restored.
OnDeactivate	On calling a child page and the current page is no more active. This event does not exist in main page.
OnActivate	When the previous opened child page is closed. This event does not appear in leaf page, that is, in the pages which do not call child pages.
OnDraw	When the page starts drawing all the objects. The page has just drawn border, background, and title.
OnTimer	Asynchronous event. You can link a procedure which is executed cyclically.

Static

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
Name	Not empty	Name of object.
Text	Text otherwise **Resource ID**	Text or **Resource ID** shown in the object.
Font	Name found in **Resources**	Font used for drawing the text in object.
Background Color	...	Background color selectable from palette.
Text Color	...	Text color selectable from palette.
Sel. Background	...	Background color selectable from palette when the object is chosen. This property is not available if the **Select** field is constant **FALSE**.
Sel. Foreground	...	Text color selectable from palette when the object is chosen. This property is not available if the **Select** field is constant **FALSE**.
Appearance	**Flat, Raised, Sunken**	• Flat • Raised • Sunken
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to Flat.
Border color	...	Border color selectable from palette. This property is available only if Appearance is set to Flat.
Number of Chars	≥ 0	Number of chars that this object can show. If the value is 0 the object shows the complete text. Otherwise with another value it can be truncated or extended.
Alignment	**Right, Center, Left**	Text alignment in the object.
Refresh	**TRUE, FALSE**	Continuous redraw of the object: • **FALSE**: the **Text value** is read from memory and updated only when opening the page or when a child page is closed, • **TRUE**: the **Text value** is read from memory and always updated.
Select	**TRUE, FALSE, var_name**	Selected status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is selected and so it shows the colors **Select Back**, **Select Fore**.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise it is hidden.

Events

Events	Description
BeforeUpdate	Before the object is redrawn.
AfterUpdate	Immediately after the object is redrawn.

Line

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
X2Pos	> 0	Bottom-right 'x coordinate' edge relative to page.
Y2Pos	> 0	Bottom-right 'y coordinate' edge relative to page.
Name	Not empty	Name of object.
Thickness pts	> 0	Line thickness (#pixel).
Border col	...	Line color selectable from palette.

Rectangle

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
XDim	> 0	Width (#pixel).
YDim	> 0	Height (#pixel).
Name	Not empty	Name of object.
Border points	> 0	Border thickness (#pixel).
Border color	...	Border color selectable from palette.
Background Color	...	Background color selectable from palette. This property is available only if Transparent is set to TRUE.
Transparent	**TRUE**, **FALSE**	Transparency: • **TRUE**: transparent background, • **FALSE**: solid background where color is Back Color.

Edit Box

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
Name	Not empty	Name of object.
Appearance	**Flat, Raised, Sunken**	• **Flat**: plain with use of **Border pts** and **Border col**, • **Raised** • **Sunken**
Font	Name found in **Resources**	Font used for drawing the text in object.
Background Color	...	Background color selectable from palette.
Text Color	...	Text color selectable from palette.
Sel. Background	...	Background color selectable from palette when the object is chosen. This property is not available if the **Selectable** field is constant **FALSE**.

Properties	Available values	Description
Sel. Foreground	...	Text color selectable from palette when the object is chosen. This property is not available if the **Selectable** field is constant **FALSE**.
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to **Flat**.
Border color	...	Border color selectable from palette. This property is available only if **Appearance** is set to **Flat**.
Number of Chars	> 0	Chars visible in the object. Width of entire object is calculated among this value and the size of Font. If **NumChar** are less than the value, the object shows this error string: #####.
Format	String as **printf** or **enum_name**	The format can be numeric, to define as **printf** of C language (refer to Format Specification - Printf, page 366), numerative, if in this field there is **enum_name** defined in **Resources** (refer to Resources, page 376).
Alignment	**Right, Center, Left**	Text alignment in the object.
Access	**RO, RW**	Accesses variable **Assoc var** used in object: • **RO** = read only, • **RW** = read/write.
Selection Order	≥ 0	Selection order of the object. It can be selected by pressing a key or by means of a procedure. In this case the selection moves from the current object to the previous or next **Sel. Order** object.
Variable	Not empty	Name of the variable that can be shown and edited with this object. It can be any variable of the project, (local, global, imported from PLC or target - refer to Edit Box and **Display** Variable Association, page 351), a parameter (refer toEdit Box and **Display** Variable Association, page 351) or an element of a set (refer to Multiple Pages Management, page 340).
Data type	**UNDEF, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, STRING**	Type of **Assoc var**. If it is a variable, the type is defined automatically. This property is available if **Assoc var** is an explicit parameter.
Low limit	**CONSTANT, var_name**	Name of variable or numeric constant. This is the least number that the object can show. It can be any variable of the project, (local, global, imported from PLC or target - refer to Edit Box and **Display** Variable Association, page 351). This object shows an error string (!!!!!!!) if condition does not holds. The * symbol means that there is no low limit.
High limit	**CONSTANT, var_name**	Name of the variable or numeric constant. This is the maximum number that the object can show. It can be any variable of the project, (local, global, imported from PLC or target - refer to Edit Box and **Display** Variable Association, page 351). This object views an error string (!!!!!!!) if condition does not hold. The * symbol means that there is no high limit.
Refresh	**TRUE, FALSE**	Enables continuous update of the value:**FALSE**: the **Assoc var** value is read from memory and updated only when open page or when a child page is closed,**TRUE**: the **Assoc var** value is read from memory and always updated.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise hidden.
Selectable	**TRUE, FALSE, var_name**	Selected status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is selected and so it shows the colors **Select Back**, **Select Fore**. If this field is **FALSE** the **Access property** is not available.
Label	-	If the target has a touchscreen display, shows keyboard and has this feature enabled.

Events

Events	Description
BeforeUpdate	Before the object is redrawn.
AfterUpdate	Immediately after the object is redrawn.

Events	Description
OnEnter	Whenever the object is selected and receives the command for entering in edit-mode.
OnClick	Touchsreen system: Whenever HMI receives a pressure on the object. Other devices: Whenever an object enters the editing mode using the local keyboard.
OnChange	Touchsreen system: Whenever you confirm the modifications and the value is different from start. Other devices: Whenever an object exits the editing mode, even if its value remains unchanged.

Format Specification - Printf

Functions and Values

If the object has not any enumerative format, the format string is composed as follows:

%[flags][width][.precision]type

The field has one or more characters, that describe the specification. The simplest format contains only percentage symbol and one char as type (for example: %s).

Next table explains in details functions and values.

Field	Available values	Description
flags	• + prints always the sign, even if the number is positive. • 0 prints zeros in head until **width** (if specified) or **NumChar**.	This char is an option for chars order, print sign, number of decimal digit. This field may have more than one flag.
width	> 0, ≤**NumChar**	Maximum chars can be printed. Allows to view values that do not fill **NumChar** fully.
precision	≥ 0	Decimal digits after the point. If the field is an integer and there is a precision the object shows a decimal point. E.g. the value is 102 integer, and precision is 2, with %.2d, the number is shown as 1.02
type	• %d: Integer with sign. • %f: Real. • %x: Hexadecimal with lowercase chars. • %x: Hexadecimal with uppercase chars. • %s: String. • %@sdf: Password. • [%d,u,f,x]: Custom measure unit format.	Mandatory field.

Image

Properties

Properties	Available values	Description
XPos	const ≥ 0, variable	Top-left ‘x coordinate’ edge relative to page. It is possible to assign a variable only if **Style** is set to **Floating**.
YPos	const ≥0, variable	Top-left ‘y coordinate’ edge relative to page. It is possible to assign a variable only if **Style** is set to **Floating**.
XDim	> 0	Width (#pixel).
YDim	> 0	Height (#pixel).
Name	Not empty	Name of object.
Appearence	**Flat, Raised, Sunken**	• **Flat**: plain with use of **Border pts** and **Border col**,

Properties	Available values	Description
		• **Raised**, • **Sunken**.
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to **Flat**.
Border color	...	Border color selectable from palette. This property is available only if **Appearance** is set to **Flat**.
Bitmap	Name found in **Resources**	Bitmap used for drawing the image in object.
Background image	Image object in the page	Name of another object that is redrawn when **Style** is set to **Floating**. It is available only if it is overlapped with this image.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise it is hidden.
Style	**Docking, Floating**	• **Docking**: fixed position, • **Floating**: variable position, according to **XPos** variable and **Ypos** variable.

Animation

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
XDim	> 0	Width (#pixel).
YDim	> 0	Height (#pixel).
Name	Not empty	Name of object.
Appearance	**Flat, Raised, Sunken**	• **Flat**: plain with use of **Border pts** and **Border col**, • **Raised**, • **Sunken**.
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to **Flat**.
Border color	...	Border color selectable from palette. This property is available only if **Appearance** is set to **Flat**.
Image list	Name found in **Resources**	It contains the images that the object can view and the value range.
Animation variable	**var_name**	Name of the variable that is compared with value range in **Image** list.
Data type	**SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD**	Type of **Animation var**. If it is a variable, the type is automatically defined.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise hidden.

Events

Events	Description
BeforeUpdate	Before the object is redrawn.
AfterUpdate	Immediately after the object is redrawn.

Button

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
XDim	> 0	Width (#pixel).
YDim	> 0	Height (#pixel).
Name	Not empty	Name of object.
Text/Img	Empty or explicit text or Resource ID or Bitmap	Text or image to view in the button: • string, • Resource ID, • bitmap.
Selection Text/Img	Empty or explicit text or Resource ID or Bitmap	Text or image to view in the button when it is selected: • string, • Resource ID, • bitmap.
Font	Name found in **Resources**	Font used for drawing the text in object. This field is not available if it shows a bitmap.
Appearance	**Flat, Raised, Sunken**	• **Flat**: plain with use of **Border pts** and **Border col**, • **Raised**, • **Sunken**.
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to **Flat**.
Border color	...	Border color selectable from palette. This property is available only if **Appearance** is set to **Flat** or Text is not empty.
Background color	...	Background color selectable from palette. This property is available only if **Transparent** is set to **TRUE**.
Selection border	...	Border color when the object is selected. This property is not available if **Selection var** is **FALSE** fixed.
Sel. background	...	Background color when the object is selected. This property is not available if **Selection var** is **FALSE** fixed.
Selection order	≥ 0	Selection order on which the object can be selected with the pressure of a key or with a procedure. In this case the selection moves from the current object to the previous or next **Sel. Order** object.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise it is hidden.
Transparent	**TRUE, FALSE, var_name**	Transparency. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is transparent.
Press variable	Empty or **var_name**	When the button is pressed **var_name** is set to **TRUE**. When the button is not pressed, **var_name** is set to **FALSE**.
Selection variable	**TRUE, FALSE, var_name**	Selected status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**. If **var_name** is **TRUE** the object is selected and so it shows the colors **Select Back**, **SelectBord**. If this field is **FALSE**, **SelectBord**and **Select Back** properties are not available.
Action	**Call, OpenPage, Close, NextField, PrevField, Edit**	Action executed on button pressure.
Action par	**page_name proc_name**	Parameter associated with the action executed on button pressure. It is available only if **Action** is **OpenPage** (**Action par** = name of the page to open) or **Call** (**Action par** = name of the procedure to execute).
Alignment	**Right, Center, Left**	Text alignment in the object.

Events

Events	Description
OnClick	Whenever HMI receives a pressure on the object, valid only for touchscreen systems.
OnRelease	Whenever HMI releases the pressure on the object, valid only for touchscreen systems.

Progress Bar

Properties

Properties	Available values	Description
XPos	≥ 0	Top-left 'x coordinate' edge relative to page.
YPos	≥ 0	Top-left 'y coordinate' edge relative to page.
XDim	> 0	Width (#pixel).
YDim	> 0	Height (#pixel).
Name	Not empty	Name of object.
Appearance	**Flat, Raised, Sunken**	• **Flat**: plain with use of **Border pts** and **Border col**, • **Raised**, • **Sunken**.
Border points	≥ 0	Border thickness (#pixel). This property is available only if **Appearance** is set to **Flat**.
Border color	...	Border color selectable from palette. This property is available only if **Appearance** is set to **Flat** or **Text** is not empty.
Bar color	...	Color of step bar, selectable from palette.
Background color	...	Background color selectable from palette.
Visible	**TRUE, FALSE, var_name**	Visible status of the object. It can be constant (**TRUE** or **FALSE**) or linked with a boolean variable **var_name**: if **var_name** is **TRUE** the object is visible, otherwise it is hidden.
Refresh trigger	**TRUE, FALSE, var_name**	Object redraw: • **FALSE**: the **Progress var** value is read from memory and updated only when opening page or when a child page is closed. • **TRUE**: the **Progress var** value is read from memory and always updated. • **var_name**: the Progress var value is read from memory and updated only when the variable becomes **TRUE**. After the update the runtime sets it to **FALSE**.
Progress variable	Not empty	Step variable. This is the filling percentage of bar in relation with the range assigned by **Lo limit** and **Hi limit**. It can be any string variable of the project (local, global, imported from PLC or target) or a parameter (Edit Box and **Display** Variable Association, page 351).
Data type	**UNDEF, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LWORD, REAL, LREAL, STRING**	Type of **Progress var**. If it is a variable, the type is automatically defined. This property is available if **Progress var** is an explicit parameter.
Low limit	Constant or **var_name**	Name of the variable or numeric constant. This is the least value for step bar. It can be any variable of the project, (local, global, imported from PLC or target) with type specified by **Data type**.
High limit	Constant or **var_name**	Name of the variable or numeric constant. This is the maximum value for step bar. It can be any variable of the project, (local, global, imported from PLC or target) with type specified by **Data type**.
Orientation	Horizontal, vertical	Direction of step bar.

Events

Events	Description
BeforeUpdate	Before the object is redrawn.
AfterUpdate	Immediately after the object is redrawn.

Declaration of Variables

Types of Variables

Overview

In a **Display** project, there are four different classes of variables.

Local Variables

Local variables are accessible via the page from which they were declared.

They are listed in the project tree, under the **Local variables** folder. Local variables can be used to carry out operations on PLC (for example to apply a different scale or to add an offset) or system variables, or to implement local procedures.

Global Variables

Global variables are declared in **Display** and they are accessible from every page of the project. Global variables are listed in the **Global variables** folder in the project tree. The function of the global variables is similar to the one of the local variable but the different visibility scope makes them unusable for the implementation of global procedures or for the parameters passing between distinct pages.

System Variables

The interaction between **Display** and target is enabled by system variables which the software publishes outside in a *.tgt file.

You may access system variables in read/write or in read-only mode. If you try to access a read-only variable in write mode, an error occurs during compilation.

Data Management

Overview

It is possible to distinguish the data in local variables (visible in the page scope only) and global variables (visible from every page). For some controls it is possible to use parameters and sets.

Declaring a Local Variable

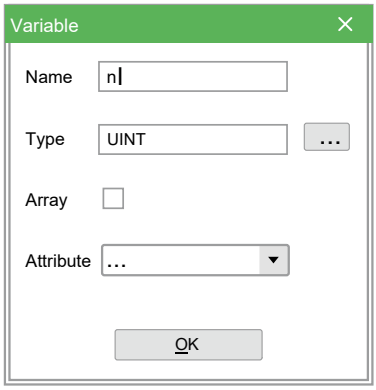
Declare a local variable, which you can use just in the specific page where the declaration takes place.

In the pages tree, under the **Init** page item, double-click the **Local variables** item.

The local variables editor window opens. It is blank at present.

Click **New record** icon in the **HMI Project** toolbar.

A dialog window opens requesting to specify the basic features of the new variable. For example, you can declare “n” as a new 16 bit unsigned integer variable.



Confirm the operation by clicking **OK** button. The new corresponding record is added to the variables editor window.

You can change the features of this new variable by editing the fields of the record which you have just created. For example, you may assign an initial value different from null and a comment.

When you save the project by clicking the apposite button or when you close the variables editor, **Display** adds a new item in the pages tree. It corresponds to the local variable which you have just declared.

Declaring a Global Variable

Let us assume that you want to declare a floating point global variable “t”: right-click on the **Variables** item under the **Global variables** node of the resources tree and select the **Open** command in the contextual menu which appears.

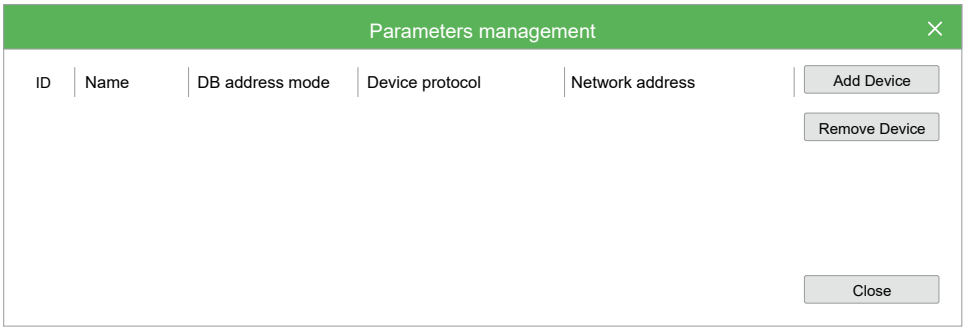
Follow the steps as shown in Declaring a Local Variable, page 370, until the new global variable appears as a new item in the pages tree.

Inserting Field Parameters

Target system usually has internal variables and is connected on a fieldbus, so it needs to show some variables of the different devices which are connected on the net.

For this reason, **Display** allows you to link a specific file which contains the variables definition on the bus. Click the **Parameters management** icon in **HMI Profile** toolbar.

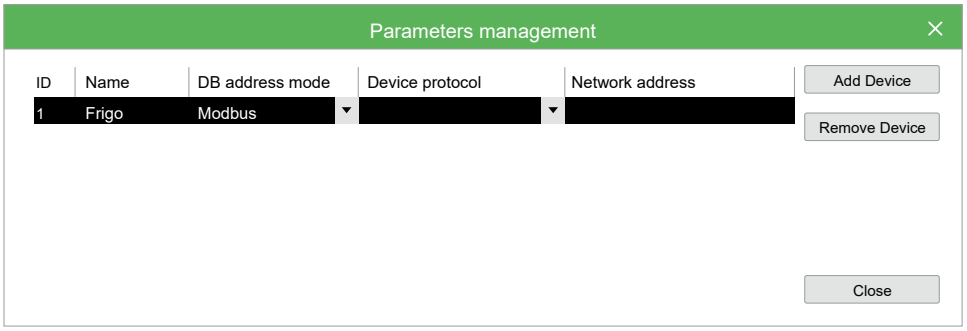
The parameters management window appears.



Through the **Add Device** button you can add a new object linked to the target on the fieldbus.

The selection window appears. Then you have to take from your PC a *.parx file (for more information, refer to Description of Parameter File, page 372).

A device called ”Frigo” has been inserted. In order to see the relevant parameters, click the **Close** button.



In **HMI Vars and Parameters** window, you can see the device and its parameters.

When you need to update the list of parameters, if the *.parx file has not been moved to another directory, it is not necessary to repeat the above mentioned procedure, but it is enough to press the button.

Description of Parameter File

Overview

As described in Inserting Field Parameters, page 371, it is possible to link in **Display** some variables from external device.

In some objects you can define an explicit or implicit syntax in order to use the parameter mode.

To use the implicit syntax, `@Device.Parameter`, **Display** requires a *.parx file in XML format.

For example:

```
<parameters>
    <par ipa=”10100” name=”Par_TAB” descr=”Tab (map code)” defval=”0” min=
    ”0” max=”65535” um=”num” typetarg=”unsignedShort” typepar=
    "unsignedShort">
        <protocol name=”Modbus” commaddr=”15716” commsubindex=”0”/>
        <protocol name=”CanOpen” commaddr=”15716” commsubindex=”0”/>
    </par>
    <par ipa=”10001” name=”Gain_Ntc_AI2” descr=”NTC calibration gain AI2”
    defval=”32768” min=”0” max=”65535” um=”num” typetarg=”unsignedShort”
    typepar="unsignedShort">
        <protocol name=”Modbus” commaddr=”15617” commsubindex=”0”/>
        <protocol name=”CanOpen” commaddr=”15617” commsubindex=”0”/>
    </par>
    <par ipa=”11308” readonly=”false” name=”Modem_InitStr1” defval=””
    descr=”InitString (1st part)” typetarg=”string” strsize=”19”>
        <protocol name=”Modbus” commaddr=”15821” commsubindex=”0”/>
        <protocol name=”CanOpen” commaddr=”15821” commsubindex=”0”/>
    </par>
</parameters>
```

Where each parameter has these fields:

- `ipa`: parameter index used as input value of `Video_SetParam()`, `Video_GetParam()`. If there are nodes with protocol type, they have more priority than `ipa`, so **Display** uses them.
- `Name`: parameter name.
- `descr`: complete description of parameter.
- `defval`: default value of parameter.

- `min`: minimum value of parameter.
- `max`: maximum value of parameter.
- `um`: measure unit of parameter.
- `typetarg`: type of parameter read as “Installer type”. The available values with the translation in PLC are:
  - `char`: SINT,
  - `unsignedChar`: USINT,
  - `short`: INT,
  - `unsignedShort`: UINT,
  - `int`: DINT,
  - `unsignedInt`: UDINT,
  - `boolean`: BOOL,
  - `digitalInput`: BOOL,
  - `digitalOutput`: BOOL,
  - `float`: REAL,
  - `double`: REAL,
  - `string`: STRING.
- `typepar`: type of parameter read as “IEC type”. In case of generic modbus slaves should be the same as `typetarg`. The available values with the translation in PLC are:
  - `char`: SINT,
  - `unsignedChar`: USINT,
  - `short`: INT,
  - `unsignedShort`: UINT,
  - `int`: DINT,
  - `unsignedInt`: UDINT,
  - `boolean`: BOOL,
  - `digitalInput`: BOOL,
  - `digitalOutput`: BOOL,
  - `float`: REAL,
  - `double`: REAL,
  - `string`: STRING.
- `strsize`: number of character if it is a string type.

Using Advanced Features

Events

Overview

There are different classes of events.

Page or Control Events

Each characteristic behavior of a specific object can raise a specific event.

Each event can be associated to a procedure Procedures that Can Be Associated to Events, page 374 that is executed each time the event takes place. The list of

all available events for each **Display** object (page or control) is reported in Page and Object Properties, page 360.

Key Pressure Events

These events take place when a key is pressed, the raising of the event starts the execution of the associated action (refer to Actions that Can Be Associated to Key Pressure, page 375) if it is. The pressure of a key can be also simulated by software.

Events Raised by Software

Programmer can raise events by software using the function `Video_SendEvent` inside the target software or in the body of the procedure, using following syntax:

`Video_SendEvent` (`event_id`, `param`),

Where `event_id` is the identifier of the type of the event and `param` is an integer 16 bit parameter.

Display supports software events defined in this table:

Event	Parameter	Description
`kWM_NULL`	Do not care	No event.
`kWM_KEY`	Key code	Simulates the pressure of the key specified as parameter then cause the associated action if it is.
`kWM_MSG`	Window ID	Causes a system message that, once got by the system, causes the instant opening of the alarm page that has `Window ID` as identifier.
`kWM_SELECT`	Edit box handle	In touchscreen systems simulates the pressure on the edit box whose handle is passed as parameter, causing its selection or its transition to edit mode.
`kWM_PUSH`	Button handle	In touchscreen systems simulates the pressure on the button whose handle is passed as parameter, causing the execution of the associated action if it is.
`kEV_WM_CHANGESETPAGE`	Page number	Shows the page specified by the parameter (if the context is a page in which sets are used).

Procedures that Can Be Associated to Events

A procedure is a program that is executed when the event that has been associated to it, takes place.

There are two classes of procedures:

- Local procedures: This kind of procedures can be called only within the scope of the page in which are declared. In particular, they can be associated to the events of the page itself and of all their controls. The same can be said for software events raised when the page they refer to is active. Procedure code can contains references to all the types of variables, with local variables of the page too.
- Global procedures: This kind of procedures can be called from every page and can be also used as periodic asynchronous routine of alarm management. They can not contain variables references.

Here follow the description of the syntax to get the properties of a control from a procedure; similarly to C language printf it is:

“`fb%s%s.%s`”, `page_name`, `ctrl_name`, `prop_name`

Where:

- `page_name` is the name of the page that has the control,
- `ctrl_name` is the name of the control,
- `prop_name` is the name of the property of the control.

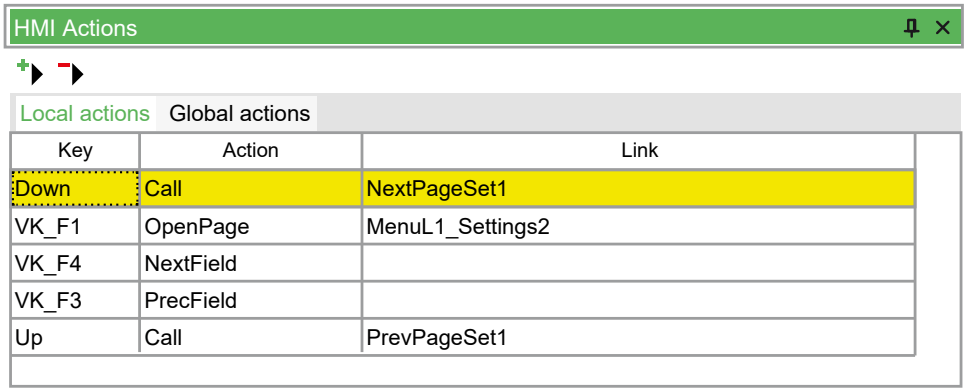
So if you want to get the property `Foreground color` of the `Static` named `String_26` in `Main` page, we have to write:`fbMainString_26.foreCol`.

> **NOTE:** The name of the property to use in the scripts of the procedures is the name of the functional block exported by the software of the target, not the name in the properties window, page 319.

Actions that Can Be Associated to Key Pressure

In common keyboard, not touchscreen systems, interaction between you and the system is normally based on keys pressure.

The **HMI Actions** window allows you to associate a code of a key to one of the actions listed in the table. In this way the pressure of that key causes the specified action.



The table of the **HMI Actions** window is composed like this:

Event	Link	Description
Call	Procedure name	Causes the invocation of the local or global procedure whose name is indicated in the **Link** field.
OpenPage	Page name	Causes the opening of the page whose name is indicated in the **Link** field.
Close	Do not care	Causes the closure of the current page
NextField	Do not care	Move the selection to the next edit box. If the system is not touchscreen moves selection to the buttons to allow their pressure.
PrevField	Do not care	Move the selection to the previous edit box.
Edit	Do not care	Access edit mode for the selected edit box. If the system is not touchscreen allows you to simulate the pressure of the button.

There are two types of associations key-action:

- **Local actions**: local associations, valid only for the page currently open in the editor of the pages.
- **Global actions**: global associations, valid in any point of the project.

If the system has the touchscreen feature, normal interaction with you is made by the pressure of sensible area on the screen. However this table does not loss its meaning because allows you to define virtual keys and to control their pressure by software causing in this way the dynamic execution of specific actions.

NOTE: If the same action is defined both at local and at global level, system does not give errors nor warnings because local declaration precedes global one.

Resources

Overview

A resource is an interface element. You can get informations from resources or can use them to do actions.

Display supports different categories of resources that are managed by **Resources** item in **HMI Project** window, page 319.

Fonts

Fonts are the different kinds of characters supported for the output of text strings on the screen. Fonts had been managed by FREE Studio Plus old versions as text files with *.plf extension and structured with the same syntax of the initialization definition of an array variable in IEC. Now, images are saved and loaded in binary format to optimize loading time on images of big size.

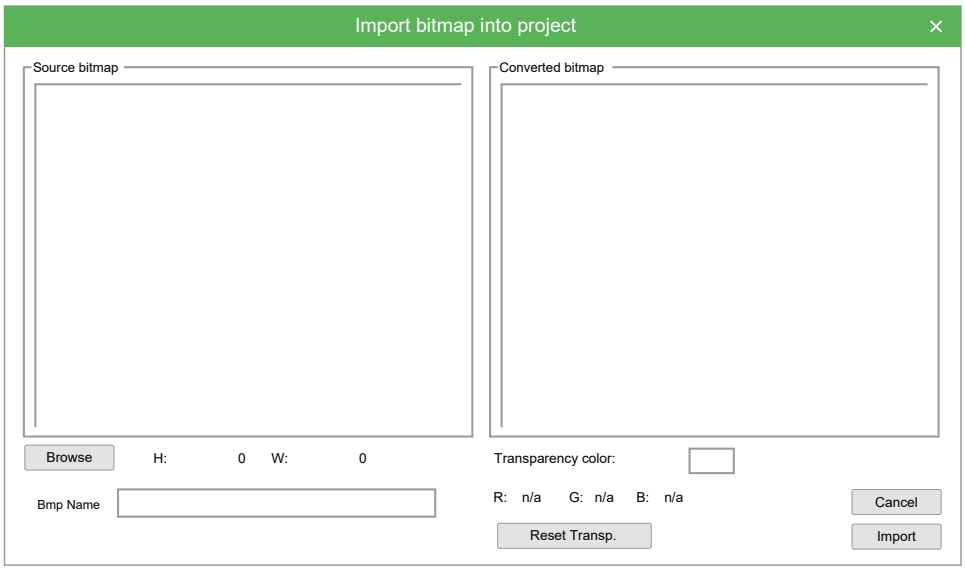
At project opening time, if FREE Studio Plus finds this declaration, it searches in project folder for a file named “font_name.plk” and loads it in memory.

Bitmaps

Bitmaps are pictures to associate to image controls, page 345. The images are saved and loaded in binary format to optimize loading time on images of big size.

At project opening time, if **Display** finds this declaration, it searches in project folder for a file named “bitmap_name.plk” and loads it in memory.

Display provides a tool to convert bitmaps from Windows format to **Display** format. To start it, right-click **Bitmaps** item of **Resources** in **HMI Project** window and click **Import bitmap...** command.



Click **Browse** button to navigate in computer resources and select the desired source file.

In **Bmp Name** field, you can personalize bitmap name that is shown on resource tree. Bitmap name is constituted by file name without extension and with “Bmp” prefix by default.

Transparency color field allows you to specify transparency color. So a color that is not really drawn on the screen but a transparent color zone that does not cover elements previously drawn.

Transparency color can be personalized by choosing it by mouse from **Converted bitmap** window.

RGB indicates transparency color Red, Green, Blue components. **n/a** value indicates that no transparency color has been selected.

Reset Transp. button allows you to undo last selected transparency color.

Once finished these operations it is possible to confirm bitmap importation by clicking **Import** button.

Strings Table

It is possible to explicitly write the text to show on a text string or on a title of the page. It is also possible to refer to one of the strings of the resources specifying its ID. For more information, refer to Language Selection, page 338.

In first case, text is always the same. In second case, the text that correspond to the active language is shown.

So, English language string table contains the following record.

IDS_State	State

And French language string table contains the following record.

IDS_State	Etat

If you refer to the identifier `ID_GDB_RXNAK` from a page control or from a page:

- If current active language is English, “Bad RX packets” is shown.
- If current active language is French, “Paquets RX défectueux” is shown instead.

Enumeratives

An enumerative is a data type defined by you, it is a set of constants that you named. Each element of an enumerative is treated as a constant and can be translated in the available languages of the project.

For example: create an enumerative called “SettingsTouch”.

Enumerative records are shown by double-clicking **SettingsTouch** node.

Then, introduce an edit box control, page 348 and insert the name of the enumerative “SettingsTouch” in its **Format**property in **HMI Properties** window. Control shows the string associated to the value as it is in the table above, not the numeric value of the variable associated to the control.

If the numeric value of the variable does not match with any record of the enumerative table, an error string ######## is shown instead.

Even enumerative are supported by multi-language feature. It is possible to personalize the name and the record values of the enumerative.

Value	Description
0	Awaiting settings ...
1	Saving settings ...
2	Touch screen set up

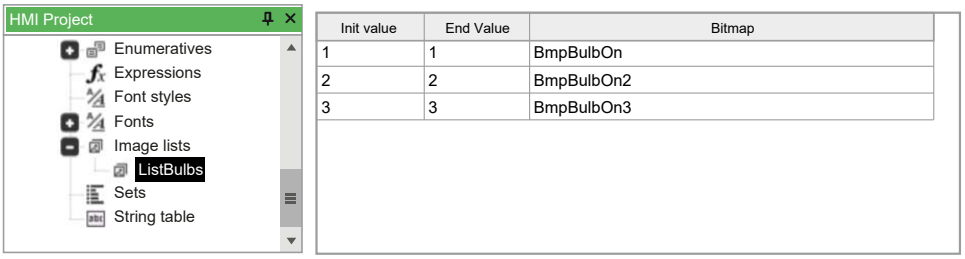
Image Lists

An images list is very similar to an enumerative but with the following differences:

- Intervals of constants are supported (not only simple values),
- Each value has an image associated,
- A list of images determines the content shown by an animation control, while an enumerative can be associated to an edit box.

For example: if you have an images list “ListBulbs” that is shown under **Resources**.

It is possible to see all the records of the list by double-clicking the node.



Introduce an animation control, page 347 in the page, and select “ListBulbs” in **Image list** property in **HMI Properties** window. The control shows the image whose specified interval includes the value of a variable associated to the control.

If the numeric value of the associated variable does not match any record in the list, a default image (with init and end value set to *) is shown if it is. If no default image is specified, no image is drawn.

Sets

Sets are ensemble of global variable even of distinct type.

In particular there are two types of set:

- Variable/parameter sets even of not equal type (**VARIANT**),

- Strings sets (**STRINGS**).

The sets of the first type are defined indicating **VARIANT** as type. This kind of set has the following attributes:

- Dynamic: indicates that every n execution cycles target automatically reloads the elements of the set and hides those elements that have no visibility (boolean constant **FALSE** or associated visibility variable set to false at that moment).
- Array: indicates that the unique element of this set is a variable of type array.

  NOTE: This kind of set can be assigned only to an edit-box control.

Once defined a set, each element of the set can be added via drag and drop from **HMI Vars and Parameters** window or can be manually inserted by you.

The **VARIANT** type is used to define the following attributes:

- **Variable/Parameter**: variable/parameter name.
- **Format**: indicates how to show associated variable value specifying a syntax analogous to C language printf (for more information, refer to Child Page, page 361).
- **Text Align**: the alignment of the text to show.
- **Min/Max**: minimum and maximum value for the element of the set.
- **Visible**: boolean variable or constant that defines the visibility of the element. If dynamic feature of the set is active, the variable is periodically checked to hide or show the element.
- **Selectable**: indicates that the element can be selected. In this case, a boolean variable or constant can be assigned too.

For a set of type **STRING**, you have to define only two attributes, the string or the ID of a string resource, page 377 and the variable/constant of visibility. An element not visible is not shown on the screen.

NOTE: This kind of set can be used with static control only.

File for Target Description

Overview

The files contain some definitions of target environment. **Display** uses this information for generating custom code.

The *.def file consists of two sections. It is allowed comment, that starts with a semicolon (;).

This file is included in *.pajx file.

Target Properties

Description

This section consist of five records, which support one or more parameters. Each record is on new line and the elements must be separated with spaces or tabs.

Record Structure			
Header	**Param. 1**	**Param. 2**	**Description**
SCREEN	**dimX**	**dimY**	Screen dimension of target measured in pixel: • **DimX**: width, • **DimY**: height.
SAVESCREEN	0/1	—	Target board can save and restore video memory: • 0: no save, • 1: save and restore.
TOUCHSCREE-N	0/1	—	Target board has touchscreen can use the pressure events: • 0: no touchscreen; • 1: exists touchscreen.
REFRESH	msec	—	Refresh time of all objects in page, measured in milliseconds.
FONT_ FORMAT	“HH”/“VH”	—	Font encoding.
ColorSET	“RGB”	—	
BMP_FORMAT	“SIMULAB”	—	Image encoding.
UNICODE	0/1	—	Target board has support for Unicode fonts.
JOYPAD	0/1	—	Target board has a joypad that can be used for moving among elements of page and can be connected to actions.
INIT	0/1	—	If set to 1 says that HMI run-time has `Video_InitHMI()`, invoked on target start-up. Typically it is used for custom commands on start-up.
BMPFULL	0/1	—	If set to 1 generates PLC code extended for bitmap instead binary bitmap.

Object Version

The graphical objects (edit box, text box, static, bitmap, and so on) can have a version, or can not exist. The syntax is:

```
CTRL “Name” “Version”
```

Where:

- `Name`: name of graphical object. Example: `Editbox`;
- `Version`: version of HMI run-time objects.

If this value is set to -1, **Display** does not make available this object.

System Enumeratives

Enumeratives of *.def file are maps for binding among numeric values and strings, or other numeric values.

Each enumerative has an identifier, that specifies a function in the map with this syntax:

`ENUM id en_key en_val`

Where:

- `id`: enumerative identifier,
- `en_key`: value-key of record (must be a number),
- `en_val`: value of value-key (can be a number or string).

Descriptions

This paragraph describes the values for system enumeratives.

- **Enumerative 100**

  With this key you can define new buttons (the names are shown in the **Key** field of actions table in **HMI Actions** window (refer toActions that Can Be Associated to Key Pressure, page 375).

The number of lines is not limited. But you must define at least the elements of 102 enumerative.

- `id`: 100,
- `en_key`: key encoding (one byte),
- `en_val`: string with key name.
- **Enumerative 101**

  With this key you can define new actions (the names are shown in **Action** field of actions table in **HMI Actions** window (refer toActions that Can Be Associated to Key Pressure, page 375.

- `id`: 101,
- `en_key`: action identifier,
- `en_val`: string with action name.

This enumerative has a well defined number of lines. The following table presents the corresponding actions:

en_key	Action
0	Calls local or global procedures.
1	Opens child page.
2	Closes current page.
3	Selects next object Edit Box, Button, and so on.
4	Selects previous object Edit Box, Button, and so on.
9	Enters editing-mode (Edit Box, Button).
10	Leaving (not implemented).

The string `en_val` is arbitrary.

- **Enumerative 102**

Selection and edit functions:

- `id`: 102,
- `en_key`: identifier of edit function,
- `en_val`: string with the name of associated string.

The name of this field `en_val` must be the same of `en_val` of 100 enumerative, so that **Display** associates an edit function with a key.

This enumerative has a well defined number of lines. See the actions in the table below:

en_key	Action
0	Confirms modifications and leaves editing-mode.
1	Loses modification and leaves editing-mode.
2	Deletes selected character.
3	Moves cursor left.
4	Moves cursor right.
5	Selects the previous element of an enumerative associated with an Editbox.
6	Selects the next element of an enumerative associated with an Editbox.
7	Deletes the first character on the left.
8	Inserts tab character.
9	Switching to uppercase alphanumeric characters for a single character.
10	Transition to permanent uppercase alphanumeric characters.

- **Enumerative 103**

Define a color palette, the encoding is RGB:

- `id`: 103,
- `en_key`: index of the color inside palette,
- `en_val`: RGB color encoding.

RGB encoding represents 24 bit of colors: `0x00bbggrr` where `bb` (1 byte) intensity of blue, `gg` (1 byte) the green and `rr` (1 byte) the red. The intensity is at least 0 and at most `0xff`.

The number of lines is not limited. You can define which colors he wants.

- **Enumerative 104**

Names of object styles (shown on **Appearance** in **HMI Properties** window):

- `id`: 104,
- `en_key`: style,
- `en_val`: string with the name of style.

This enumerative contains at most 3 records, supported by **Display**.

en_key	Style
0	Flat, plane.
1	Raised.
2	Sunken.

Example

```
;
;Target properties
;
SCREEN 128 64
SAVESCREEN 1
```

```
REFRESH 50
FONT_FORMAT "VH"
JOYPAD 1
INIT 1
BMPFULL 1
UNICODE 1
;
;Versions of controls
;
CTRL "Static" 1
CTRL "EditBox" 1
CTRL "TextBox" -1
CTRL "Button" 2
CTRL "Progress" 0
CTRL "Animation" 0
CTRL "Image" 0
CTRL "CustomCtrl" -1
CTRL "Chart" -1
CTRL "Trend" -1



;
;Enumeratives
; ENUM 100: key codes
;
ENUM 10013 "Enter"
ENUM 1008 "Left"
ENUM 10012 "Right"
ENUM 10011 "Up"
ENUM 10010 "Down"
ENUM 10019 "LongEnter"
ENUM 10015 "LongLeft"
ENUM 10016 "LongRight"
ENUM 10017 "LongUp"
ENUM 10018 "LongDown"
ENUM 10030 "VK_F1"
ENUM 10031 "VK_F2"
ENUM 10032 "VK_F3"
ENUM 10033 "VK_F4"
ENUM 10034 "VK_F5"
ENUM 10035 "VK_F6"
ENUM 10036 "VK_F7"
ENUM 10037 "VK_F8"
ENUM 10038 "VK_F9"
ENUM 10039 "VK_F10"
;
; ENUM 101: key-related actions
;
ENUM 1010 "Call"
ENUM 1011 "OpenPage"
ENUM 1012 "Close"
ENUM 1013 "NextField"
ENUM 1014 "PrevField"
ENUM 1019 "Edit"
;
; ENUM 102: editing-mode keys
;
ENUM 1020 "Enter"
ENUM 1021 "LongLeft"
ENUM 1023 "Left"
ENUM 1024 "Right"
ENUM 1025 "Up"
ENUM 1026 "Down"
;
; ENUM 103: color codes
;BBGGRR
ENUM 1030 "0x00000000" ; Bianco
```

```
ENUM 1031"0x00FFFFFF" ; Nero
;
; ENUM 104: controls appearance
;
ENUM 1040"Flat"
ENUM 1041"Raised"
ENUM 1042"Sunken"
```

Functions and Function Blocks for HMI

What's in This Chapter

Overview

Here is the lists of the functions that HMI run-time exports to **Display**. You can use them into script and procedures.

These functions are divided into several categories which are shown in details in the following paragraphs.

Functions for HMI

System Functions: Hardware and Operating System

Video_InitHMI (dmy)			
Initialization for HMI runtime			
I/O	**Name**	**Type**	**Description**
Input	dmy	unsigned char	Reserved. Set 0.
Output	Video_InitHMI	unsigned char	TRUE if successful, FALSE otherwise.

Video_Switch (on)			
Turn on/off the display			
I/O	**Name**	**Type**	**Description**
Input	on	unsigned char	TRUE: turns on the display. FALSE: turns off the display.
Output	Video_Switch	unsigned char	Not sensible (always TRUE).

Video_LCDContrast (more)			
Display contrast			
I/O	**Name**	**Type**	**Description**
Input	on	unsigned char	TRUE: turns on the display. FALSE: turns off the display.
Output	Video_Switch	unsigned char	Not sensible (always TRUE).

Video_SaveRect (x1, y1, x2, y2)			
Save display area to memory			
I/O	**Name**	**Type**	**Description**
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
Output	Video_SaveRect	unsigned char	Not sensible (always TRUE).

Video_WriteFromBuff (x1, y1, x2, y2)			
Restore display area from memory (previously saved with Video_SaveRect).			
I/O	**Name**	**Type**	**Description**
Input	x1	unsigned short	Not sensible (saved area has the original coordinates).
	y1	unsigned short	Not sensible (saved area has the original coordinates).
	x2	unsigned short	Not sensible (saved area has the original coordinates).
	y2	unsigned short	Not sensible (saved area has the original coordinates).
Output	Video_ WriteFromBuff	unsigned char	Not sensible (always TRUE).

Video_Lock (res)			
Lock the display resources for exclusive access			
I/O	**Name**	**Type**	**Description**
Input	res	unsigned char	Reserved. Set 0.
Output	Video_Lock	unsigned char	Not sensible (return input parameter res).

Video_Unlock (res)			
Unlock the display resource after exclusive access			
I/O	**Name**	**Type**	**Description**
Input	res	unsigned char	Reserved. Set 0.
Output	Video_Unlock	unsigned char	Not sensible (return input parameter res).

Video_Sleep (msec)			
Suspend the task where the function is used			
I/O	**Name**	**Type**	**Description**
Input	msec	unsigned short	Suspends time measured in milliseconds.
Output	Video_Unlock	unsigned short	Not sensible (always TRUE).

Function for Managing Project Resources and Common Properties

Video_SetWndSysProps (pFont, colFore, colBack)			
Unlock the display resource after exclusive access			
I/O	**Name**	**Type**	**Description**
Input	pFont	unsigned long	Address of font for printing text in title bar (the font must be added with Video_AddFont function).
	colFore	unsigned long	Text color of Title Bar.
	colBack	unsigned long	Background color of Title Bar.
Output	Video_ SetWndSysProps	unsigned char	Not sensible (always TRUE).

Video_SetEditKey (id, code)			
Set key-code for editing functions			
I/O	**Name**	**Type**	**Description**
Input	id	unsigned char	Identifier of editing function (see. Enumerative table, page 381).
	code	unsigned char	Key code associated with editing function.
Output	Video_ SetEditKey	unsigned char	Not sensible (always TRUE).

Video_AddFont (pFont, charLen, charHei, offs)			
Publish a new font in HMI run-time			
I/O	**Name**	**Type**	**Description**
Input	pFont	unsigned long	Address of first byte of font.
	charLen	unsigned long	Character width of font (#pixel).
	charHei	unsigned long	Character height of font (#pixel).
	offs	unsigned long	Byte offset of a font that starts with ASCII 0x00 (subset of characters).
Output	Video_AddFont	unsigned char	TRUE if successful, FALSE otherwise.

Video_AddFontUnicode (pFont, charLen, charHei)			
Publish a new unicode font in HMI run-time			
I/O	**Name**	**Type**	**Description**
Input	pFont	unsigned long	Address of first byte of font.
	charLen	unsigned long	Character width of font (#pixel).
	charHei	unsigned long	Character height of font (#pixel).
Output	Video_ AddFontUnicode	unsigned char	TRUE if successful, FALSE otherwise.

Video_LoadLanguage (pResStrings, pEnums)			
Load strings and enumeratives of any language			
I/O	**Name**	**Type**	**Description**
Input	pResStrings	unsigned long	Address of first resources string for current language.
	pEnums	unsigned long	Address of first resources string for current language.
Output	Video_ LoadLanguage	unsigned char	TRUE if successful, FALSE otherwise.

Video_DrawFrames (left, top, right, bottom, colBack, fBar, pTitle, fResStr, fSysBtn, style)			
Function for draw frame-set			
I/O	**Name**	**Type**	**Description**
Input	left	unsigned short	Width of left frame (#pixel).
	top	unsigned short	Height of top frame (#pixel).
	right	unsigned short	Width of right frame (#pixel).
	bottom	unsigned short	Height of bottom frame (#pixel).
	colBack	unsigned long	Background color.
	fBar	unsigned char	TRUE: shows title bar. FALSE: hides title-bar.
	pTitle	unsigned long	Text of title bar: NULL: No string in title.
	fResStr	unsigned char	TRUE: pTitle is a resource string. FALSE: pTitle is an address of constant string.

Video_DrawFrames (left, top, right, bottom, colBack, fBar, pTitle, fResStr, fSysBtn, style)			
Function for draw frame-set			
I/O	**Name**	**Type**	**Description**
	fSysBtn	unsigned char	TRUE: shows system. FALSE: hides system button.
	style	unsigned char	0: Flat 1: Raised 2: Sunken
Output	Video_ DrawFrames	unsigned char	Not sensible (always TRUE).

Functions for Operating with Pages

Video_InitPage (x1, y1, x2, y2, pTitle, wData)			
Show a page on display			
I/O	**Name**	**Type**	**Description**
Input	x1	unsigned short	Top-left ‘x coordinate’ edge relative to full page.
	y1	unsigned short	Top-left ‘y coordinate’ edge relative to full page.
	x2	unsigned short	Bottom-down ‘x coordinate’ edge relative to full page.
	y2	unsigned short	Bottom-down ‘y coordinate’ edge relative to full page.
	pTitle	unsigned long	Address of Text of title bar. NULL: no text in title bar.
	wData	unsigned short	**Feature declaration** bit 0...7: • 0: flat • 1: raised • 2: sunken bit 8: • 0: no title bar • 1: shows title bar bit 9: • 0: pTitle is an address of constant string • 1: pTitle is a resource string bit 10: • 0: no system button • 1: shows system button bit 11: • 0: window not modal • 1: modal window (sensible only for pop-ups windows)
Output	Video_InitPage	unsigned char	Not sensible (always TRUE).

Video_SetPageColors (colFore, colBack)			
Assign all colors for current page			
I/O	**Name**	**Type**	**Description**
Input	colFore	unsigned long	Color of the text of page.
	colBack	unsigned long	Background color of page.
Output	Video_ SetPageColors	unsigned char	TRUE if successful, FALSE otherwise.

Video_ClrScreen ()			
Delete entire display area and fill with background color defined with Video_SetPage-Colors			
I/O	**Name**	**Type**	**Description**
Output	Video_ClrScreen	unsigned char	TRUE if successful, FALSE otherwise.

Video_ClrRect (x1, y1, x2, y2)			
Delete only a portion of display and fill with background color defined with Video_Set-PageColors			
I/O	**Name**	**Type**	**Description**
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
Output	Video_ClrRect	unsigned char	TRUE if successful, FALSE otherwise.

Video_SetFont (fontPtr)			
Load a font as current font for drawing objects. To correctly execute this function, the font must be declared with Video_ AddFont			
I/O	**Name**	**Type**	**Description**
Input	fontPtr	unsigned long	Address of first byte of font.
Output	Video_SetFont	unsigned char	TRUE if successful, FALSE otherwise.

Video_SetColors (colForeTxt, colBackTxt, colForeSel, colBackSel)			
Assign the current colors for drawing objects			
I/O	**Name**	**Type**	**Description**
Input	colForeTxt	unsigned long	Text color.
	colBackTxt	unsigned long	Background color.
	colForeSel	unsigned long	Text color for selection.
	colBackSel	unsigned long	Background color for selection.
Output	Video_SetColors	unsigned char	TRUE if successful, FALSE otherwise.

Video_ResetMaps (res)			
Delete the maps saved for every object. The maps are created adding an object at once, with access mode kACS_INIT			
I/O	**Name**	**Type**	**Description**
Input	res	unsigned long	Reserved. Set 0.
Output	Video_ResetMaps	unsigned char	Not sensible (return input parameter res).

Function for Objects

Video_NextEdit (fRWOnly)			
Enable selection for next objects identified by Sel. Order attribute			
I/O	**Name**	**Type**	**Description**
Input	fRWOnly	unsigned char	Limit for selecting the next edit-box: FALSE: next edit-box must be selectable. TRUE: the next edit-box must be selectable and writable.
Output	Video_NextEdit	unsigned char	Handle of selected objects; if -1 the function has an error.

Video_PrevEdit (fRWOnly)			
Enable selection for previous objects identified by `Sel.Order` attribute			
I/O	**Name**	**Type**	**Description**
Input	`fRWOnly`	unsigned char	Limit for selecting the next edit-box: • FALSE: the next edit-box must be selectable. • TRUE: the next edit-box must be selectable and writable.
Output	`Video_PrevEdit`	unsigned char	Handle of selected objects; if -1 the function has an error.

Video_EnterEdit (wHnd)			
Enter edit-mode of an Edit Box otherwise execute the action for a button. The object holds the task until exits from edit-mode.			
I/O	**Name**	**Type**	**Description**
Input	`wHnd`	unsigned short	Handle of object that must be edited or execute his action.
Output	`Video_EnterEdit`	unsigned char	Return pressed key code for exiting edit-mode. If return -1 is an error only if the object is an edit-box.

Video_EnterEditSel (wHnd, onlySelect)			
Select object or enter edit-mode of an Edit box otherwise execute the action for a button. The object holds the task until exit from edit-mode.			
I/O	**Name**	**Type**	**Description**
Input	`wHnd`	unsigned short	Handle of object that must be edited or execute his action.
	`OnlySelect`	unsigned char	• FALSE: as `VideoEnterEdit()`. • TRUE: enables only the selection without entering edit-mode.
Output	`Video_EnterEditSel`	unsigned char	Return pressed key code for exiting edit-mode. If return -1 is an error only if the object is an edit-box.

Video_PushButton (wHnd)			
Enter press-mode for buttons. The object holds the task until exit from press-mode. This function is sensible only for touchscreen systems.			
I/O	**Name**	**Type**	**Description**
Input	`wHnd`	unsigned short	Handle of button.
Output	`Video_PushButton`	unsigned char	• TRUE: last pressure event was in button area. • FALSE: last pressure event was outside button area. -1: error.

Video_FirstLastEdit (rwReq, last)			
Return the handle of first or last selectable controls.			
I/O	**Name**	**Type**	**Description**
Input	`rwReq`	unsigned char	Boolean parameter. It indicates if the function checks for the objects that have read-write access mode.
	`last`	unsigned char	• TRUE: last selectable object. • FALSE: first selectable object.
Output	`Video_FirstLastEdit`	short	Handle of the object; -1 if errors or do not exist selectable objects.

Drawing Functions

Video_Line (x1, y1, x2, y2, pts, color)			
Draw a line			
I/O	**Name**	**Type**	**Description**
Input	`x1`	unsigned short	Top-left 'x coordinate' edge relative to full page.

Video_Line (x1, y1, x2, y2, pts, color)			
Draw a line			
I/O	**Name**	**Type**	**Description**
	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
	pts	unsigned char	Thickness.
	color	unsigned color	Line color.
Output	Video_ClrRect	unsigned char	TRUE if successful, FALSE otherwise.

Video_Rectangle (x1, y1, x2, y2, pts, transp, bordCol, fillCol)			
Draw a rectangle			
I/O	Name	Type	Description
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
	pts	unsigned char	Border thickness.
	transp	unsigned char	• TRUE: transparent square. • FALSE: solid square.
	bordCol	unsigned long	Border color.
	fillCol	unsigned long	Fill color. The value is not sensible if transp is TRUE.
Output	Video_Rectangle	unsigned char	TRUE if successful, FALSE otherwise.

Video_DrawBorder (style, x1, y1, x2, y2, pts, color)			
Draw a border outside the rectangle area			
I/O	Name	Type	Description
Input	style	unsigned char	0: flat. 1: raised. 2: sunken.
	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
	pts	unsigned char	Border thickness. It is sensible only if style = 0.
	color	unsigned char	Border color. It is sensible only if style = 0.
Output	Video_ DrawBorder	unsigned char	TRUE if successful, FALSE otherwise.

Video_DelBorder (style, x1, y1, x2, y2, pts)			
Delete a border outside the rectangle area. The color of fill is the page color assigned with Video_SetPageColors			
I/O	Name	Type	Description
Input	style	unsigned char	• 0: Flat • 1: Raised • 2: Sunken
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
Input	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
Input	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.

Input	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
Input	pts	unsigned char	Border thickness It's sensible only if style = 0
Output	Video_DelBorder	unsigned char	TRUE if successful, FALSE otherwise.

Video_PrintBitmap (ptrBmp, x, y)			
Print a bitmap coded with run-time HMI format			
I/O	Name	Type	Description
Input	ptrBmp	unsigned long	Address of first byte of bitmap.
	x	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y	unsigned short	Top-left 'y coordinate' edge relative to full page.
Output	Video_ PrintBitmap	unsigned char	Not sensible (always TRUE).

Video_DelBitmap (ptrBmp, x, y)			
Delete a bitmap where it is not transparent, coded with run-time HMI format			
I/O	Name	Type	Description
Input	ptrBmp	unsigned long	Address of first byte of bitmap.
Input	x	unsigned short	Top-left 'x coordinate' edge relative to full page.
Input	y	unsigned short	Top-left 'y coordinate' edge relative to full page.
Output	Video_DelBitmap	unsigned char	Not sensible (always TRUE).

Video_InitBmpTreeRefresh (x1, y1, x2, y2)			
Delete a bitmap where it is not transparent, coded with run-time HMI format			
I/O	Name	Type	Description
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
Input	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
Input	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
Input	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
Output	Video_DelBitmap	unsigned long	Address of invisible device context.

Video_EndBmpTreeRefresh (pDC, x1, y1, x2, y2)			
Restore original device context and copy the area from invisible context to display context			
I/O	Name	Type	Description
Input	pDC	unsigned short	Address of invisible device context.
Input	x1	unsigned short	Top-left 'x coordinate' edge relative to full page.
Input	y1	unsigned short	Top-left 'y coordinate' edge relative to full page.
Input	x2	unsigned short	Bottom-down 'x coordinate' edge relative to full page.
Input	y2	unsigned short	Bottom-down 'y coordinate' edge relative to full page.
Output	Video_ EndBmpTreeRe- fresh	unsigned long	Not sensible (always TRUE).

Functions for Text

Video_PrintStr (str, x, y)
Print a string using the current font set with SetFont and current colors set with Set-Colors()

I/O	Name	Type	Description
Input	str	char *	Text to print.
	x	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y	unsigned short	Top-left 'y coordinate' edge relative to full page.
Output	Video_PrintStr	unsigned char	Number of chars printed.

Video_PrintResStr(idRes, x, y)			
Print a resources string using the current font set with SetFont and current colors set with SetColors()			
I/O	Name	Type	Description
Input	idRes	unsigned short	Identifiers of resource.
	x	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y	unsigned short	Top-left 'y coordinate' edge relative to full page.
Output	Video_ PrintResStr	unsigned char	Number of chars printed.

Video_PrintNChar(str, accMode, x, y, nChar, format)			
Print at most nChar characters of a string, using the current font set with SetFont and current colors set with SetColors(). It uses also a format for drawing the text. If nChar is less than string length, it truncates the string; otherwise apply the alignment.			
I/O	Name	Type	Description
Input	str	char *	Text to print.
	accMode	unsigned char	kACS_PRINT: print with colForeTxt and colBackTxt colors. kACS_SELECT: print with colForeSel and colBackSel colors.
	x	unsigned short	Top-left 'x coordinate' edge relative to full page.
	y	unsigned short	Top-left 'y coordinate' edge relative to full page.
	nChar	unsigned char	Maximum number of chars to print.
	format	unsigned long	Alignment of text. It is sensible only if nChar > length of str: • 0x08 = right alignment; • 0x10 = center alignment; • 0x20 = left alignment.
Output	Video_ PrintResStr	unsigned char	Number of chars of truncated string.

Functions for Parameter Access

Video_GetParam(idxDevice, idxParam, subIdxParam, pVal, type)			
Read a variable at address idxParam from Modbus idxDevice and put the result to variable addressed by pointer pVal			
I/O	Name Wr	Type	Description
Input	idxDevice	unsigned char	Index of device connected.
	idxParam	unsigned short	Index of parameter.
	subIdxParam	unsigned char	Sub-index of parameter.
	pVal	unsigned long	Address of variable that contains the read value.
	type	unsigned char	Parameter type. Available values: tyBool, tySInt, tyUSInt, tyByte, tyInt, tyUInt, tyWord, tyDInt, tyUDInt, tyDWord, tyReal,tyString.
Output	Video_GetParam	unsigned short	Number of chars printed.

Video_SetParam(idxDevice, idxParam, subIdxParam, pVal, type)			
Write the value of a variable addressed by pointer pVal to the variable of address idxParam of Modbus idxDevice			
I/O	Name	Type	Description
Input	idxDevice	unsigned char	Index of device connected.
	idxParam	unsigned short	Index of parameter.
	subIdxParam	unsigned char	Sub-index of parameter.
	pVal	unsigned long	Address of variable that contains the value to write.
	type	unsigned char	Parameter type. Available values: tyBool, tySInt, tyUSInt, tyByte, tyInt, tyUInt, tyWord, tyDInt, tyUDInt, tyDWord, tyReal, tyString.
Output	Video_SetParam	unsigned short	Integer values: • 0 = successful; • 1 = index of parameter not found; • 2,8,9 = system errors; • 3 = type not valid; • 4 = read-only parameter; • 5 = cannot write now; • 6 = the value is less than the min value; • 7 = the value is more than the max value.

Functions for Events

Video_SendEvent(msgID, wParam)			
Send an event from code			
I/O	Name	Type	Description
Input	msgID	unsigned char	Available values: • kWM_NULL = no event; • kWM_KEY = key pressure; • kWM_MSG = open message; • kWM_SELECT = select an edit-box, a button; • kWM_PUSH = pressure on button.
	wParam	unsigned short	Event parameter. It has a different meaning according to msgID: if kWM_NULL = not sensible; if kWM_KEY = pressed key. For the key a constant value exists. The syntax is: kKEY_<key> Ex. LongLeft -> kKEY_LongLeft if kWM_MSG =ID of message page to open; if kWM_SELECT= handle of selected edit-box, button; if kWM_PUSH = handle of pressed button.
Output	Video_SendEvent	unsigned short	TRUE if successful, FALSE otherwise.

Video_GetEvent(dmy)			
Pop an event from queue			
I/O	Name	Type	Description
Input	dmy	unsigned char	Reserved. Set 0.
Output	Video_GetEvent	unsigned short	Double word with inside the encoding. 16 low bit = type of event: • kWM_NULL = no event; • kWM_KEY = key pressure; • kWM_MSG = open message; • kWM_SELECT = select an edit-box, a button; • kWM_PUSH = pressure on button. 16 high bit = event parameter: • if kWM_NULL = not sensible; • if kWM_KEY = pressed key; • if kWM_MSG = ID of message page to open; • if kWM_SELECT = handle of selected edit-box, button; • if kWM_PUSH = handle of pressed button.

Function Blocks for HMI

Function Blocks

Video_GetPageColors			
Get the page colors of the page where called			
I/O	**Name**	**Type**	**Description**
Local variables	—	—	—
Input variables	—	—	—
Output variables	color	word32	Text color in the page.
	back	word32	Background of the page.

Static01			
Text strings with variable visibility			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
Input variables	wHnd	word16	Handle of the object. Must be unique among static objects.
	x	word16	Top-left 'x coordinate' edge relative to full page.
	y	word16	Top-left 'y coordinate' edge relative to full page.
	accMode	byte	• kACS_IDLE = no effect; • kACS_INIT = first draw on display; • kACS_PRINT = update draw on display.
	fResStr	byte	Boolean value: • FALSE = pString is the address of string to draw; • TRUE = pString is the identifier of resource string.
	pString	word32	Text to draw. It is different according to fResStr.
	pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
	foreCol	word32	Text color.
	bckCol	word32	Background color.
	pVisVar	word32	Visibility. Available values: • FALSE = text not visible; • TRUE = text always visible; • var_addr = address of boolean variable.
	format	word16	Format for numeric values, encoded in 32 bit: • bit 3: 1 = right alignment • bit 4: 1 = center alignment • bit 5: 1 = left alignment
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0 and not pSelVar = 1 fixed.
	selBackCol	word32	Background color when object is selected. It is not sensible if pSelVar = 0 fixed.
	selForeCol	word32	Text color when selected. It is not sensible if pSelVar = 0 fixed.
	pRefrVar	word32	Variable for update: • FALSE = the object is redrawn only when the page is opening or when returning from child page; • TRUE = the object is always redrawn.
	pSelVar	word32	Selection flag for the object. Suggest if the object must uses {'selBackCol'} and {'selForeCol'}. Available values: • FALSE = object is never selected; • TRUE = object is always selected; • var_addr = address of boolean variable.
	numChars	word16	Number of max characters. 0 indicates that the string is drawn with the entire value of pString.
Output variables	—	—	—

Image			
Image object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
	memSel	byte	Selection status of the previous execution.
Input variables	wHnd	word16	Handle of the object. Must be unique among image objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	px1	word32	Address of variable for moving image on X-Axis. It is sensible only if floating = TRUE
	py1	word32	Address of variable for moving image on Y-Axis. It is sensible only if floating = TRUE
	type_x	byte	Type for px1. Available values: tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord. It is sensible only if floating = TRUE and px1 <> NULL
	type_y	byte	Type for py1. Available values: tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord. It is sensible only if floating = TRUE and py1 <> NULL
	dx	word16	Width (#pixel).
	dy	word16	Height (#pixel).
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	floating	byte	Position of object: • FALSE = docking • TRUE = floating
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0 and not pSelVar = 1 fixed.
	bordSelCol	word32	Border color for selected object. It is sensible when style = 0 and bordPts > 0 and not pSelVar = 0 fixed
	accMode	byte	• kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display • kACS_QUERY = request for updating output variables • kACS_BCKQUERY = request for updating output variables when the object is in background pages • kACS_DELETE = delete object
	pBmp	word32	Address of first byte of bitmap to view. It is not sensible if pSelBmp = 1 fixed.
	pSelBmp	word32	Address of first byte of bitmap to view when selected. It is not sensible if pSelBmp = 0 fixed.
	pSelVar	word32	Selection flag for the object. Suggest if the object must uses {'bordCol', 'pBmp'} or {'bordSelCol', 'pSelBmp'}. Available values: • FALSE = object is never selected • TRUE = object is always selected • var_addr = address of boolean variable
	pVisVar	word32	Flag of visibility. Available values: • FALSE = image not visible • TRUE = image always visible • var_addr = address of boolean variable
Output variables	reqRefr	byte	Request refresh, updated when the object is called with accMode = kACS_QUERY or accMode = kACS_BCKQUERY.

Image			
Image object			
I/O	**Name**	**Type**	**Description**
	abs_x1	word16	Top-left 'x coordinate' edge relative to full page obtained with the sum among ‘x1’ and px1’. The value is updated when the object is called with accMode = kACS_INIT or accMode = kACS_QUERY.
	abs_y1	word16	Top-left 'y coordinate' edge relative to full page obtained with the sum among ‘y1’ and ‘py1’. The value is updated when the object is called with accMode = kACS_INIT or accMode = kACS_QUERY.
	mem_x1	word16	Value read from abs_x1 when the object is called with accMode = kACS_INIT or accMode = kACS_PRINT.
	mem_y1	word16	Value read from abs_y1 when the object is called with accMode = kACS_INIT or accMode = kACS_PRINT.

Animation			
Animation object			
I/O	**Name**	**Type**	**Description**
Local variables	memBmp	word32	Address of bitmap of the previous execution
Input variables	wHnd	word16	Handle of the object. Must be unique among animation objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0 and not pSelVar = 1 fixed
	accMode	byte	• kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display
	pBmpArr	word32	Address of first image to view.
	pCaseArr	word32	Address of first element of selection.
	nArrEl	byte	Number of elements in image list.
	pBmpDef	word32	Address of bitmap to view pSelVar not in pCaseArr.
	pSelVar	word32	Address of variable for selection.
	type	byte	Type of pSelVar. Available values: tyBool; tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord.
	pVisVar	word32	Flag of visibility. Available values: • FALSE = image not visible • TRUE = image always visible • var_addr = address of boolean variable
Output variables	—	—	—

Button02			
Button object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
	memTransp	byte	Transparency status of the previous execution.

Button02			
Button object			
I/O	**Name**	**Type**	**Description**
	memSel	byte	Selection status of the previous execution.
Input variables	wHnd	word16	Handle of the object. Must be unique among buttons objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	fResStr	byte	Boolean value: • FALSE = pString is the address of string to draw • TRUE = pString is the identifier of resource string
	pText	word32	Text to draw on the button. It has different meaning according to fResStr. If this field is NULL, no text is drawn.
	pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color and text color. It is sensible only if style = 0 and bordPts > 0, or pString different as NULL, and not pSelVar = 1 fixed.
	fillCol	word32	Color of button area. It is sensible only if pTransp different as 1 fixed, and not pSelVar = 1 fixed.
	bordSelCol	word32	Border color and text color when selected. It is sensible only if style = 0 and bordPts > 0, or pString different as NULL, and not pSelVar = 0 fixed.
	fillSelCol	word32	Color of button area when selected. It is sensible only if pTransp different as 1 fixed, and not pSelVar = 0 fixed.
	accMode	byte	• kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display
	pVisVar	word32	Flag of visibility. Available values: • FALSE = image not visible • TRUE = image always visible • var_addr = address of boolean variable
	pTransp	word32	Flag of transparency. Available values: • FALSE = button always solid • TRUE = button always transparent • var_addr = address of boolean variable
	pPressVar	word32	Address of a boolean variable. • Pressed button= *pPressVar = TRUE • Released button= *pPressVar = FALSE If the field is NULL there is no variable.
	pSelVar	word32	Selection flag for the object. Suggest if the object must uses {'bordCol', 'fillCol'} or {'bordSelCol', 'fillSelCol'}. Available values: • FALSE = object is never selected • TRUE = object is always selected • var_addr = address of boolean variable
	format	word16	Format of numeric values encoded with 16 bit: • bit 4: 1 = right alignment • bit 5: 1 = center alignment

Button02			
Button object			
I/O	**Name**	**Type**	**Description**
			• bit 6: 1 = left alignment
	order	word16	Number for establishing a sequential selection.
Output variables	—	—	—

EditBox01			
Edit object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
	wHnd	word16	Handle of the object. Must be unique among edit-box objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
Input variables	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	foreCol	word32	Text color.
	bckCol	word32	Background color.
	foreSelCol	word32	Text color when selected. It is sensible only if pCanSel is not 0 fixed.
	bckSelCol	word32	Background color when selected. It is sensible only if pCanSel is not 0 constant.
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color. It is sensible when style = 0 bordPts> 0
	rw	byte	• FALSE = read-only mode • TRUE = read-write mode
	refr	byte	Request refresh: • FALSE = the object is redrawn only when the page is opening or return from child page • TRUE = the object is always redrawn
	pVar	word32	Address of variable or parameter according to format. It cannot be NULL. If it is a parameter is encoded in this way: • bit 0...7 = Subindex parameter • bit 8...23 = IPA parameter • bit 24...32 = Device address
	type	byte	Type of data. Available values: tyBool; tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord; tyReal
	pVarMin	word32	Min value for edit-box variable. If bit 16...17 (LSB) of field format contains 0 the limit is not set, if contains 1 is a constant limit, if contains 2 it is a variable limit.
	pVarMax	word32	Max value for edit-box variable. If bit 14...15 (LSB) of field format contains 0 the limit is not set, if contains 1 is a constant limit, if contains 2 it's a variable limit.
	enumId	int16	Identifier of enumerative. If 0 no enumerative associated with this field exists.

EditBox01			
Edit object			
I/O	**Name**	**Type**	**Description**
	format	word32	View format encoded in 32 bit: • **bit 0** ◦ 0 = draw sign only if number is negative ◦ 1 = draw sign also for positive numbers • **bit 1** ◦ 0 = does not print most significant null digits ◦ 1 = draw zeroes on most significant null digits • **bit 2** ◦ 0 = ‘pVar’ is a variable ◦ 1 = ‘pVar’ is a parameter • **bit 3**: 1 = right alignment • **bit 4**: 1 = center alignment • **bit 5**: 1 = left alignment • **bit 10**: Hexadecimal format, with a...f lowercase • **bit 11**: Hexadecimal format, with A...F uppercase • **bit 14...15** ◦ 0 = no max limit ◦ 1 = constant max limit ◦ 2 = variable max limit • **bit 16...17** ◦ 0 = no min limit ◦ 1 = constant min limit ◦ 2 = variable min limit • **bit 24...26**: Precision (real numbers) • **bit 27...31**: Width
	pVisVar	word32	Flag of visibility. Available values: • FALSE = object not visible • TRUE = object always visible • var_addr = address of boolean variable
	pCanSel	word32	Available values: • FALSE = object not selected • TRUE = object always selected • var_addr = address of boolean variable
	order	byte	Number for establish a sequential selection.
	accMode	byte	• kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display • kACS_SELECT = update draw on display when selected • kACS_MODIFY = enter in editing mode
Output variables	outKey	char	Key code for exiting editing-mode.

TextBox			
Text box object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
	base	word16	Number of first line seen in object.
Input variables	wHnd	word16	Handle of the object. Must be unique among textbox objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	foreCol	byte	Text color.
	bckCol	byte	Background color.
	bordPts	byte	Border thickness It is sensible only if style = 0
	bordCol	byte	Border color. It is sensible when style = 0 bordPts > 0
	LineNr	byte	• FALSE = hide line number • TRUE = show line number
	rw	byte	• FALSE= read-only mode • TRUE= read-write mode
	pVar	word32	Address of string variable. It cannot be NULL.
	szpVar	word32	Size of pVar.
	pVisVar	word32	Flag of visibility. Available values: • FALSE= object not visible • TRUE= object always visible • var_addr= address of boolean variable
	order	byte	Number for establishing a sequential selection.
	accMode	byte	Access mode. Available values: • kACS_IDLE = no effect when selected • kACS_INIT = first draw on display • kACS_PRINT = update draw on display • kACS_SELECT = update draw on display • kACS_MODIFY = enter editing mode • kACS_SCROLLUP = scroll up one line • kACS_SCROLLDW = scroll down one line
	rqCursPos	word16	Char Index where move the cursor.
	rqCursRow	word16	Row to select.
	dispCurs	byte	• TRUE= the cursor is always visible even if it is not enabled editing mode • FALSE= the cursor is visible only if it is enabled editing mode
	dispRow	byte	• TRUE= the row selection is always visible even if it is not enabled editing mode • FALSE= the row selection is visible only if it is enabled editing mode
	bckSelCol	word32	Future developments.
	wParam	word32	Future developments.
	lParam	word32	Future developments.

TextBox			
Text box object			
I/O	**Name**	**Type**	**Description**
Output variables	outKey	char	Key code for exiting editing-mode.
	outCursPos	word16	Char index where there is the cursor.
	outCursRow	word16	Index of selected row.

Progress			
Progress bar object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution
	memVal	word32	Progress status of the previous execution
Input variables	wHnd	word16	Handle of the object. Must be unique among progress objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	style	byte	• 0 = flat • 1 = raised • 2 = sunken
	barCol	word32	Color of step bar.
	bckCol	word32	Background color.
	bordPts	byte	Border thickness. It is sensible only if style = 0
	bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0
	pVar	word32	Step variable. This is the filling percentage of bar in relation with the range assigned by pMin and pMax.
	type	byte	Type of pVar. Assigned values: tyBool; tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord
	pMin	word32	Min value for edit-box variable. If bit 0 (LSB) of field format contain 0 is a constant limit, if contain 1 it is a variable limit.
	pMax	word32	Min value for edit-box variable. If bit 1 (LSB) of field format contain 0 is a constant limit, if contain 1 it is a variable limit.
	format	word32	View format encoded in bit: • **bit 0**: ◦ 0 = pMin contains a constant value of Type ‘type’ ◦ 1 = pMin contains the address of variable of Type ‘type’ • **bit 1**: ◦ 0 = pMax contains a constant value of Type type ◦ 1 = pMax contains the address of variable of Type type • **bit 2**: ◦ 0 = horizontal orientation ◦ 1 = vertical orientation
	pVisVar	word32	Flag of visibility. Available values: • FALSE = object not visible • TRUE = object always visible • var_addr = address of boolean variable
	accMode	byte	Access mode. Available values: • kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display
Output variables	—	—	—

CustomCtrl			
Progress bar object			
I/O	**Name**	**Type**	**Description**
Local variables	memVis	byte	Visibility status of the previous execution.
	ptrFunct	word32	Address of function that implements Type wCtrlID.
	data0	word32	Local variable.
	data1	word32	Local variable.
	data2	word32	Local variable.
	data3	word32	Local variable.
Input variables	wHnd	word16	Handle of the object. Must be unique among custom control objects.
	x1	word16	Top-left 'x coordinate' edge relative to full page.
	y1	word16	Top-left 'y coordinate' edge relative to full page.
	x2	word16	Bottom-right 'x coordinate' edge relative to full page.
	y2	word16	Bottom-right 'y coordinate' edge relative to full page.
	wCtrlID	word16	Identifier of custom control.
	pVisVar	word32	Flag of visibility. Available values: • FALSE = object not visible • TRUE = object always visible • var_addr = address of boolean variable
	refr	byte	Request refresh: • FALSE = the object is redrawn only when the page is opening or return from child page • TRUE = the object is always redrawn
	accMode	byte	Access mode. Available values: • kACS_IDLE = no effect • kACS_INIT = first draw on display • kACS_PRINT = update draw on display The value greater than 200 can be used for custom purpose.
	wParam	word16	16 bit data without sign, used for custom purpose
	lParam	int32	32 bit data with sign, used for custom purpose
	rParam	float	32 bit real data with sign, used for custom purpose
Output variable	—	—	—

Commissioning

What's in This Part

The Commissioning Tab

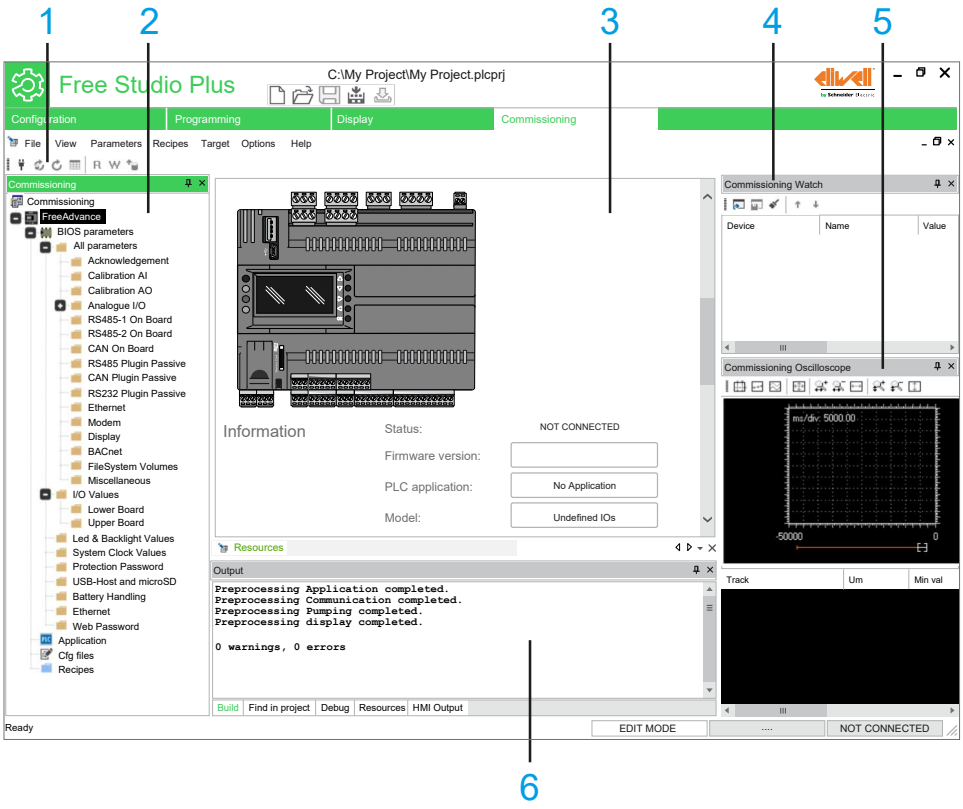
What's in This Chapter

Overview of the Commissioning Window

General Description

Commissioning is the entry point to deploy projects on real devices.

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



Item	Description	
1	Toolbar	This toolbar shows the tools in form of icons. For more information, refer to Toolbars, page 407.
2	**Commissioning** window	This window shows the configurable BIOS parameters of the target device. For more information, refer to Content of the Commissioning Window, page 409.
3	Editor window	This window allows you to edit the content of the current selection in **Commissioning** window.
4	**Commissioning 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 **Commissioning Watch** window, refer to Commissioning Watch Window, page 416

Item	Description	
5	**Commissioning Oscilloscope** window	This window enables you to plot the evolution of the values of a set of variables. Being an asynchronous tool, the Oscilloscope cannot establish synchronization of samples. For more information about how to use the **Commissioning Oscilloscope** window, refer to Commissioning Oscilloscope Window, page 417
6	**Output** window	This window shows the messages relating to the development of the project (file opening, reading/writing detected faults, status of connection to device, and so on). **NOTE:** The connection to the device is also visible in the status bar, page 35.

Menu Bar

Overview

The menu bar of **Commissioning** tab is composed of these menus:

- File, page 28
- View, page 33
- Parameters, page 30
- Recipes, page 31
- Target, page 32
- Options, page 29
- Help, page 28

Toolbar

Introduction

The toolbar appears at the top of the FREE Studio Plus window to provide access to frequently used functions.

For generalities of toolbars, refer to Toolbars description, page 34.

Commissioning Toolbar

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

Icon	Description	Shortcut
	**Connects to the target** Starts the communication with the device.	-
	**Toggle Auto refresh mode** Starts or stops (toggle) the Auto refresh mode.	-
	**Refresh page** Reloads the values for the current page.	-
	**Select all parameters** Select all the parameters from the currently displayed parameter table.	Ctrl+A
	**Read parameter** Read the value of the selected parameters.	Ctrl+R

Icon	Description	Shortcut
	**Write parameter** Write the value of the selected parameters.	Ctrl+W
	**Read all** Read all BIOS parameters from the target.	Ctrl+Shift+R

Managing Commissioning Elements

What's in This Chapter

Overview

Commissioning Window

Overview

Once the target device is connected, the **Commissioning** window allows you to:

- Read and write one or more BIOS parameters,
- Modify and restore the BIOS parameters to default,
- Read the current value of application variables (but not modify them).

  NOTE: These variables can be displayed by dragging them to the **Watch** and **Oscilloscope** windows.

Content of the Commissioning Window

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

Item	Icon	Description
Target device		Shows the picture of the target device and allows you to configure some settings. **NOTE:** For more information, refer to Target Device, page 412.
BIOS parameters		Shows the list of BIOS settings that you can modify.
Application		Shows the menu list. **NOTE:** For more information, refer to Target Menus, page 62.
Cfg files		Shows the CONNEC.PAR configuration file. **NOTE:** To download this file to the target device, right-click on **Cfg files** and click **Download files**.
Recipes		Shows the list of recipes. **NOTE:** You can add a parameter directly to a recipe by dragging and dropping.

NOTE: The **Commissioning** 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 %SW 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.

Read and Write BIOS Parameters

Overview

The term “parameter” refers to any resource present on the target device: parameters, I/O, and variables of the application.

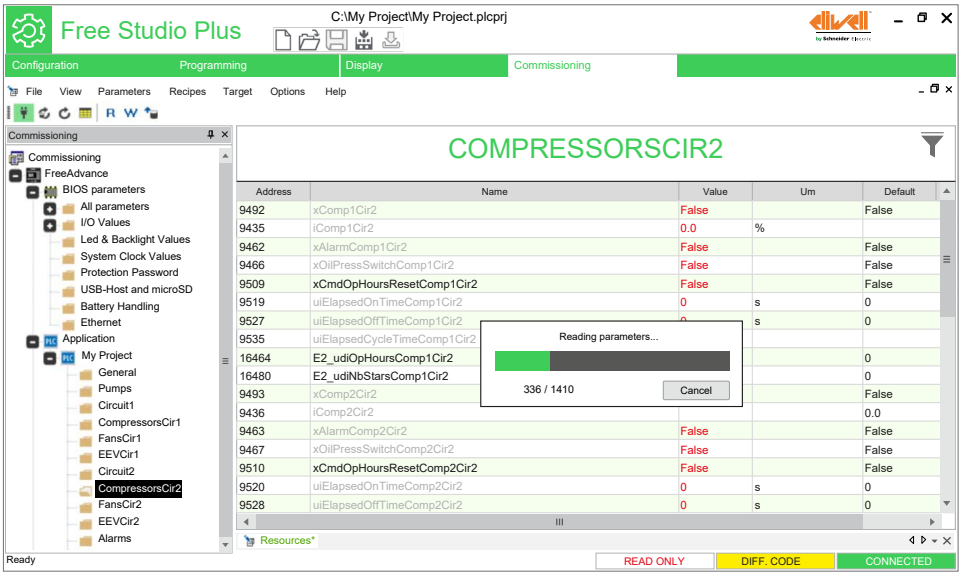
By default, the project contains a list of the BIOS parameters and I/O values with their respective default settings.

Read Parameters

From the target device, you can read the parameters.

To read a parameter, select it and click the **R** icon.

To read the parameters, click **Parameters > Read all**. The operation takes a few seconds.



The default value of the analog inputs and outputs is 0.0. The digital inputs and outputs are set to FALSE.

Write Parameters

You can download the parameters to the target device, either with their default values or with modified values inserted by you.

To write a parameter, select it and click the **W** icon.

To download the parameters (local BIOS), click **Parameters > Select all** or click **Select all parameters** icon. The parameters are highlighted in yellow. Then, click **Parameters > Write selected** or click the **Write parameter** icon. The parameters are downloaded to the target device.

Downloading the parameters by clicking **Parameters > Write all** replaces the values present on the target device with the values listed in the **Value** column.

If you select **Parameters > Write all default values**, the default values including I/O is downloaded to the target device.

Color Rules of Values

The inputs are read-only and are shown in gray. See the following table of parameter values:

Color	Column	Description	Cases
Black	Value	Value aligned with default	Target already read
Blue	Default	Default column value different from value in value column	Target already read
Red	Value	Value NOT aligned with default	Device just opened value changed by you
Gray	Name	Read-only parameters	I/O values analog inputs AI digital inputs DI
Green	Name	Parameters not visible on target display (only for FREE Smart)	See visibility table

Target Device

Overview

Description

In the **Commissioning** 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.

These settings may differ depending on the target device chosen for the project.

General

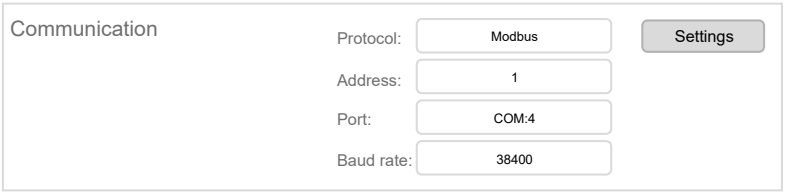
Overview

The **General** box displays the name of the project target device and the version number of its excepted firmware.

Communication

Overview

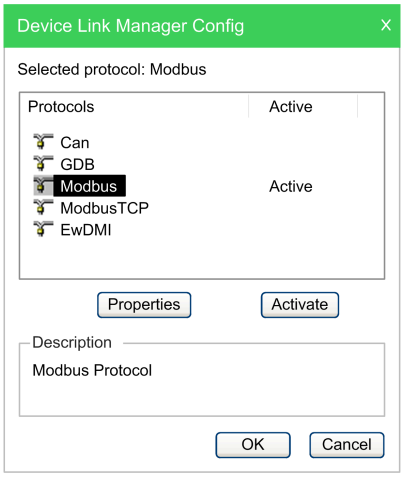
The **Communication** box displays information about the communication between the PC and the target device (protocol, address, port, and baud rate).



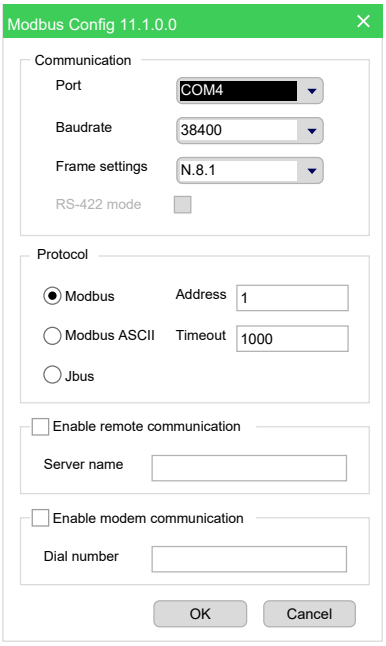
Communication Settings

To modify the communication settings, click the **Settings** button or click **Target > Communication settings**.

To modify the communication protocol, select the desired protocol in the **Protocols** list and click the **Activate** button.



The communication modalities can be modified by clicking the **Properties** button.

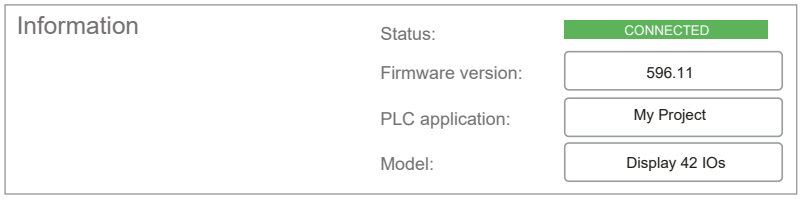


For more information on how to establish communication between FREE Studio Plus and the target device, refer to Setting Up the Communication, page 176.

Information

Overview

The **Information** box displays the information of the device connected to the PC (firmware version, current device application, and model).

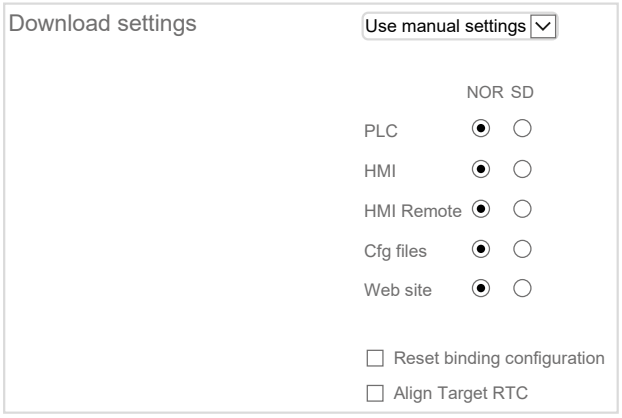


NOTE: Make sure the firmware version installed on the target device matches the firmware version used by the project. For more informations, refer to Other Operations, page 414.

Download Settings

Overview

The **Download settings** box displays where data is downloaded.



To modify the data download location, select from the drop-down list **Use manual settings** and choose:

- **NOR**: to download to internal memory,
- **SD**: to download to SD card.

Two additional options are available:

- **Reset binding configuration**: remove the bindings defined in the installer,
- **Align Target RTC**: align controller clock with PC clock.

Other Operations

Overview

The **Other operations** box allows you to perform the following actions:

Action	Goal
BIOS download	Update the BIOS of the target device using a *.bin file.
Open file browser	Explore the storage space of the device through a Windows Explorer.
Web site download	Downloads webpages created in the project by user.
Web site preview	Preview created webpages.
Generate XIF file	Create a *.xif file which needs to be manually transferred to the target device. **NOTE:** The XIF is a text-based file that defines the network image of the device to a network management tool (for example, Niagara, LonMaker, and so on). The XIF along with files termed device resource files (DRFs) are used by the management tools to work with the LON device.
Generate parameters files	Create parameters files Param.dat, Param.raw and Param.csv.
Generate USB files on target	Force target controller to generate Param.dat and Param.bin directly in an USB memory key connected to the port. **NOTE:** The controller must be powered to allow for this feature.

Action	Goal
Import parameters files	Allows to import a pre-generated parameters file in the target controller. **NOTE:** After import procedure the software asks to generate a recipe in the project.
Create USB programming files	Create a backup of the project for later restore.

Bios Download Feature

If an incorrect firmware is downloaded into a device, the device could be damaged.

Debugging

What's in This Chapter

Overview

Description

FREE Studio Plus provides several debugging tools, which help you to verify whether the application behaves as intended.

These debugging tools basically allow you to watch the value of selected variables while the PLC application is running.

Commissioning debugging tools are 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.

This chapter presents how to debug your application using debugging tools.

Commissioning Watch Window

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.

For more information about the **Commissioning Watch** window, refer to Watch Window, page 194.

> **NOTE:** The functions available in the **Programming Watch** window are only available for variables with Modbus address (**EEPROM Parameters**, **Status variables**, and **BIOS Parameters**).

Commissioning Oscilloscope Window

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.
- 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.

For more information about **Commissioning Oscilloscope** window, refer to Oscilloscope, page 200.

NOTE: The functions available in the **Programming Watch** window are only available for variables with Modbus address (**EEPROM Parameters**, **Status variables**, and **BIOS Parameters**).

Appendices

What’s in This Part

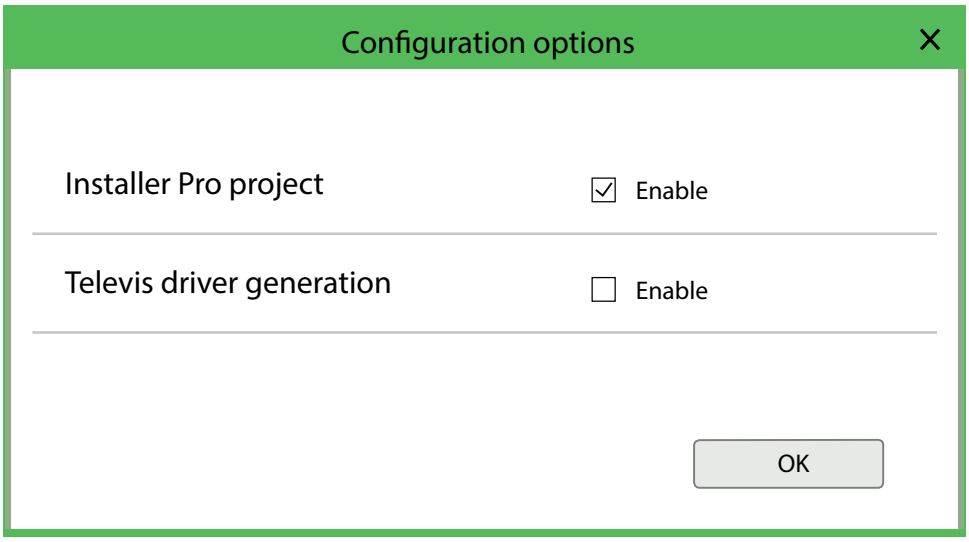
Installer Pro Project

What's in This Chapter

Overview

Installer Pro project is an additional feature which implements new functionalities in Free Studio Plus.

Installer Pro project is activated in the Configuration page by clicking on **Project > Options > Installer Pro project** in the menu toolbar.



NOTE: In order to enable/disable the Installer Pro project it is first necessary to remove all the expansions in project.

Compatibility Range

Installer Pro project is compatible with the following products:

Reference	Description
AV•126•0•I500	FREE Advance AV••••••5•500 Logic Controller
AV•126•060500	FREE Advance AV••••••6•500 Logic Controller
EWCM 9000 PRO (HF)	EWCM 9000 PRO (HF) Logic Controller

Reference	Description
EVE4200	FREE Expansion EVE4200
EVE1020000500	FREE Expansion EVE10200
EP4000000B0	EWCM Expansion EP4000

NOTE: All the I/O Expansions must be of the same type, otherwise a compilation error is generated.

Installer Pro Project Features

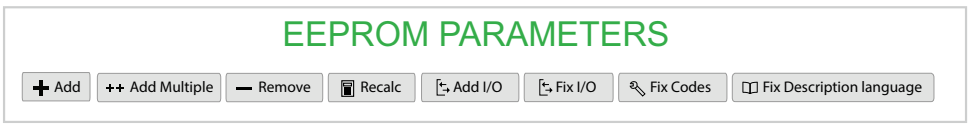
By enabling **Installer Pro Project** flag additional columns in the parameter grid are shown in **Commissioning** and **Installer** pages:

Column	Label	Description
1	Label	In the **Label** column can be filled the Label code, without spaces or special characters. If the cell is left empty the values will automatically be overwritten with the value in **Name** column.
2	Unit par Folder	These columns allow to organize parameters in easier viewing folders which has been previously created. Refer to PLC Parameters Navigation Tree, page 421
3	Unit par Sub Folder	
4	I/O Type	This column defines the type of input/output related to the selected parameter; it can be AI, AO, DI or DO. This definition is useful for the new feature **I/O Definition** Refer to I/O Definition, page 421 for more information on the **I/O Definition**.
5	Softscope	This column enables the related parameter to be recorded and viewed with the **Softscope** commissioning tool. Refer to Softscope, page 427.

Label	Unit par Folder	Unit par Sub Folder	IO Type	Softscope
01.002-Sbp	SYSTEM	GENERAL		False
01.003-LFr	SYSTEM	GENERAL		False
01.004-Ert	SYSTEM	GENERAL		False
01.005-rot	SYSTEM	GENERAL		False
01.006-rSE	SYSTEM	GENERAL		True
01.007-rdi	SYSTEM	GENERAL		False
01.008-ECS	SYSTEM	GENERAL		False
01.009-ECd	SYSTEM	GENERAL		True
01.010-Att	SYSTEM	GENERAL		False
01.011-En	SYSTEM	CONFIGURATION		False
01.013-tr1	SYSTEM	CONFIGURATION		False
01.014-Sr1	SYSTEM	CONFIGURATION		False
01.015-dr1	SYSTEM	CONFIGURATION		True

Furthermore **Installer Pro Project** flag enables additional button :

Button	Description
Add I/O	Adds an input/output variable. A couple of parameters with consecutive address are created.
Fix I/O	Allows to modify an existing input/output variable. The couple of parameters related to the variable must be selected in order to modify the variable.
Fix Codes	Allows to regenerate the **Description Code** of an inserted description if it is present in the Eliwell dictionary or in the custom dictionary (Custom Dictionaries, page 439).
Translate	The **Translate** button translates the **Description** column into the selected software language, where it is possible.

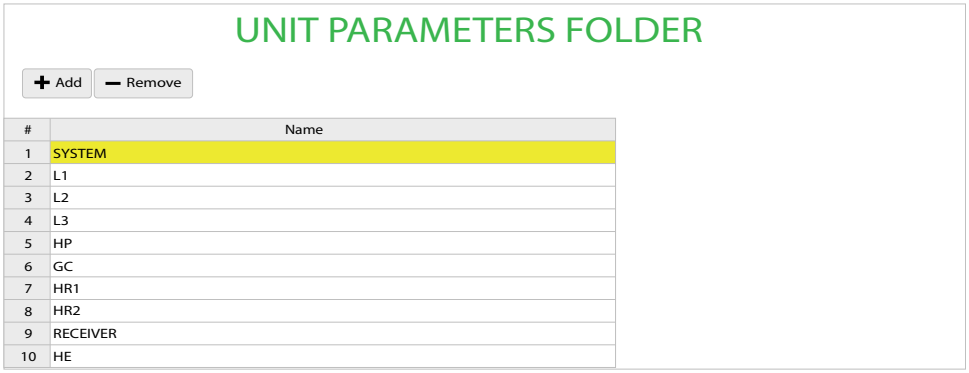


PLC Parameters Navigation Tree

Unit parameters Folder and Sub-folder allow the automatic generation in **Commissioning** page of a tree menu with selected and organized PLC parameters.

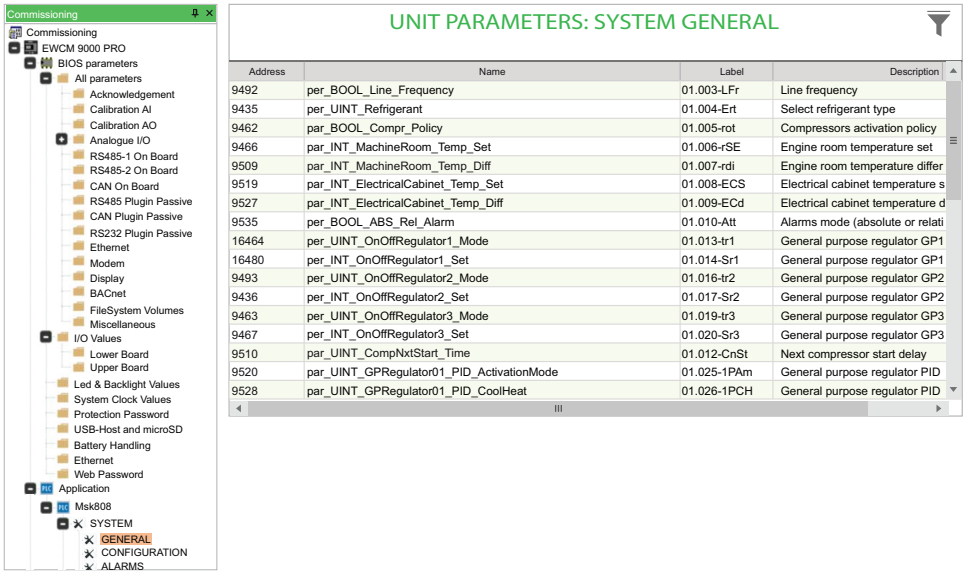
NOTE: Label and **Description** fields must be filled for a correct visualization.

Unit parameters Folder and Sub-folder values must be previously created.



#	Name
1	SYSTEM
2	L1
3	L2
4	L3
5	HP
6	GC
7	HR1
8	HR2
9	RECEIVER
10	HE

As in **Configuration** page parameters have been organized in Folder and Sub Folders, in **Commissioning** window these parameters can be easily viewed by clicking on the related parameters Folder. That grants a easier and faster modifying of project settings (for example Compressors parameters can be organized all together in a folder).

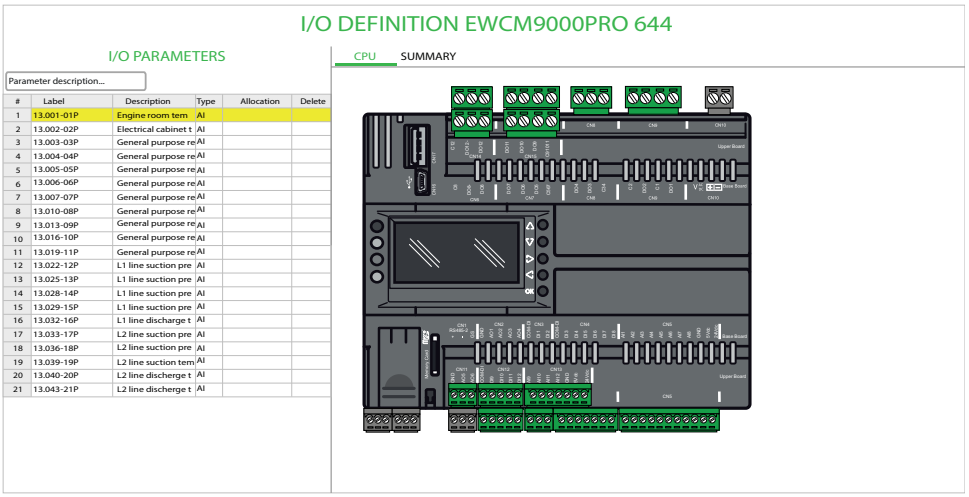


Address	Name	Label	Description
9492	per_BOOL_Line_Frequency	01.003-LFr	Line frequency
9435	per_UINT_Refrigerant	01.004-Ert	Select refrigerant type
9462	par_BOOL_Compr_Policy	01.005-rot	Compressors activation policy
9466	par_INT_MachineRoom_Temp_Set	01.006-rSE	Engine room temperature set
9509	par_INT_MachineRoom_Temp_Diff	01.007-rdi	Engine room temperature differ
9519	par_INT_ElectricalCabinet_Temp_Set	01.008-ECS	Electrical cabinet temperature s
9527	par_INT_ElectricalCabinet_Temp_Diff	01.009-ECd	Electrical cabinet temperature d
9535	per_BOOL_ABS_Rel_Alarm	01.010-Att	Alarms mode (absolute or relati
16464	per_UINT_OnOffRegulator1_Mode	01.013-tr1	General purpose regulator GP1
16480	per_INT_OnOffRegulator1_Set	01.014-Sr1	General purpose regulator GP1
9493	per_UINT_OnOffRegulator2_Mode	01.016-tr2	General purpose regulator GP2
9436	per_INT_OnOffRegulator2_Set	01.017-Sr2	General purpose regulator GP2
9463	per_UINT_OnOffRegulator3_Mode	01.019-tr3	General purpose regulator GP3
9467	per_INT_OnOffRegulator3_Set	01.020-Sr3	General purpose regulator GP3
9510	par_UINT_CompNxtStart_Time	01.012-CnSt	Next compressor start delay
9520	par_UINT_GPRegulator01_PID_ActivationMode	01.025-1PAm	General purpose regulator PID
9528	par_UINT_GPRegulator01_PID_CoolHeat	01.026-1PCH	General purpose regulator PID

I/O Definition

I/O Definition is a tool implemented to simplify the assignment of machine inputs and outputs.

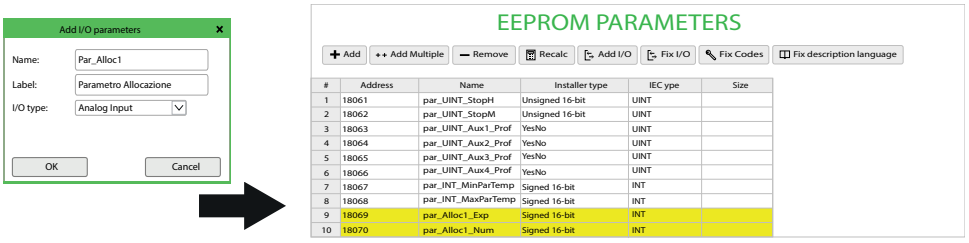
I/O Definition can be correctly used only if the allocation parameters have been previously declared.



New Project

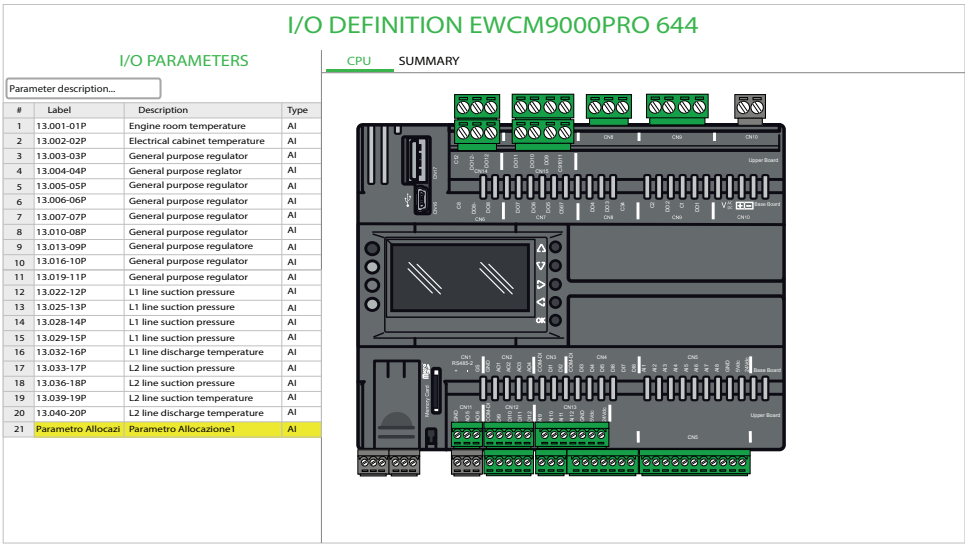
In case of a new project use the **Add I/O** button to add an input/output parameter. A couple of parameters will be created with consecutive address and I/O Type selected, for example:

- Par_Alloc1_Exp
- Par_Alloc1_Num



Fill in the empty fields in the parameter grid to correctly define parameters and compile the project.

After compiling the project the newly created logic I/O is added to the **I/O Definition** tool.

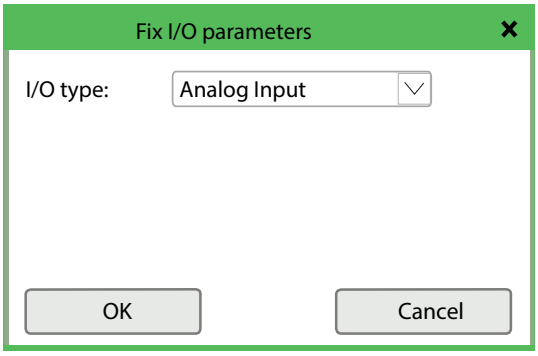


Existing Project

In case of an existing project to change a input/output variable it must be used the **Fix I/O** button.

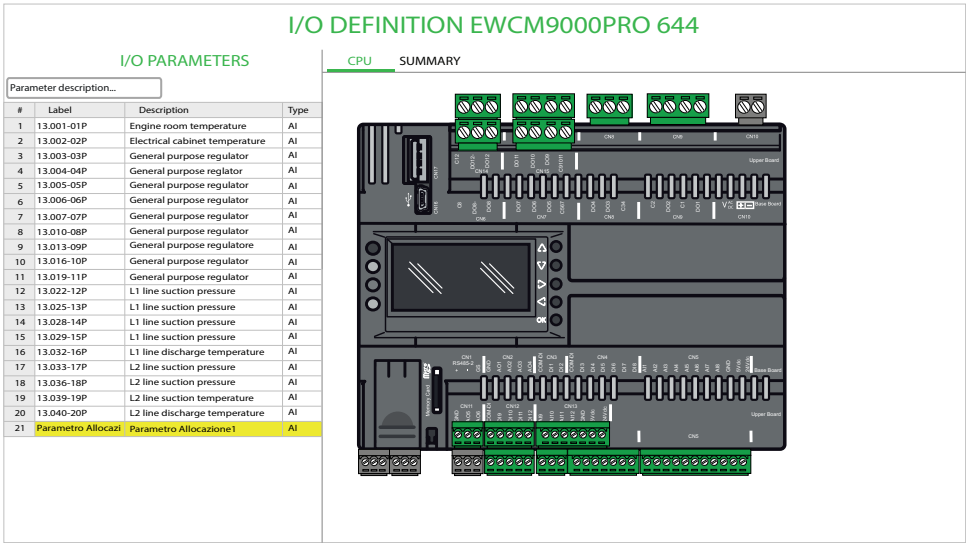
Select the couple of parameters (one ending with **_Exp** and one ending with **_Num**) and press the **Fix I/O** button. In the triggered window select the I/O type.

The selected type of I/O will be associated with the couple of parameters and it will be automatically filled the **I/O Type** column value.



Fill in the blank fields in the parameters grid, then recompile the project.

After compiling the project the modified logic I/O will be updated into the **I/O Definition** tool.

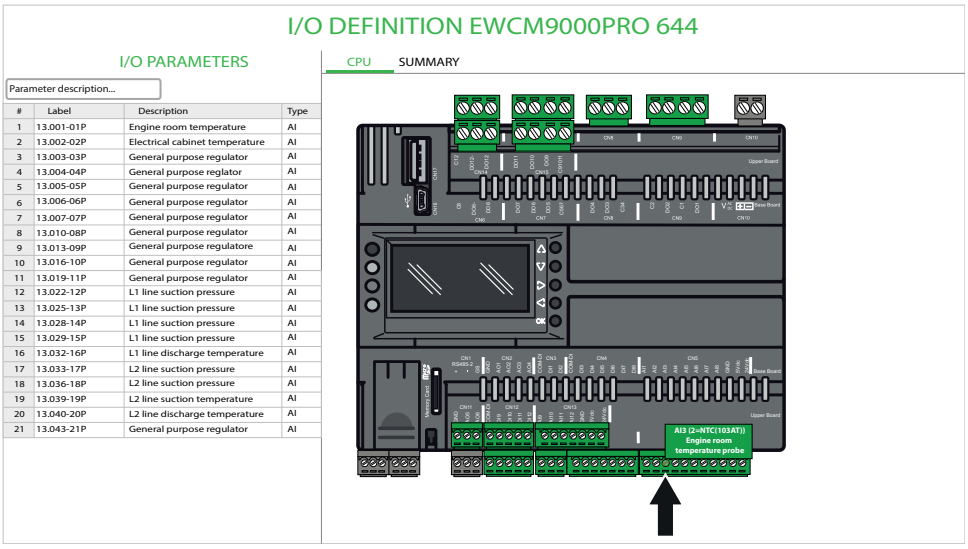


#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	Parametro Allocazi	Parametro Allocazione1	AI

Allocation of I/O

To allocate I/O parameters drag the logical I/O from the left grid onto the desired connector screw.

The tooltip indicates that the assignment has been done.



#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	13.043-21P	General purpose regulator	AI

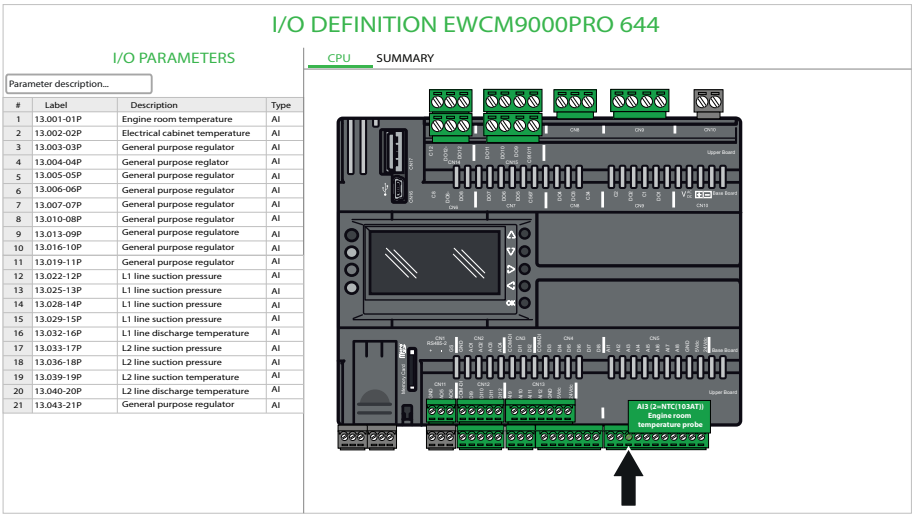
The pair of allocation parameters (**_Exp** and **_Num**) is updated automatically.

Double clicking on a AI screw displays the parameter BIOS CFG_AIX allocated, where it is possible to change its value.

Limitations

The **I/O Definitions** has the following limitations:

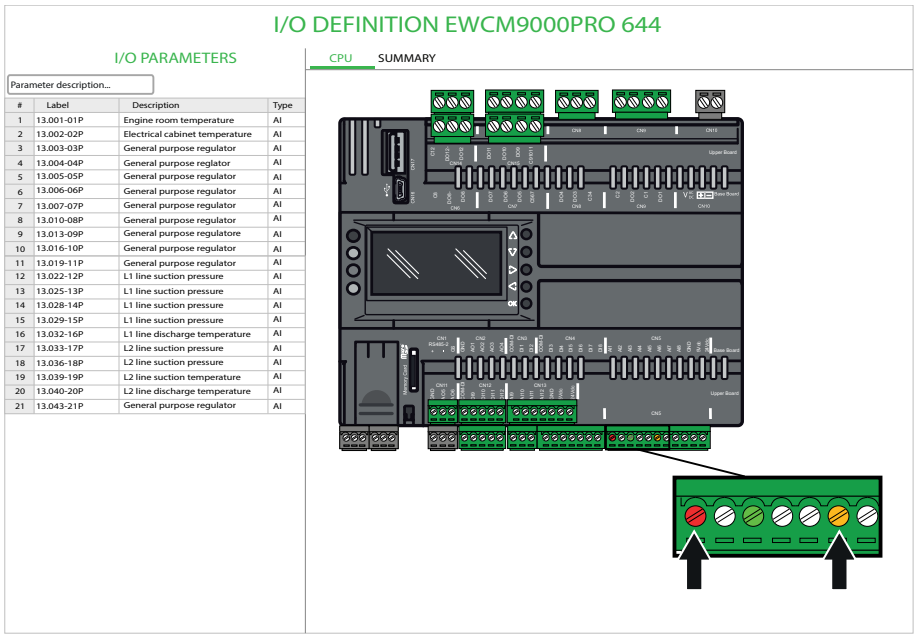
- It is necessary to drag an I/O logic from the grid to the screw in correspondence with an I/O of the same type. For example, you cannot drag an I/O declared DO onto a DI position.
- It is possible drag a DI to an AI if the corresponding BIOS parameter configures it as a DI.
- An AI can be dragged onto a screw corresponding to an AI as long as the AI is not configured as DI. For example, you can allocate a temperature probe even if the AI = 4-20 mA.



#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	13.043-21P	General purpose regulator	AI

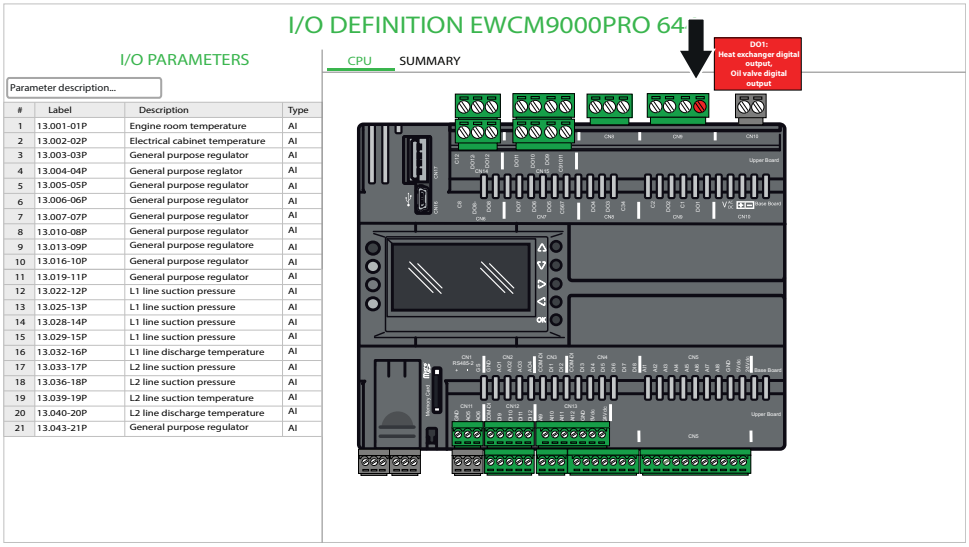
- Error conditions are indicated with the screw in red.

- Warning conditions are indicated with the screw in orange.



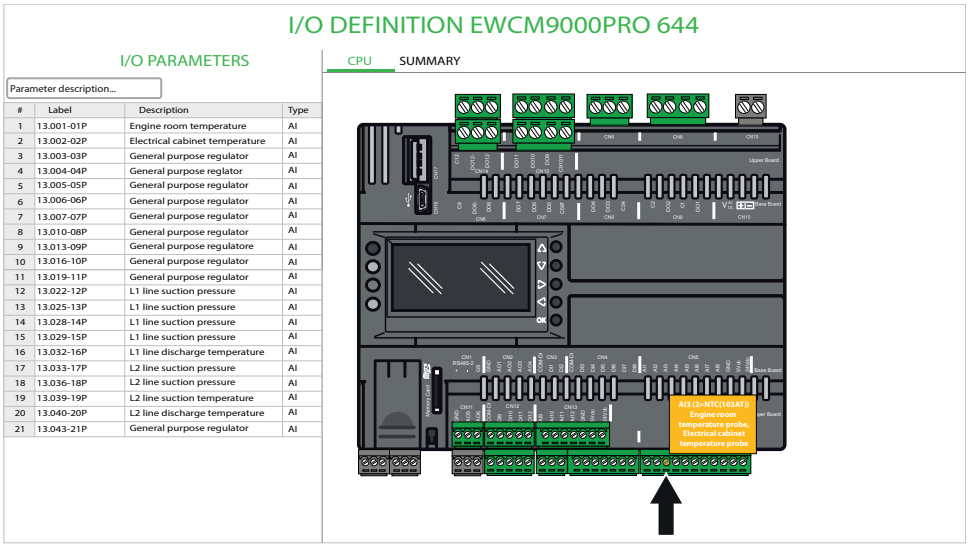
#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	13.043-21P	General purpose regulator	AI

If more logical outputs are assigned to the same DO, an error condition is signaled.



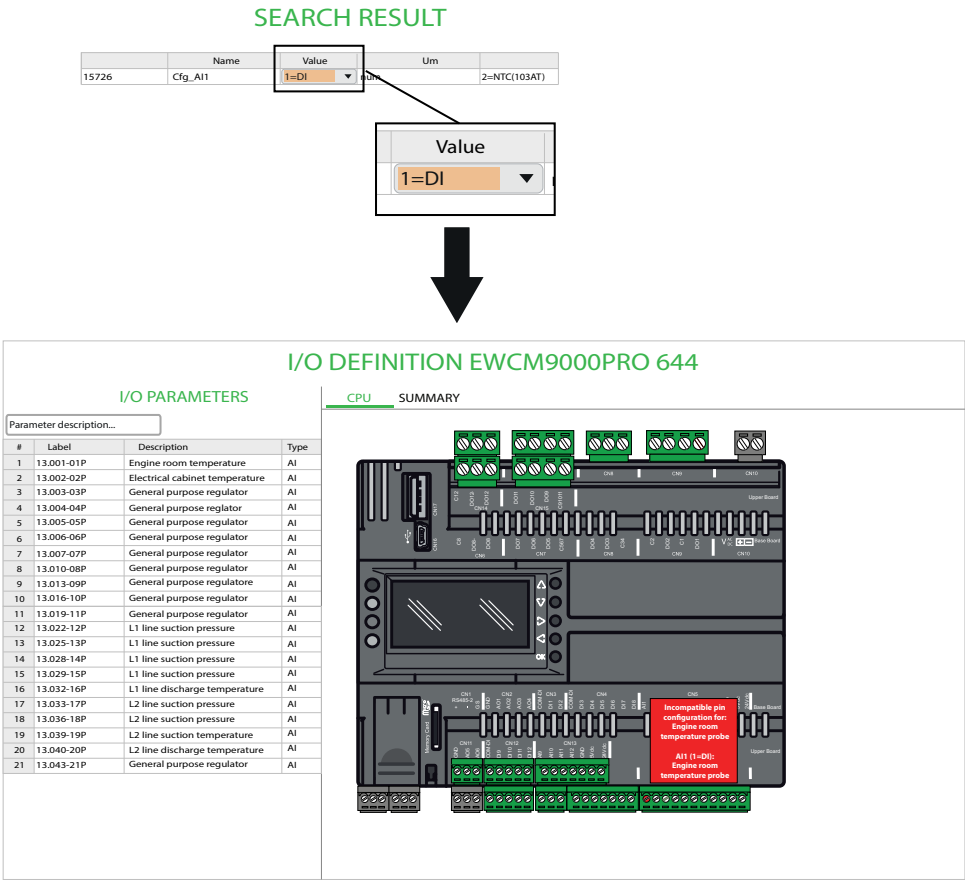
#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	13.043-21P	General purpose regulator	AI

If more logical inputs (AI or DI) are assigned to the same physical input, a warning condition is signaled.



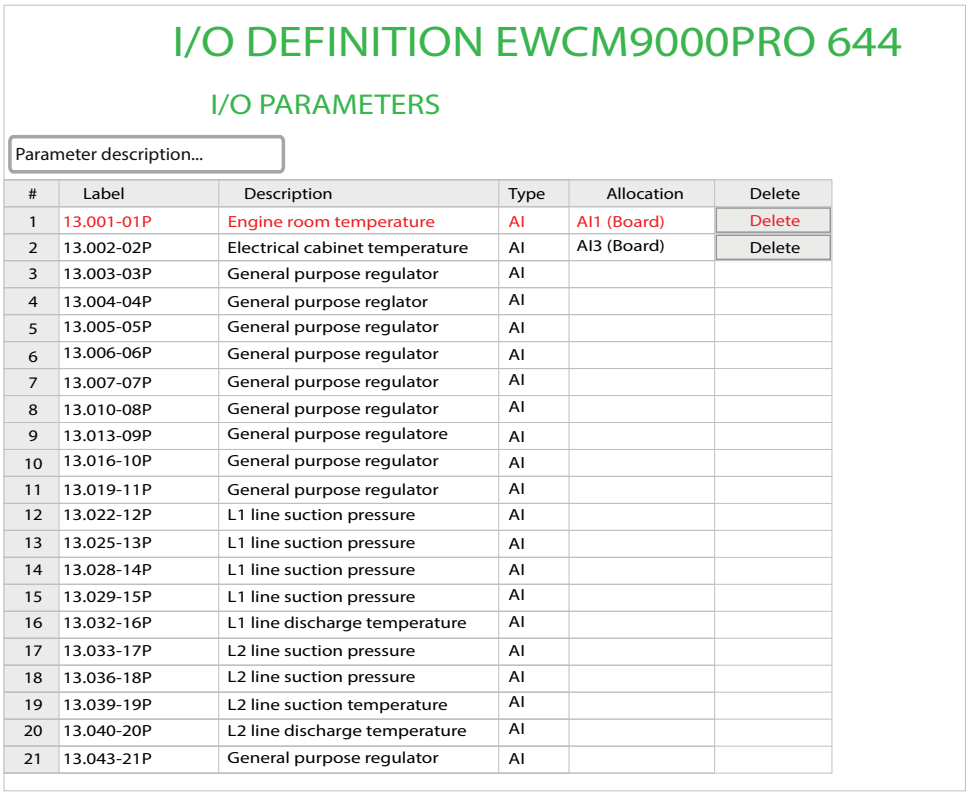
#	Label	Description	Type
1	13.001-01P	Engine room temperature	AI
2	13.002-02P	Electrical cabinet temperature	AI
3	13.003-03P	General purpose regulator	AI
4	13.004-04P	General purpose reglator	AI
5	13.005-05P	General purpose regulator	AI
6	13.006-06P	General purpose regulator	AI
7	13.007-07P	General purpose regulator	AI
8	13.010-08P	General purpose regulator	AI
9	13.013-09P	General purpose regulatore	AI
10	13.016-10P	General purpose regulator	AI
11	13.019-11P	General purpose regulator	AI
12	13.022-12P	L1 line suction pressure	AI
13	13.025-13P	L1 line suction pressure	AI
14	13.028-14P	L1 line suction pressure	AI
15	13.029-15P	L1 line suction pressure	AI
16	13.032-16P	L1 line discharge temperature	AI
17	13.033-17P	L2 line suction pressure	AI
18	13.036-18P	L2 line suction pressure	AI
19	13.039-19P	L2 line suction temperature	AI
20	13.040-20P	L2 line discharge temperature	AI
21	13.043-21P	General purpose regulator	AI

Another error condition can occur if user double-clicks a screw to modify the corresponding BIOS parameter in the parameter grid with an incompatible I/O type. For example an AI, previously configured as NTC, is changed into DI, when getting back to **I/O Definition** page it will be triggered an error message, shown by a red screw, and the tooltip highlights an incompatible pin configuration.



To delete an allocation:

- Double click on the Delete button corresponding to the I/O to be de-allocated.
- The screw automatically turns white.
- This operation must be done for each single I/O.



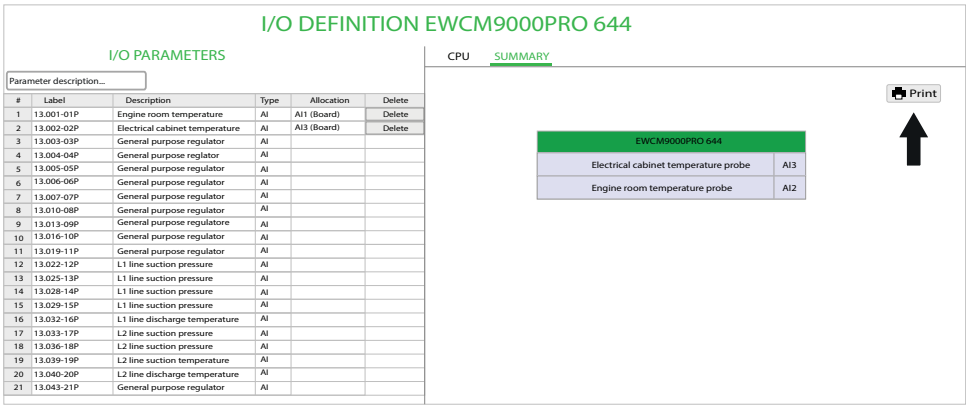
I/O DEFINITION EWCM9000PRO 644

I/O PARAMETERS

Parameter description...

#	Label	Description	Type	Allocation	Delete
1	13.001-01P	Engine room temperature	AI	AI1 (Board)	Delete
2	13.002-02P	Electrical cabinet temperature	AI	AI3 (Board)	Delete
3	13.003-03P	General purpose regulator	AI		
4	13.004-04P	General purpose reglator	AI		
5	13.005-05P	General purpose regulator	AI		
6	13.006-06P	General purpose regulator	AI		
7	13.007-07P	General purpose regulator	AI		
8	13.010-08P	General purpose regulator	AI		
9	13.013-09P	General purpose regulatore	AI		
10	13.016-10P	General purpose regulator	AI		
11	13.019-11P	General purpose regulator	AI		
12	13.022-12P	L1 line suction pressure	AI		
13	13.025-13P	L1 line suction pressure	AI		
14	13.028-14P	L1 line suction pressure	AI		
15	13.029-15P	L1 line suction pressure	AI		
16	13.032-16P	L1 line discharge temperature	AI		
17	13.033-17P	L2 line suction pressure	AI		
18	13.036-18P	L2 line suction pressure	AI		
19	13.039-19P	L2 line suction temperature	AI		
20	13.040-20P	L2 line discharge temperature	AI		
21	13.043-21P	General purpose regulator	AI		

In the summary page it is possible to print the summary of the allocation.



I/O DEFINITION EWCM9000PRO 644

I/O PARAMETERS

Parameter description...

#	Label	Description	Type	Allocation	Delete
1	13.001-01P	Engine room temperature	AI	AI1 (Board)	Delete
2	13.002-02P	Electrical cabinet temperature	AI	AI3 (Board)	Delete
3	13.003-03P	General purpose regulator	AI		
4	13.004-04P	General purpose reglator	AI		
5	13.005-05P	General purpose regulator	AI		
6	13.006-06P	General purpose regulator	AI		
7	13.007-07P	General purpose regulator	AI		
8	13.010-08P	General purpose regulator	AI		
9	13.013-09P	General purpose regulatore	AI		
10	13.016-10P	General purpose regulator	AI		
11	13.019-11P	General purpose regulator	AI		
12	13.022-12P	L1 line suction pressure	AI		
13	13.025-13P	L1 line suction pressure	AI		
14	13.028-14P	L1 line suction pressure	AI		
15	13.029-15P	L1 line suction pressure	AI		
16	13.032-16P	L1 line discharge temperature	AI		
17	13.033-17P	L2 line suction pressure	AI		
18	13.036-18P	L2 line suction pressure	AI		
19	13.039-19P	L2 line suction temperature	AI		
20	13.040-20P	L2 line discharge temperature	AI		
21	13.043-21P	General purpose regulator	AI		

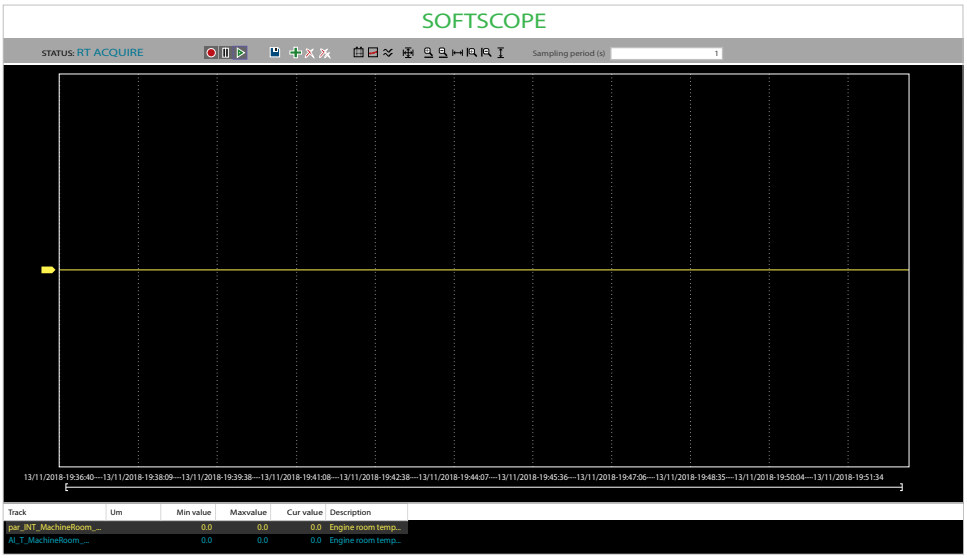
CPU SUMMARY

Print

EWCM9000PRO 644	
Electrical cabinet temperature probe	AI3
Engine room temperature probe	AI2

Softscope

This function is available only for BIOS version 644, **Installer Pro Project** flag enables the use of the **Softscope** tool.



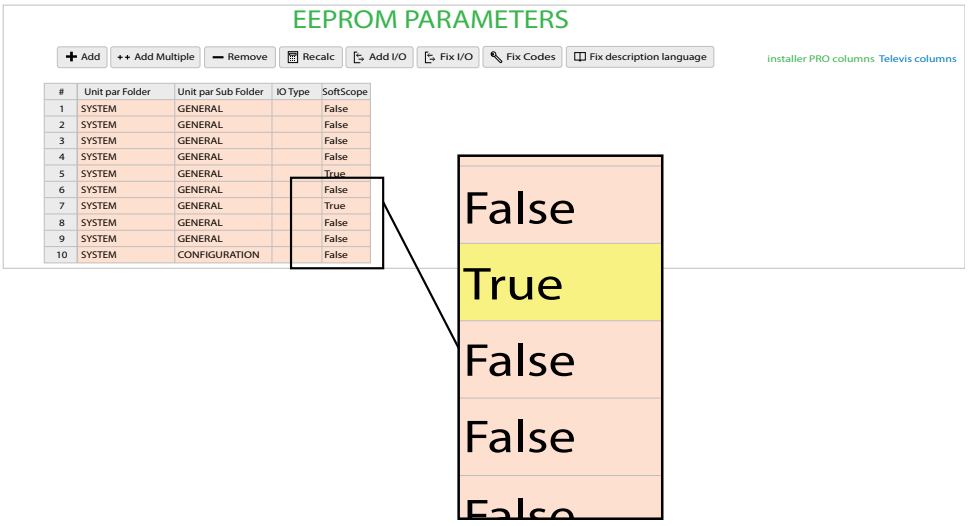
The **Softscope** tool allows to plot and record the evolution of the selected set of variables in real time for a period (maximum 48 hours). While the **Oscilloscope** tool reads the quantities asynchronously, **Softscope** reads the selected quantities with a set frequency, up to 10 tracks, at the same instant.

Recorded data can be saved as graphs and analyzed later, when the controller is offline.

Commissioning Softscope Window

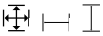
Description

The variables viewed and recorded through the **SoftScope** must be declared in the **SoftScope** Column as True in **Configuration** page.



The **Softscope** consists of three elements:

- The toolbar allows to control the **Softscope**.

Button	Function
	Start/stop data recording on the controller.
	Start/pause the display of currently acquired data in the main graph.
	Save the acquired data in a OSCX file.
	Add a track to be recorded.
	Remove the selected track from the list of tracks to be recorded.
	Remove all tracks from the list of tracks to be recorded.
	Show/hide more information on the recording interval.
	Show/hide the track acquisition points.
	Display all recorded tracks as separate in the graph.
	Scaling functions to resize the trends in the graph.
	Zooming functions
Sampling period (s)	Data sampling interval in seconds

- 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 **Softscope** is a table consisting of a row for each variable.

Softscope Communication Status

Status	Description
IDLE	FREE Studio Plus is ready to start date recording.
NOT CONNECTED	FREE Studio Plus is not connecting to the controller.
RT ACQUIRE	FREE Studio Plus is displaying the data recorded by the controller.
RECORDING	The connected controller is recording the data for the selected tracks.
UNDEFINED	FREE Studio Plus cannot connect to the controller.

Quick Start

To start using the **Softscope** it is necessary to:

1. Connect to the target.
2. Add an available track by clicking ✚.
3. Define the sample period and start recording the acquisition.

   NOTE: The target records the tracks autonomously in his own memory.

4. Press to view the tracks in real time. Press to suspend the tracks in real time. Press to start/stop recording data.
5. Save these settings to reuse the same set of measures in the next recording session.

It is possible to keep an acquisition running just by leaving the target in **RT_ ACQUIRE**. Then it is possible to close Free Studio Plus application for the necessary amount of time. While restarting Free Studio Plus and opening the initial project, the system remembers the activation of **RT_ACQUIRE** and it is possible to resume the tracks by pressing .

NOTE: Tracks can be saved into files (with *.OSCX extension); saved traces can be viewed with **Offline Graph Viewer** feature in Free Studio Plus.

NOTE: The storage capacity is 432.000 samples, that means 10 tracks every second for 12 hours, as well as 1 track every second for 5 days.

Automatic Generation of Expansion Code

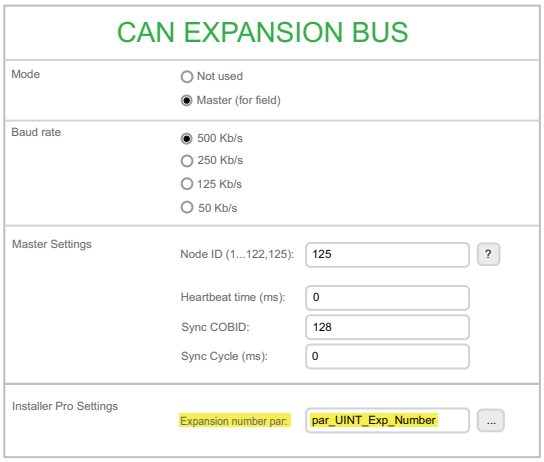
Installer Pro Project allows an easier addition of a CAN expansion to the project.

When an expansion is added a set of field variables linked to the AI, DI, AO, DO of the expansion is automatically generated.

Furthermore a program is automatically generated and added to the **INIT** task, though it is undetectable.

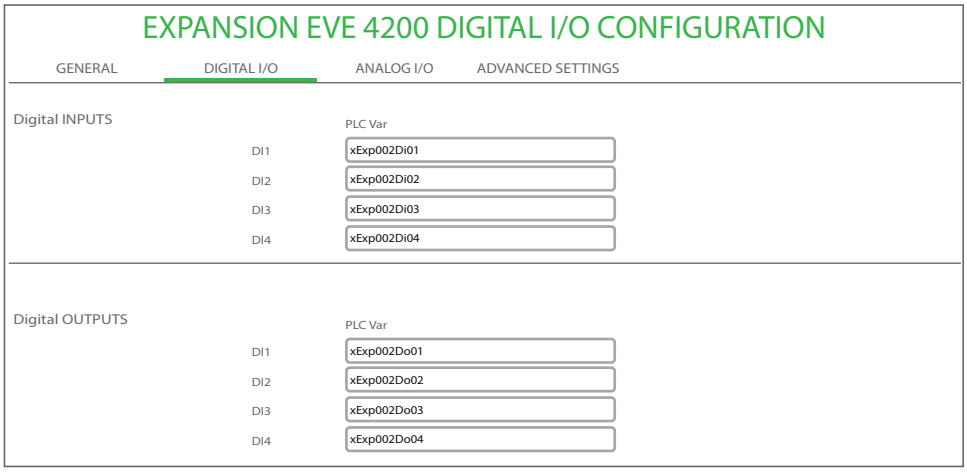
This program sends the values of the PLC parameters **CFG_AIx** and **CFG_AOx** to the expansions.

NOTE: It must be created a new parameter which will state the number of expansions required. This parameter is limited by the number of physical expansions in the CAN network, therefore it cannot be greater than the number of expansions. There can be up to 12 expansions. The CAN address of the expansion is automatically assigned.



To add an expansion follow the same procedures as in Using an Expansion Module as CAN Expansion Bus Field Slave, page 69, except for the configuration of **DIGITAL I/O** and **ANALOG I/O**.

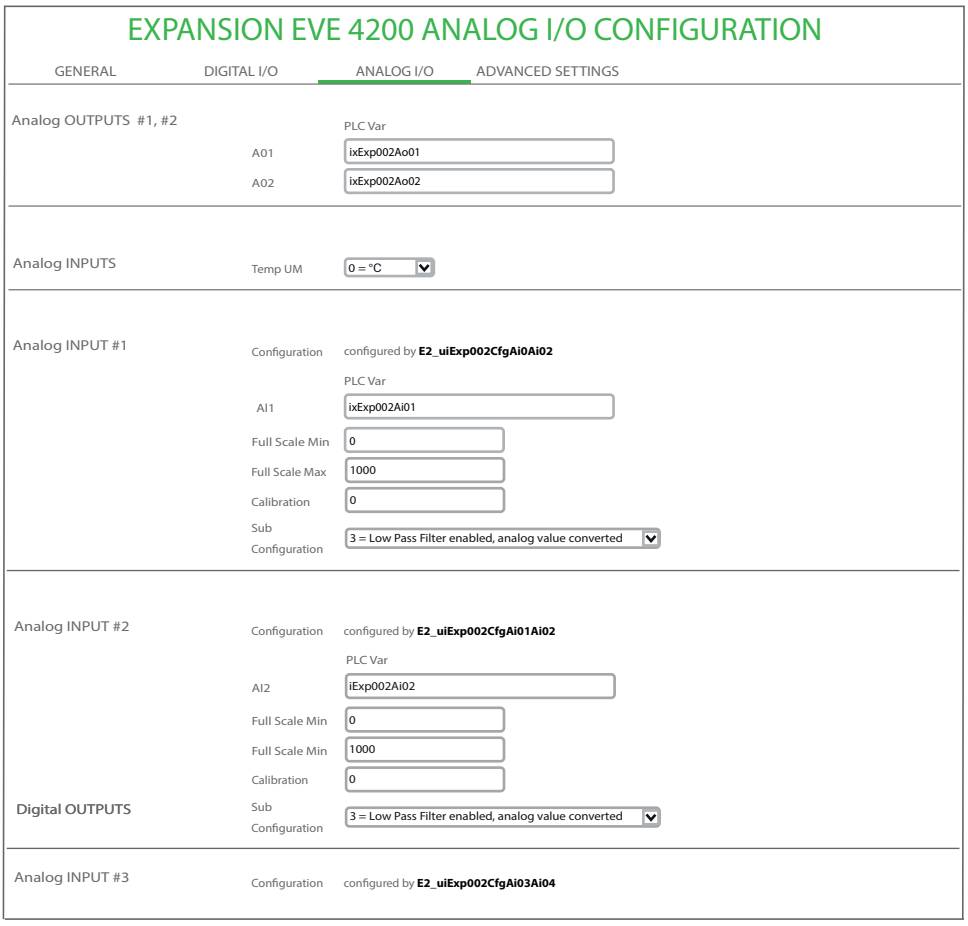
In the new **DIGITAL I/O** tab are showed the field variables relating to DI and DO generated variables.



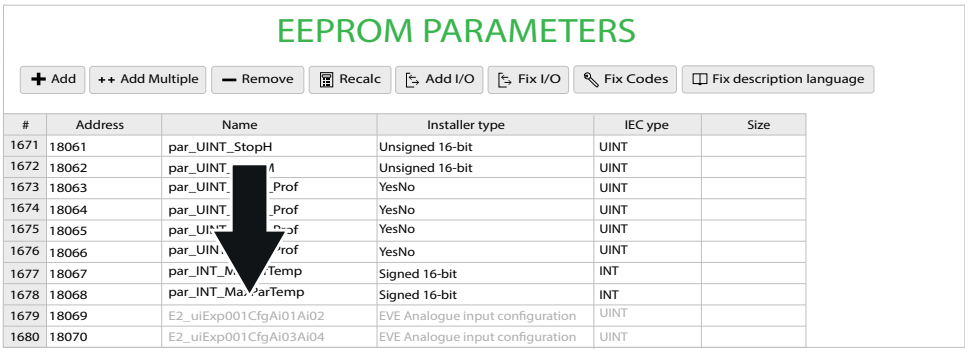
A set of **CFG_AIx** and **CFG_AOx** parameters will be automatically created, and **Folder** and **Sub Folder** fields are default assigned. **Folder** will be **BIOS** and **Sub Folder** will be **CONFIG AI** or **CONFIG AO**

Instead of indicating the value of the **CFG_AIx** or **CFG_AOx** parameter to be sent via CAN message to the expansion, will be indicated the name of the PLC parameter and field variable just created.

User just need to configure the variable settings linked to the automatically created expansion parameters which will exchange the information via CAN bus with the controller

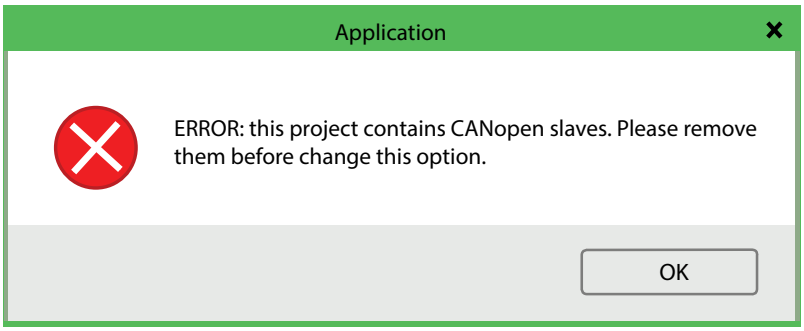


Default values are assigned to **Name**, **Description** and **Label** fields of auto-generated **CFG_AIx** and **CFG_AOx** parameters, while the auto-generated code is not visible.



NOTE:

- Removing an expansion removes alto auto-generated parameters and codes.
- All the expansions must be of the same type, otherwise a compilation error will be triggered.
- When one or more expansions are removed from the physical CAN network, the system asks to limit, if necessary, the PLC parameter that indicates the number of expansions.
- The **Installer Pro Project** flag cannot be changed if there are expansions in the CAN network. In order to disable or enable the **Installer Pro Project** flag it is first necessary to remove all the expansions.



Televis Driver Generation

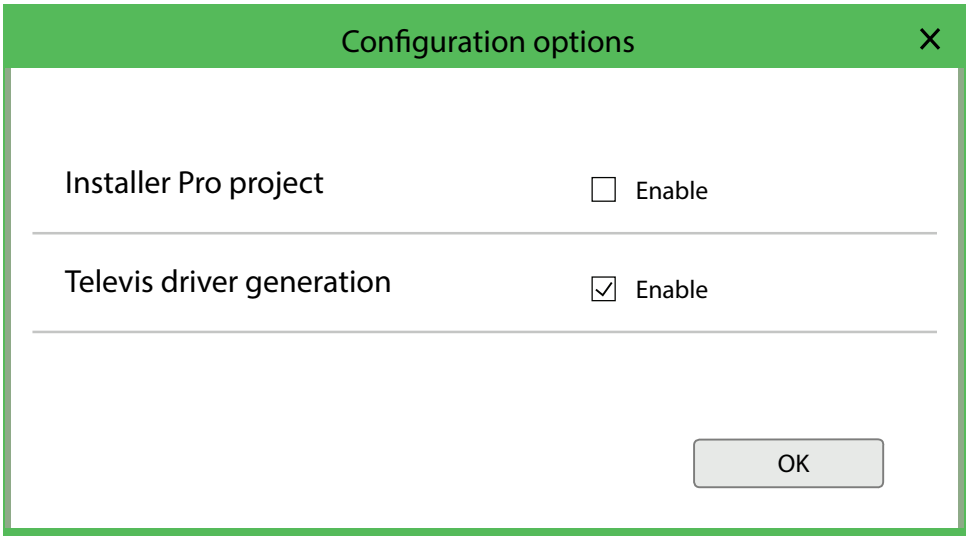
What's in This Chapter

Overview

With the activation of the **Televis driver generation** it is possible create the drivers required by supervisors, Televis Go and Televis Air, to automatically read the parameters.

To create the drivers **Developer > Build Televis Driver**.

Televis Driver Generation is activated in the Configuration page by clicking on **Project > Options > Televis driver generation** in the menu toolbar.



By enabling **Televis driver generation** flag additional columns in the parameter grid are shown in **Commissioning** and **Installer** pages:

Column	Description	
1	Progressive	In the **Description** column some strings have the symbol {0}. If one of these descriptions is chosen, the symbol is replaced by the supervisors with the number written in the **Progressive** column.
2	Description Code	The **Code Description** column cannot be changed. It represents the identification code of the string in the **Description** column.
3	Label	In the **Label** column can be filled the Label code, without spaces or special characters. If the cell is left empty the values will automatically be overwritten with the value in **Name** column.
4	Groups	The **Groups** column allows to insert groups to which the parameter belongs. It is accessed via a modal window. One or more groups can be entered and they can be selected from the groups of the Eliwell dictionary and from the strings of the custom dictionary (refer to Custom Dictionaries, page 439).
5	Groups Code	The Groups Code column cannot be changed. It represents the identification code of the string in the Groups column.
6	Measurement Unit	Values in the **Measurement Unit** column can only be filled by the related modal window. It can be a fixed unit of measure that can be chosen between the units of measure in the Eliwell dictionary and between the strings of the custom dictionary, or it can be a variable unit of measure conditional on the value of another variable. If the the value in the column is left empty it will automatically be overwritten by **Unit** column value, if present, otherwise it will just be "num".
7	Measurement Unit Code	The **Measurement Unit Code** cannot be changed. It represents the identification code of the string in **Measurement unit** column.
8	Visibility	Values in the **Visibility** column can be True or False. If True, the parameter is exported to the Televis driver, otherwise if False, the parameter is not exported to the Televis driver.

#	Progressive	Description Code	Label	Groups	Groups Code	Meausement Unit	Meausement Unit Code	Visibility
1		INO00184	01.002-SbP	Impianto;Impianto - funzion	PGR00166;PGR00196	num	FIX(VMU00020)	True
2		STA00400	01.003-LFr	Impianto;Impianto - funzion	PGR00166;PGR00196	HZ	FIX(VMU00010)	True
3	1	STA40196	01.004-Ert	Impianto;Impianto - funzion	PGR00166;PGR00196	num	FIX(VMU00020)	True
4		AIM00430	01.005-rot	Impianto;Impianto - funzion	PGR00166;PGR00196	num	FIX(VMU00020)	True
5		INO00181	01.006-rSE	Impianto;Impianto - funzion	PGR00166;PGR00196	INT(VMU00000,{1	INT(VMU00000,{1=VMU00	True
6		STI00314	01.007-rdi	Impianto;Impianto - funzion	PGR00166;PGR00196	INT(VMU00000,{1	INT(VMU00000,{1=VMU00	True
7		STA20124	01.008-ECS	Impianto;Impianto - funzion	PGR00166;PGR00196	INT(VMU00000,{1	INT(VMU00000,{1=VMU00	True
8		AIS00421	01.009-ECd	Impianto;Impianto - funzion	PGR00166;PGR00196	INT(VMU00000,{1	INT(VMU00000,{1=VMU00	True
9		AFE00178	01.010-Att	Impianto;Impianto - funzion	PGR00166;PGR00196	flag	FIX(VMU00021)	True
10		INA00159	01.011-En	Impianto;Impianto - funzion	PGR00166;PGR00196	num	FIX(VMU00020)	True

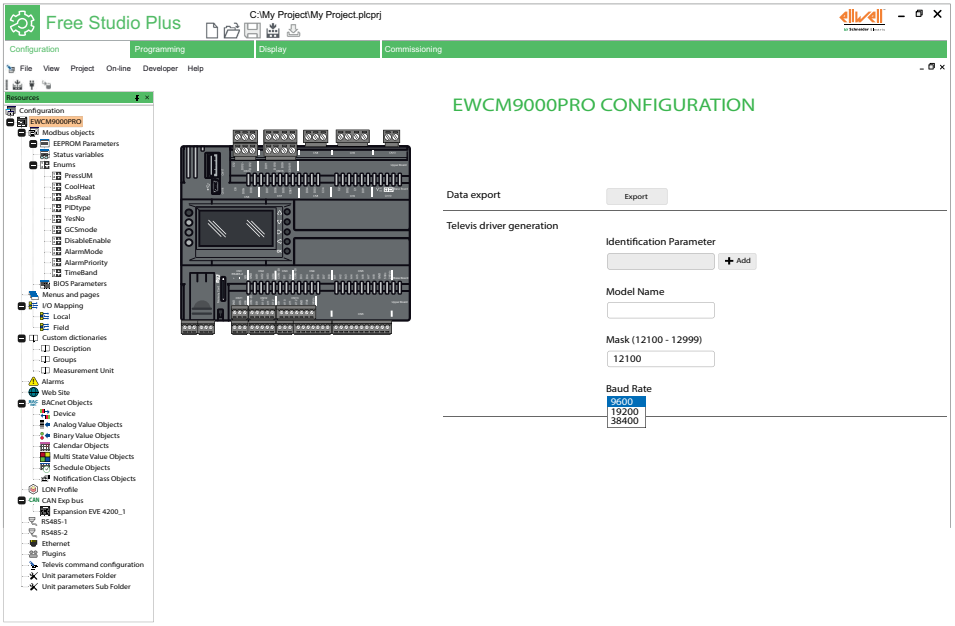
Furthermore **Televis Driver Generation** flag enables additional button :

Button	Description
Add I/O	Adds an input/output variable. A couple of parameters with consecutive address are created.
Fix I/O	Allows to modify an existing input/output variable. The couple of parameters related to the variable must be selected in order to modify the variable.
Fix Codes	Allows to regenerate the **Description Code** of an inserted description if it is present in the Eliwell dictionary or in the custom dictionary (Custom Dictionaries, page 439).
Translate	The **Translate** button translates the **Description**, **Groups** and **Measurement Unit** columns into the selected software language, where it is possible.

Configuration Page Layout

On the main node of the project:

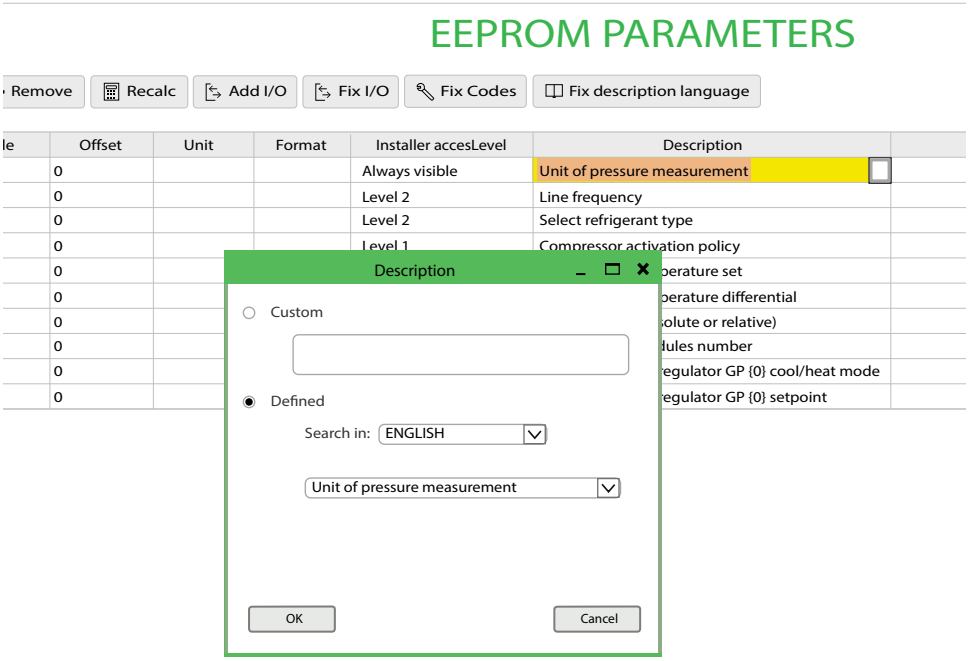
- Identification Parameter: must be present to generate the driver. It is added with the add button and the relative parameter is created among the status variables.
- Model Name: commercial name with which the application is recognized.
- Mask: the mask that identifies the Project. It must be between 12100 and 12999 to allocate a range of masks not superimposed on the official Eliwell masks.
- Baud Rate: it is possible to choose in a drop-down menu between 9600, 19200 or 38400.



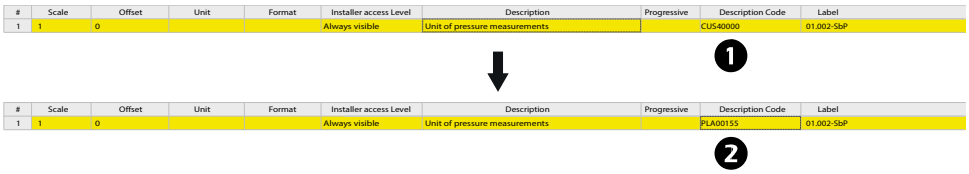
EEPROM Parameters

On the node **EEPROM Parameters** it is possible to fill in the information of each parameter.

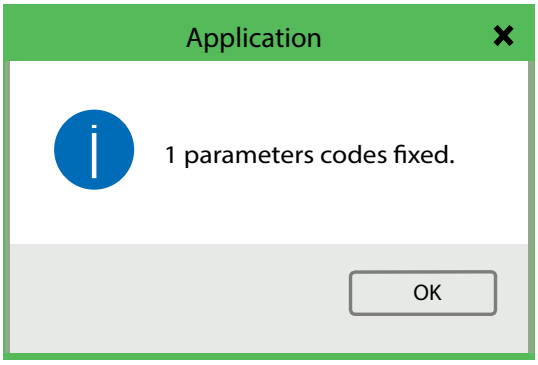
In the Description column must be entered the description of the parameter. It can be inserted by writing in the cell or by opening the modal window that allows you to choose between the strings of the Eliwell dictionary and the strings of the custom dictionary (refer to Custom Dictionaries, page 439). If the string is inserted by the custom dictionary, the description will be translated into various languages, otherwise if the string is inserted by writing the description will always be fixed as it is inserted.



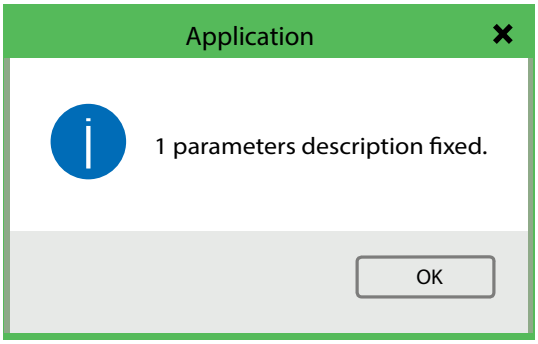
The **Code Description** column cannot be changed. It represents the identification code of the string in the **Description** column. If the string is manually compiled, the description code automatically becomes CUS40000 . If instead the string is entered by selecting from the modal window the relative description code of the chosen string is associated (for example PLA00155) ❷.



The **Fix Codes** button allows to regenerate the description code of an inserted description if it is present in the Eliwell dictionary or in the custom dictionary (Custom Dictionaries, page 439).



The **Translate** button translates the **Description**, **Groups** and **Measurement Unit** columns into the selected software language, where it is possible.



Usually the default language is set in English and so are **Description**, **Groups** and **Measurement Unit** columns, but in compatible controllers it is possible to change these columns language.

To change the parameter description language: **File > Options > Language**; once selected the language flag the **Accept** box to enable the selection.

NOTE: To make the language change effective it is necessary to close and reopen the project.

NOTE: The languages that have the specification "**(Parameters)**" indicate that the software will not be translated into language but will be by default in English, only the **Descriptions**, **Groups** and **Measurement Units** of parameters and variables will be translated.

Status Variable

On the node **Status Variable** it is possible to fill in the information of each variable.

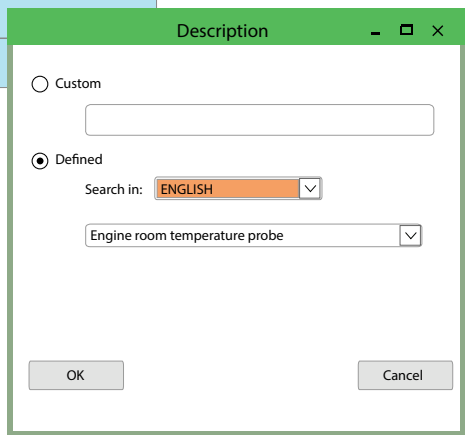
In the **Description** column must be entered the description of the variable. It can be inserted by writing in the cell or by opening the modal window that allows you to choose between the strings of the Eliwell dictionary and the strings of the custom dictionary (refer to Custom Dictionaries, page 439). If the string is inserted by the custom dictionary, the description will be translated into various languages, otherwise if the string is inserted by writing the description will always be fixed as it is inserted.

In the Description column some strings have the symbol {0}. If one of these descriptions is chosen, the symbol is replaced by the supervisors with the number written in the Progressive column .

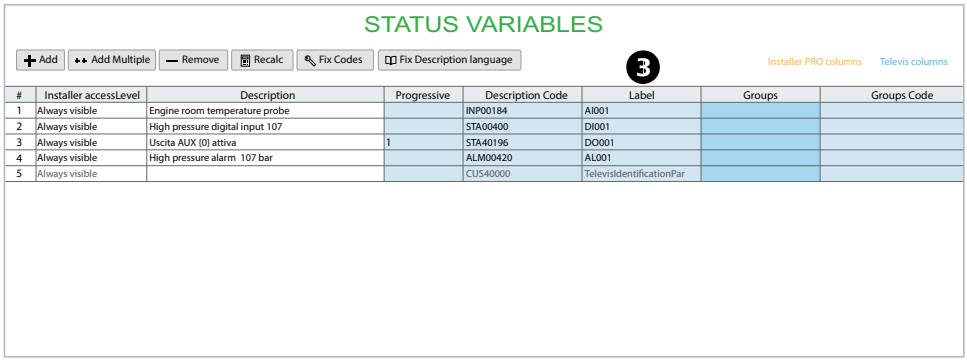
Description	❶ Progressive
Active AUX {0} Output	1

The **Code Description** column ❷ cannot be changed. It represents the identification code of the string in the **Description** column. If the string is entered manually, the description code automatically becomes **CUS40000**. If instead the string is entered by selecting from the modal window the relative description code of the chosen string is associated (for example **PLA00155**).

❷ Description Code
INP00184
CUS40000

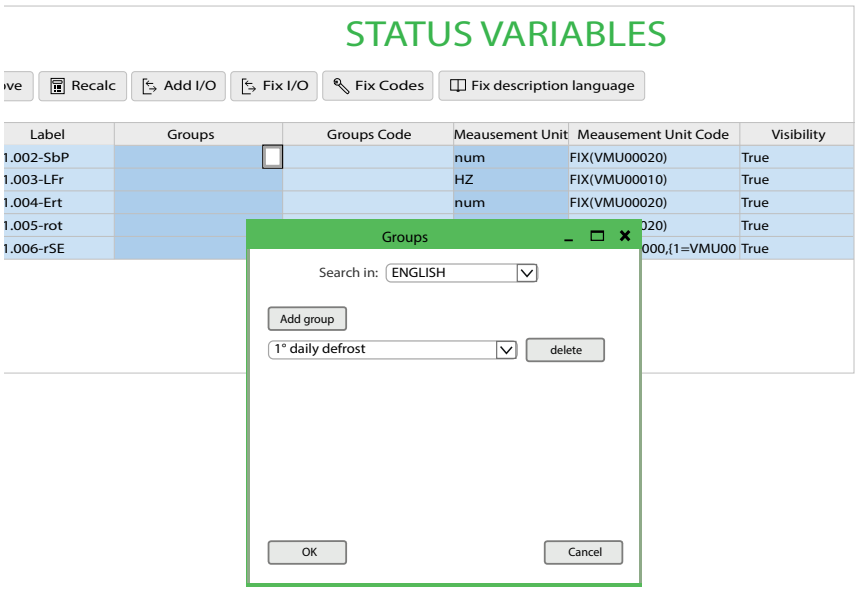


In the **Label** column ❸ there are mandatory cells. These cells cannot contain spaces or special characters.

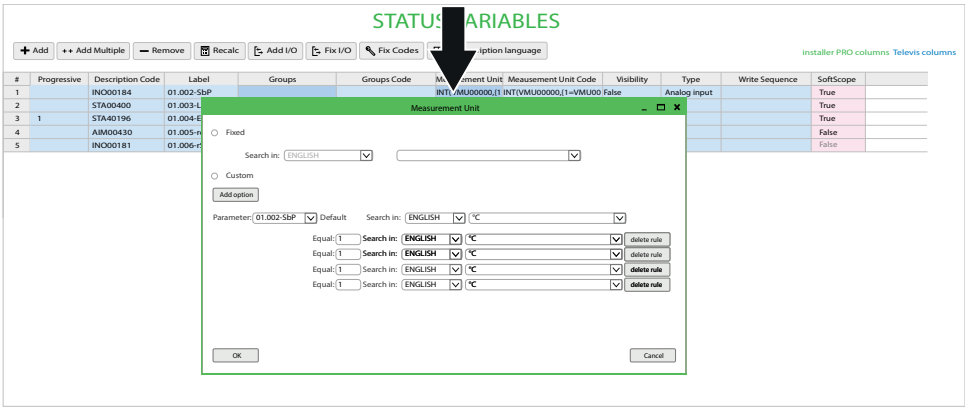


The **Groups** column allows you to insert groups to which the variable belongs. It is accessed via a modal window. One or more groups can be entered and they can be selected from the groups of the Eliwell dictionary and from the strings of the custom dictionary (refer to Custom Dictionaries, page 439).

The **Groups Code** column cannot be changed. It represents the identification code of the string in the **Groups** column.



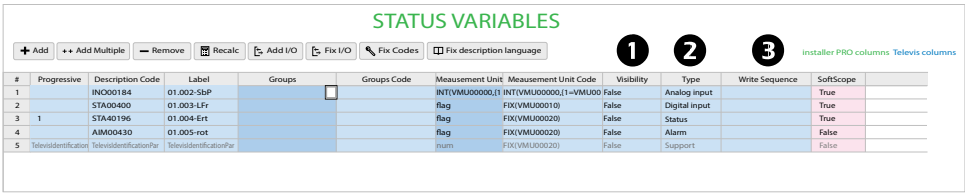
The **Measurement Unit Code** cannot be changed. It represents the identification code of the string in **Measurement unit** column.



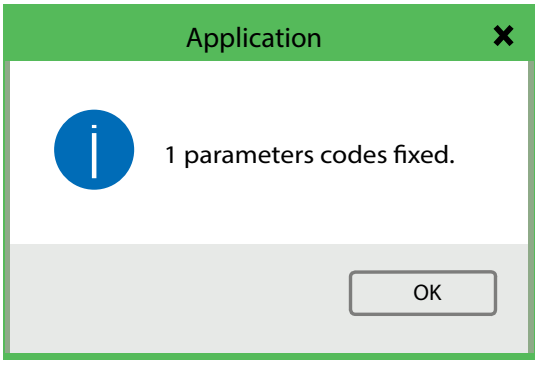
Values in the Visibility column can be **True** or **False**. If **True**, the variable is exported to the Televis drivers, otherwise if **False**, the variable is not exported to the Televis drivers.

The **Type** column ❷ defines the type of the variable.

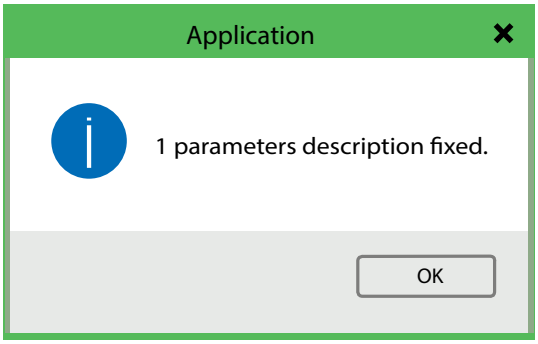
The **Write Sequence** column ❸ allows to enter a series of commands taken from the **Televis Command Configuration** node.



The **Fix Codes** button allows you to regenerate the description code of an inserted description if it is present in the Eliwell dictionary or in the custom dictionary (Custom Dictionaries, page 439).

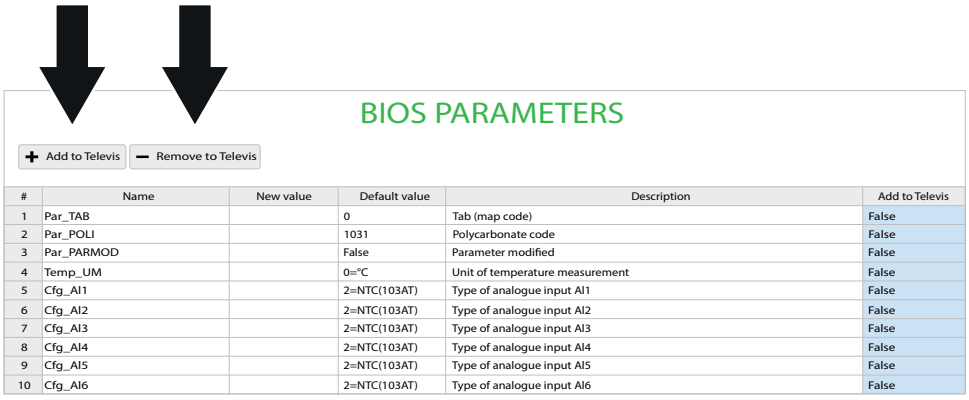


The **Translate** button translates the **Description**, **Groups** and **Measurement Unit** columns into the selected software language, where it is possible.



BIOS Parameters

Add/Remove to Televis button defines if the BIOS parameter is exported or not in the Televis driver. The export to the driver is useful for being able to modify this parameter in real time by the supervisors.



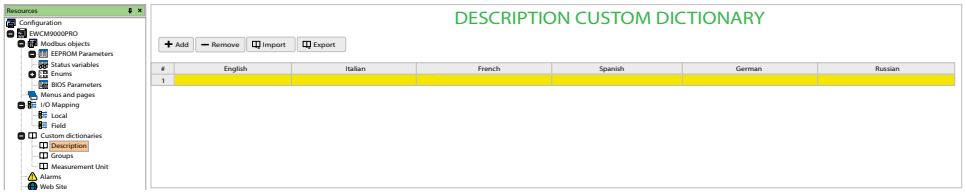
#	Name	New value	Default value	Description	Add to Televis
1	Par_TAB		0	Tab (map code)	False
2	Par_POLI		1031	Polycarbonate code	False
3	Par_PARMOD		False	Parameter modified	False
4	Temp_UM		0=°C	Unit of temperature measurement	False
5	Cfg_AI1		2=NTC(103AT)	Type of analogue input AI1	False
6	Cfg_AI2		2=NTC(103AT)	Type of analogue input AI2	False
7	Cfg_AI3		2=NTC(103AT)	Type of analogue input AI3	False
8	Cfg_AI4		2=NTC(103AT)	Type of analogue input AI4	False
9	Cfg_AI5		2=NTC(103AT)	Type of analogue input AI5	False
10	Cfg_AI6		2=NTC(103AT)	Type of analogue input AI6	False

Custom Dictionaries

The Custom Dictionary page allows you to enter a series of custom strings to be used as description, groups or measurement unit. Each string is added with the **Add button** and must include at least one language. To remove a string use the **Remove button**.

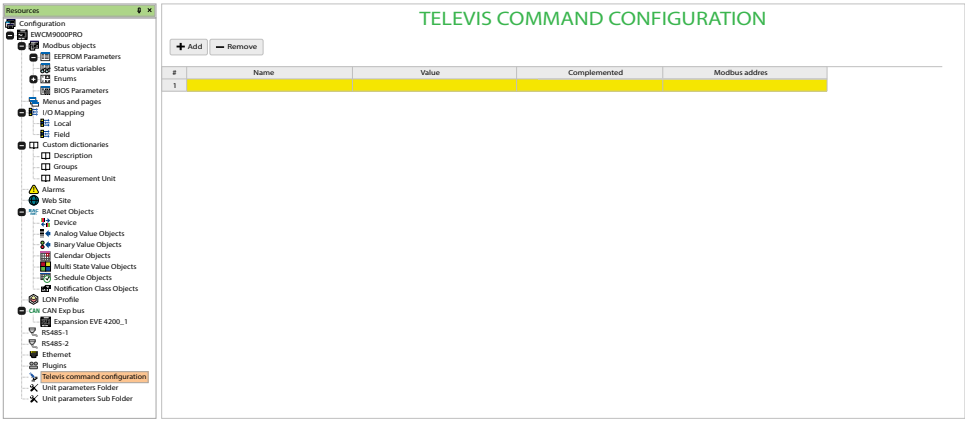
Import button allows to import a custom dictionary.

Export button allows to export a custom dictionary.



Televis Command Configuration

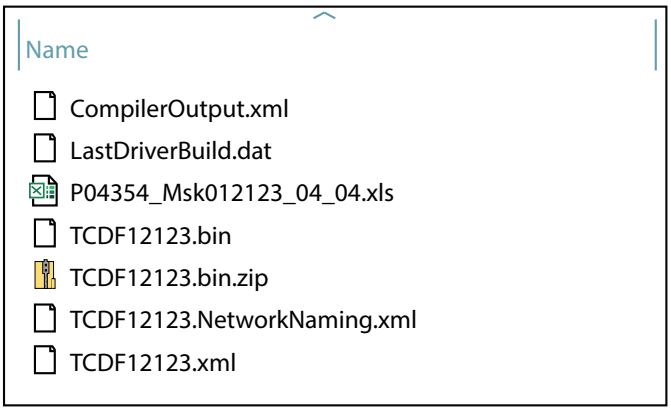
Televis Command Configuration node allows to enter a Televis command.



Projects Outputs

At each compilation of the driver, from menu **Developer > Build Televis Driver**, the following files are created in the **"Televis Driver"** folder in the Project directory:

- TelevisGo driver as **TCDF[Mask].bin**
- TelevisAir driver as **TCDF[Mask].xml**
- **P0435_Msk[Masknumber]_[Target]_[Version].xsl** as hidden file
- Custom dictionaries, that can be imported into TelevisGo as .txt file for each language



Glossary

A

%:

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.

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.

NOTE:

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

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.

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.

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.

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:

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)

H

hex:

(*hexadecimal*)

I

I/O:

(*input/output*)

%I:

According to the IEC standard, %I represents an input bit (for example, a language object of type digital IN).

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.

%IW:

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

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.

%MW:

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

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. POUs facilitate the modular re-use of software programs, functions, and function blocks. Once declared, POUs are available to one another.

program:

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

protocol:

A convention or standard definition that controls or enables the connection, communication, and data transfer between 2 computing system and devices.

Q

%Q:

According to the IEC standard, %Q represents an output bit (for example, a language object of type digital OUT).

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.

string:

A variable that is a series of ASCII characters.

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.

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.

Index

O

P

S

T

V

W

Eliwell Controls s.r.l.
Via dell'Industria, 15 • Z.I. Paludi
32016 Alpago (BL)
Italy

+39 0437 986 111 (Operator)
+39 0437 986 100 (Italy)

www.eliwell.com

As standards, specifications, and design change from time to time, please ask for confirmation of the information given in this publication.

9MA10256.04