SoMachine HVAC Application

User Manual

Revision 4.2 - February 2015

SoMachine HVAC - Application User Manual
Revision 4.2 - February 2015
Published by Schneider Electric
© Schneider Electric 2014-2015
All Rights Reserved.

Contents

1.	**Introduction**	1
1.1	Conventions used in this document	1
2.	**Overview**	3
2.1	The workspace	3
2.1.1	The output window	4
2.1.2	The status bar	4
2.1.3	The document bar	4
2.1.4	The watch window	5
2.1.5	The library window	5
2.1.6	The workspace window	6
2.1.7	The source code editors	7
3.	**Using the environment**	9
3.1	Layout customization	9
3.2	Toolbars	9
3.2.1	Showing/hiding toolbars	9
3.2.2	Moving toolbars	9
3.3	Docking windows	10
3.3.1	Showing/hiding tool windows	10
3.3.2	Floating tool windows	10
3.3.3	Docking tool windows	10
3.3.4	Auto-Hide tool windows	11
3.4	Working with windows	11
3.4.1	The document bar	11
3.4.2	The window menu	12
3.5	Full screen mode	12
3.6	Environment options	12
3.6.1	General	12
3.6.2	Graphic Editor	13
3.6.3	Text Editors	13
3.6.4	Language	13
3.6.5	Tools	14
3.6.6	Merge	16
4.	**Managing projects**	17
4.1	Creating a new project	17
4.2	Uploading the project from the target device	17
4.3	Saving the project	18
4.3.1	Persisting changes to the project	18

4.3.2 Saving to an alternative location 18
4.3.3 Autosave 18
4.3.4 Backup Copies 19
4.4 Managing existing projects 19
4.4.1 Opening an existing Application project 19
4.4.2 Editing the project 19
4.4.3 Closing the project 20
4.5 Distributing projects 20
4.6 Project options 21
4.6.1 Project info 21
4.6.2 Code generation 21
4.6.3 Build output 22
4.6.4 Download 23
4.6.5 Debug 23
4.6.6 Build events 24
4.7 Selecting the target device 24
4.8 Working with libraries 25
4.8.1 The library manager 25
4.8.2 Exporting to a library 26
4.8.3 Importing from a library or another source 27
4.8.4 Updating existing libraries 29
5. Managing project elements 31
5.1 Program Organization Units 31
5.1.1 Creating a new Program Organization Unit 31
5.1.2 Editing POUs 32
5.1.3 Source code encryption/DECRYPTION 32
5.2 Variables 33
5.2.1 Global variables 33
5.2.2 Local variables 36
5.2.3 Creating multiple 36
5.3 Tasks 37
5.3.1 Assigning a program to a task 37
5.3.2 Task configuration 38
5.4 Derived data types 38
5.4.1 Typedefs 38
5.4.2 Structures 39
5.4.3 Enumerations 40
5.4.4 Subranges 41
5.5 Browse the project 42
5.5.1 Object Browser 43

5.5.2 Search with the Find in project command 49

5.6 Working with Application extensions 51

5.7 Project Custom Workspace 52

5.7.1 Enable The Custom Workspace Into An Existing Project 52

5.7.2 Workspaces Migration 52

5.7.3 Custom Workspace Basic Units 53

5.7.4 Custom Workspace Operations 53

5.7.5 Workspace Elements With Limitations 54

6. Editing the source code 55

6.1 Instruction List (IL) editor 55

6.1.1 Editing functions 55

6.1.2 Reference to PLC objects 55

6.1.3 Automatic error location 56

6.1.4 Bookmarks 56

6.2 Structured Text (ST) Editor 56

6.2.1 Creating and editing ST objects 56

6.2.2 Editing functions 56

6.2.3 Reference to PLC objects 57

6.2.4 Automatic error location 57

6.2.5 Bookmarks 57

6.3 Ladder Diagram (LD) editor 58

6.3.1 Creating a new LD document 58

6.3.2 Adding/Removing networks 58

6.3.3 Labeling networks 59

6.3.4 Inserting contacts 59

6.3.5 Inserting coils 60

6.3.6 Inserting blocks 60

6.3.7 Editing coils and contacts properties 61

6.3.8 Editing networks 61

6.3.9 Modifying properties of blocks 61

6.3.10 Getting information on a block 62

6.3.11 Automatic error retrieval 62

6.3.12 Inserting variables 62

6.3.13 Inserting constants 62

6.3.14 Inserting expression 62

6.3.15 Comments 63

6.3.16 Branches 63

6.4 Function Block Diagram (FBD) editor 64

6.4.1 Creating a new FBD document 64

6.4.2 Adding/Removing networks 64

6.4.3 Labeling networks 65

6.4.4 Inserting and connecting blocks 65
6.4.5 Editing networks 66
6.4.6 Modifying properties of blocks 66
6.4.7 Getting information on a block 66
6.4.8 Automatic error retrieval 67
6.5 Sequential Function Chart (SFC) Editor 67
6.5.1 Creating a new SFC document 67
6.5.2 Inserting a new SFC element 67
6.5.3 Connecting SFC elements 67
6.5.4 Assigning an action to a step 67
6.5.5 Specifying a constant/a variable as the condition of a transition 69
6.5.6 Assigning conditional code to a transition 69
6.5.7 Specifying the destination of a jump 71
6.5.8 Editing SFC networks 71
6.6 Variables editor 71
6.6.1 Opening a variables editor 72
6.6.2 Creating a new variable 73
6.6.3 Editing variables 73
6.6.4 Deleting variables 75
6.6.5 Sorting variables 75
6.6.6 Copying variables 76
7. Compiling 77
7.1 Compiling the project 77
7.1.1 Image file loading 77
7.2 Compiler output 77
7.2.1 Compiler errors 78
7.3 Command-line compiler 80
8. Launching the application 81
8.1 Setting up the communication 81
8.1.1 Saving the last used communication port 83
8.2 On-line status 83
8.2.1 Connection status 83
8.2.2 Application status 83
8.3 Downloading the application 84
8.3.1 Controlling source code download 84
8.4 Simulation 87
8.5 Control the PLC execution 87
8.5.1 Halt 87
8.5.2 Cold restart 87
8.5.3 Warm restart 87

8.5.4 Hot restart 87
8.5.5 Reboot target 87

9. Debugging 89

9.1 Watch window 89
9.1.1 Opening and closing the watch window 89
9.1.2 Adding items to the watch window 89
9.1.3 Removing a variable 92
9.1.4 Refreshment of values 93
9.1.5 Changing the format of data 94
9.1.6 Working with watch lists 94
9.1.7 Autosave watch list 96

9.2 Oscilloscope 96
9.2.1 Opening and closing the oscilloscope 97
9.2.2 Adding items to the oscilloscope 97
9.2.3 Removing a variable 100
9.2.4 Variables sampling 100
9.2.5 Controlling data acquisition and display 100
9.2.6 Changing the polling rate 107
9.2.7 Saving and printing the graph 107

9.3 Edit and debug mode 109

9.4 Live debug 109
9.4.1 SFC animation 109
9.4.2 LD animation 110
9.4.3 FBD animation 110
9.4.4 IL and ST animation 111

9.5 Triggers 111
9.5.1 Trigger window 111
9.5.2 Debugging with trigger windows 117

9.6 Graphic triggers 127
9.6.1 Graphic trigger window 127
9.6.2 Debugging with the graphic trigger window 133

10. Application reference 143

10.1 Menus reference 143
10.1.1 File menu 143
10.1.2 Edit menu 144
10.1.3 View menu 145
10.1.4 Project menu 146
10.1.5 Online Menu 147
10.1.6 Debug menu 148
10.1.7 Scheme FBD menu 149
10.1.8 Scheme LD menu 151

10.1.9 Scheme SFC menu 153
10.1.10 Variables menu 154
10.1.11 Window menu 154
10.1.12 Help menu 154
10.2 Toolbars reference 155
10.2.1 Main toolbar 155
10.2.2 FBD toolbar 155
10.2.3 LD toolbar 155
10.2.4 SFC toolbar 155
10.2.5 Project toolbar 155
10.2.6 Network toolbar 155
10.2.7 Debug toolbar 155
11. Language reference 157
11.1 Common elements 157
11.1.1 Basic elements 157
11.1.2 Elementary data types 157
11.1.3 Derived data types 158
11.1.4 Literals 160
11.1.5 Variables 161
11.1.6 Program Organization Units 164
11.1.7 IEC 61131-3 standard functions 167
11.2 Instruction List (IL) 181
11.2.1 Syntax and semantics 181
11.2.2 Standard operators 182
11.2.3 Calling Functions and Function blocks 183
11.3 Function Block Diagram (FBD) 184
11.3.1 Representation of lines and blocks 184
11.3.2 Direction of flow in networks 184
11.3.3 Evaluation of networks 184
11.3.4 Execution control elements 186
11.4 Ladder Diagram (LD) 187
11.4.1 Power rails 187
11.4.2 Link elements and states 188
11.4.3 Contacts 188
11.4.4 Coils 189
11.4.5 Operators, functions and function blocks 190
11.5 Structured Text (ST) 190
11.5.1 Expressions 190
11.5.2 Statements in ST 191
11.6 Sequential Function Chart (SFC) 196
11.6.1 Steps 197

11.6.2 Transitions 199
11.6.3 Rules of evolution 199
11.6.4 SFC control flags 202
11.6.5 Check a SFC POU from other programs 203
11.7 Application Language Extensions 205
11.7.1 Macros 205
11.7.2 Pointers 205
11.7.3 Waiting statement 206
12. Errors Reference 207
12.1 Compile time error messages 207

SAFETY INFORMATION

Important Information

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

The addition of this symbol to a Danger 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 an imminently hazardous situation which, if not avoided, **results in** death or serious injury**.**

⚠ WARNING

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

⚠ CAUTION

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

NOTICE

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

PLEASE NOTE

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel.

No responsibility is assumed by Schneider Electric for any consequences arising out of the use of this material.

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

www.schneider-electric.com

PRODUCT RELATED INFORMATION

⚠ WARNING

LOSS OF CONTROL

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

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

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

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

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

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

1. INTRODUCTION

1.1 CONVENTIONS USED IN THIS DOCUMENT

Text Type	Description
Command, Key	Name of the command or keyboard shortcuts key.
Code	Source code text.
[Context menu]	Toolbar icon and context menu voice.
[Context menu]	Context menu voice without any icon.
Menu>Item	For menu items hierarchy, the ">" symbol is used. A record File>Open Project is equivalent to "the Open Project item under the File menu".
Menu>Item	Same as above including the icon shown in the toolbar.
(see Paragraph) (see Chapter)	Link to related subject within this guide.
Terminology	Important term or concept.

2. OVERVIEW

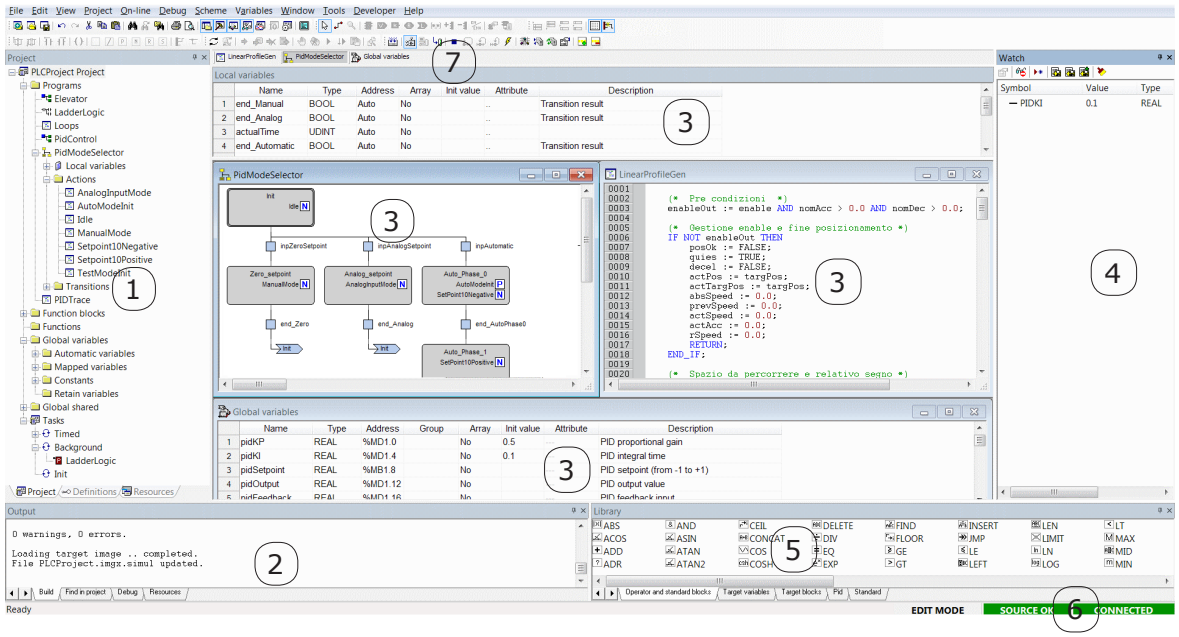
Application is an IEC61131-3 Integrated Development Environment supporting the whole range of languages defined in the standard.

In order to support the user in all the activities involved in the development of an application, Application includes:

- textual source code editors for the Instruction List (briefly, IL) and Structured Text (briefly, ST) programming languages (see Chapter 6.);
- graphical source code editors for the Ladder Diagram (briefly, LD), Function Block Diagram (briefly, FBD), and Sequential Function Chart (briefly, SFC) programming languages (see Chapter 6.);
- a compiler, which translates applications written according to the IEC standard directly into machine code, avoiding the need for a run-time interpreter, thus making the program execution as fast as possible (see Chapter 7);
- a communication system which allows the download of the application to the target environment (see Chapter 8);
- a rich 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, ensuring the information is accurate and reliable (see Chapter 9).

2.1 THE WORKSPACE

The figure below shows a view of Application's workspace, including many of its more commonly used components.

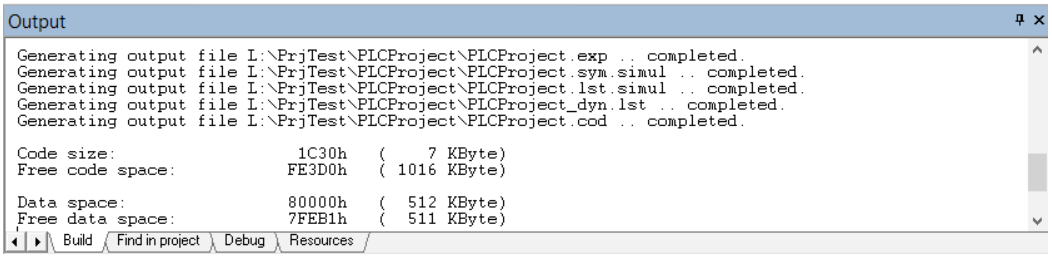


1. *Workspace window* **2.** *Output window* **3.** *Source code editors* **4.** *Watch window* **5.** *Library window* **6.** *Status bar* **7.** *Document bar*

The following paragraphs give an overview of these elements.

2.1.1 THE OUTPUT WINDOW

The *Output* window is the place where Application prints its output messages. This window contains four tabs: *Build*, *Find in project*, *Debug*, and *Resources*.



Build

The *Build* panel displays the output of the following activities:

- opening a project;
- compiling a project;
- downloading code to a target.

Find in project

This panel shows the result of the *Find in project* activity.

Debug

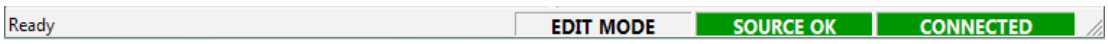
The *Debug* panel displays information about advanced debugging activities (for example, breakpoints). Depending on the target device you are interfacing with, Application can print on this output window every PLC run-time error (for example, division by zero), locating the exact position where the error occurred.

Resources

The *Resources* panel displays messages related to the specific target device Application is interfacing with.

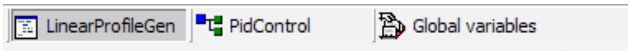
2.1.2 THE STATUS BAR

The *Status* bar displays the state of the application at its left border, and an animated control reporting the state of communication at its right border.



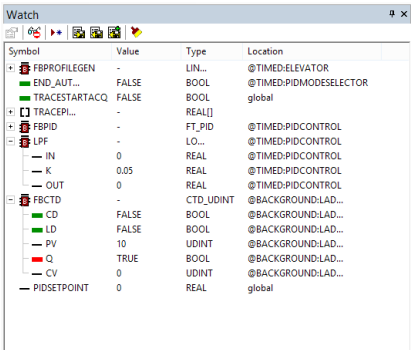
2.1.3 THE DOCUMENT BAR

The *Document* bar lists all the documents currently open for editing in Application.



2.1.4 THE WATCH WINDOW

The *Watch* window is one of the many debugging tools supplied by Application. Among the other debugging tools, it is worth mentioning the Oscilloscope (see Paragraph 9.2), triggers, and the live debug mode (see Paragraph 9.2).



2.1.5 THE LIBRARY WINDOW

The *Library* window contains a set of different panels, which fall into the categories explained in the following paragraphs.

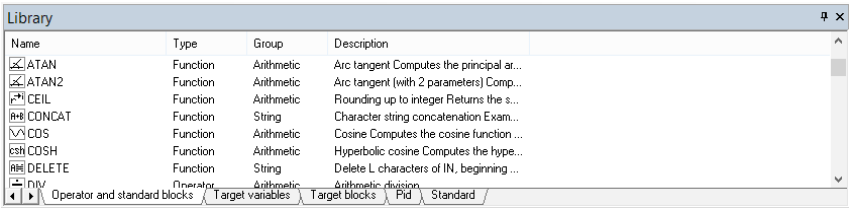
You can choose the display mode by clicking the right button of your mouse. In the *[View list]* mode, each element is represented by its name and icon. Instead, a table appears in the *[View detail]* mode, each row of which is associated with one of the embedded elements. The latter mode also displays the *Type* (Operator/Function) and the description of each element.

If you right-click one of the elements of this panel, and you click *[Object properties]* from the dialog box, then a window appears with further details on the element you selected (input and output supported types, name of input and output pins, etc.).

In the *[View folder]* mode each element is grouped into the folder to which it belongs. These folders are useful to logically group the library elements.

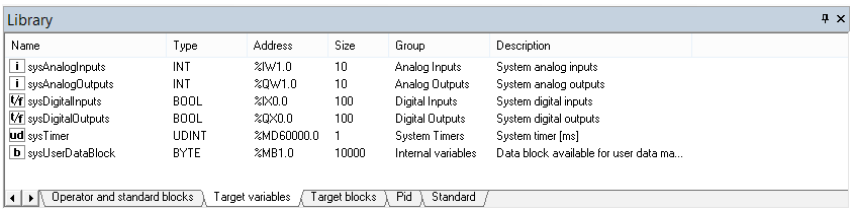
2.1.5.1 OPERATORS AND STANDARD BLOCKS

This panel lists basic language elements, such as operators and functions defined by the IEC 61131-3 standard.



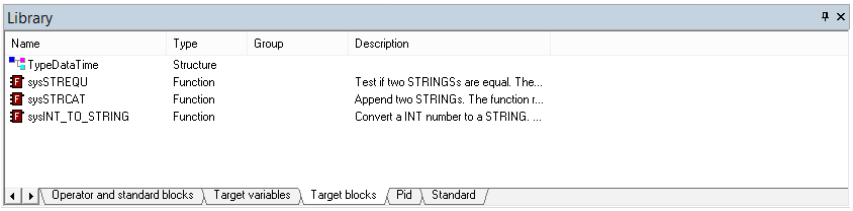
2.1.5.2 TARGET VARIABLES

This panel lists all the system variables, also called target variables, which are the interface between firmware and PLC application code.



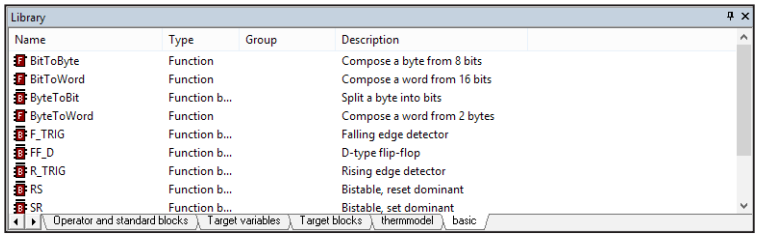
2.1.5.3 TARGET BLOCKS

This panel lists all the system functions and function blocks available on the specific target device.



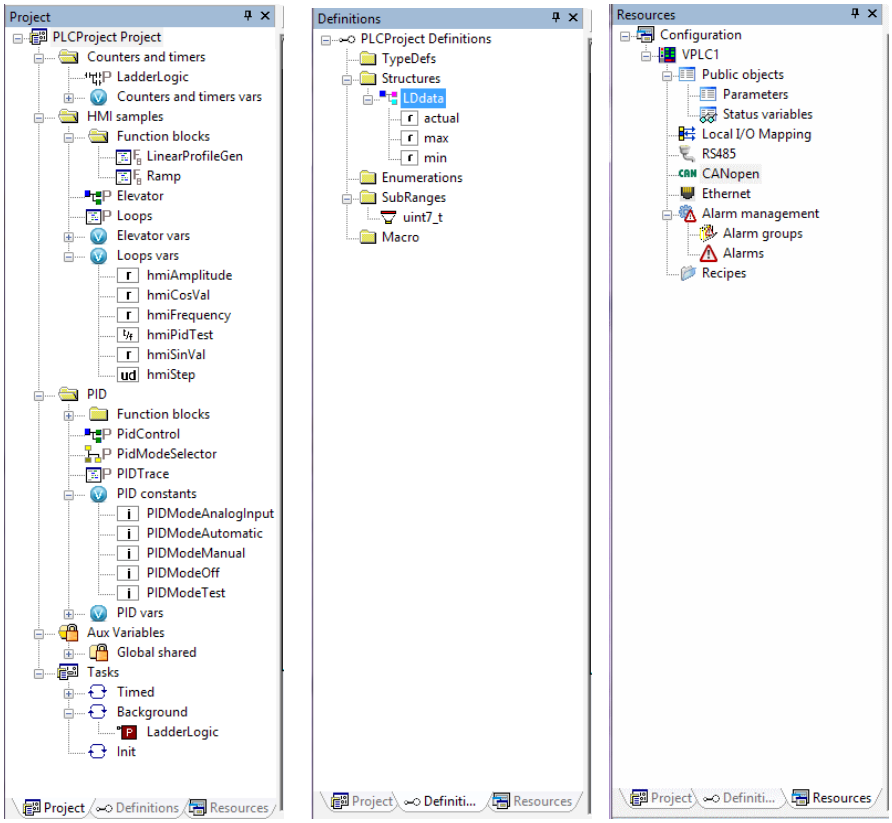
2.1.5.4 INCLUDED LIBRARY PANELS

The panels described in the preceding paragraphs are usually always available in the *Library* window. However, other panels may be added to this window, one for each library included in the current Application project. For example, the picture above was taken from a Application project having two included libraries, *basic.pll* and *thermmodel.pll* (see Paragraph 4.7).



2.1.6 THE WORKSPACE WINDOW

The *Workspace* window consists of three distinct panels, as shown in the following picture.



2.1.6.1 PROJECT

The *Project* panel contains a set of folders:

- *Tasks*: this item lists the system tasks and the programs assigned to each task (see Paragraph 5.3).

2.1.6.2 DEFINITIONS

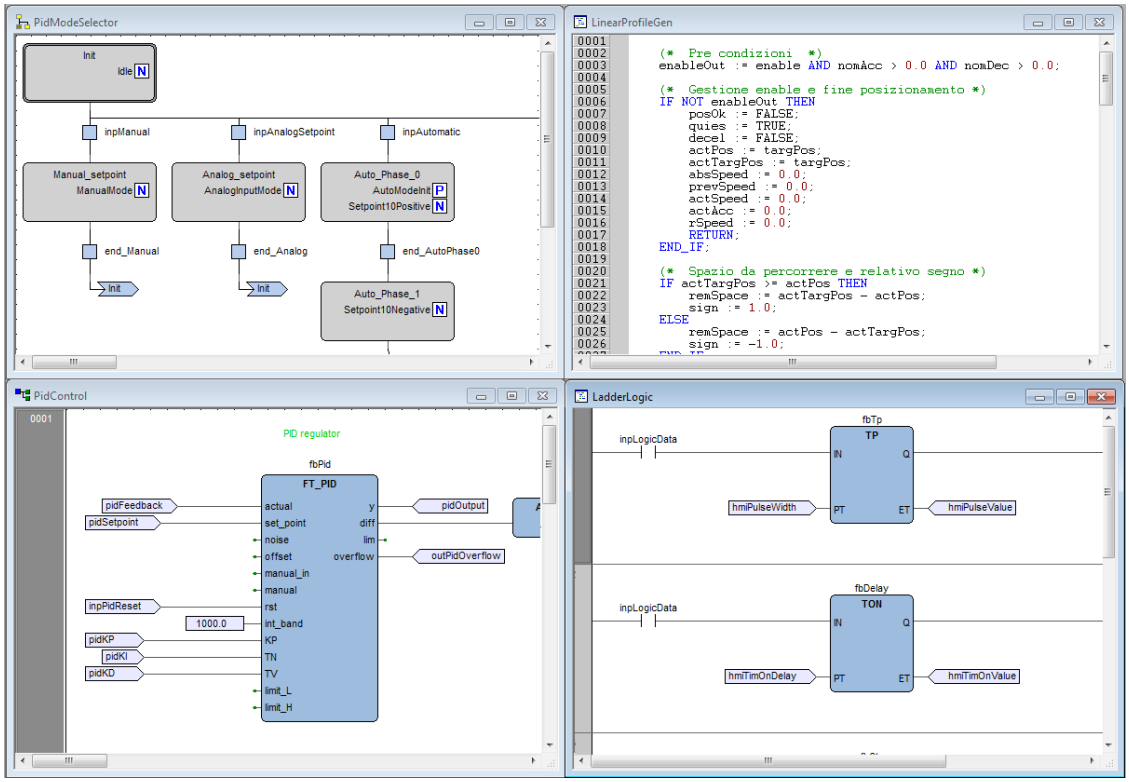
The *Definitions* panel contains the definitions of all user-defined data types, such as structures or enumerated types.

2.1.6.3 RESOURCES

The contents of the *Resources* panel depends on the target device Application is interfacing with: it may include configuration elements, schemas, wizards, and so on.

2.1.7 THE SOURCE CODE EDITORS

The Application programming environment includes a set of editors to manage, edit, and print source files written in any of the 5 programming languages defined by the IEC 61131-3 standard (see Chapter 6).



The definition of both global and local variables is supported by specific spreadsheet-like editors

	Name	Type	Address	Group	Array	Init value	Attribute	Description
1	hmiElevatorOn	BOOL	%MX1.78		No		---	Positioning enable
2	hmiElevatorStanding	BOOL	%MX1.83		No		---	If TRUE, elevator is not moving
3	hmiPidTest	BOOL	%MX1.1319		No		---	Starts execution of PID test
4	traceStartAcq	BOOL	Auto		No		---	Start of test PID acquisition
5	traceTrigger	BOOL	%MX1.1286		No		---	
6	hmiPIDMode	INT	%MW1.32		No		---	
7	tracePIDLen	UINT	%MW1.1284		No		---	

3. USING THE ENVIRONMENT

This chapter shows you how to deal with the many UI elements Application is composed of, in order to let you set up the IDE in the way which best suits to your specific development process.

3.1 LAYOUT CUSTOMIZATION

The layout of Application's workspace can be freely customized in order to suit your needs.

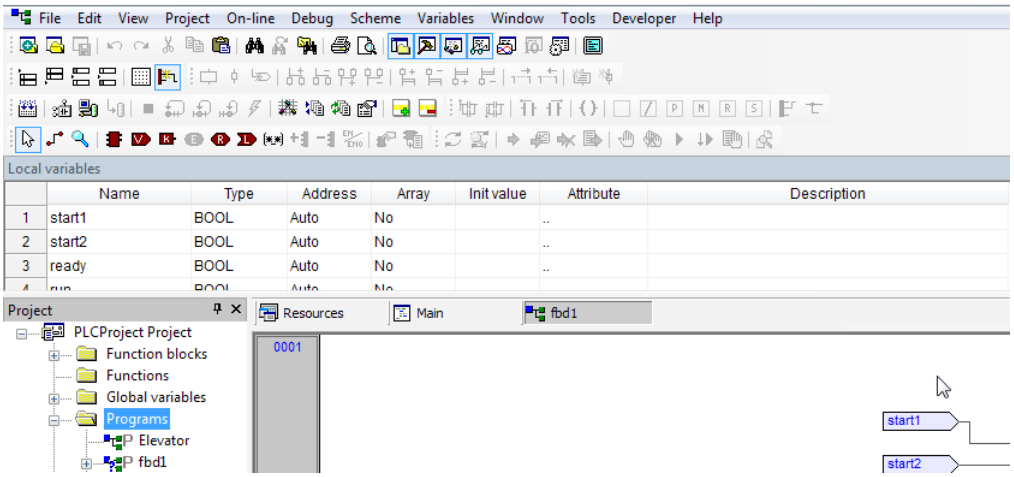
Application takes care to save the layout configuration on application exit, in order to persist your preferences between different working sessions.

3.2 TOOLBARS

3.2.1 SHOWING/HIDING TOOLBARS

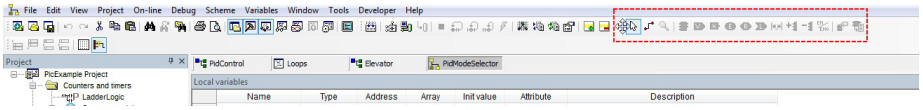
In details, in order to show (or hide) a toolbar, open the *View>Toolbars* menu and select the desired toolbar (for example, the *FBD* bar).

The toolbar is then shown (hidden).

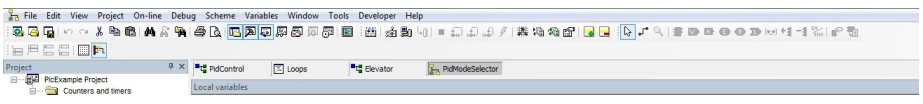


3.2.2 MOVING TOOLBARS

You can move a toolbar by clicking on its left border and then dragging and dropping it to the destination.



The toolbar shows up in the new position.

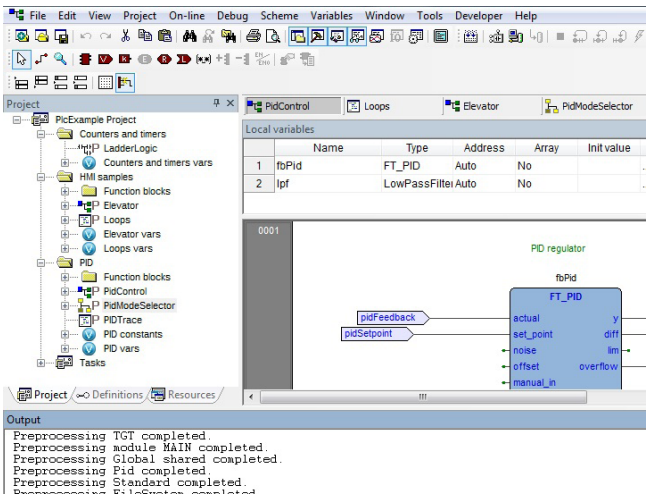


3.3 DOCKING WINDOWS

3.3.1 SHOWING/HIDING TOOL WINDOWS

The *View>Tool windows* menu allows you to show (or hide) a tool window (for example, the *Output* window).

The tool window is then shown (hidden).



3.3.2 FLOATING TOOL WINDOWS

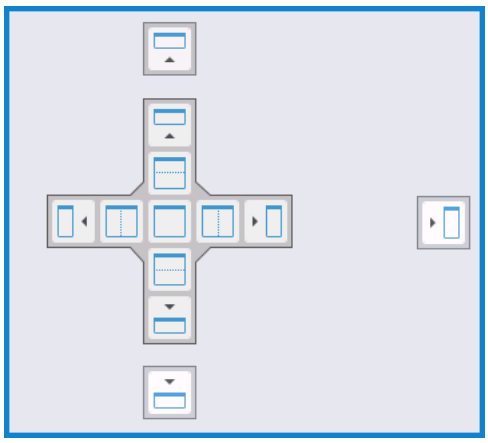
You can undock any window from its default location in Application and move it anywhere by dragging it to the location you want.

Take back a window to its most recent docked location simply double-click the title bar of the window.

3.3.3 DOCKING TOOL WINDOWS

Application shows you a guide diamond when you drag a window to another location to help you easily re-dock the window.

While dragging a window move the mouse cursor on the position of the guide diamond you want to use as new window position.



Tool windows can be fastened to one side of a frame in Application or within a frame.

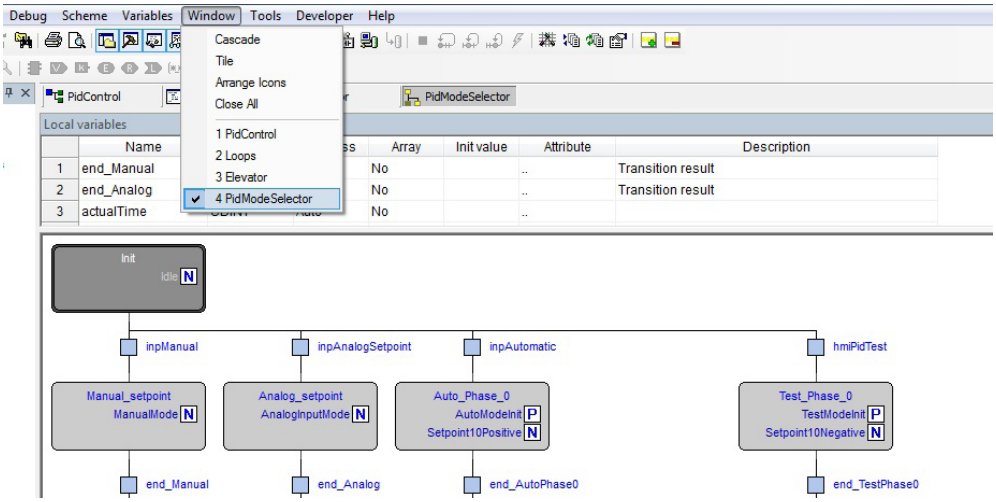
3.3.4 AUTO-HIDE TOOL WINDOWS

By the pin button on the top right corner of the window you can switch the window to auto-hide mode or to regular docking mode.

3.4 WORKING WITH WINDOWS

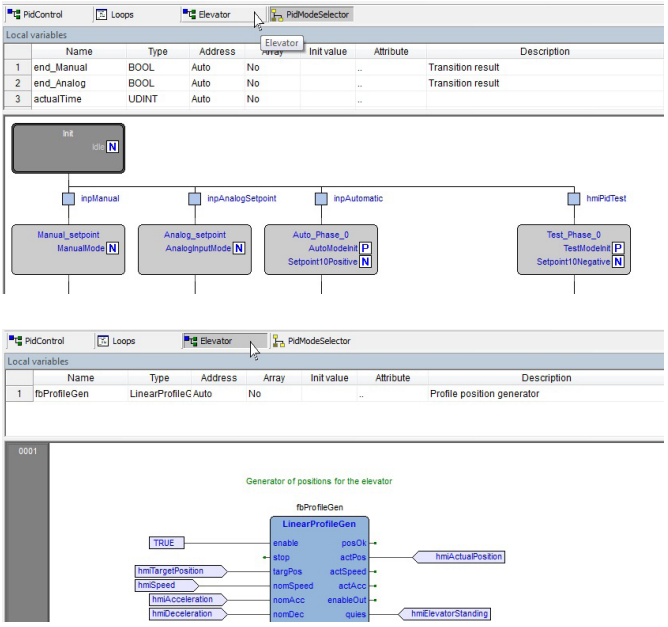
Application allows to open many source code editors so that the workspace could get rather messy.

You can easily navigate between these windows through the *Document* bar and the *Window* menu.



3.4.1 THE DOCUMENT BAR

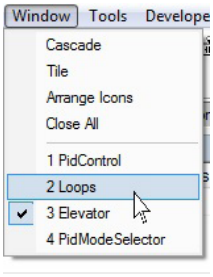
The *Document* bar allows to switch between all the currently open editors, simply by clicking on the corresponding name.



You can show or hide the *Document bar* with the menu option of the same name in the menu *View>Toolbars>Document bar*.

3.4.2 THE WINDOW MENU

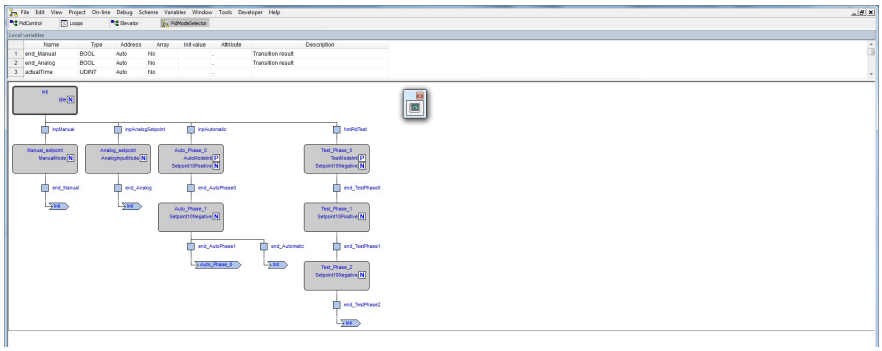
The *Window* menu is an alternative to the *Document* bar: it lists all the currently open editors and allows to switch between them.



Moreover, this menu supplies a few commands to automate some basic tasks, such as closing all windows.

3.5 FULL SCREEN MODE

In order to ease the coding of your application, you may want to switch on the full screen mode. In full screen mode, the source code editor extends to the whole working area, making easier the job of editing the code, notably when graphical programming languages (that is, LD, FBD, and SFC) are involved.



You can switch on and off the full screen mode with the *View>Full screen*.

3.6 ENVIRONMENT OPTIONS

If you click *File>Options...*, a multi-tab dialog box appears and lets you customize some options of Application.

3.6.1 GENERAL

3.6.1.1 SAVE OPTIONS

Autosave: if the *Enable Autosave* box is checked, Application 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 the *Autosave interval* text box.

Max previous version to keep: if set greater than 0 indicates the maximum number of copies of the project that must be zipped and stored in the *PreviousVersions* folder.

3.6.1.2 OUTPUT WINDOW

You can specify the family and the size of the font used for output window.

3.6.1.3 COMMUNICATION

If enabled, the last used port will be set as the default one.

3.6.1.4 TOOLTIP

If enabled, small information boxes will appear when user places the cursor over a symbol in the editors.

3.6.1.5 TOOL WINDOWS

You can specify the family and the size of the font used for tool windows.

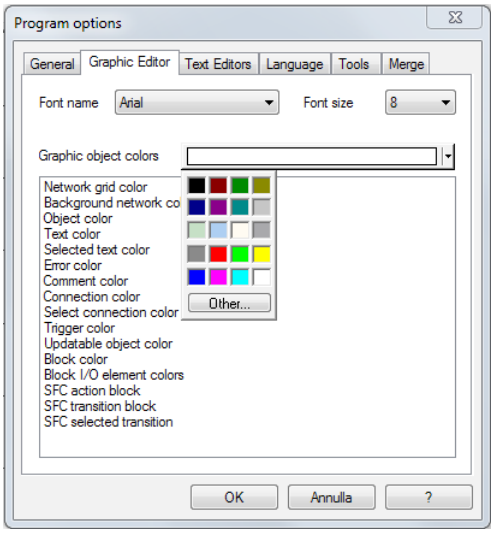
Reset bars positions: the layout of the dock bars in the IDE will be resetted to default positions and dimensions. In order to take effect Application must be restarted.

3.6.2 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 modify also the colours of the graphical object.



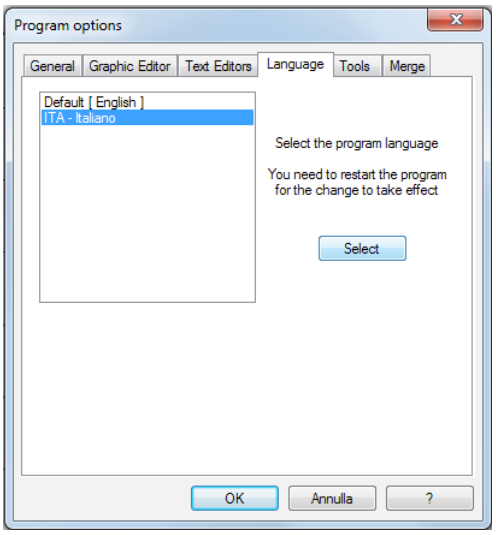
3.6.3 TEXT EDITORS

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

3.6.4 LANGUAGE

You can change the language of the environment by selecting a new one from the list shown in this panel.

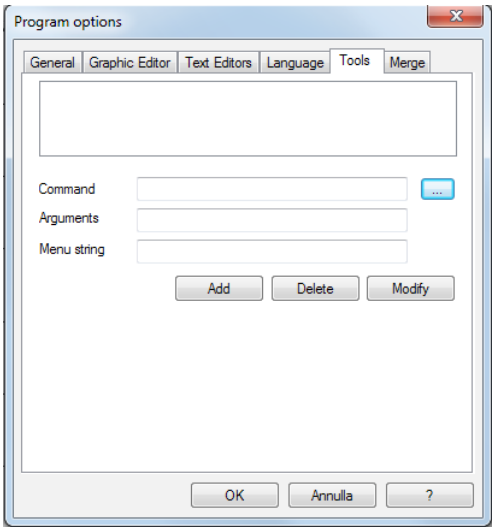
After selecting the new language, press the *Select* button and confirm by clicking *OK*. This change will be effective only the next time you start Application.



3.6.5 TOOLS

You can add up to 16 commands to the *Tools* menu. These commands can be associated with any program that will run on your operating system. You can also specify arguments for any command that you add to the *Tools* menu. The following procedure shows you how to add a tool to the *Tools* menu.

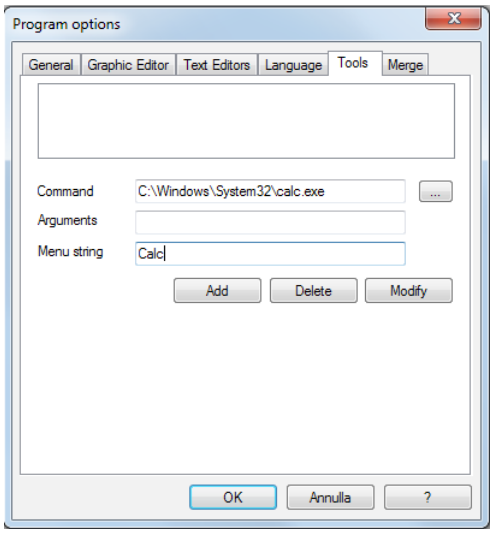
1) Type the full path of the executable file of the tool in the *Command* text box. Otherwise, you can specify the filename by selecting it from Windows Explorer, which you open by clicking the *Browse* button.



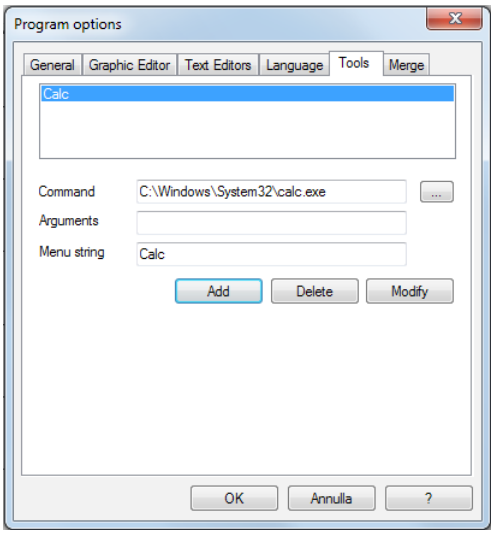
2) In the *Arguments* text box, type the arguments - if any - to be passed to the executable command mentioned at step 1. They must be separated by a space.
3) Enter in *Menu string* the name you want to give to the tool you are adding. This is the string that will be displayed in the *Tools* menu.
4) Press *Add* to effectively insert the new command into the suitable menu.
5) Press *OK* to confirm, or *Cancel* to quit.

For example, let us assume that you want to add *Windows calculator* to the *Tools* menu:

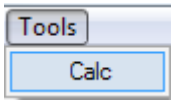
- Fill the fields of the dialog box as displayed.



- Press *Add*. The name you gave to the new tool is now displayed in the list box at the top of the panel.

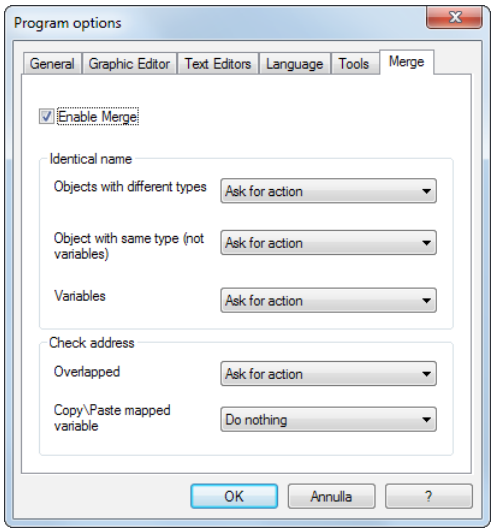


And in the *Tools>Calc* menu as well.



3.6.6 MERGE

Here you can set the merge function behavior (see Paragraph 4.8.3.2 for more details).



4. MANAGING PROJECTS

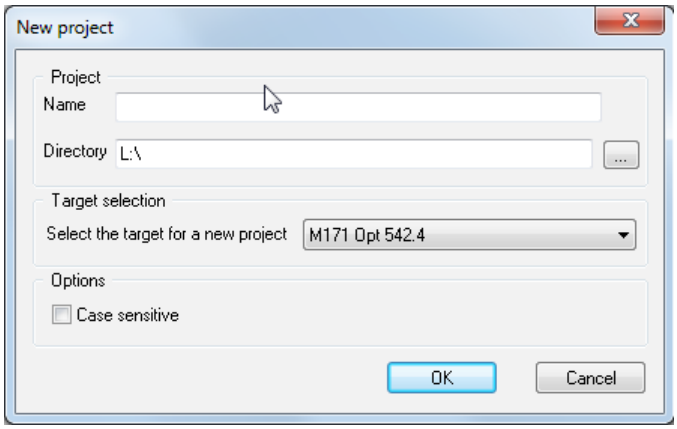
This chapter focuses on Application projects.

A project corresponds to a PLC application and includes all the required elements to run that application on the target device, including its source code, links to libraries, information about the target device and so on.

The following paragraphs explain how to properly work with projects and their elements.

4.1 CREATING A NEW PROJECT

To start a new project, click *File>New project* of the Application main window.



You are required to enter the name of the new project in the *Name* control. The string you enter will also be the name of the folder which will contain all the files making up the Application project. The pathname in the *Directory* control indicates the default location of this folder.

Target selection allows you to specify the target device which will run the project.

Finally, you can make the project case-sensitive by activating the related option. Note that, by default, this option is not active, in compliance with IEC 61131-3 standard: when you choose to create a case-sensitive project, it will not be standard-compliant.

When you confirm your decision to create a new project and the whole required information has been provided, Application completes the operation, creating the project directory and all project files; then, the project is opened.

The list of devices from which you can select the target for the project you are creating depends on the contents of the catalog of target devices available to Application.

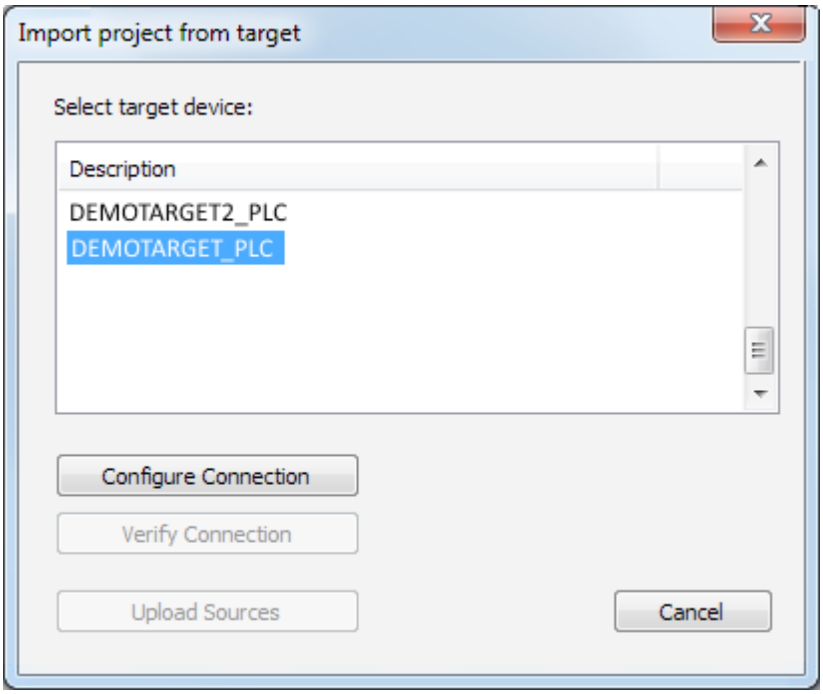
When the desired target is missing, either you have run the wrong setup executable or you have to run a separate setup which is responsible to update the catalog to include the target device. In both cases, you should contact your hardware supplier for support.

4.2 UPLOADING THE PROJECT FROM THE TARGET DEVICE

Depending on the target device you are interfacing with, you may be able to upload a working Application project from the target itself.

In order to upload the project from the target device, follow the procedure below:

1) Click the *File>Import project from target* menu voice of the Application main window, which opens the Target list dialog box.



2) From the shown list select the target device from which you want to upload the project.
3) *Configure Connection* with correct parameters (see Paragraph 8.1 for more details).
4) You can test the connection with the target device by *Verify Connection* button. Application tries to establish the connection and reports the test result.
5) If the connection is available confirm the operation by clicking on the *Upload Sources* button. When the application upload completes successfully, the project is open and ready for editing.

4.3 SAVING THE PROJECT

4.3.1 PERSISTING CHANGES TO THE PROJECT

When you make any change to the project (for example, you add a new Program Organization Unit) you are required to save the project in order to persist that change.

To save the project, you can select the corresponding item *File>Save project*.

4.3.2 SAVING TO AN ALTERNATIVE LOCATION

You can also use the *File>Save project As ...* command to rename the project, change its format or modify the location of where you want save the file.

Application asks you to select the new destination (which must be an empty directory), then saves a copy of the project to that location and opens this new project file for editing.

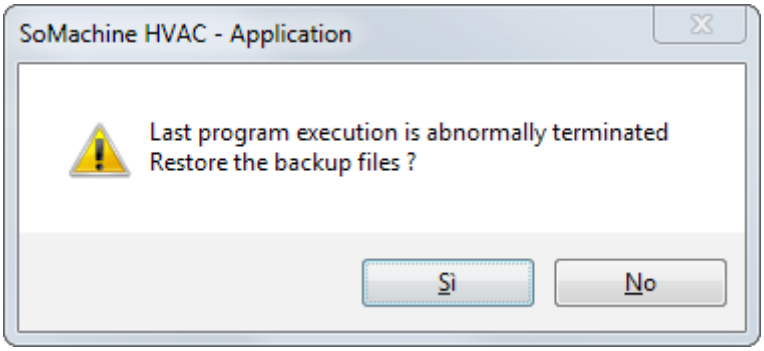
4.3.3 AUTOSAVE

Application 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.

AutoSave protects your project in the event that Application unexpectedly quits. When Application is started again, if the *Backup* folder is present, you are asked to restore the

last valid backup file of the project.



When you close Application correctly the *Backup* folder and its contents are deleted. You can specify the interval time (in minutes) between saving.

By default *AutoSave* is running with 1 minute of interval (see Paragraph 3.6 for more details).

4.3.4 BACKUP COPIES

Application includes a backup feature of the previous version of the project on which you are working.

When you explicitly save the project, Application 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 (see Paragraph 3.6 for more details).

4.4 MANAGING EXISTING PROJECTS

4.4.1 OPENING AN EXISTING APPLICATION PROJECT

To open an existing project, click *File>Open project* of Application's main window, or in the *Welcome page* (when no project is open). This causes a dialog box to appear, which lets you load the directory containing the project and select the relative project file.

4.4.2 EDITING THE PROJECT

In order to modify an element of a project, you need first to open that element by double-clicking its name, which you can find by browsing the tree structure of the project tab of the *Workspace* bar.

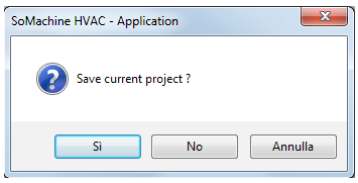
By double-clicking the name of the object you want to modify, you open an editor consistent with the object type: for example, when you double-click the name of a project POU, the appropriate source code editor is shown; if you double-click the name of a global variable, the variable editor is shown.

Note that Application prevents you from applying changes to elements of a project, when at least one of the following conditions holds:

- you are 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).

4.4.3 CLOSING THE PROJECT

You can terminate the working session either by explicitly closing the project or by exiting Application. In both cases, when there are changes not yet persisted to file, Application asks you to choose between saving and discarding them.



To close the project, select the item *File>Close project*; Application shows the *Welcome page*, so that you can rapidly start a new working session.

4.5 DISTRIBUTING PROJECTS

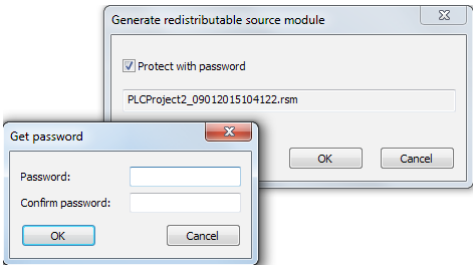
When you need to share a project with another developer you can send him/her either a copy of the project file(s) or a redistributable source module (RSM) generated by Application.

In the former case, the number of files you have 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) including all source code modules, so that you need to share only the *.ppjs* file.
- 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. You need to share the whole directory.
- 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.

Alternatively, you can generate a redistributable source module (RSM) with the corresponding item *Project>Generate redistributable source module*.

Application notifies you of the name of the RSM file and lets you choose whether to protect the file with a password or not. If you choose to protect the file, Application asks you to insert the password.



The advantages of the RSM file format are:

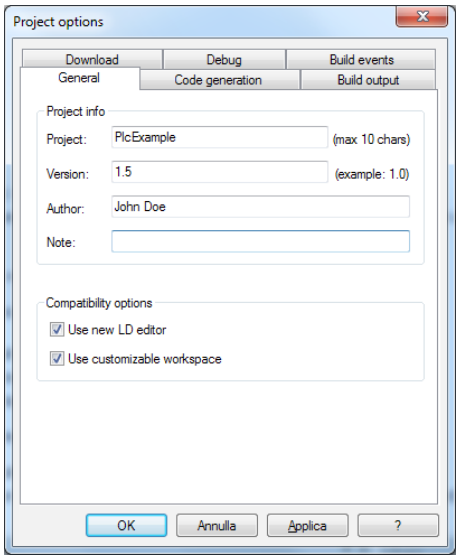
- the source code is encoded in binary format, thus it cannot be read by third parties which do not use Application, making a transfer over the Internet more secure;
- it can be protected with a password, which will be required by Application on file opening;
- being a binary file, its size is reduced.

4.6 PROJECT OPTIONS

You can edit some significant project properties choosing *Project>Options...*.

4.6.1 PROJECT INFO

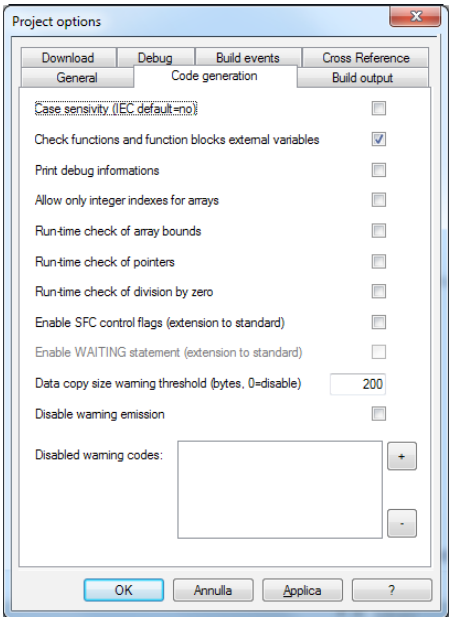
Here you can set some basic properties related to the project, such as its application name and version.



- *Use new LD editor*: the new Ladder Diagram editor is easier to use, by helping you in common operations working on the diagram will be faster and more efficient. Note that, by default, this option is active.
- *Use customizable workspace*: allows you to manage your project tree in order to reach a more efficient workspace. Note that, by default, this option is active.

4.6.2 CODE GENERATION

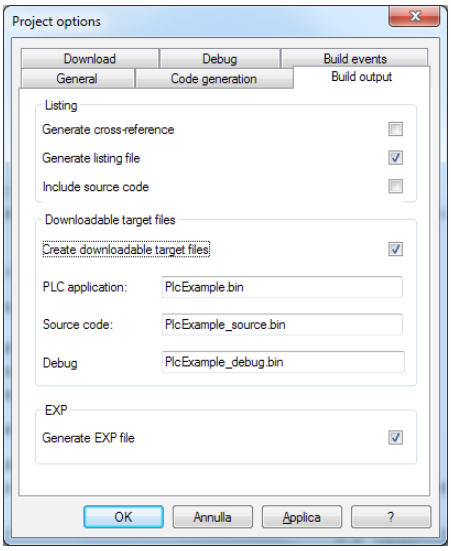
Here you can edit some properties about code generation.



- *Case sensitivity*: you can set the project as case-sensitive checking this option. Note that, 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 that, by default, this option is enabled respecting the IEC 61131-3 standard.
- *Print debug information*: prints on the output window some significant debug info.
- *Allow only integer indexes for arrays*: if this option is checked you cannot use *BYTE*, *WORD* or *DWORD* as array indexes.
- *Run-time check of array bounds*: if this option is checked some check code is added to verify that array indexes are not out of bounds during run-time. This option is settable depending on target device.
- *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 is settable depending on target device.
- *Run-time check of pointers*: if this option is checked the pointers will be test for their validity before their use, calling a user-defined function `checkptr` on target. Therefore this option is settable depending on target device.
- *Enable SFC control flags (extension to standard)*: if this option is checked, HOLD and RESET flags for SFC POU are enabled.
- *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 (see Paragraph 11.7.3 for more details).
- *Data copy size warning threshold (bytes, 0=disable)*: when arrays or structures are copied, if their dimension exceed the specified threshold, a warning is emitted in order to inform the possible loss of performance of the PLC. If the threshold is set to 0, no warnings are emitted.
- *Disable warning emission*: if this option is checked warning emissions are not printed on the output window.
- *Disable warning codes*: if this option is checked some specified warning emissions are not printed on the output window.

4.6.3 BUILD OUTPUT

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



Listing section

- *Generate listing file*: if this option is checked the compiler will generate a listing file named as *projectname.lst*.
- *Include source code* (active only if *Generate listing file* is checked): if this option is checked the source code will be inserted as visible in the *lst* file. Otherwise the source code will be hidden.

Downloadable target files section

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

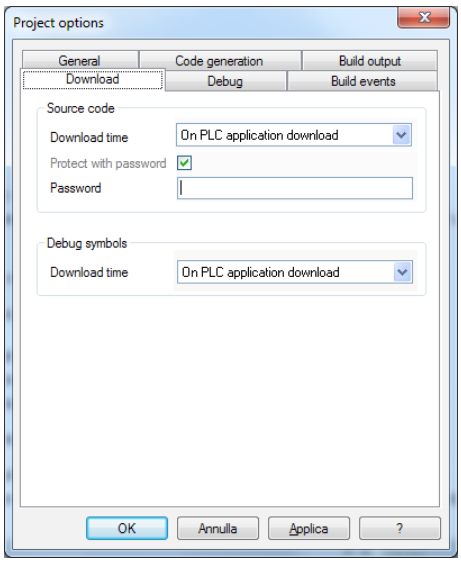
  Please note that only valid Windows filename are accepted!
- *PLC application* (active only if *Create downloadable target files* is checked): this field specifies the name of the PLC application binary file. By default *projectname.bin*.
- *Source code* (active only if *Create downloadable target files* is checked): this field specifies the name of the Source code binary file. By default *projectname._source.bin*.
- *Debug* (active only if *Create downloadable target files* is checked): this field specifies the name of the Debug symbol binary file. By default *projectname._debug.bin*

Generate EXP file section

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

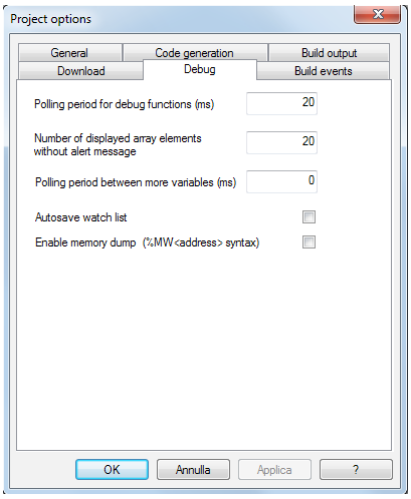
4.6.4 DOWNLOAD

Here you can edit some significant properties of the download behavior (see Paragraph 8.3.1 for more information).



4.6.5 DEBUG

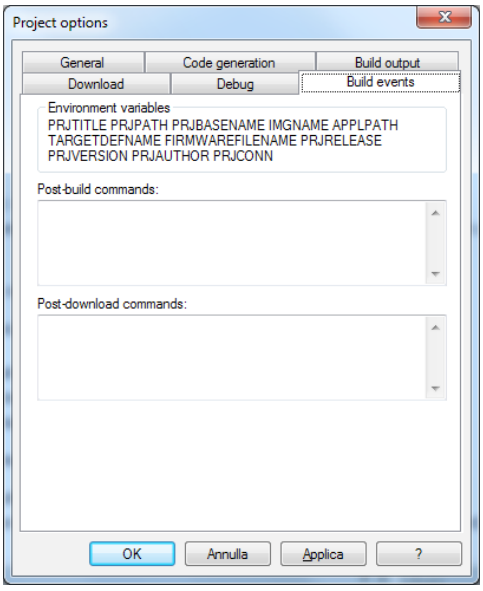
Here you can edit some significant properties of the debug behavior.



- *Polling period for debug function* (ms): set the active sampling period of the functions's status.
- *Number of displayed array elements without alert message*: specifies the maximum number of array elements to be added in watch window without being alerted.
- *Polling period between more variables* (ms): set the sleep period between sampling two variables.
- *Autosave watch list*: if checked (not by default) the watch list status will be saved into a file, when the project is closed (see Paragraph 9.1.7 for more details).

4.6.6 BUILD EVENTS

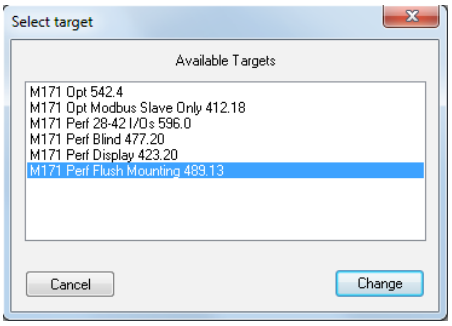
Here you can specify commands that run before the build starts or after the build finishes. You can also use a set of defined environment variables listed on the top of the window.



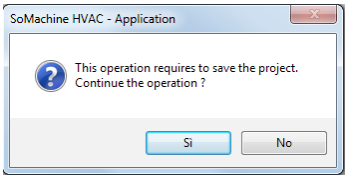
4.7 SELECTING THE TARGET DEVICE

You may need to port a PLC application on a target device which differs from the one you originally wrote the code for. Follow the instructions below to adapt your Application project to a new target device.

1) Click *Project>Select target* menu of the Application main window. This causes the following dialog box to appear.



2) Select one of the target devices listed in the combo box.
3) Click *Change* to confirm your choice, *Cancel* to abort.
4) If you confirm, Application displays the following dialog box.



Press *Yes* to complete the conversion, *No* to quit.

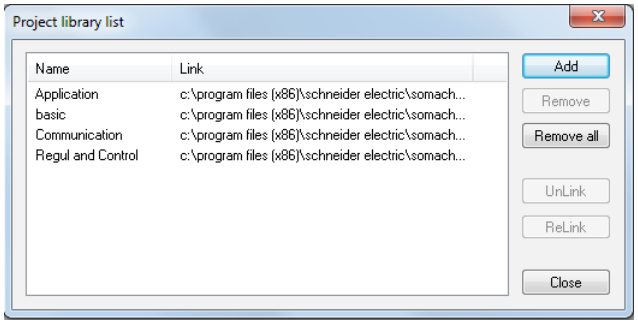
If you press *Yes*, Application updates the project to work with the new target.

It also makes a backup copy of the project file(s) in a sub-directory inside the project directory, so that you can roll-back the operation by manually (i.e., using Windows Explorer) replacing the project file(s) with the backup copy.

4.8 WORKING WITH LIBRARIES

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

4.8.1 THE LIBRARY MANAGER



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

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

4.8.1.1 INCLUDING A LIBRARY

The following procedure shows you how to include a library in a Application project, which results in all 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. Note that you cannot edit the elements of an included library, unlike imported objects.

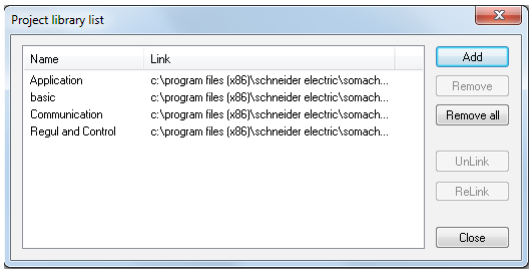
If you want to copy or move a project which includes one or more libraries, make sure that references to those libraries are still valid in the new location.

1) Click *Project>Library manager*, which opens the *Library manager* dialog box.
2) Press 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 pressing 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 in the white box.
4) Repeat step 1, 2, and 3 for all the libraries you wish to include.
5) When you have finished including libraries, press either *OK* to confirm, or *Cancel* to quit.

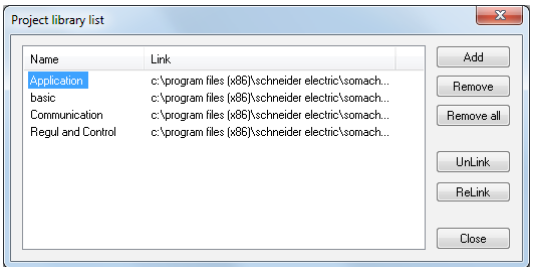
4.8.1.2 REMOVING A LIBRARY

The following procedure shows you how to remove an included library from the current project. Remember that removing a library does not mean erasing the library itself, but the project's reference to it.

1) Click *Project>Library manager* menu of the Application main window, which opens the *Library manager* dialog box.



Select the library you wish to remove by clicking its name once. The *Remove* button is now enabled.



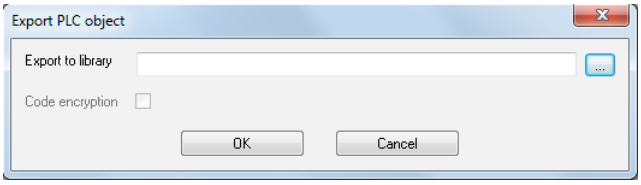
2) Click the *Remove* button, which causes the reference to the selected library to disappear from the *Project library* list.
3) Repeat for all the libraries you wish to remove. Alternatively, if you want to remove all the libraries, you can press the *Remove all* button.
4) When you have finished removing libraries, press either *OK* to confirm, or *Cancel* not to apply changes.

4.8.2 EXPORTING TO A LIBRARY

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 shows you how to export objects to a library.

1) Look for the object you want to export by browsing the tree structure of the project tab of the *Workspace* bar, then click once the name of the object.

2) Click *Project>Export object to library*. This causes the following dialog box to appear.



3) Enter the destination library by specifying the location of its *.pll* file. You can do this by:
   - typing the full pathname in the white text box;
   - clicking the *Browse* button , in order to open an explorer dialog box which allows you 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.
5) Click *OK* to confirm the operation, otherwise press *Cancel* to quit.

If at Step 3 of this procedure you enter the name of a non-existing *.pll* file, Application creates the file, thus establishing a new library.

4.8.2.1 UNDOING EXPORT TO A LIBRARY

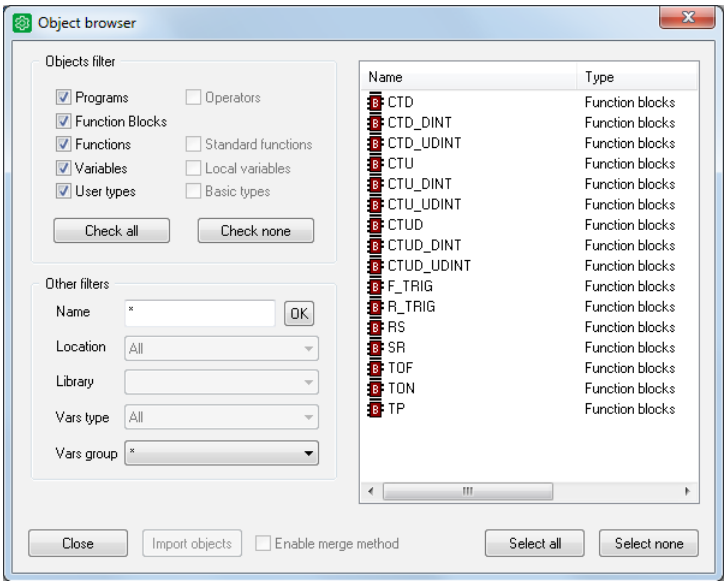
So far, it is not possible to undo export to a library. The only possibility to remove an object is to create another library containing all the objects of the current one, except the one you wish to delete.

4.8.3 IMPORTING FROM A LIBRARY OR ANOTHER SOURCE

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. Therefore, you can edit imported objects, unlike objects of included libraries.

There are two ways of getting a POU from a library. The following procedure shows you how to import objects from a library.

1) Click *Project>Import object from library*. 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* file, open it either by double-clicking it or by pressing the *Open* button. The dialog box of the library explorer appears in foreground. Each tab in the dialog box contains a list of objects of a type consistent with the tab's title.



3) Select the tab of the type of the object(s) you want to import. You can also make simple queries on the objects in each tab by using *Filters*. However, note that only the *Name* filter actually applies to libraries. To use it, select a tab, then enter the name of the desired object(s), even using the * wildcard, if necessary.
4) Select the object(s) you want to import, then press the *Import object* button.
5) When you have finished importing objects, press indifferently *OK* or *Cancel* to close the *Library* browser.

4.8.3.1 UNDOING IMPORT FROM A LIBRARY

When you import an object in a Application project, you actually make a local copy of that object. Therefore, you just need to delete the local object in order to undo import.

4.8.3.2 MERGE FUNCTION

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

By setting the corresponding environment options (see Paragraph 3.6 for more details) you can choose the behavior that Application should keep when encountering those problems.

The possible actions are:

		Ask	Automatic	Take from library	Do nothing
Naming behavior	If different types	X	X		X
	If same type but not variables	X	X	X	
	If both variables	X	X	X	
Address behavior	If address overlaps	X	X	X	
	Copy/paste mapped variable		X		X

- *Ask* (default): user has to decide every time an action is required.
- *Automatic*: a valid name or address is automatically generated by Application 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, Application generates a log file in the project folder with detailed info.

4.8.4 UPDATING EXISTING LIBRARIES

If you edit a linked library file you can refresh its content on the project without closing Application.

1) Click *Project>Refresh all libraries*.
2) If the file is correct, Application updates the linked library content and prints a successful message in the output window, otherwise no changes are made on the existing linked library.

5. MANAGING PROJECT ELEMENTS

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

5.1 PROGRAM ORGANIZATION UNITS

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

This paragraph shows you how to add new POUs to the project, how to edit and eventually remove them.

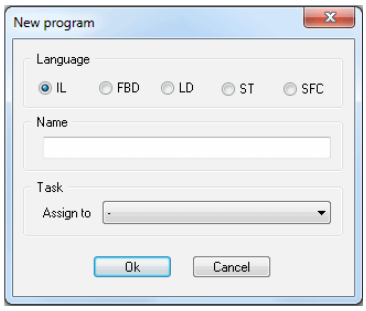
5.1.1 CREATING A NEW PROGRAM ORGANIZATION UNIT

In order to Add a POU select the appropriate voice of the menu

Project>New Object>New program

Please note that the item of the sub-menu may change according to the type of the POU you want to create.

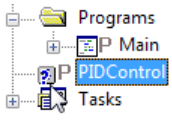
Application will show you a dialog box in where you must select the specific language for the new POU and enter its name.



Confirm the operation by clicking on the OK button.

Alternatively, you can create a new POU from the context menu by selecting a folder or the root element of the project (see Paragraph 5.7.4).

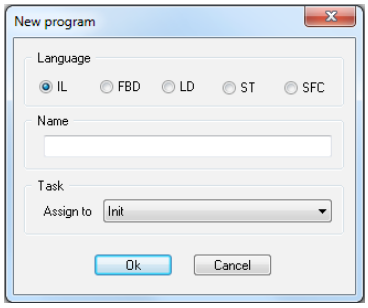
After creating a new program, an alert icon (interrogation mark) appears below the new program icon.



This alert icon indicates that the program is not yet associated to a task. Refer to paragraph 5.3.1 to assign the program to the desired task.

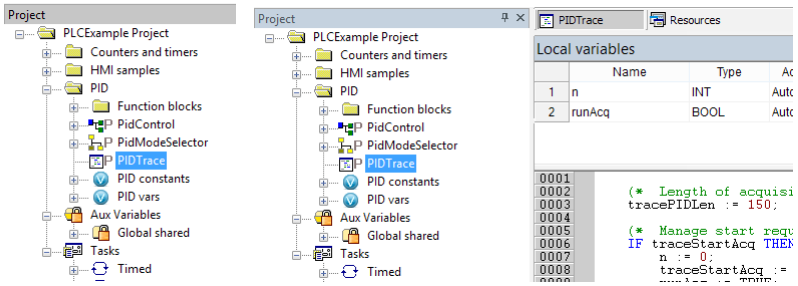
5.1.1.1 ASSIGNING A PROGRAM TO A TASK AT CREATION TIME

When creating a new program, Application gives you the chance to assign that program to a task at the same time: select the task you want the program to be assigned to from the list shown in the *Task* section of the *New program* window.



5.1.2 EDITING POUS

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 choose the appropriate voice of the menu *Project > PLC Object Properties*. Please note that the menu voice may change according to the type of the selected POU.

Duplicating a POU:

Select a POU from the project tree then choose the appropriate voice of the menu *Project>Duplicate object*. Please note that the menu voice may change according to the type of the selected POU.

Enter the name of the new duplicated POU and confirm the operation.

Deleting POUs

Select a POU from the project tree then choose the appropriate voice of the menu *Project>Delete Object*.

Please note that the item of the sub-menu may change according to the type of the POU you have selected.

Confirm the operation to delete the POU.

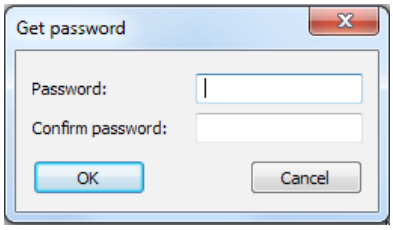
5.1.3 SOURCE CODE ENCRYPTION/DECRYPTION

Application can encrypt POUs and protect them with a password, hiding the source code of the POU.

Encrypting a POU:

Select a POU from the project tree then choose the *[Crypt]* voice of the context menu

Double enter the password and confirm the operation.



Application shows in the project tree a special marker icon that overlays the standard POU icon in order to inform the user that the POU is crypted.

Decrypting a POU:

Select a POU from the project tree then choose the *[Decrypt]* voice of the context menu

Encrypting all POUs:

Select the root element from the project tree then choose the *[Crypt all objects]* voice of the context menu.

All POUs will be encrypted with the same password.

Decrypt all POUs:

Select the root element from the project tree then choose the *[Decrypt all objects]* voice of the context menu.

5.2 VARIABLES

There are two classes of variables in Application: global variables and local variables.

This paragraph shows you how to add to the project, edit, and eventually remove both global and local variables.

5.2.1 GLOBAL VARIABLES

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

5.2.1.1 CLASSES OF GLOBAL VARIABLES

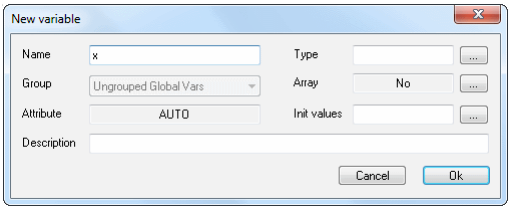
Global variables are organized in special folders of the project tree called *Global variables group*. Those variables are classified according to their properties as:

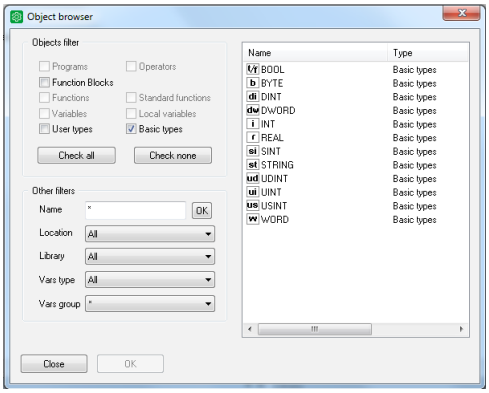
- Automatics: the compiler automatically allocates them to an appropriate location in the target device memory.
- Mapped: they have an assigned address in the target device logical addressing system, which shall be specified by the developer.
- Constants: are declared having the `CONSTANT` attribute; They cannot be written.
- Retains: they are declared having the `RETAIN` attribute; Their values are stored in a persistent memory area of the target device.

5.2.1.2 CREATING A NEW GLOBAL VARIABLE

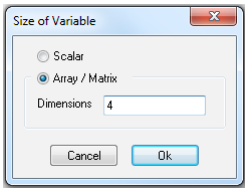
1) In order to create a new global variable you need to define almost one *Global variables group* in your project then select it from the project tree then choose the appropriate voice of the menu *Project>New Object>New variable* (see Paragraph 5.7.4).
2) Application will show you a dialog box where you must enter the name of the variable (remember that some characters, such as '?', '.', '/', and so on, cannot be used: the variable name must be a valid IEC 61131-3 identifier).

3) Specify the type of the variable either by typing it or by selecting it from the list that Application displays when you click on the *Browse* button.

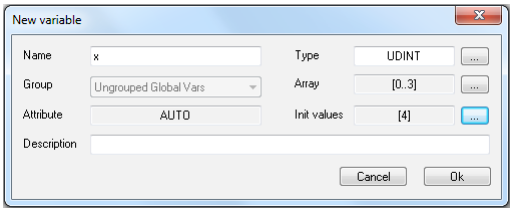


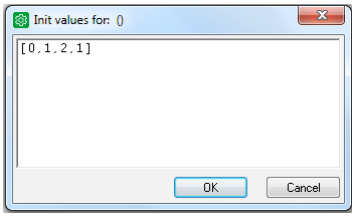


If you want to declare an array, you must specify its size.



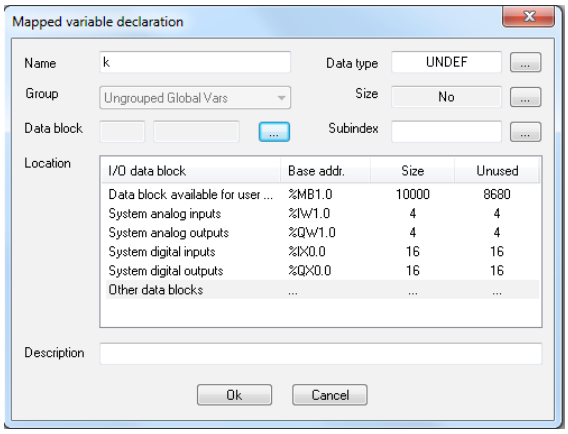
4) You may optionally assign the initial value to the variable or to the single elements of the array.

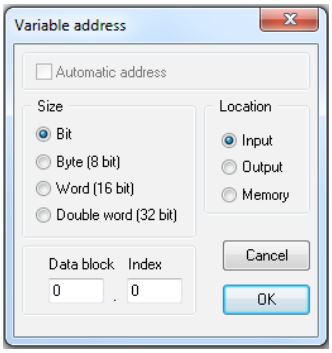




If you create a new mapped variable, you are required to specify the address of the variable during its definition. In order to do so, you may do one of the following actions:

- Click on the button to open the editor of the address, then enter the desired value.

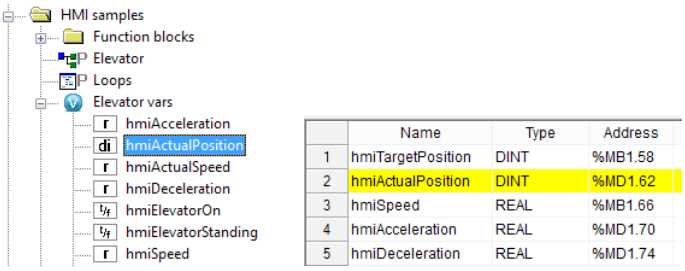




- Select from the list that Application shows you the memory area you want to use: the tool automatically calculates the address of the first free memory location of that area.

5.2.1.3 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.



Changing the name of the variable:

Select the variable you want to rename from the project tree then choose the appropriate voice of the menu *Project>View PLC Object Properties*.

Duplicating a variable:

Select the variable you want to duplicate from the project tree then choose the appropriate voice of the menu *Project>Duplicate variable*.

Enter the name of the new duplicated variable and confirm.

5.2.1.4 DELETING A GLOBAL VARIABLE

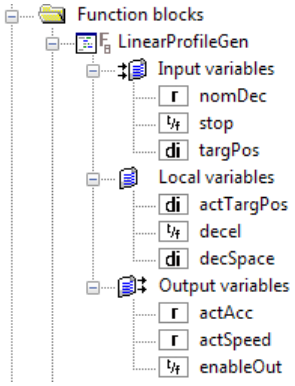
Select the variable you want to delete from the project tree then choose the appropriate voice of the menu *Project>Delete variable*.

Confirm the operation to delete the variable.

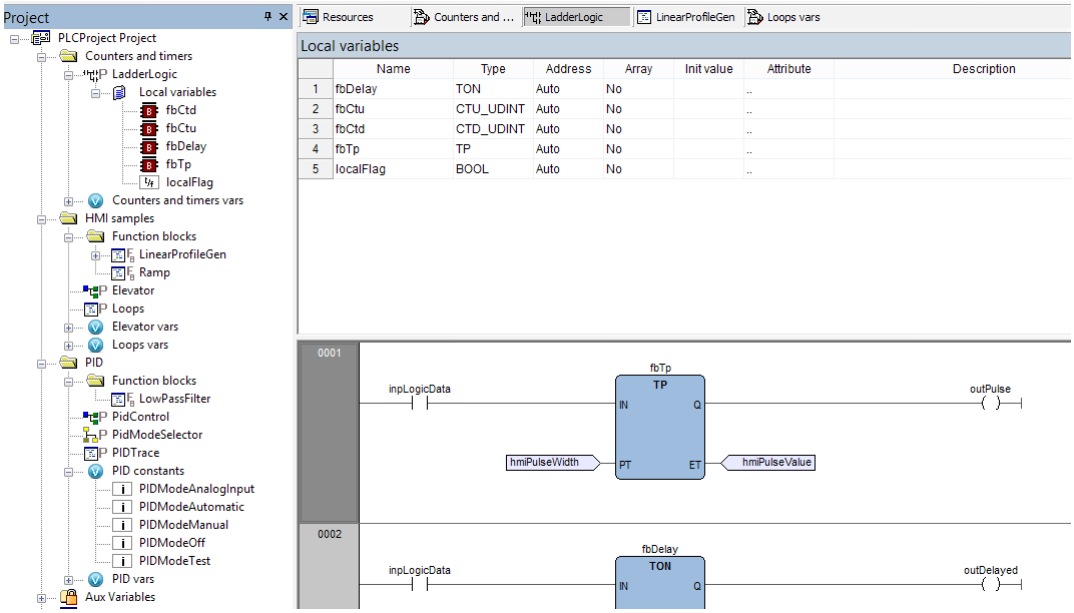
5.2.2 LOCAL VARIABLES

Local variables are declared within a POU (either program, or function, or function block), the module 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), where they are further classified according to their class (e.g., as input or inout variables).



In order to create, edit, and delete local variables, you have to open the Program Organization Unit 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 changes applied to the local variables.



Refer to the corresponding section in this manual for details (see Paragraph 6.6.1.2).

5.2.3 CREATING MULTIPLE

Application allows you to create multiple variables in one shot.

Open the POU for editing then choose the appropriate voice of the menu *Variables>Create multiple*.

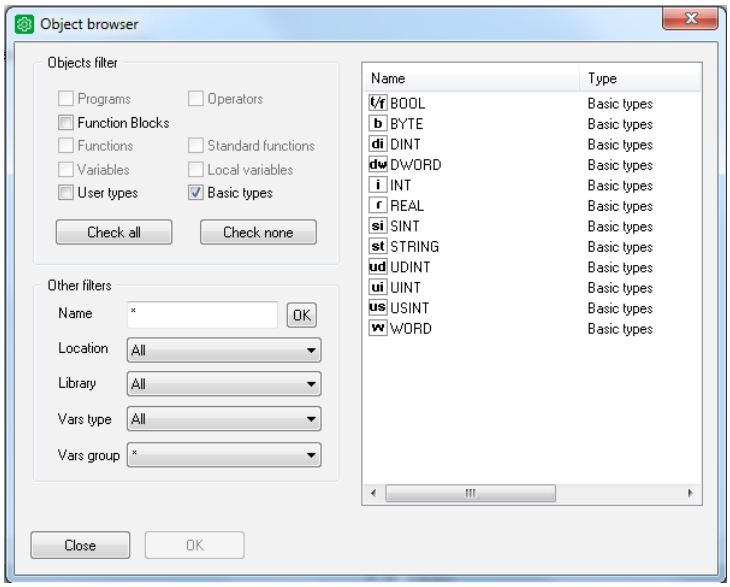
Application will show you a dialog box where you must specify the prefix and the suffix to name of the new variables.

1) Select the type of the variables.
2) 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.

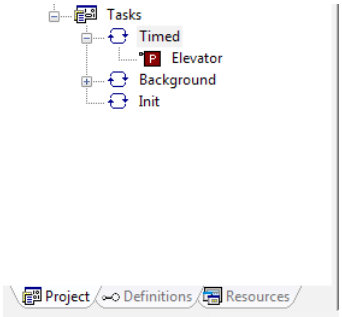
5.3 TASKS

5.3.1 ASSIGNING A PROGRAM TO A TASK

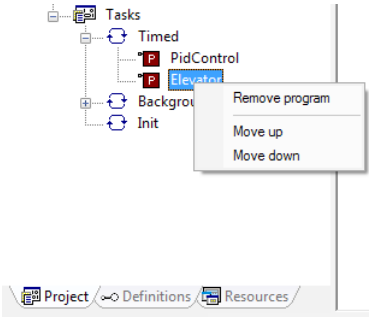
1) Select the task where you want to add the program from the project tree then choose the *[Add program]* voice of the context menu.
2) Select the program you want to be executed by the task from the list which shows up and confirm your choice.



3) The program has been assigned to the task, as you can see in the project tree.



Note that you can assign more than a program to a task. From the contextual menu you can sort and, eventually, remove program assignments to tasks.

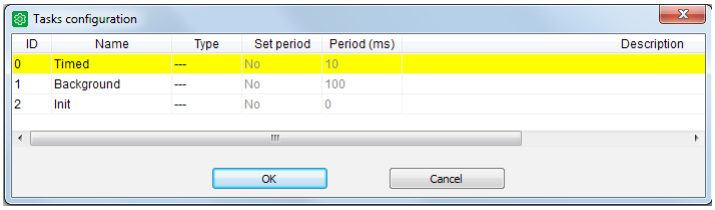


5.3.2 TASK CONFIGURATION

Depending on the target device you are interfacing with, you may have the chance to configure some of the PLC tasks' settings.

Select the tasks element from the project tree then choose the *[Task configuration]* voice of the context menu.

In the *Task configuration* window you can edit the task execution period.



5.4 DERIVED DATA TYPES

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 made up of primitive data types.

User has the flexibility to create those own types that have advanced properties and uses far beyond those of the basic primitive data types.

5.4.1 TYPEDEFS

The following paragraphs show you how to manage Typedefs.

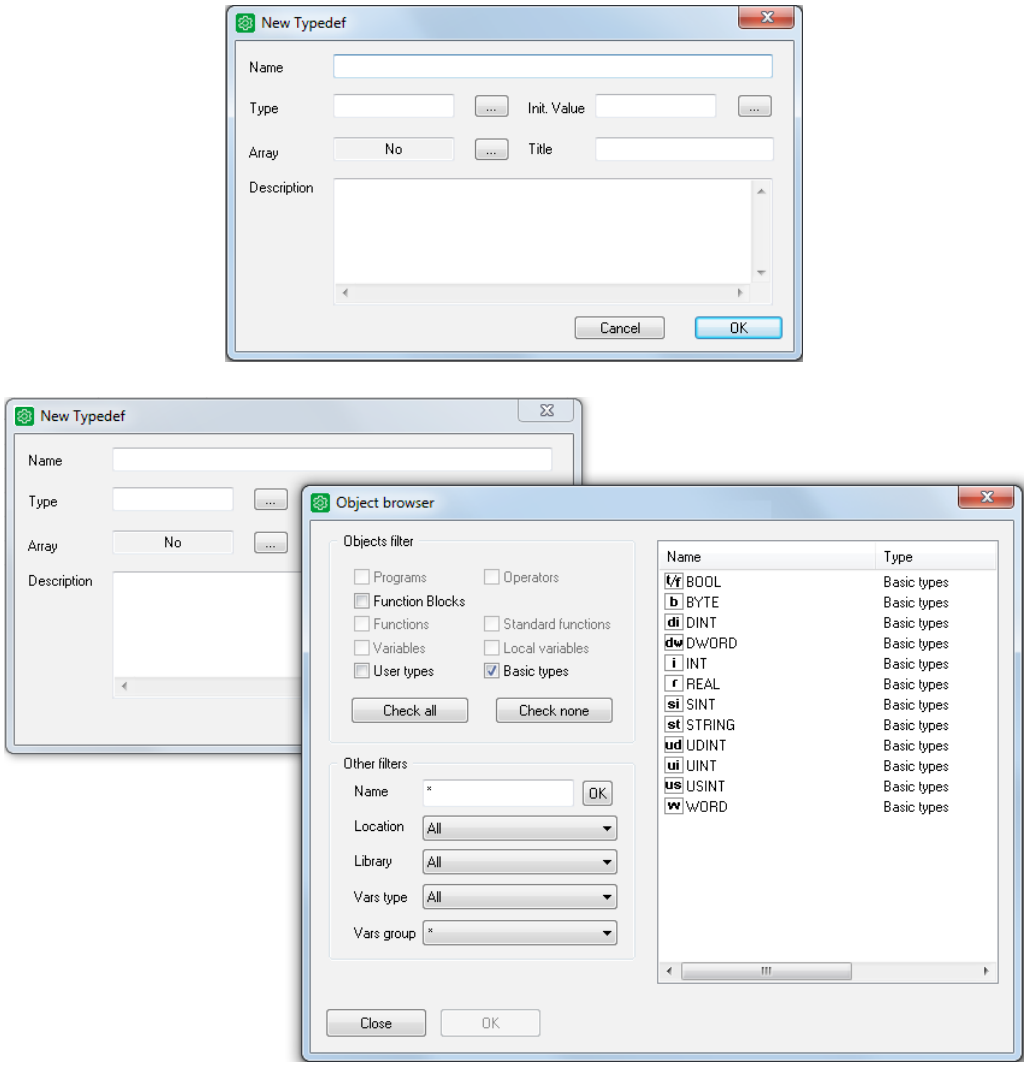
Application can manage Typedefs, Structures, Enumeration and Subranges.

In order to create, edit or delete those data types, use the *Definitions* section of the *Worskpace* window.

5.4.1.1 CREATING A NEW TYPEDEF

In order to create a Typedef select *TypeDefs* folder item in the *Definitions* tree then choose the *[New TypeDef]* voice of the context menu.

Application will show you a dialog box where you must specify the name of the new typedef and select the type you are defining an alias for:

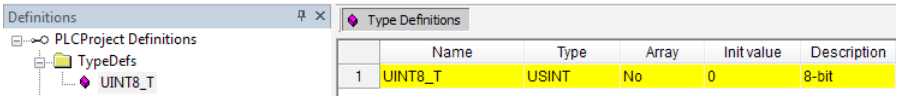


(if you want to define an alias for an array type, you shall choose the array size).

Enter a meaningful description (optional) and confirm the operation.

5.4.1.2 EDITING A TYPEDEF

In order to edit an existing typedef you have to double-click it from the *Definitions* tree. The associated editor opens and lets you modify its definition.



5.4.1.3 DELETING A TYPEDEF

In order to delete a Typedef select it from the *Definitions* tree then choose the *[Delete]* voice of the context menu.

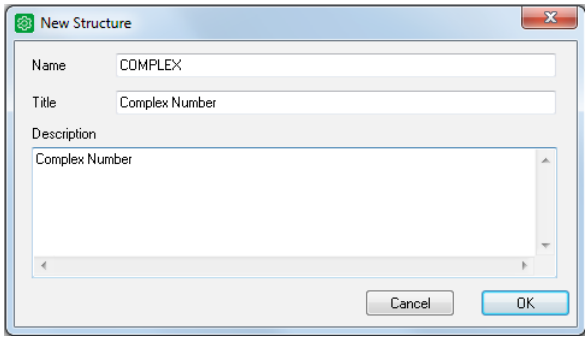
5.4.2 STRUCTURES

The following paragraphs show you how to manage structures.

5.4.2.1 CREATING A NEW STRUCTURE

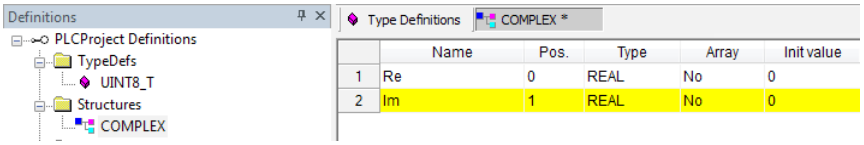
In order to create a Structure select *Structure* folder item in the *Definitions* tree then choose the *[New Structure]* voice of the context menu.

Application will show you a dialog box where you must specify the name of the new structure and a meaningful description, then confirm the operation.



5.4.2.2 EDITING A STRUCTURE

In order to edit an existing structure, open it by double-clicking it from the *Definitions* tree. The associated editor opens and lets you modify its definition and fields.



5.4.2.3 DELETING A STRUCTURE

In order to delete an existing structure select it from *Structures* folder item in the *Definitions* tree then choose the *[Delete]* voice of the context menu.

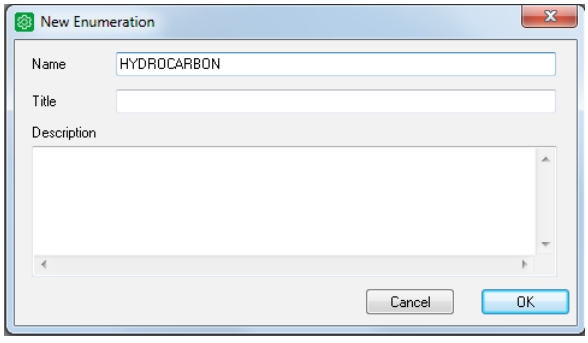
5.4.3 ENUMERATIONS

The following paragraphs show you how to manage enumerations.

5.4.3.1 CREATING A NEW ENUMERATION

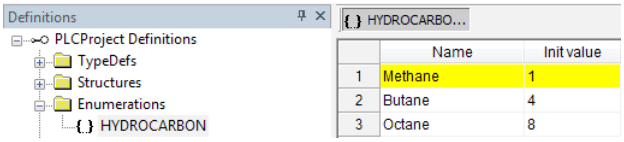
In order to create an enumeration select *Enumerations* folder item in the *Definitions* tree then choose the *[New Enumeration]* voice of the context menu.

Application will show you a dialog box where you must specify the name of the new enumeration and a meaningful description, then confirm the operation.



5.4.3.2 EDITING AN ENUMERATION

In order to edit an existing enumeration, open it by double-clicking it from the *Definitions* tree. The associated editor opens and lets you modify its definition and the initialization values of its elements.



5.4.3.3 DELETING AN ENUMERATION

In order to delete an existing enumeration select it from *Enumeration* folder item in the *Definitions* tree then choose the *[Delete]* voice of the context menu.

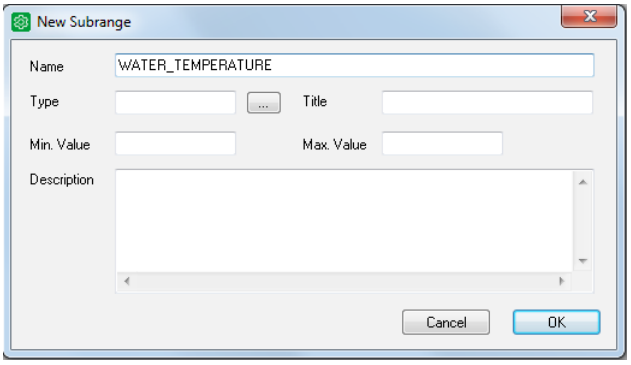
5.4.4 SUBRANGES

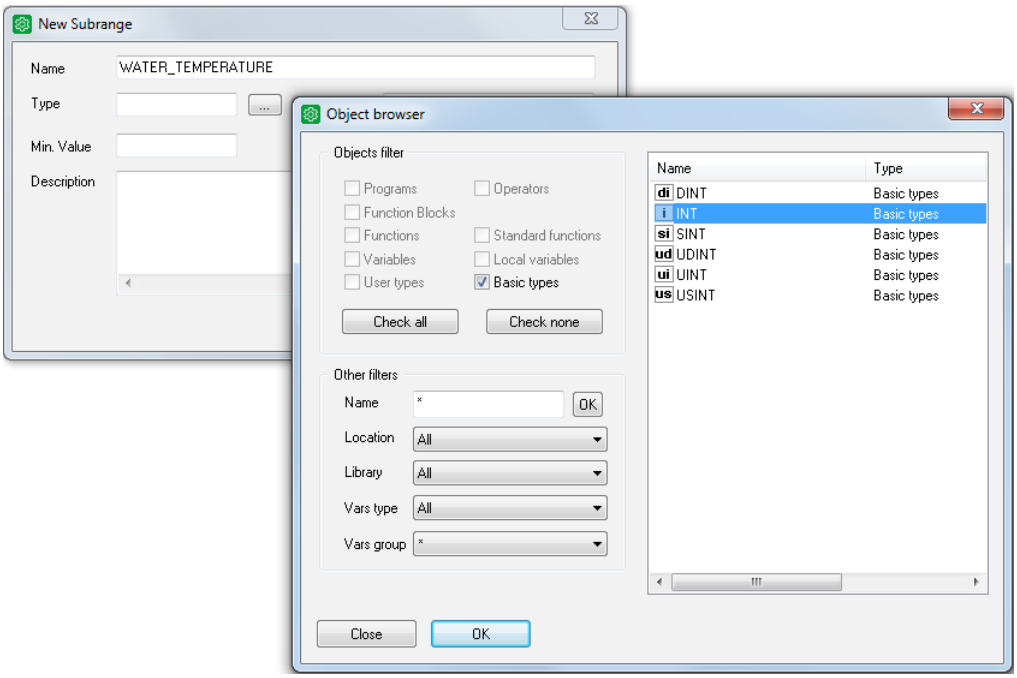
The following paragraphs show you how to manage subranges.

5.4.4.1 CREATING A NEW SUBRANGE

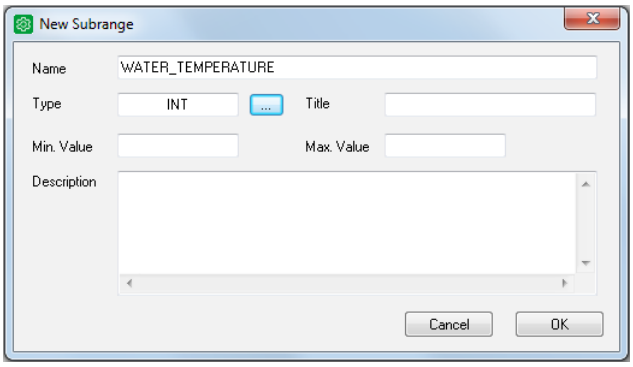
In order to create a subrange select *Subranges* folder item in the *Definitions* tree then choose the *[New Subrange]* voice of the context menu.

Application will show you a dialog box where you must specify the name of the new subrange and select its basic type.



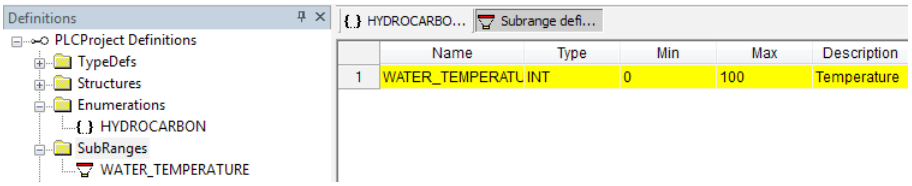


Enter minimum and maximum values of the subrange, a meaningful description (optional), and confirm the operation.



5.4.4.2 EDITING A SUBRANGE

In order to edit an existing subrange, open it by double-clicking it from the *Subranges* folder of the *Definitions* tree. The associated editor opens and lets you modify its definition.



5.4.4.3 DELETING A SUBRANGE

In order to delete an existing subrange select it from *Subranges* folder item in the *Definitions* tree then choose the *[Delete]* voice of the context menu.

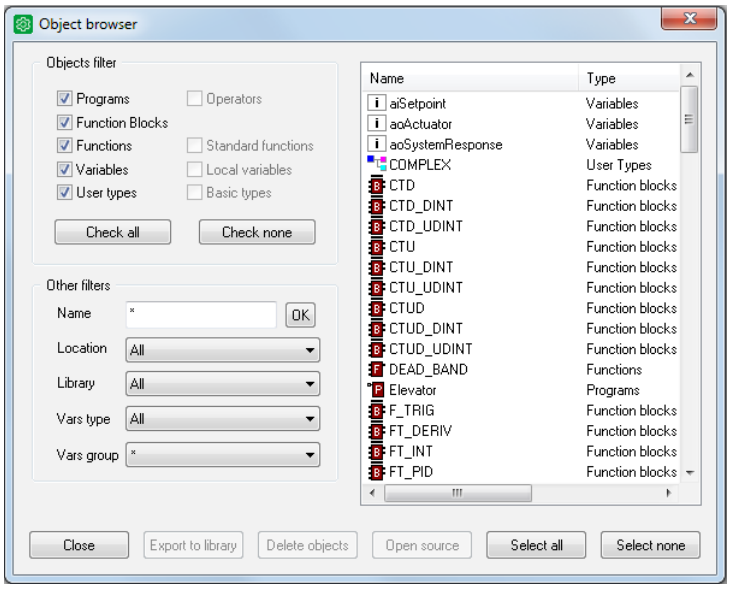
5.5 BROWSE THE PROJECT

Projects may grow huge, hence Application provides two tools to search for an object

within a project: the *Object browser* and the *Find in project* feature.

5.5.1 OBJECT BROWSER

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



This tool is context dependent, this implies that the kind of objects that can be selected and that the available operations on the objects in the different contexts are not the same.

Object browser can be opened in these three main ways:

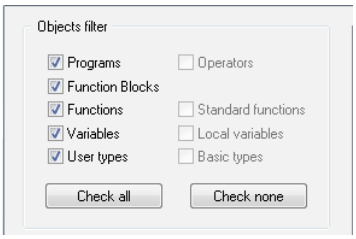
- *Browser* mode.
- *Import object* mode.
- *Select object* mode.

User interaction with *Object browser* is mainly the same for all the three modes and is described in the next paragraph.

5.5.1.1 COMMON FEATURES AND USAGE OF OBJECT BROWSER

This section describes the features and the usage of the *Object browser* that are common to every mode in which *Object browser* can be used.

Objects filter

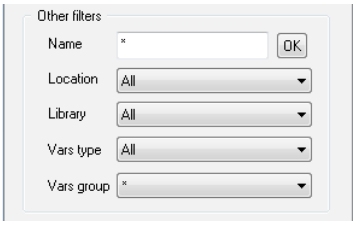


This is the main filter of the *Object browser*. User can check one of the available (enabled) object items.

In this example, *Programs*, *Function Blocks*, *Functions* are selected, so objects of this type are shown in the object list. *Variables* and *User types* objects can be selected by user but objects of that type are not currently shown in the object list. *Operators*, *Standard functions*, *Local variables*, and *Basic types* cannot be checked by user (because of the context) so cannot be browsed.

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

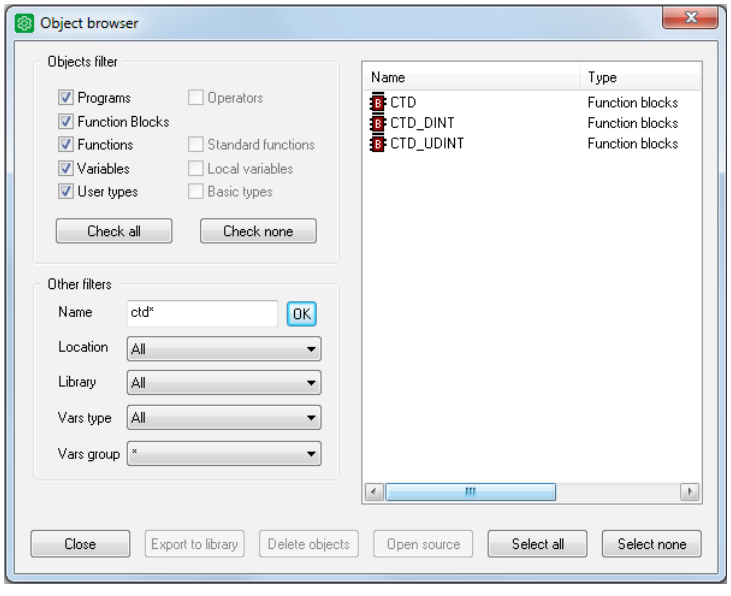
Other filters



Selected objects can be also filtered by name, symbol location, specific library and var type.

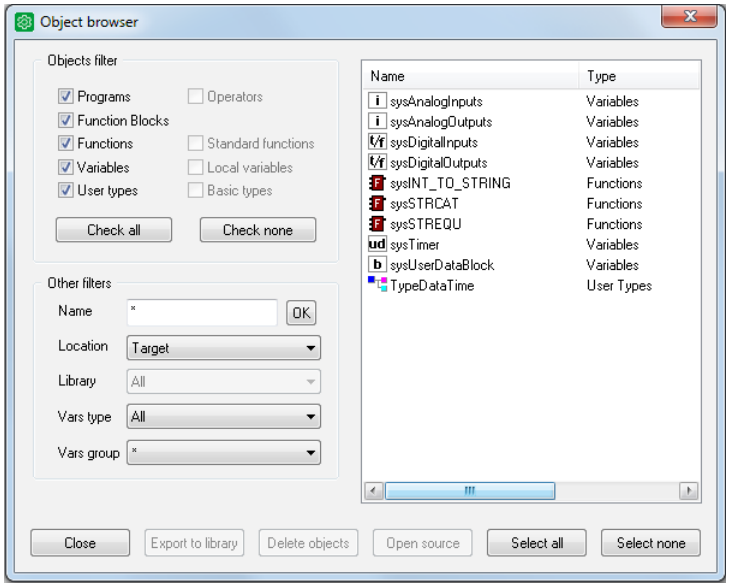
Filters are all additive and are immediately applied after setting.

Name	
Function	Filters objects on the base of their name.
Set of legal 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 on the *OK* button near the edit box to apply the filter.
Applies to	All object types.

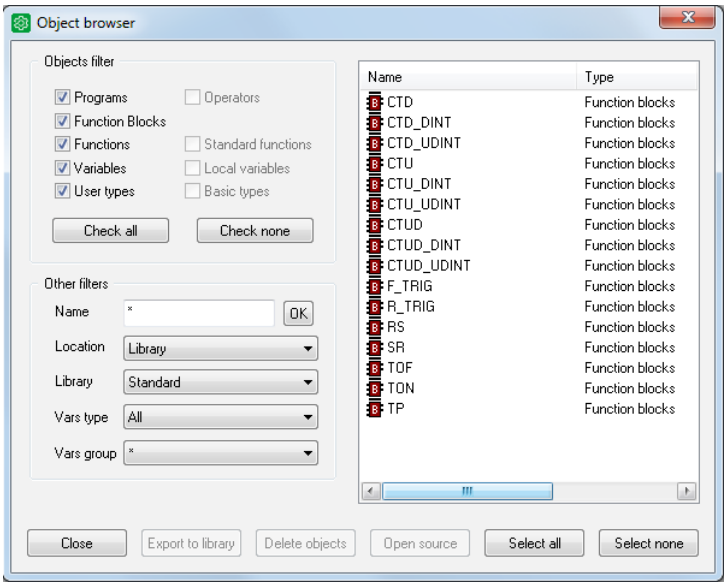


Symbol location	
Function	Filters objects on the base of their location.
Set of legal values	All, Project, Target, Library, Aux. Sources.

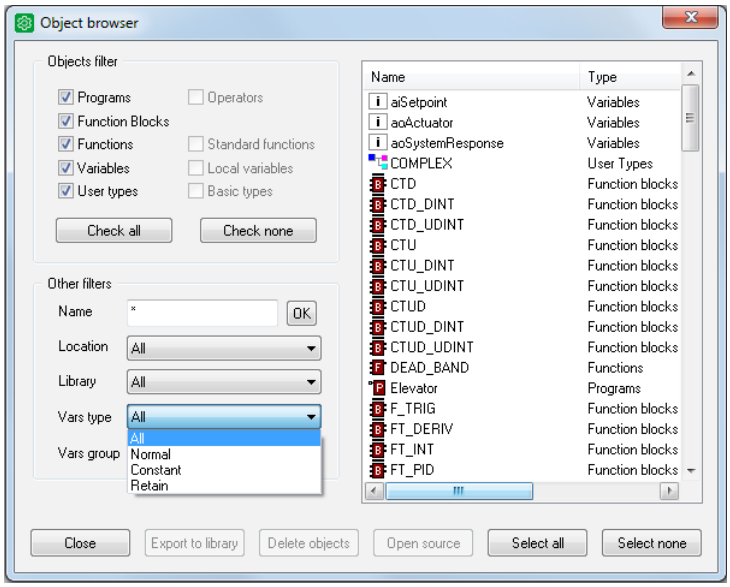
Use	All= Disables this filter. Project= Objects declared in the Application project. Target= Firmware objects. Library= Objects contained in a library. In this case, use simultaneously also the *Library* filter, described below. Aux sources= Shows aux sources only.
Applies to	All objects types.



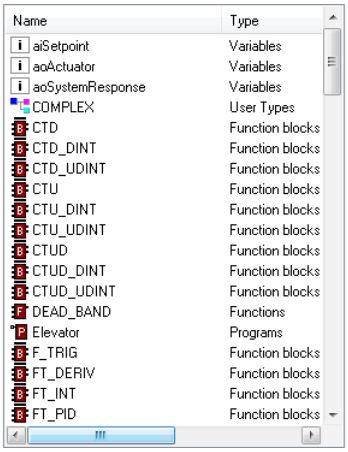
Library	
Function	Completes the specification of a query on objects contained in libraries. The value of this control is relevant only if the *Symbol location* filter is set to *Library*.
Set of legal values	All, libraryname1, libraryname2, ...
Use	All= Shows objects contained in whatever library. LibrarynameN= Shows 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.
Set of legal values	All, Normal, Constant, Retain
Use	All= Shows all the global and system variables. Normal= Shows normal global variables only. Constant= Shows constants only. Retain= Shows retain variables only.
Applies to	Variables.



Object list



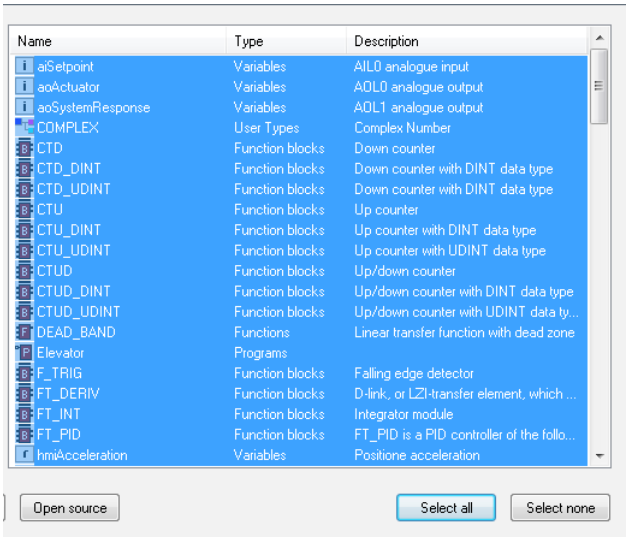
Object list shows all the filtered objects. List can be ordered in ascending or descending way by clicking on the header of the column. So it is possible to order items by *Name*, *Type*, or *Description*.

Double-clicking on an item allows the user to perform the default associated operation (the action is the same of 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 on *Select all* button. *Select none* deselects all objects.

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



Resize

Window can be resized, the cursor changes along the border of the dialog and allows the user to resize window. When reopened, *Object browser* dialog takes the same size and position of the previous usage.

5.5.1.2 USING OBJECT BROWSER AS A BROWSER

In order to use the object browser to simply look over through the element of the project choose the appropriate voice of the menu *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 *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 so they are unchecked and disabled).

Available operations

Allowed operations in this mode are:

Open source, default operation for double-click on 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 on the *Open source* button.

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

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

Multi selection

Multi selection is allowed for this mode, *Select all* and *Select none* buttons are visible.

5.5.1.3 USING OBJECT BROWSER FOR IMPORT

Object browser is also used to support objects importation in the project from a desired external library.

In order to use the object browser to import external library to the project choose the appropriate voice of the menu *Project>Import object from library*.

Available objects

In this mode you can list objects of these types:

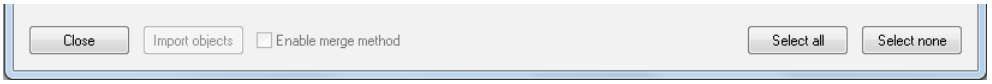
- Programs.
- Function blocks.
- Functions.

- 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 on *Import objects* button or by double-clicking on one of the objects in the list.

Multi selection

Multi selection is allowed for this mode, *Select all* and *Select none* buttons are visible.

5.5.1.4 USING OBJECT BROWSER FOR OBJECT SELECTION

Object browser dialog is useful for many operations that requires the selection of a single PLC object. So 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, etc..

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 programs objects.

It is possible that not all available objects are selected by default.

Available operations

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

Multi selection

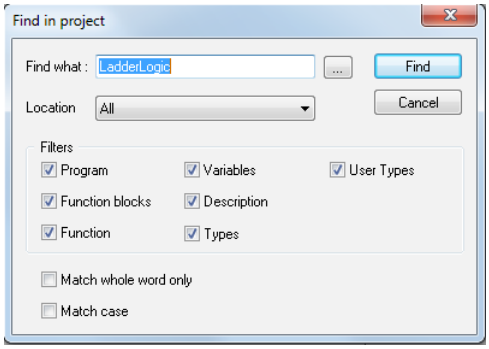
Multi selection is not allowed for this mode, *Select all* and *Select none* buttons are not visible.

5.5.2 SEARCH WITH THE FIND IN PROJECT COMMAND

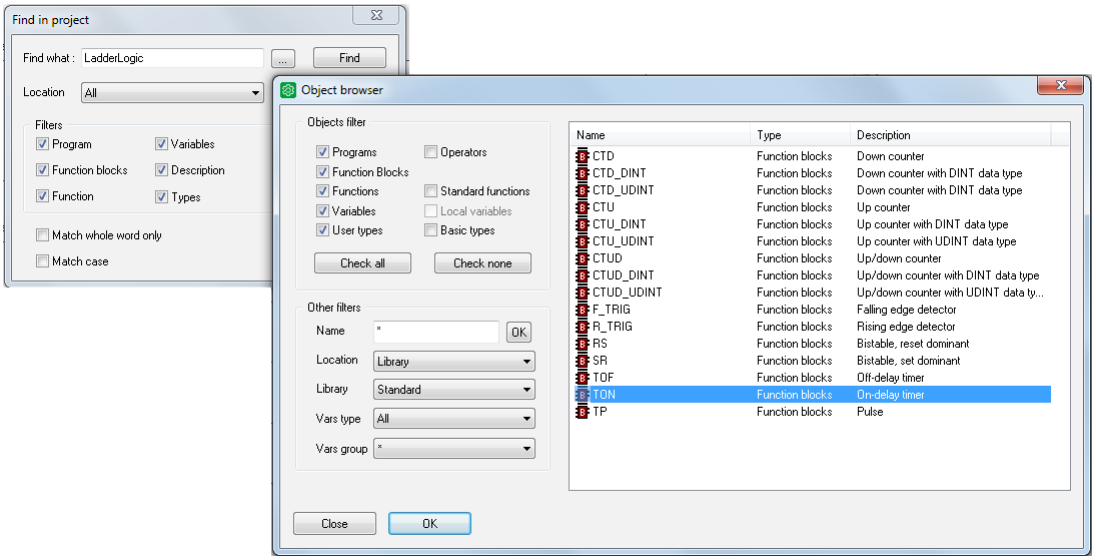
The *Find in project* command retrieves all the instances of a specified character string in the project.

In order to use this functionality choose the appropriate voice of the menu *Edit > Find in project* .

Application will show you the following dialog box:

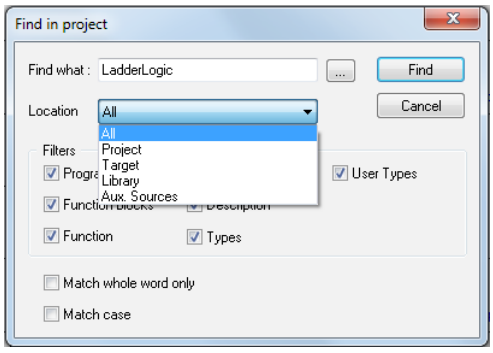


1) In the *Find what* text box, type the name of the object you want to search.



Otherwise, click the *Browse* button to the right of the text 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* combo box, so as to specify a constraint on the location of the objects to be inspected.

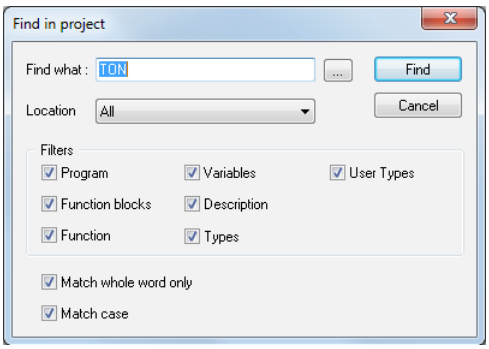


3) The frame named *Filters* contains 7 checkboxes, each of which, if ticked, enables research of the string among the object it refers to.

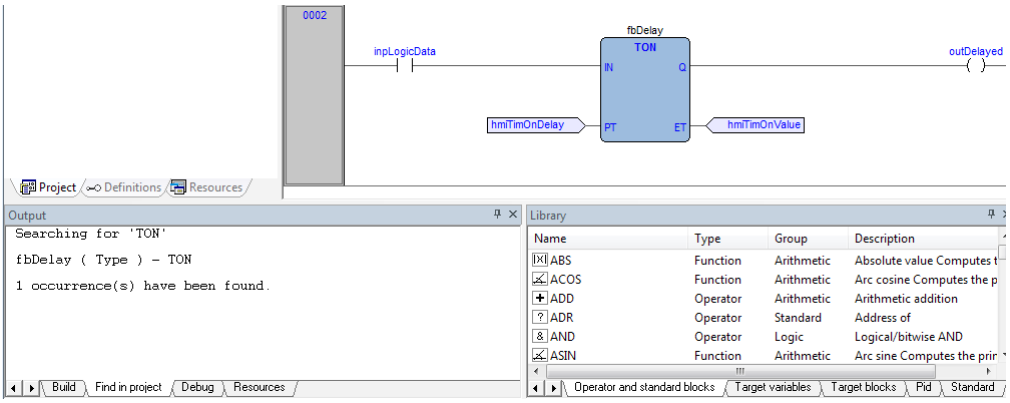
4) Tick *Match whole word only* if you want to compare your string to entire word only.

5) Tick *Match case* if you want your search to be case-sensitive.

6) Press *Find* to start the search, otherwise click *Cancel* to abandon.



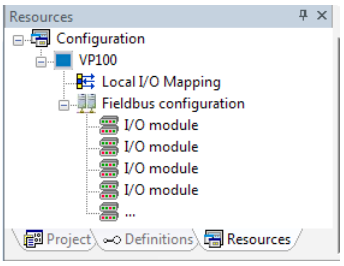
The results will be printed in the *Find in project* tab of the *Output* window.



5.6 WORKING WITH APPLICATION EXTENSIONS

Application's *Workspace* window may include a section whose contents completely depend on the target device the IDE is interfacing with: the *Resources* panel.

If the *Resources* panel is visible, you can access some additional features related to the target device (configuration elements, schemas, wizards, and so on).

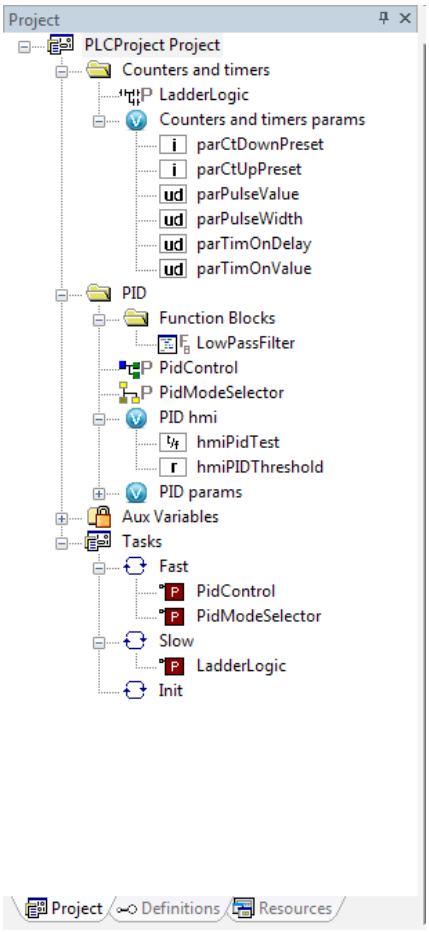


Information about these features may be found in a separate document: refer to your hardware supplier for details.

5.7 PROJECT CUSTOM WORKSPACE

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

All organizationals units of the custom workspace are logical: creating and editing those units will no triggers any effects on the PLC code.



5.7.1 ENABLE THE CUSTOM WORKSPACE INTO AN EXISTING PROJECT

To enable this feature see the *Project>Options...* (see Paragraph 4.6.1), once enabled the project needs to be reloaded.

By default this features is enabled depending on targets.

5.7.2 WORKSPACES MIGRATION

Whenever this feature is switched, Application tries to reorder the workspace maintaining the user customization by this logic:

Static (old) workspace to custom (new)

Fixed logic units (Ex. Function blocks folder) are converted into new dynamic folders with the same names. Fixed global group units (Ex. Mapped variables) are converted into new global dynamic groups with the same names. All global variables that do not belong to any group will be grouped into a new group called *Ungrouped global vars*.

Custom (new) workspace to static (old)

All custom units will be destroyed and all POUs and global variables will be grouped into the default fixed units (Ex. Function blocks folder and Mapped Variables).

5.7.3 CUSTOM WORKSPACE BASIC UNITS

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.

5.7.4 CUSTOM WORKSPACE OPERATIONS

Different useful operations can be performed in order to give a better 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 choose the *[Add>New folder]* voice of the context menu.

This operation adds a new customizable folder (by default named *New folder*) unit ready to be renamed.

Creating a Global variables Group

In order to create a global variables group select the root item of the project tree or, if you want to nest it, an existing folder the *[Add>New global variables group]* voice of the context menu.

This operation adds a new customizable folder (by default named *New var group*) unit ready to be renamed.

Rename a unit (folder or Global variables group)

In order to rename a global variables group or a folder select it than choose *[Rename]* voice of the context menu.

This operations 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 than choose *[Delete]* voice of the context menu.

If the units contains any child you will be prompted for three possibilities:

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

Export all children to library

In order to export all elements of a global variables group or a folder select it than choose *[Export all children to library]* voice of the context menu.

This operation allows you to export recursively all child elements of the selected item into a library (see Paragraph 4.8.2 for more information about new library).

Moving Unit

You can simply drag&drop 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) (see Paragraph 6.6 for more information about variables editor).

5.7.5 WORKSPACE ELEMENTS WITH LIMITATIONS

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

Root Project Element

You can not move, rename or delete this element. It can contain customizable units as children.

POUs Children Elments

These elements are generated following the structure of the POU they belong to. You can not move, rename or delete these elements directly from the tree. For more information about POUs (see Paragraph 5.1).

SFC Children Elements

These elements follow the aforesaid 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 (see Paragraph 6.5).

Aux Variables Element

You can not move, rename or delete this element and his children. They are automatically generated by Application.

Tasks Element

You can not move, rename or delete these elements. They are automatically generated by Application. For more information about SFC language (see Paragraph 5.3).

6. EDITING THE SOURCE CODE

PLC editors

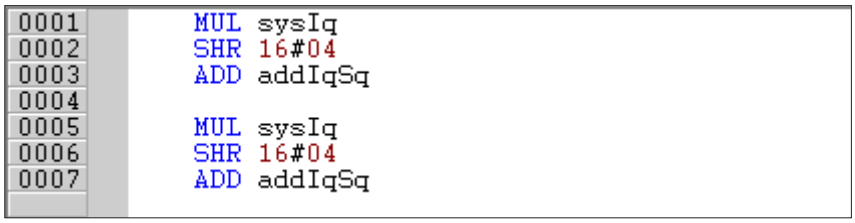
Application includes five source code editors, which support the whole range of IEC 61131-3 programming languages: Instruction List (IL), Structured Text (ST), Ladder Diagram (LD), Function Block Diagram (FBD), and Sequential Function Chart (SFC).

Moreover, Application includes a grid-like editor to support the user in the definition of variables.

All editors, both graphical and text one, support tooltips. By enabling them (see Paragraph 3.6.1.4), Application will show some information about symbols on which the user move the mouse.

This chapter focuses on all these editors.

6.1 INSTRUCTION LIST (IL) EDITOR



The IL editor allows you to code and modify POUs using IL (i.e., Instruction List), one of the IEC-compliant languages.

6.1.1 EDITING FUNCTIONS

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.

6.1.2 REFERENCE TO PLC OBJECTS

If you need to add to your IL 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 standard operators and embedded functions can be dragged from the *Libraries* window, whereas local variables can be selected from the local variables editor.

6.1.3 AUTOMATIC ERROR LOCATION

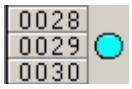
The IL editor also automatically displays the location of compiler errors. To know where a compiler error occurred, double-click the corresponding error line in the *Output* bar.

6.1.4 BOOKMARKS

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.

6.1.4.1 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.



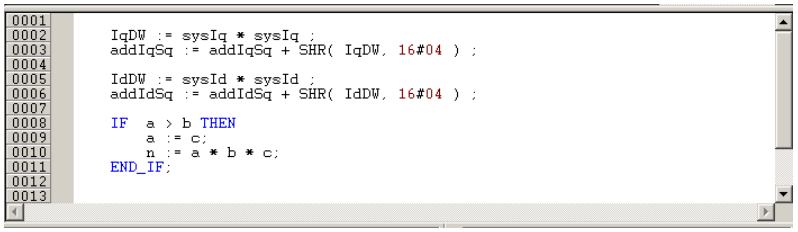
6.1.4.2 JUMPING TO A BOOKMARK

Press *F2* repeatedly, until you reach the desired line

6.1.4.3 REMOVING A BOOKMARK

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

6.2 STRUCTURED TEXT (ST) EDITOR



The ST editor allows you to code and modify POUs using ST (i.e. Structured Text), one of the IEC-compliant languages.

6.2.1 CREATING AND EDITING ST OBJECTS

See the Creating and Editing POUs section (see Paragraphs 5.1.1 and 5.1.2).

6.2.2 EDITING FUNCTIONS

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.

6.2.3 REFERENCE TO PLC OBJECTS

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.

6.2.4 AUTOMATIC ERROR LOCATION

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.

6.2.5 BOOKMARKS

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.

6.2.5.1 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.

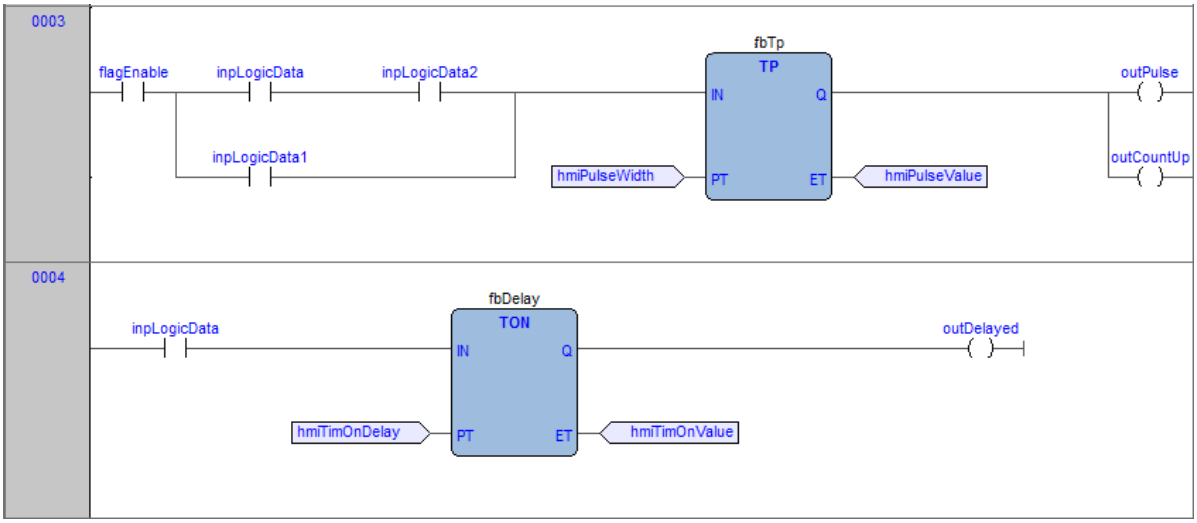
6.2.5.2 JUMPING TO A BOOKMARK

Press *F2* repeatedly, until you reach the desired line.

6.2.5.3 REMOVING A BOOKMARK

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

6.3 LADDER DIAGRAM (LD) EDITOR



The LD editor allows you to code and modify POUs using LD (i.e. Ladder Diagram), one of the IEC-compliant languages.

6.3.1 CREATING A NEW LD DOCUMENT

See the Creating and Editing POUs section (see Paragraphs 5.1.1 and 5.1.2).

6.3.2 ADDING/REMOVING NETWORKS

Each POU coded in LD 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 grey raised button 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) have to be placed.

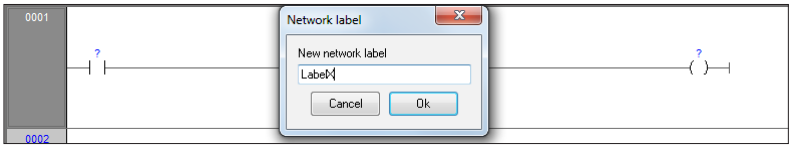
You can perform the following operations on networks:

- To add a new blank network, click *Scheme>Network>New*, or press one of the equivalent buttons in the *Network* toolbar.
- To assign a label to a selected network, give the *Scheme>Network>Label*. This enables jumping to the labeled network.
- To display a background grid which helps you to align objects, click *View>Grid*.
- To add a comment, click *Scheme>Object>New Comment*.

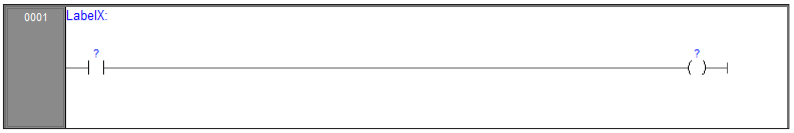
6.3.3 LABELING NETWORKS

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, double-click the raised grey button on the left, which bears the network number.

This causes a dialog box to appear, where you can type the label you want to associate with the selected network.



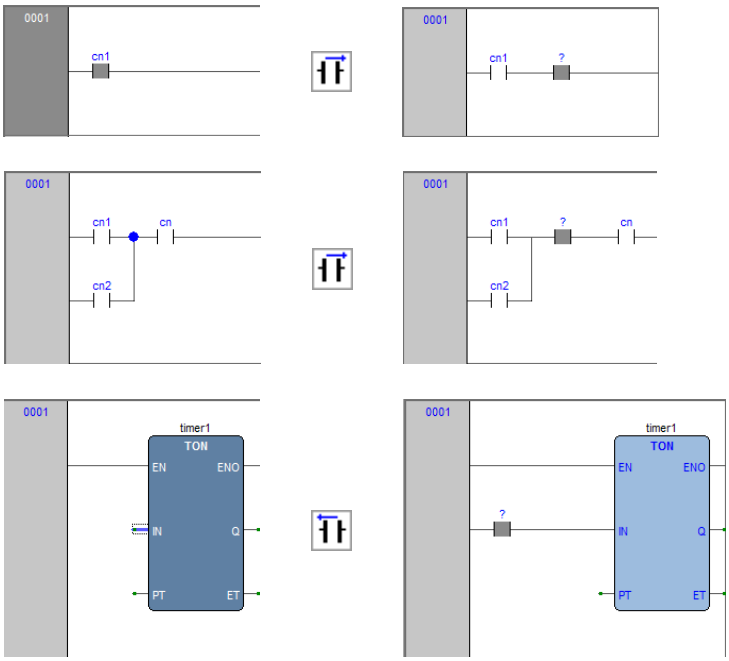
If you press *OK*, the label is printed in the top left-hand corner of the selected network.

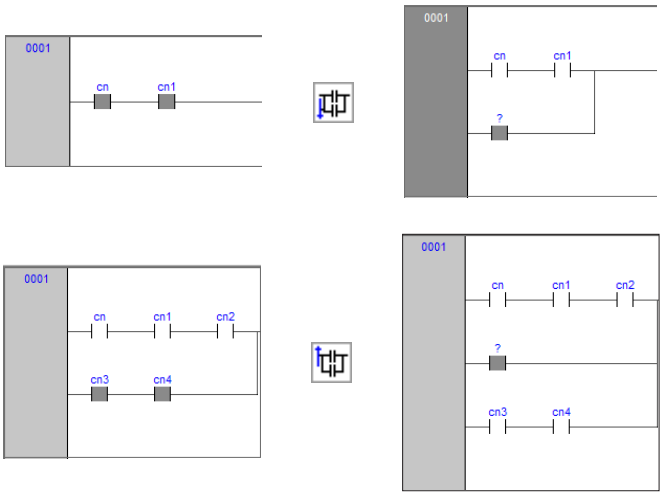


6.3.4 INSERTING CONTACTS

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

- Select a contact, a block, a pin of block, or a connection point, that will act 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 *Sheme>Object>New*. For serial insertion, the new contact will be 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. For parallel insertions, several contacts can be selected before performing the insertion; the new contact will be inserted above or below the group of selected contacts.



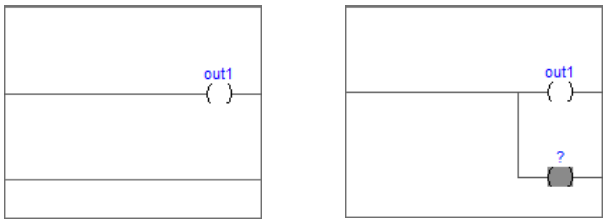


- Drag a boolean variable to the desired place over an object. For example, global variables can be taken from the *Workspace* window, whereas local variables can be selected from the local variables editor. Contacts inserted with drag and drop will always be inserted in series after the destination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

6.3.5 INSERTING COILS

To insert new coils on the network apply one of the following options:

- Click *Scheme>Object>New>Coil*. The new coil will be inserted and linked to the right power rail. If other coils, return or jumps are already present in the network, the new coil will be added in parallel with the previous ones.



- 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.

6.3.6 INSERTING BLOCKS

To insert blocks on the network apply one of the following options:

- Select a contact, connection or block then click *Scheme>Object>New>Block*, which causes a dialog box to appear listing all the objects of the project, then 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 will be connected to the power link (and it will possible to add *EN/ENO* pins later with the provided command); otherwise the *EN/ENO* pins will be 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 can not be inserted on the 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.

6.3.7 EDITING COILS AND CONTACTS PROPERTIES

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 on 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]*.

An apposite dialog box will appear. Select the desired element type from the presented list and then press *OK*.

Otherwise, select the desired contact or coil, and change its type using the six provided buttons in the *LD* toolbar or the six commands in the *Scheme* menu.

6.3.8 EDITING NETWORKS

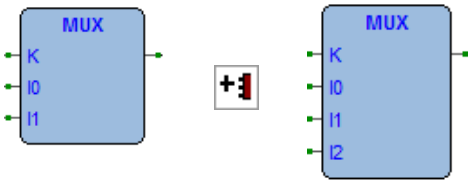
The *LD* editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

- 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 will be 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.

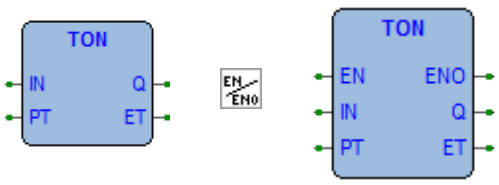
Adding, moving, deleting or copy/pasting objects will automatically recalculate the layout of the network objects; because of this, it is not possible to manually "draw" connection lines or freely placing objects without connecting them to the network.

6.3.9 MODIFYING PROPERTIES OF BLOCKS

- 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.



EN/ENO pins can be removed only if the selected block has at least one *BOOL* input and one *BOOL* output; otherwise, they will be automatically added when creating the block and it will not be possible to remove them (the *Enable EN/ENO pins* command will be disabled).

If a block has more than one *BOOL* output pin, it is possible to choose which pin will bring 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>Instance name*, to change the name of an instance of a function block.

6.3.10 GETTING INFORMATION ON A BLOCK

You can always get information on a block that you added to an LD document, by selecting it and then performing one of the following operations:

- click *Scheme>Object>Open source*, to open the source code of a block.
- Click *Scheme>Object properties* in the menu, to see properties and input/output pins of the selected block.

6.3.11 AUTOMATIC ERROR RETRIEVAL

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.

6.3.12 INSERTING VARIABLES

To connect a variable to an input or output pin of a block apply one of the following options:

- 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*) 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.

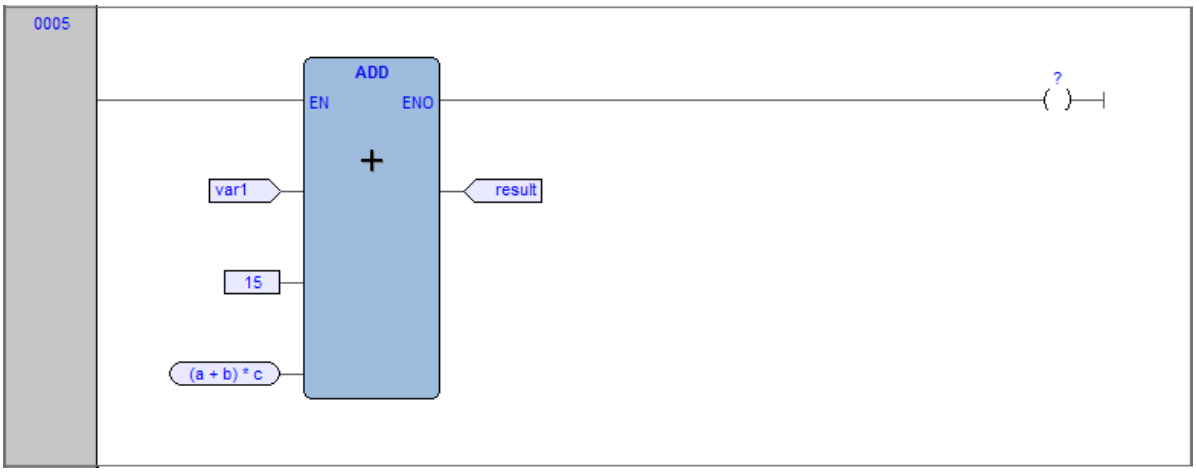
6.3.13 INSERTING CONSTANTS

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*) and enter the numeric constant value.

6.3.14 INSERTING EXPRESSION

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*) and enter any *ST* expression:

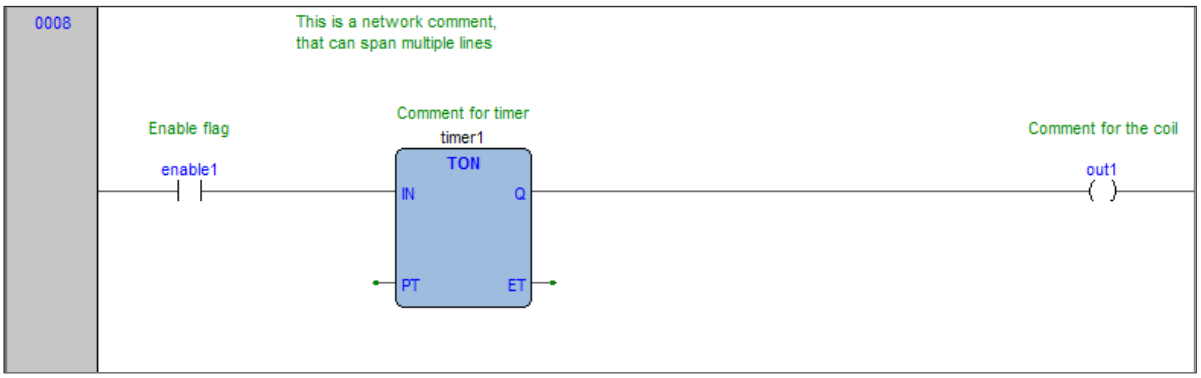
```
(a+b)*c

TO_INT(n)

ADR(x)
```



6.3.15 COMMENTS

It is possible to insert two types of comments:

- *network* comments: activate the network by clicking on 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 will be displayed at the top of the network, and if necessary will be expanded to show all the text lines of the comment.
- *Object* comments: they are activated with the apposite menu command in *View>Show comments for objects*; above any contact, function block or coil the description of the associated PLC variable (if present) will be initially shown, but with the *Comment* command you can modify it to enter a specific object comment that will override the PLC variable description.

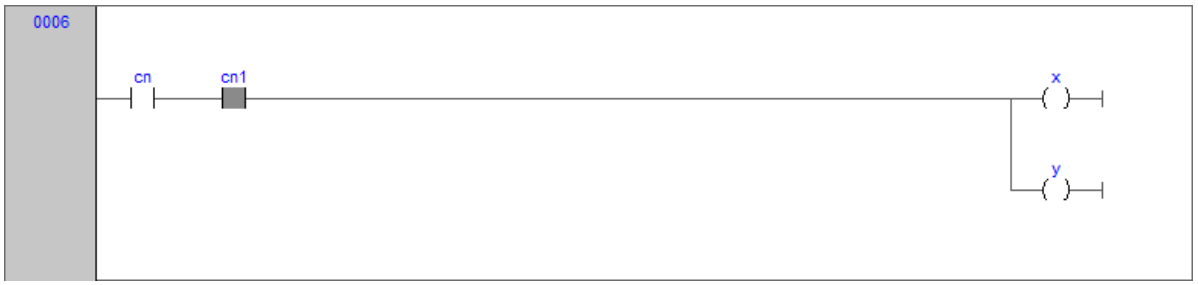


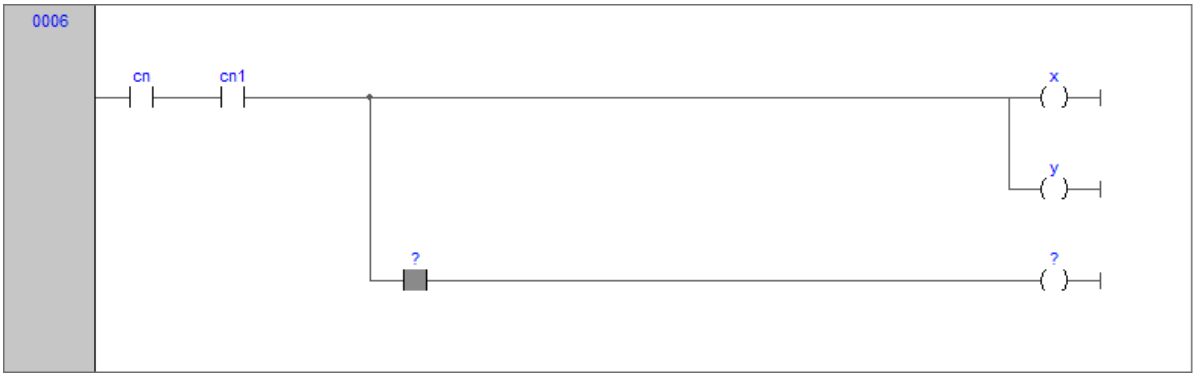
6.3.16 BRANCHES

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.

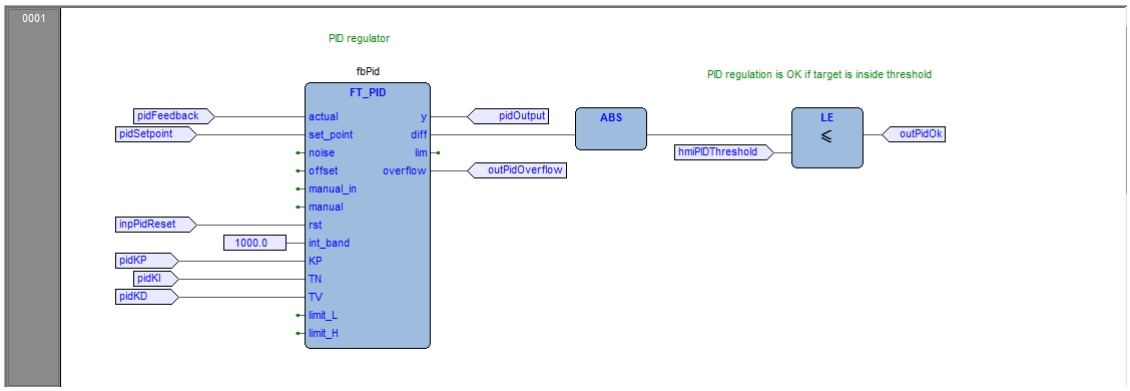
The start of the new branch is marked as a big dot on the source line; deleting all objects on a branch deletes the branch itself.

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.





6.4 FUNCTION BLOCK DIAGRAM (FBD) EDITOR



The FBD editor allows you to code and modify POUs using FBD (i.e. Function Block Diagram), one of the IEC-compliant languages.

6.4.1 CREATING A NEW FBD DOCUMENT

See the Creating and editing POUs section (see Paragraphs 5.1.1 and 5.1.2).

6.4.2 ADDING/REMOVING NETWORKS

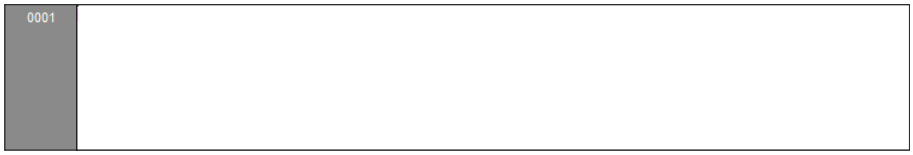
Every POU coded in FBD consists of a sequence of networks. A network is defined as a maximal set of interconnected graphic elements. The upper and lower bounds of every network are fixed by two straight lines, while each network is delimited on the left by a grey raised button containing the network number.

You can perform the following operations on networks:

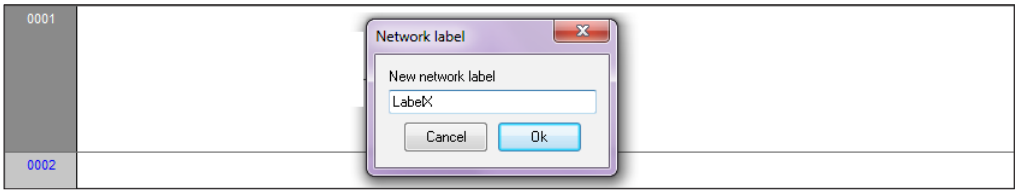
- To add a new blank network, click *Scheme>Network>New*.
- To assign a label to a selected network, give the *Scheme>Network>Label* command. This enables jumping to the labeled network.
- To display a background grid which helps you to align objects, click *View>View greed*.
- To add a comment, click *Scheme>Object>New>Comment*.

6.4.3 LABELING NETWORKS

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, double-click the raised grey button on the left, that bears the network number.



This causes a dialog box to appear, which lets you type the label you want to associate with the selected network.



If you press *OK*, the label is printed in the top left-hand corner of the selected network.

6.4.4 INSERTING AND CONNECTING BLOCKS

This paragraph shows you how to build a network.

Add a block to the blank network, by applying one of the following options:

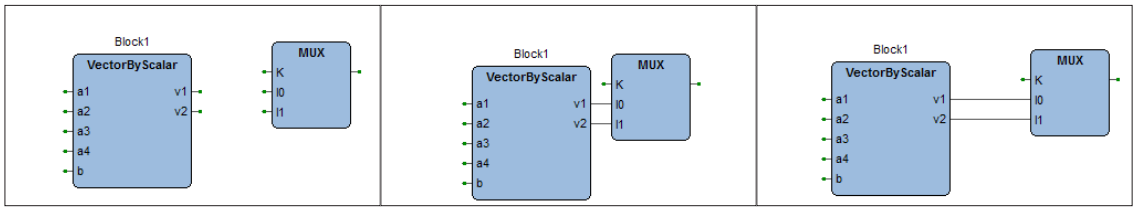
- Click *Scheme>Object>New>Function Block* which causes a dialog box to appear listing all the objects of the project, then choose one item from the list. If the block is a constant, a return statement, or a jump statement, you can directly press the relevant buttons in the *FBD* toolbar.
- Drag the selected object to 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 that will make up the network.

Then connect blocks:

- Click *Edit>Connection mode*, or simply press the space bar of your keyboard. 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.
- If you want to connect two blocks having a one-to-one correspondence of pins, you can enable the auto connection mode by clicking *Scheme>Auto connect*. Then take the two

blocks, drag them close to each other so as to let the corresponding pins coincide. The FBD editor automatically draws the logical wires.



If you delete a block, its connections are not removed automatically, but they become invalid and they are redrawn red. Click *Scheme>Delete invalid connection*.

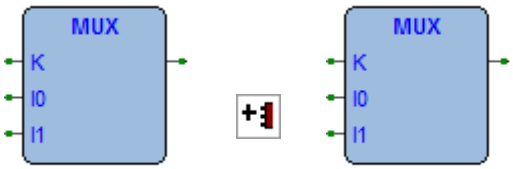
6.4.5 EDITING NETWORKS

The FBD editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

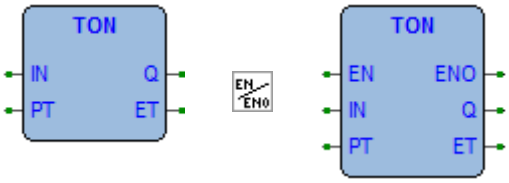
- Selection of a block.
- Selection of a set of blocks by pressing *Shift* + left button and by drawing a frame including the blocks to select.
- *Edit>Cut*, *Edit>Copy*, *Edit>Paste* operations of a single block as well as of a set of blocks.
- Drag-and-drop.

6.4.6 MODIFYING PROPERTIES OF BLOCKS

- 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 change the name of an instance of a function block.

6.4.7 GETTING INFORMATION ON A BLOCK

You can always get information on a block that you added to an FBD document, by selecting it and then performing 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.

6.4.8 AUTOMATIC ERROR RETRIEVAL

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.

6.5 SEQUENTIAL FUNCTION CHART (SFC) EDITOR

The SFC editor allows you to code and modify POUs using SFC (i.e. Sequential Function Chart), one of the IEC-compliant languages.

6.5.1 CREATING A NEW SFC DOCUMENT

See the creating and editing POUs section (see Paragraphs 5.1.1 and 5.1.2).

6.5.2 INSERTING A NEW SFC ELEMENT

- 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.

6.5.3 CONNECTING SFC ELEMENTS

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 so as to let the respective pins coincide, which makes the SFC editor draw automatically the logical wire.

6.5.4 ASSIGNING AN ACTION TO A STEP

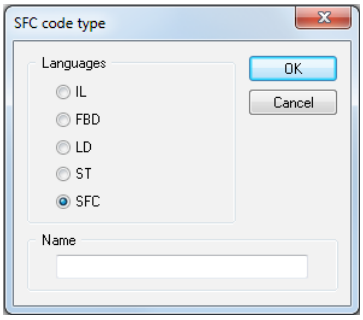
This paragraph explains how to implement an action and how to assign it to a step.

6.5.4.1 WRITING THE CODE OF AN ACTION

To start implementing an action, you need to open an editor. Do it by 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, Application displays a dialog box like the one shown below.



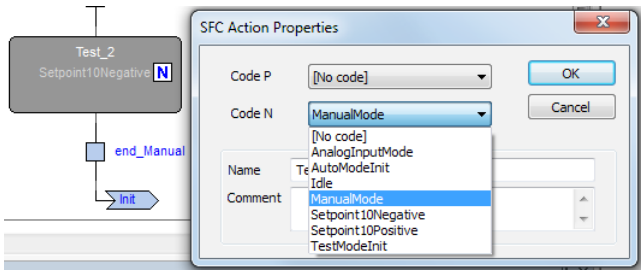
Select one of the languages and type the name of the new action in the text box at the bottom of the dialog box. Then either confirm by pressing *OK*, or quit by clicking *Cancel*.

If you press *OK*, Application 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.

Note that 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.

6.5.4.2 ASSIGNING AN ACTION TO A STEP

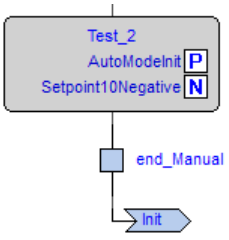
When you have finished writing the code, double-click the step you want to assign the new action to. This causes the following dialog box to appear.



From the list shown 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 shown 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 pressing *OK*.

In the SFC schema, action to step assignments are represented by letters on the step block:

- action *N* by letter N in the top right corner;
- action *P* by letter P in the bottom right corner.

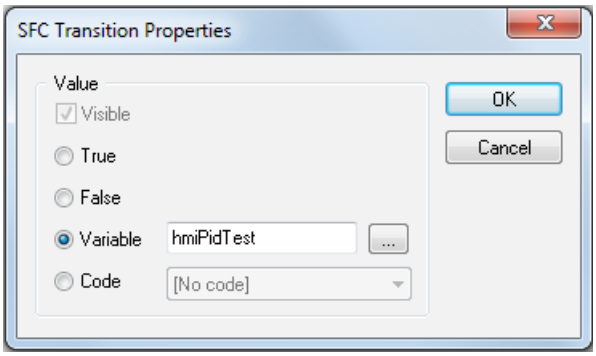


If later you need to edit the source code of the action, you can just double-click these letters. Alternatively, you can double-click the name of the action in the *Actions* folder of the *Workspace* window.

6.5.5 SPECIFYING A CONSTANT/A VARIABLE AS THE CONDITION OF A TRANSITION

As stated in the relevant section of the language reference, 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 will depend 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.

To this purpose, you can also make use of the objects browser, that you can invoke by pressing the *Browse* button shown here below.

Click *OK* to confirm, or *Cancel* to quit without applying changes.

6.5.6 ASSIGNING CONDITIONAL CODE TO A TRANSITION

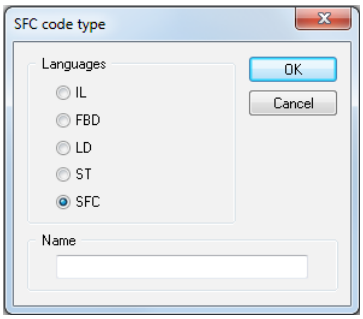
This paragraph explains how to specify a condition through a piece of code, and how to assign it to a transition.

6.5.6.1 WRITING THE CODE OF A CONDITION

Start by opening an editor, following one of these procedures:

- Click *Scheme>Code object>New transition*.
- Right-click on the name of the SFC POU in the *Workspace* window *[New transition]*.

In either case, Application displays a dialog box similar the one shown in the following picture.



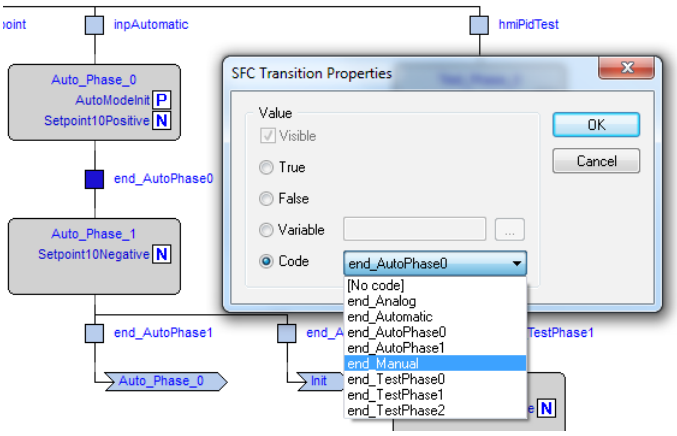
Note that you can use any language except SFC to code a condition. Select one of the languages and type the name of the new condition in the text box at the bottom of the dialog box. Then either confirm by pressing *OK*, or quit by clicking *Cancel*.

If you press OK, Application 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.

Note that 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.

6.5.6.2 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.

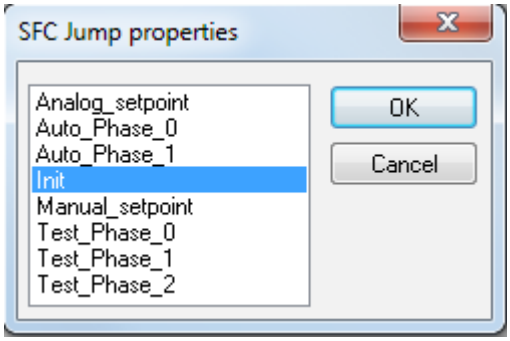


Select the name of the condition you want to assign to this step. Then confirm by pressing *OK*.

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.

6.5.7 SPECIFYING THE DESTINATION OF A JUMP

To specify the destination step of a jump, double-click the jump block in the *Chart* area. This causes the dialog box shown below to appear, listing the name of all the existing steps. Select the destination step, then either press *OK* to confirm or *Cancel* to quit.



6.5.8 EDITING SFC NETWORKS

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.

6.6 VARIABLES EDITOR

Application includes a graphical editor for both global and local variables that supplies a user-friendly interface for declaring and editing variables: the tool takes care of the translation of 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	Group	Array	Init value	Attribute	Description
1	pidKP	REAL	%MD1.0	PID	No		---	PID proportional gain
2	pidKI	REAL	%MD1.4	PID	No		---	PID integral time
3	pidSetpoint	REAL	%MB1.8	PID	No		---	PID setpoint (from -1 to +1)
4	pidOutput	REAL	%MD1.12	PID	No		---	PID output value

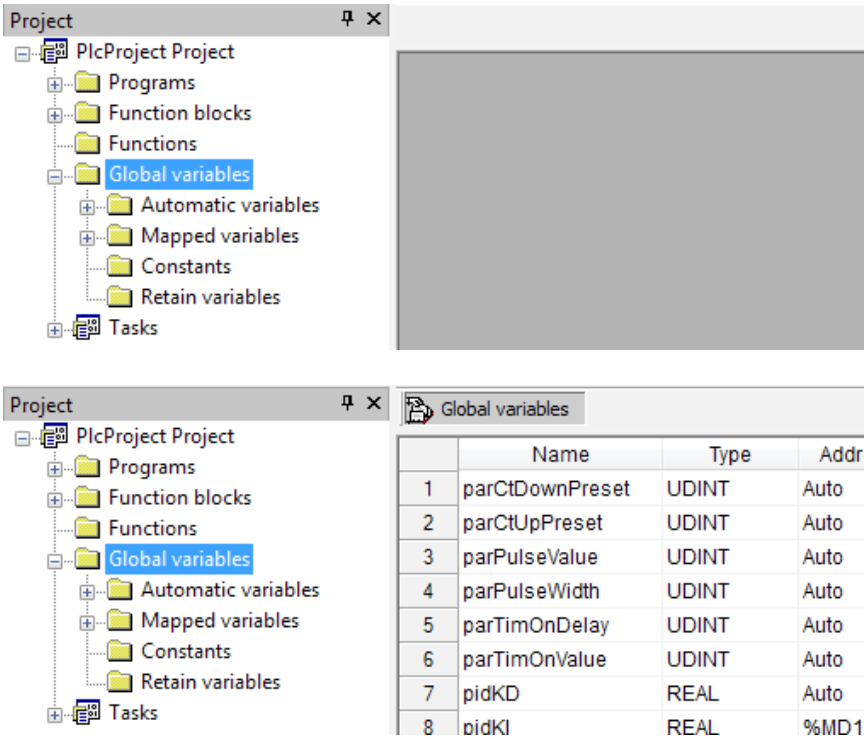
The corresponding source code will look like this:

```
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
```

6.6.1 OPENING A VARIABLES EDITOR

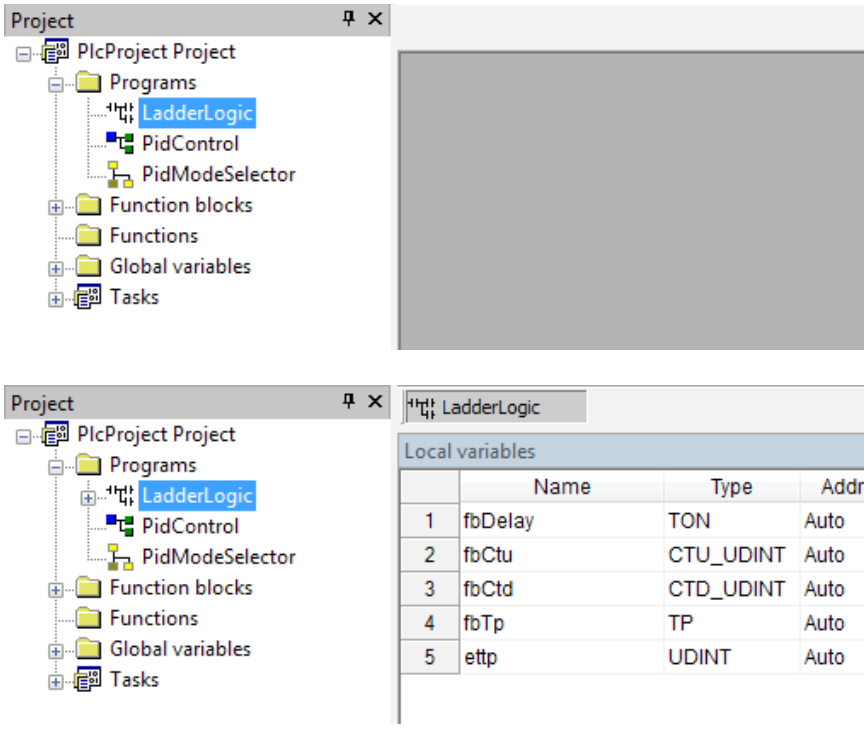
6.6.1.1 OPENING THE GLOBAL VARIABLES EDITOR

In order to open the Global variables editor, double-click on *Global variables* in the project tree.



6.6.1.2 OPENING A LOCAL VARIABLES EDITOR

To open a local variables editor, just open the Program Organization Unit the variables you want to edit are local to.



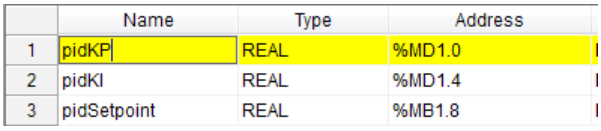
6.6.2 CREATING A NEW VARIABLE

In order to create a new variable, you may click *Variables>Insert*.

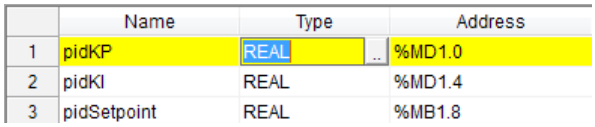
6.6.3 EDITING VARIABLES

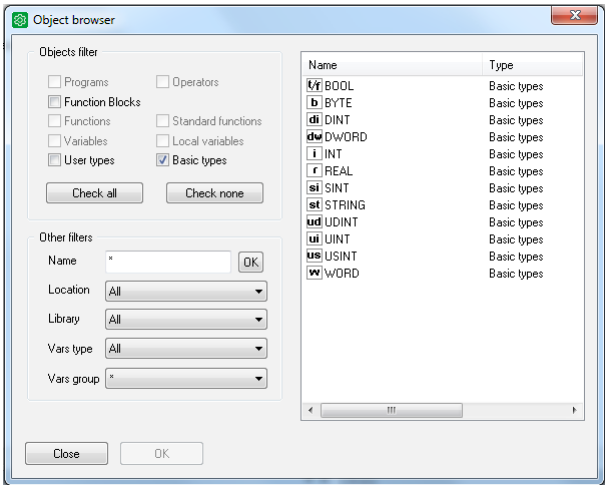
Follow this procedure to edit the declaration of a variable in a variables editor (all the following steps are optional and you will typically skip most of them when editing a variable):

1) Edit the name of the variable by entering the new name in the corresponding cell.

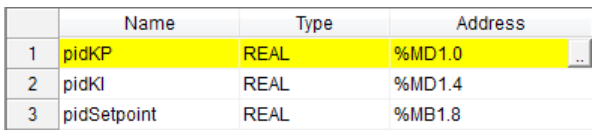


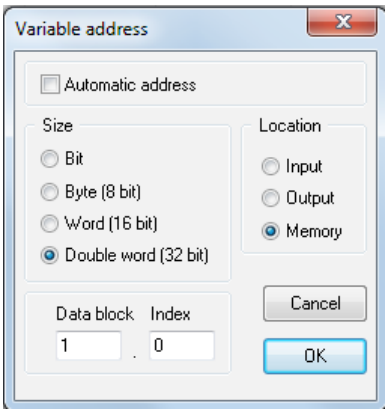
2) Change the variable type, either by editing the type name in the corresponding cell or by clicking on the button in that cell and select the desired type from the list that pops up.



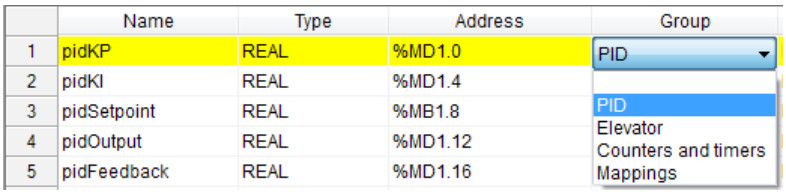


3) Edit the address of the variable by clicking on the button in the corresponding cell and entering the required information in the window that shows up. Note that, in the case of global variables, this operation may change the position of the variable in the project tree.

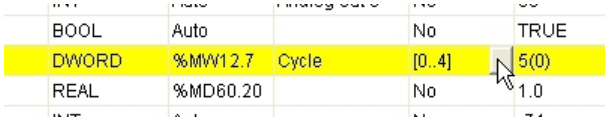


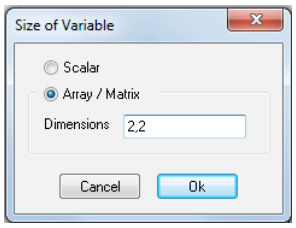


4) In the case of global variables, you can assign the variable to a group, by selecting it from the list which opens when you click on the corresponding cell. This operation will change the position of the variable in the project tree.

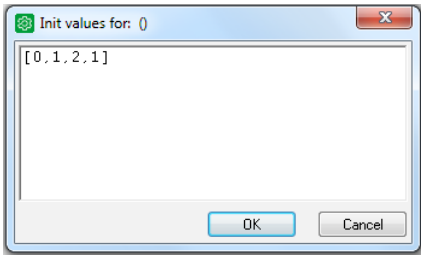


5) Choose whether a variable is an array or not; if it is, edit the size of the variable.

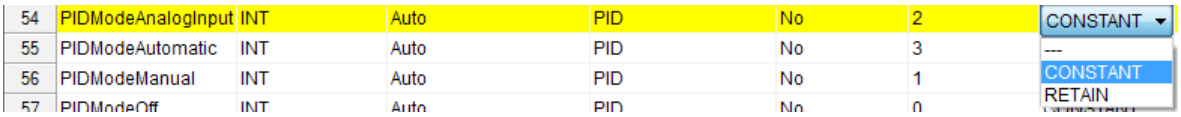




6) Edit the initial values of the variable: click on the button in the corresponding cell and enter the values in the window that pops up.



7) Assign an attribute to the variable (for example, `CONSTANT` or `RETAIN`), by selecting it from the list which opens when you click on the corresponding cell.



8) Type a description for the variable in the corresponding cell. Note that, in the case of global variables, this operation may change the position of the variable in the project tree.

	No	2	CONSTANT	Indicates PID analog input reference mode
	No	3	CONSTANT	Indicates PID automatic reference mode
	No	1	CONSTANT	Indicates PID manual reference mode
	No	0	CONSTANT	Indicates PID reference mode disabled

9) Save the project to persist the changes you made to the declaration of the variable.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Ensure that all variables are initialized to an appropriate value before their first use as array indices.
- Write programming instructions to test the validity of operands intended to be used as array indices.
- Do not attempt to access an array element outside the defined bounds of the array.
- Do not attempt to assign a value to an array name without using an appropriate index into the array.

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

6.6.4 DELETING VARIABLES

In order to delete one or more variables, select them in the editor: you may use the *CTRL* or the *SHIFT* keys to select multiple elements.

	Name	Type	Address
1	pidKP	REAL	%MD1.0
2	pidKI	REAL	%MD1.4
3	pidSetpoint	REAL	%MB1.8
4	pidOutput	REAL	%MD1.12
5	pidFeedback	REAL	%MD1.16
6	pidKD	REAL	%MD1.20
7	hmiPIDSetpoint	REAL	%MB1.24

Then, click *Variables>Delete*.

Notice that you cannot delete the `RESULT` of an IEC61131-3 `FUNCTION`.

6.6.5 SORTING VARIABLES

You can sort the variables in the editor by clicking on the column header of the field you want to use as the sorting criterion.

	Name	Type	Ac
1	valueFilt	REAL	%M
2	tau	REAL	%MI
3	valueRef	REAL	%MI
4	period	REAL	%MI
5	klnilncr	INT	Auto
6	ki	UINT	Auto
7	incr	INT	Auto
8	freeRunCounter	DINT	Auto

	Name	Type	Ac
1	freeRunCounter	DINT	Auto
2	incr	INT	Auto
3	ki	UINT	Auto
4	klnilncr	INT	Auto
5	period	REAL	%MI
6	tau	REAL	%MI
7	valueFilt	REAL	%M
8	valueRef	REAL	%MI

6.6.6 COPYING VARIABLES

The variables editor allows you to quickly copy and paste elements. You can either use keyboard shortcuts or the *Edit>Copy* , *Edit>Paste* menu.

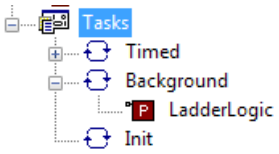
Note: overlapping addresses problems may occur by copying mapped variables. Application can automatically assign new free address to the new pasted variable and fix the overlap. In order to enable this functionality please refer to paragraph 3.6 and 4.8.3.2 for further details.

7. COMPILING

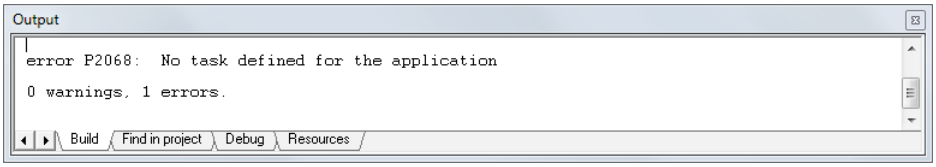
Compilation consists of taking the PLC source code and automatically translating it into binary code, which can be executed by the processor on the target device.

7.1 COMPILING THE PROJECT

Before starting actual compilation, make sure that at least one program has been assigned to a task.



When this pre-condition does not hold, compilation aborts with a meaningful error message.

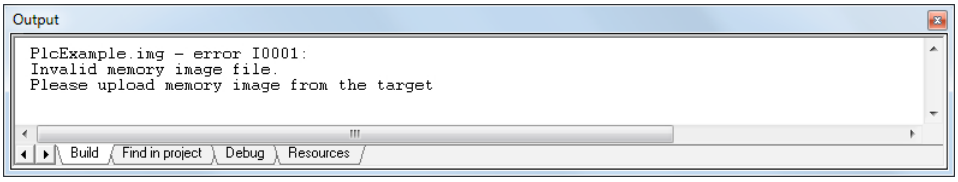


In order to start compilation, click *Project>Compile*.

Note that Application automatically saves all changes to the project before starting the compilation.

7.1.1 IMAGE FILE LOADING

Before performing the actual compilation, the compiler needs to load the image file (*img file*), which contains the map of memory 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.



7.2 COMPILER OUTPUT

If the previous step was accomplished, the compiler performs the actual compilation, then prints a report in the *Output* window. The last string of the report has the following format:

```
m warnings, n errors
```

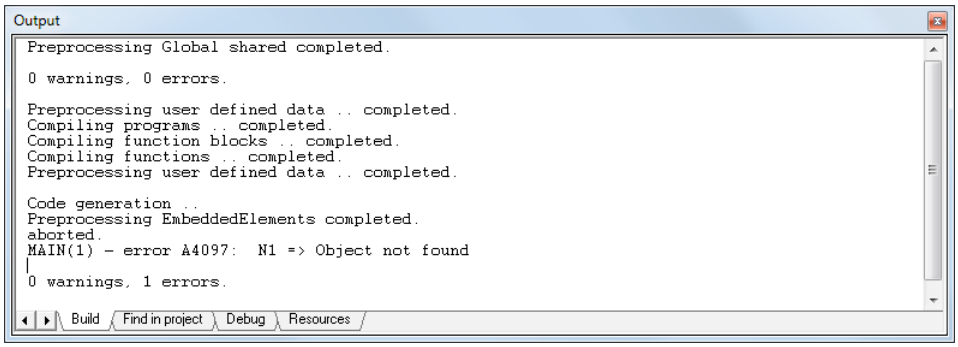
It tells the user the outcome of compilation.

Condition	Description
n>0	Compiler error(s). The PLC code contains one or more serious errors, which cannot be worked around by the compiler.

Condition	Description
n=0, m>0	Emission of warning(s). The PLC code contains one or more minor errors, which the compiler automatically spotted and worked around. However, you are informed that the PLC program may act in a different way from what you expected: you are encouraged to get rid of these warnings by editing and re-compiling the application until no warning messages are emitted.
n=m=0	PLC code entirely correct, compilation accomplished. You should always work with 0 warnings, 0 errors.

7.2.1 COMPILER ERRORS

When your application contains one or more errors, some useful information is printed in the *Output* window for each of those errors.

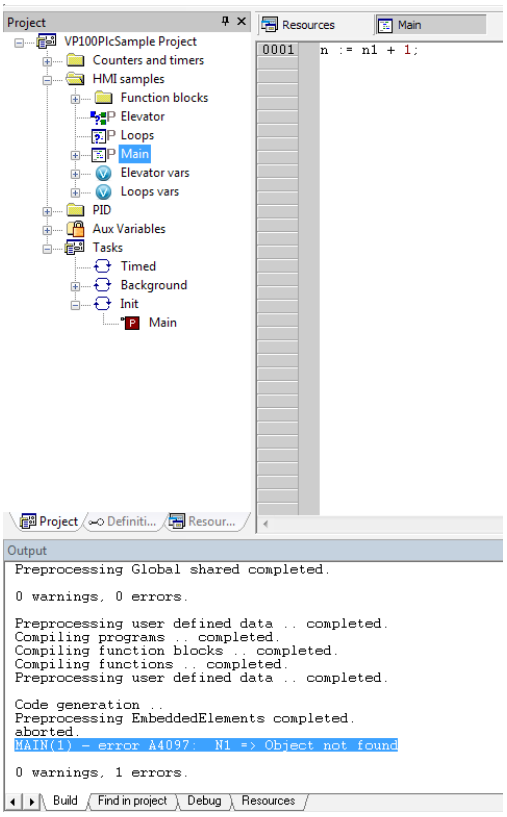


As you can see, the information includes:

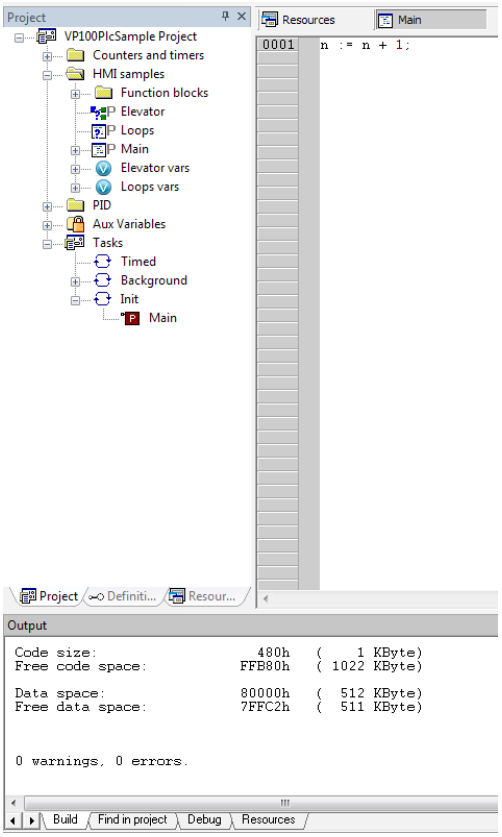
- the name of the Program Organization Unit affected by the error;
- the number of the source code line which procured the error;
- whether it is a fatal error (`error`) or one that the compiler could work around (`warning`);
- the error code;
- the error description.

Refer to the appropriate section for the compiler error reference.

If you double-click the error message in the *Output* bar, Application opens the source code and highlights the line containing the error.



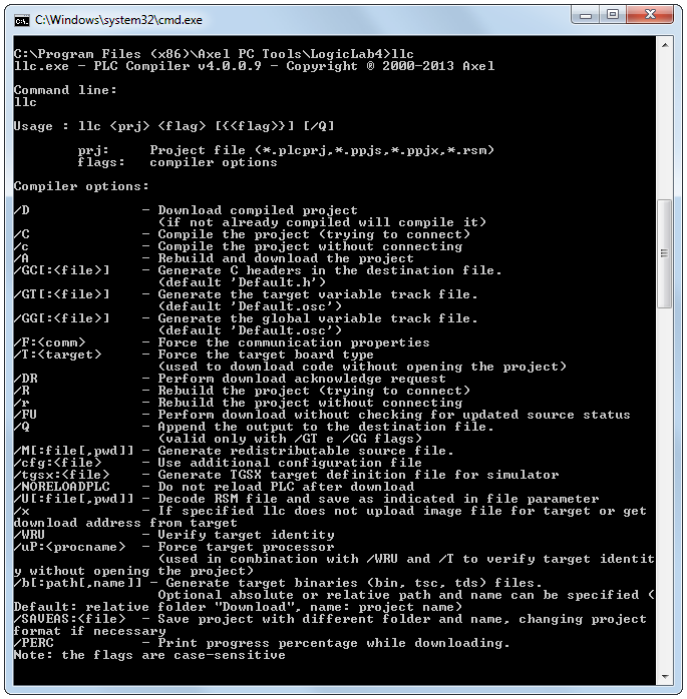
You can then fix the problem and re-compile.



7.3 COMMAND-LINE COMPILER

Application's compiler can be used independently from the IDE: in Application's directory, you can find an executable file, *Command-line compiler*, 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, just launch the executable without parameters.



8. LAUNCHING THE APPLICATION

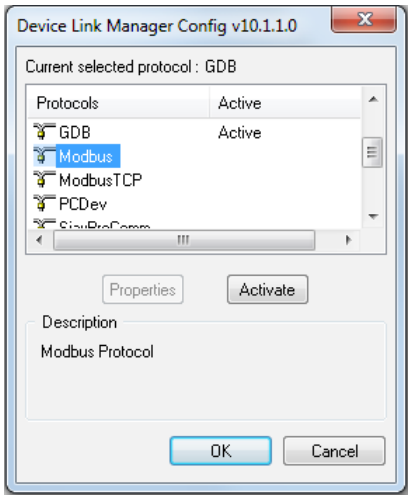
In order to download and debug the application, you have to establish a connection with the target device. This chapter focuses on the operations required to connect to the target and to download the application, while the wide range of Application's debugging tools deserves a separate chapter (see Chapter 9).

8.1 SETTING UP THE COMMUNICATION

In order to establish the connection with the target device, make sure the physical link is up (all the cables are plugged in, the network is properly configured, and so on).

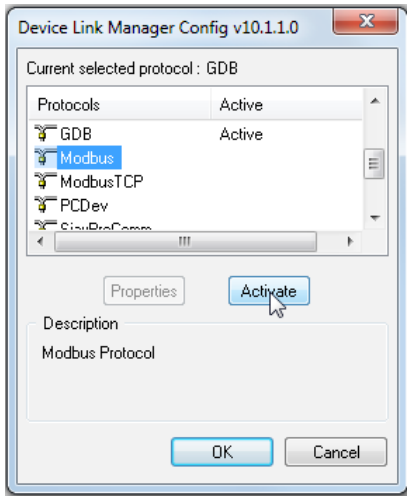
Follow this procedure to set up and establish the connection to the target device:

1) Click *On-line>Set up communication...* menu of the Application main window. This causes the following dialog box to appear.



The elements in the list of communication protocols you can select from depend on the setup executable(s) you have run on your PC (refer to your hardware provider if a protocol you expect to appear in the list is missing).

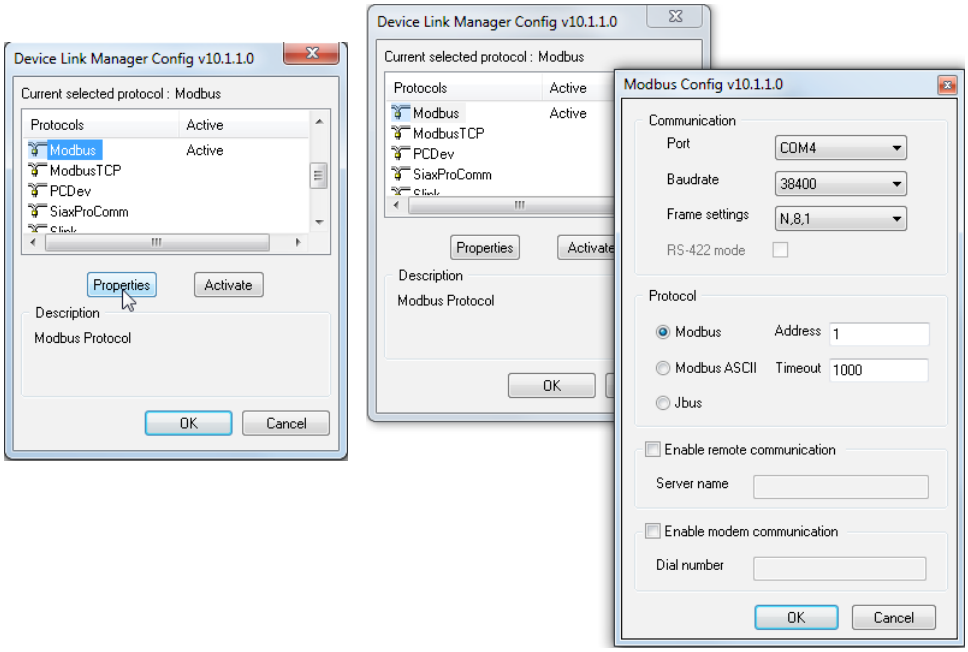
2) Choose the appropriate protocol and make it the active protocol.



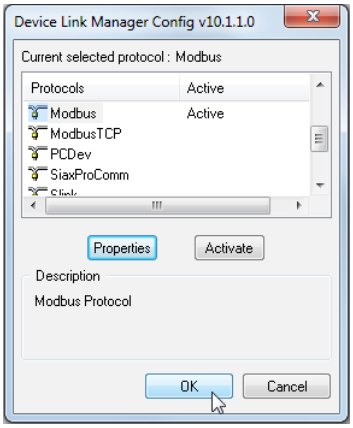
Note: in case the connection is made through a USB cable, using a virtual COM port on your PC, Modbus protocol is to be activated.

3) Fill in all the protocol-specific settings (e.g., the address or the communication timeout - that is how long Application must wait for an answer from the target before displaying a communication error message).

   If the selection is on Modbus, with a USB cable connection, only the proper virtual COM port is to be selected, leaving all the other parameters as default.



4) Apply the changes you made to the communication settings.

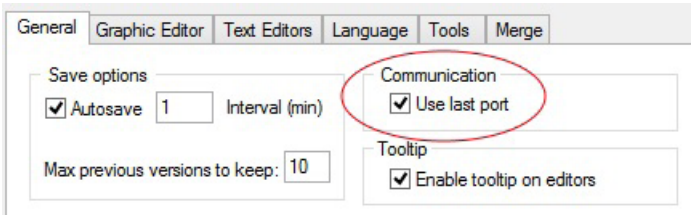


Now you can establish communication by clicking *On-line>Connect* menu.

8.1.1 SAVING THE LAST USED COMMUNICATION PORT

When you connect to target devices using a serial port (COM port), you usually use the same port for all devices (many modern PCs have only one COM port). You may save the last used COM port and let Application use that port to override the project settings: this feature proves especially 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...* menu.

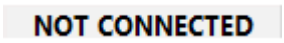


8.2 ON-LINE STATUS

8.2.1 CONNECTION STATUS

The state of communication is shown in a small 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*.



When you try to connect to the target device, the state of communication becomes one of the following:

- `Error`: the communication cannot be established. You should check both the physical link and the communication settings.

- `Connected`: the communication has been established.

8.2.2 APPLICATION STATUS

Next to the communication status there is another small box which gives information about the status of the application currently executing on the target device.

When the connection status is `Connected`, the application status takes on one of the following values.

- `No code`: no application is executing on the target device.

- `Diff. code`: the application currently executing on the target device is not the same as the one currently open in the IDE; moreover, no debug information consistent with the running application is available: thus, the values shown in the watch window or in the oscilloscope are not reliable and the debug mode cannot be activated.

- `Diff. code, Symbols OK`: the application currently executing on the target device is

not the same as the one currently open in the IDE; however, some debug information consistent with the running application is available (for example, because that application has been previously downloaded to the target device from the same PC): the values shown in the watch window or in the oscilloscope are reliable, but the debug mode still cannot be activated.

- `Source OK`: the application currently executing on the target device is the same as the one currently open in the IDE: the debug mode can be activated.

8.3 DOWNLOADING THE APPLICATION

A compiled PLC application must be downloaded to the target device in order to have the processor execute it. This paragraph shows you how to send a PLC code to a target device. Note that Application can download the code to the target device only if the latter is connected to the PC where Application is running. See the related section for details.

To download the application, click *On-line>Download code*.

Application checks whether the project has unsaved changes. If this is the case, it automatically starts the compilation of the application. The binary code is eventually sent to the target device, which then undergoes automatic reset at the end of transmission. Now the code you sent is actually executed by the processor on the target device.

8.3.1 CONTROLLING SOURCE CODE DOWNLOAD

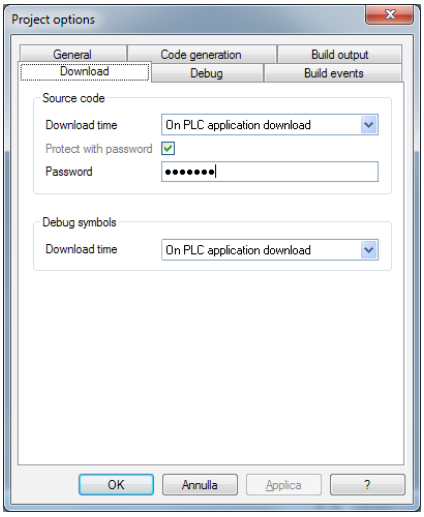
Whether the source code of the application is downloaded along with the binary code or not, depends on the target device you are interfacing with: some devices host the application source code in their storage, in order to allow the developer to upload the project in a later moment.

If this is the case, you can control some aspects of the source code download process, as explained in the following paragraphs.

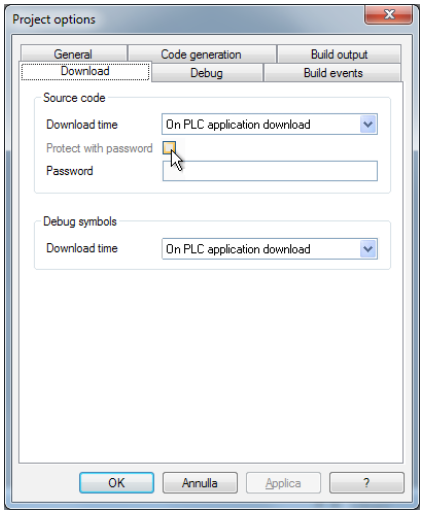
8.3.1.1 PROTECTING THE SOURCE CODE WITH A PASSWORD

You may want to protect the source code downloaded to the target device with a password, so that Application will not open the uploaded project unless the correct password is entered.

Click the *Project>Options...* menu and set the password.

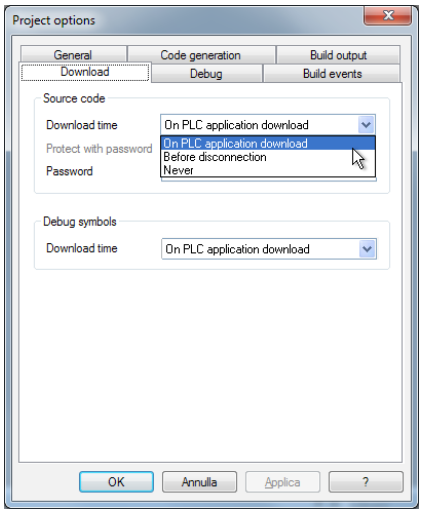


You may opt to disable the password, instead.



8.3.1.2 SOURCE CODE AND DEBUG SYMBOLS DOWNLOAD TIME

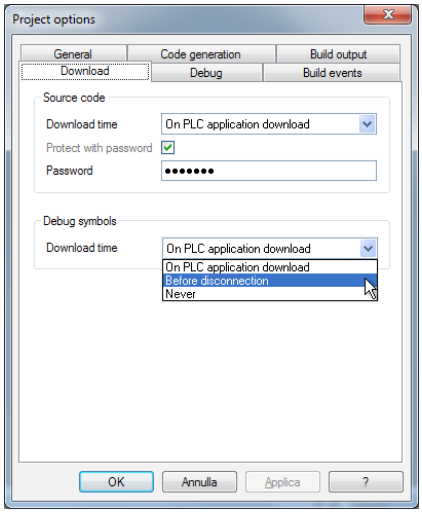
From the following select menu you can set the *Source code download time*.



Choosing:

- *On PLC application download*: the Source code will be downloaded to the target together with PLC application.
- *Before disconnection*: the Source code will be downloaded before target disconnection.
- *Never*: the Source code will be never downloaded to the target.

As well as Source code the Debug symbols download time can be set using the following select menu with the same options.



8.4 SIMULATION

Depending on the target device you are interfacing with, you may be able to simulate the execution of the PLC application with Application's integrated simulation environment: Simulation.

In order to start the simulation, just click *Debug>Simulation mode*.

Refer to Simulation's manual to gain information on how to control the simulation.

8.5 CONTROL THE PLC EXECUTION

The PLC application execution can be controlled using the related functions in the project bar or by the command presents in the On-line menu.

8.5.1 HALT

You can stop the PLC execution by clicking *On-line>Halt*

8.5.2 COLD RESTART

The PLC application execution will be restarted and both retain and non-retain variables will be resetted.

You can cold restart the PLC execution by clicking *On-line>Cold restart*.

8.5.3 WARM RESTART

The PLC application execution will be restarted and only non-retain variables will be resetted.

You can warm restart the PLC execution by clicking *On-line>Warm restart*.

8.5.4 HOT RESTART

The PLC application execution will be restarted and no variables will be resetted.

You can hot restart the PLC execution by clicking *On-line>Hot restart*.

8.5.5 REBOOT TARGET

You can reboot the target by clicking *On-line>Reboot target*.

9. DEBUGGING

Application provides several debugging tools, which help the developer to check whether the application behaves as expected or not.

All these debugging tools basically allow the developer to watch the value of selected variables while the PLC application is running.

Application debugging tools can be gathered in two classes:

- Asynchronous debuggers. They read the values of the variables selected by the developer 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. Thus, there is no guarantee about the values of two distinct variables being sampled in the same moment, 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 reliable, especially when it changes fast.
- 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 shows you how to debug your application using both asynchronous and synchronous tools.

9.1 WATCH WINDOW

The *Watch* window allows you to monitor the current values of a set of variables. Being an asynchronous tool, the *Watch* window does not guarantee synchronization of values. Therefore, when reading the values of the variables in the *Watch* window, be aware of the possibility that they 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 shown in the *Watch* window includes the name of the variable, its value, its type, and its location in the PLC application.

Symbol	Value	Type	Location
HMIPIDTEST	FALSE	BOOL	global
HMIPIDTHRESHOLD	0.2	REAL	global
PARCTDOWNPRESET	100	INT	global
BASETIME	0	UDINT	@FAST:PIDMODESELECTOR

9.1.1 OPENING AND CLOSING THE WATCH WINDOW

To open, close the *Watch* window, click *View>Tool windows>Watch*.

Closing the *Watch* window means simply hiding it, not resetting it. As a matter of fact, 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.

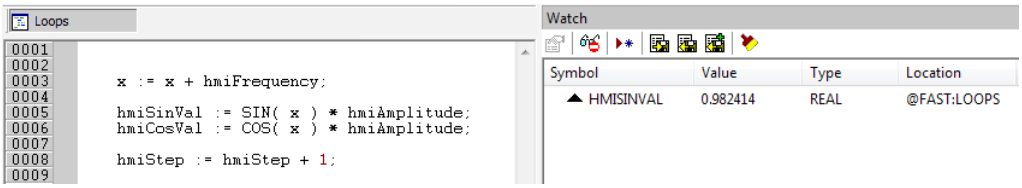
9.1.2 ADDING ITEMS TO THE WATCH WINDOW

To watch a variable, you need to add it to the watch list.

Note that, unlike trigger windows and the *Graphic trigger* window, you can add to the *Watch* window all the variables of the project, regardless of where they were declared.

9.1.2.1 ADDING A VARIABLE FROM A TEXTUAL SOURCE CODE EDITOR

Follow this procedure to add a variable to the *Watch* window from a textual (that is, IL or ST) source code editor: select a variable, by double-clicking on it, and then drag it into the watch window.

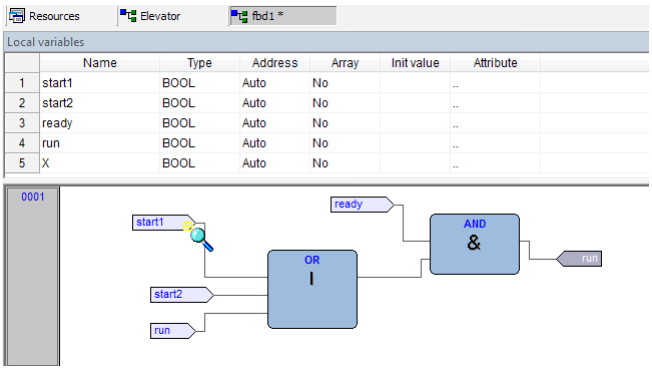


The same procedure applies to all the variables you wish to inspect.

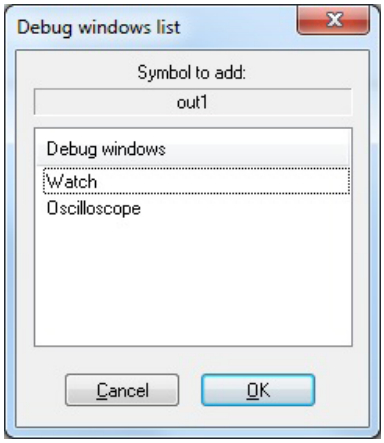
9.1.2.2 ADDING A VARIABLE FROM A GRAPHICAL SOURCE CODE EDITOR

Follow this procedure to add a variable to the *Watch* window from a graphical (that is, LD, FBD, or SFC) source code editor:

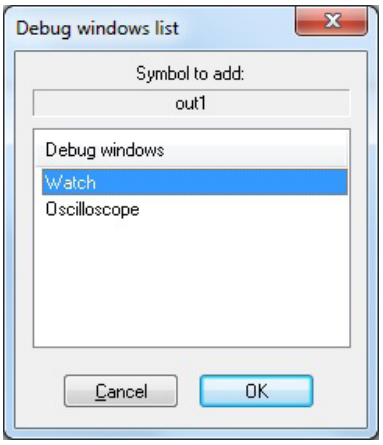
1) Click *Edit>Watch mode*.
2) Click on the block representing the variable you wish to be shown in the *Watch* window.



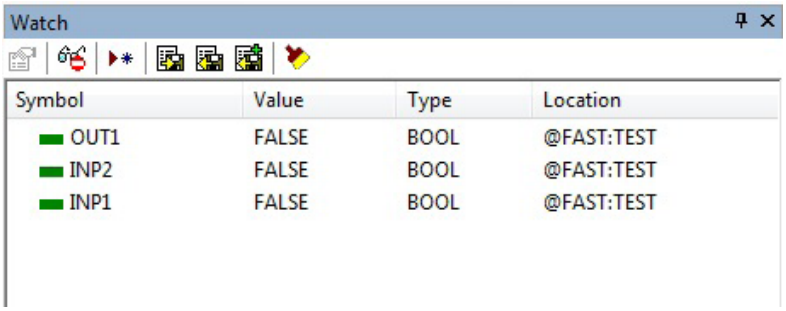
A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked on.



In order to display the variable in the *Watch* window, select *Watch*, then press *OK*.



The variable name, value, and location are now displayed in a new row of the *Watch* window.

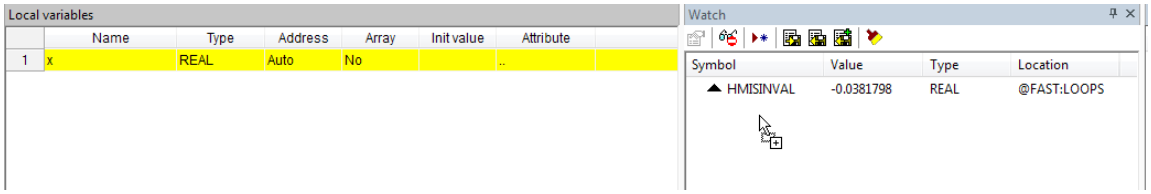


The same procedure applies to all the variables you wish to inspect.

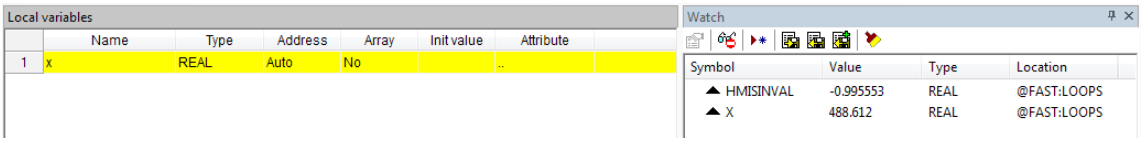
Once you have added to the *Watch* window all the variables you want to observe, you should click *Edit>Insert/Move mode*: the mouse cursor turns to its original shape.

9.1.2.3 ADDING A VARIABLE FROM A VARIABLES EDITOR

In order to add a variable to the *Watch* window, you can select the corresponding record in the variables editor and then either drag-and-drop it in the *Watch* window

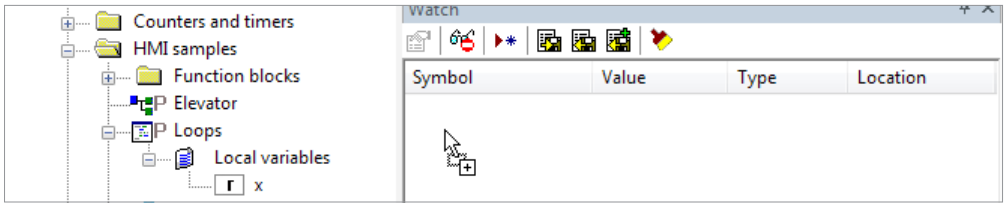


or press the *F8* key.

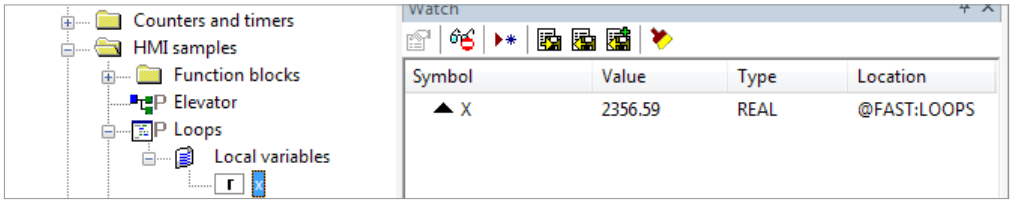


9.1.2.4 ADDING A VARIABLE FROM THE PROJECT TREE

In order to add a variable to the *Watch* window, you can select it in the project tree and then either drag-and-drop it in the *Watch* window

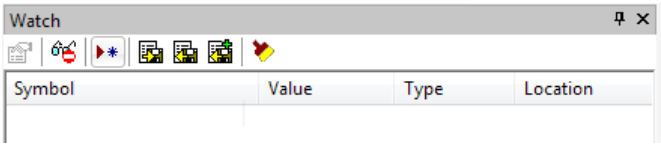


or press the *F8* key.

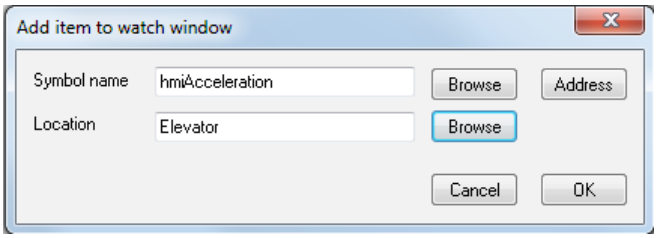


9.1.2.5 ADDING A VARIABLE FROM THE WATCH WINDOW TOOLBAR

You can also click on the appropriate item of the Watch window inner toolbar, in order to add a variable to it.

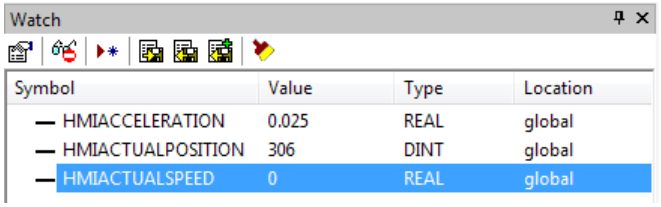


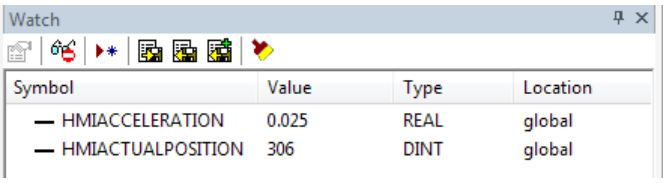
You shall type (or select by browsing the project symbols) the name of the variable and its location (where it has been declared).



9.1.3 REMOVING A VARIABLE

If you want a variable not to be displayed any more in the *Watch* window, select it by clicking on its name once, then press the *Del* key.

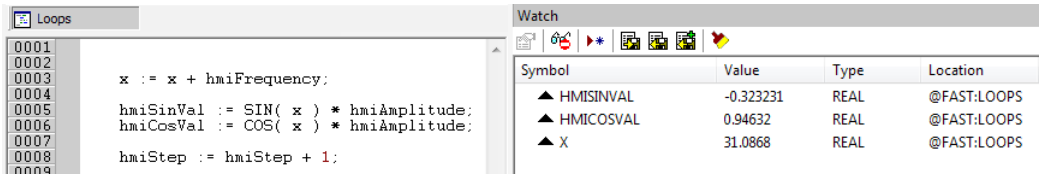




9.1.4 REFRESHMENT OF VALUES

9.1.4.1 NORMAL OPERATION

Let us consider the following example.

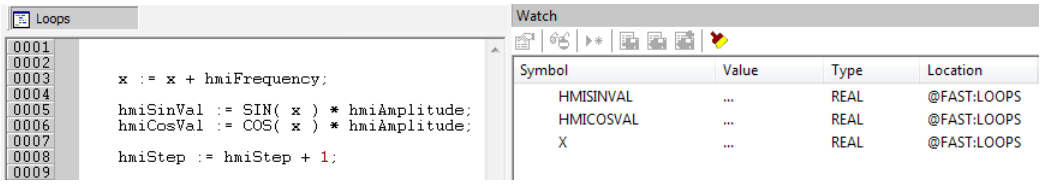


The watch window manager reads periodically from memory the value of the variables.

However, this action is carried out asynchronously , that is 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.

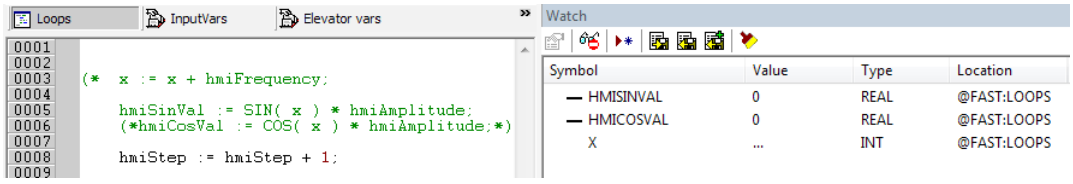
9.1.4.2 TARGET DISCONNECTED

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

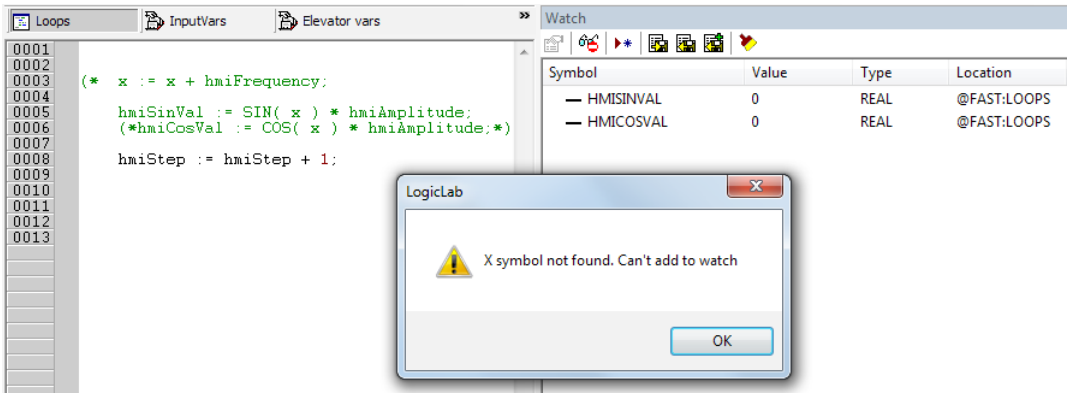


9.1.4.3 OBJECT NOT FOUND

If the PLC code changes and Application 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, Application gives the following error message.

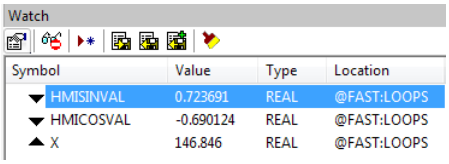


9.1.5 CHANGING THE FORMAT OF DATA

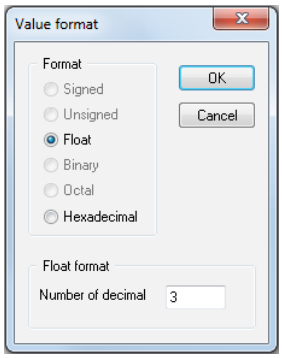
When you add a variable to the *Watch* window, Application automatically recognizes its type (unsigned integer, signed integer, floating point, hexadecimal), and displays its value consistently. Also, if the variable is floating point, Application assigns it a default number of decimal figures.

However, you may need the variable to be printed in a different format.

To impose another format than the one assigned by Application, press the *Format value* button in the toolbar.



Choose the format and confirm your choice.

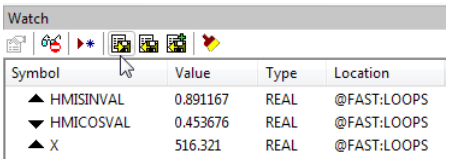


9.1.6 WORKING WITH WATCH LISTS

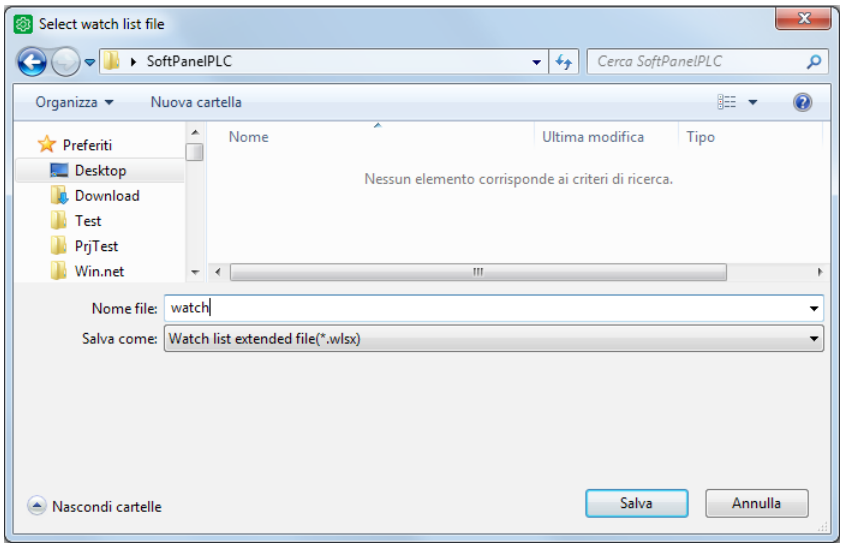
You can store to file the set of all the items in the *Watch* window, in order to easily restore the status of this debugging tools in a successive working session.

Follow this procedure to save a watch list:

1) Click on the corresponding item in the *Watch window* toolbar.

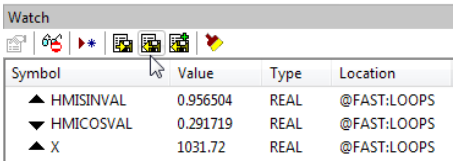


2) Enter the file name and choose its destination in the file system.

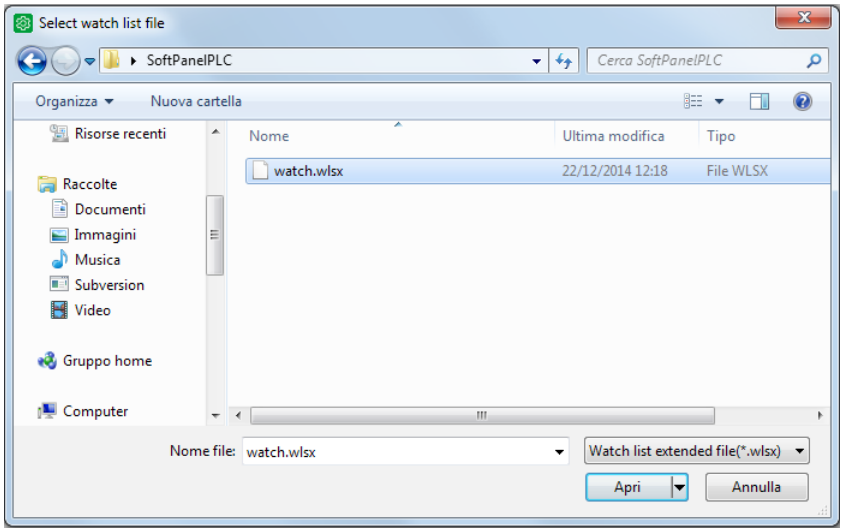


You can load a watch list from file, removing the opened one, following this procedure:

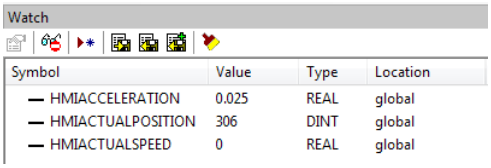
1) Click on the corresponding icon in the *Watch* window toolbar.



2) Browse the file system and select the watch list file.

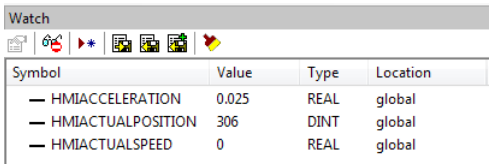


The set of symbols in the watch list is added to the *Watch* window.

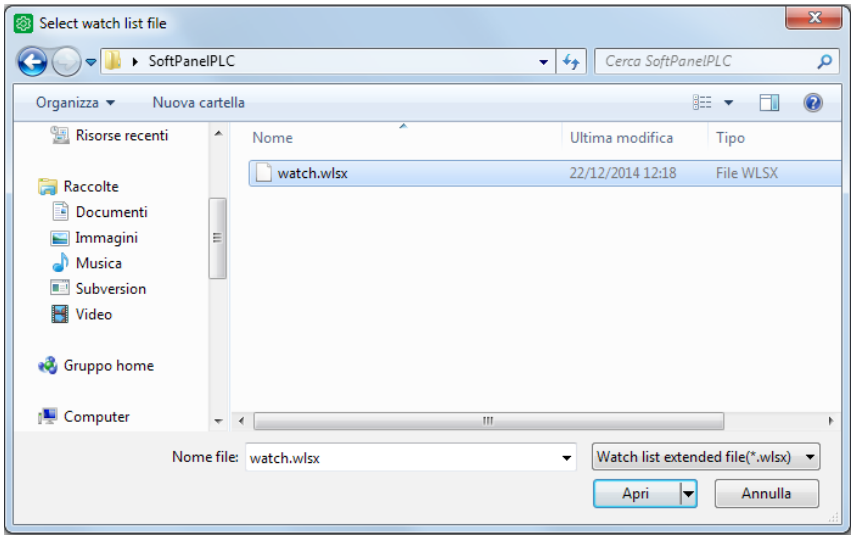


You can load a watch list from file, appending to the opened one, following this procedure:

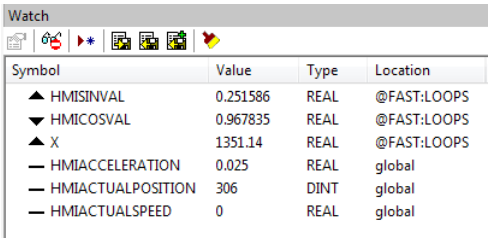
1) Click on the corresponding icon in the *Watch* window toolbar.



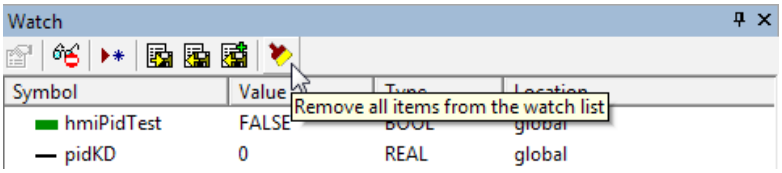
2) Browse the file system and select the watch list file.



The set of symbols in the watch list is added to the *Watch* window.



You can clear the current opened watch list by clicking on the following icon:



9.1.7 AUTOSAVE WATCH LIST

By selecting the associated option in the project options dialog (see Paragraph 4.6.5 for more info) the watch list will be automatically saved on the project closing.

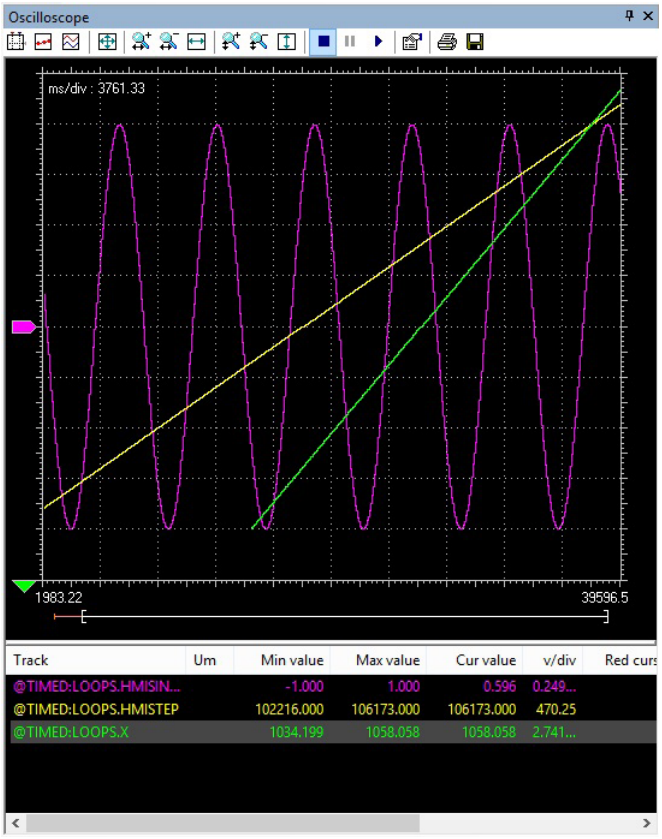
The saved watch list will be automatically loaded (with no append option) on the first connection to target when the project will be re-opened.

9.2 OSCILLOSCOPE

The Oscilloscope allows you to plot the evolution of the values of a set of variables. Being an asynchronous tool, the Oscilloscope cannot guarantee synchronization of samples.

Opening the Oscilloscope causes a new window to appear next to the right-hand border of the Application frame. This is the interface for accessing the debugging functions that the Oscilloscope makes available. The Oscilloscope consists of three elements, as shown

in the following picture.



The toolbar allows you to better 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.

9.2.1 OPENING AND CLOSING THE OSCILLOSCOPE

To open, close the Oscilloscope, click *View>Tool windows>Oscilloscope*.

Closing the Oscilloscope means simply hiding it, not resetting it. As a matter of fact, 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.

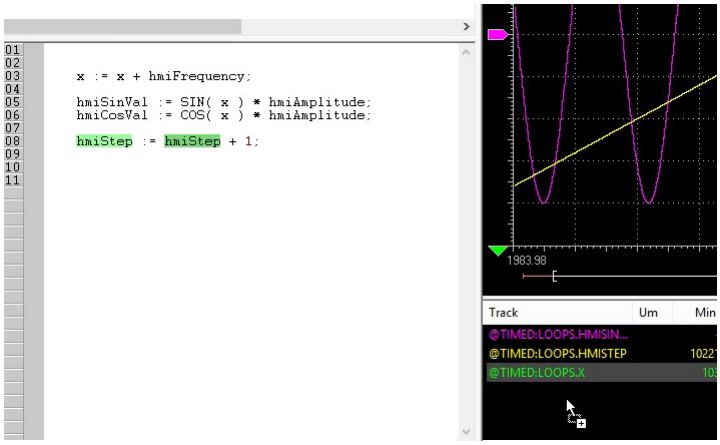
9.2.2 ADDING ITEMS TO THE OSCILLOSCOPE

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

Note that 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.

9.2.2.1 ADDING A VARIABLE FROM A TEXTUAL SOURCE CODE EDITOR

Follow this procedure to add a variable to the Oscilloscope from a textual (that is, IL or ST) source code editor: select a variable by double-clicking on it, and then drag it into the *Oscilloscope* window.

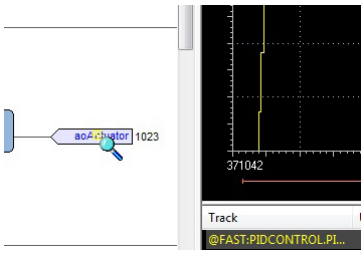


The same procedure applies to all the variables you wish to inspect.

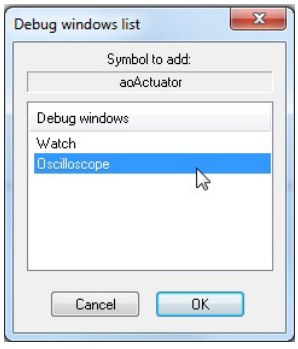
9.2.2.2 ADDING A VARIABLE FROM A GRAPHICAL SOURCE CODE EDITOR

Follow this procedure to add a variable to the Oscilloscope from a graphical (that is, LD, FBD, or SFC) source code editor:

1) Click *Edit>Watch mode*.
2) Click on the block representing the variable you wish to be shown 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 on.



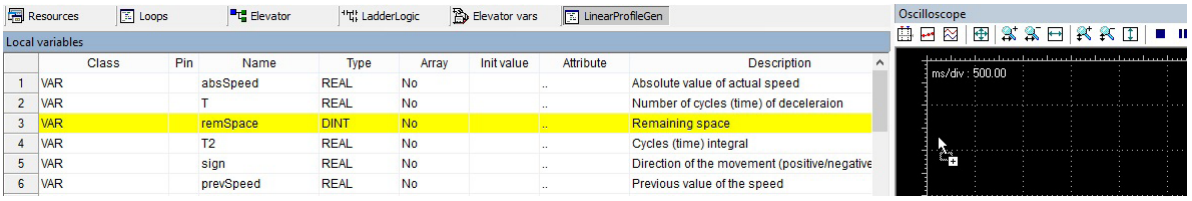
Select *Oscilloscope*, the press *OK*. The name of the variable is now displayed in the *Track* column.

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

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.

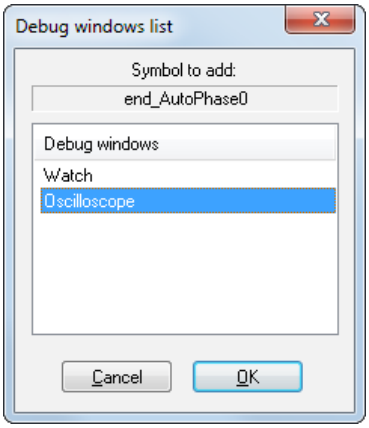
9.2.2.3 ADDING A VARIABLE FROM A VARIABLES EDITOR

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



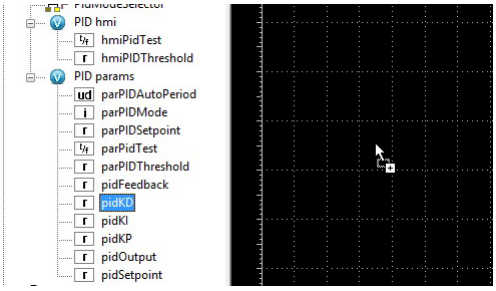
	Class	Pin	Name	Type	Array	Init value	Attribute	Description
1	VAR		absSpeed	REAL	No		..	Absolute value of actual speed
2	VAR		T	REAL	No		..	Number of cycles (time) of deceleraion
3	VAR		remSpace	DINT	No		..	Remaining space
4	VAR		T2	REAL	No		..	Cycles (time) integral
5	VAR		sign	REAL	No		..	Direction of the movement (positive/negative
6	VAR		prevSpeed	REAL	No		..	Previous value of the speed

or press the *F10* key and choose *Oscilloscope* from the list of debug windows which pops up.

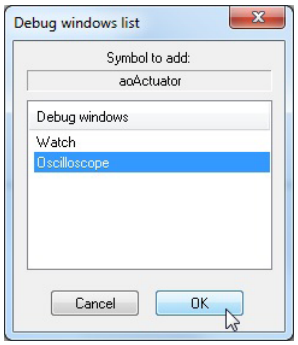


9.2.2.4 ADDING A VARIABLE FROM THE PROJECT TREE

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.



9.2.3 REMOVING A VARIABLE

If you want to remove a variable from the Oscilloscope, select it by clicking on its name once, then press the *Del* key.

9.2.4 VARIABLES SAMPLING

9.2.4.1 NORMAL OPERATION

The Oscilloscope manager periodically reads from memory the value of the variables.

However, this action is carried out asynchronously, that is 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 sampling process, data associated with the same value of the x-axis may actually refer to different execution states of the PLC code.

9.2.4.2 TARGET DISCONNECTED

If the target device is disconnected, the curves of the dragged-in variables get frozen, until communication is restored.

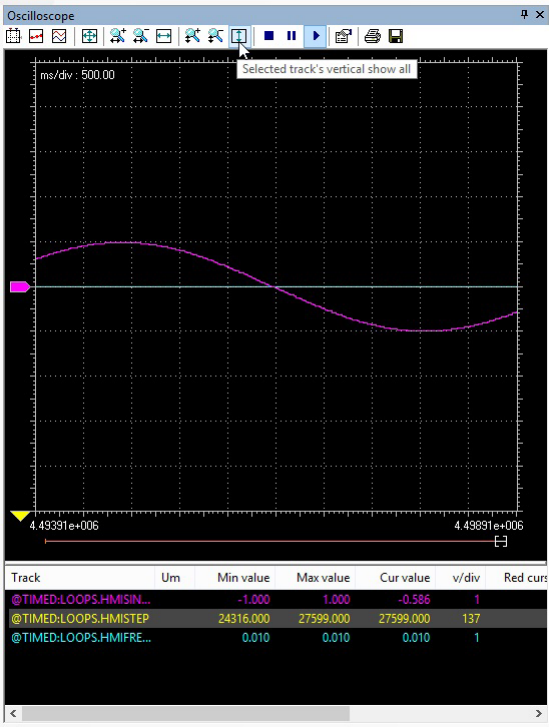
9.2.5 CONTROLLING DATA ACQUISITION AND DISPLAY

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.

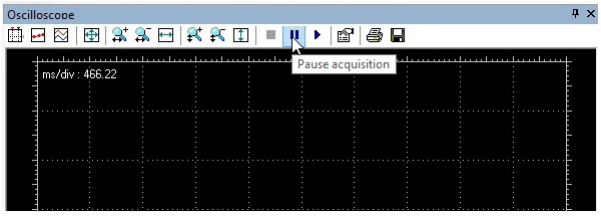
Note that all the commands in the toolbar are disabled if no variable has been added to the Oscilloscope.

9.2.5.1 STARTING AND STOPPING DATA ACQUISITION

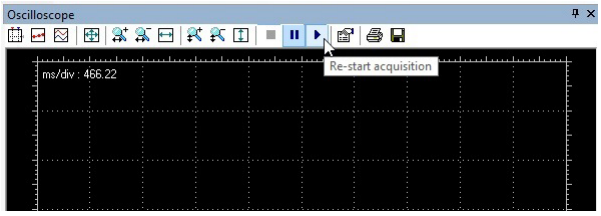
When you add a variable to the Oscilloscope, data acquisition begins immediately.



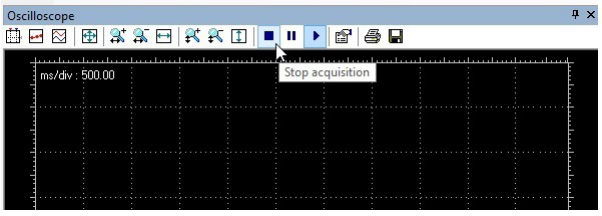
However, you can suspend the acquisition by clicking on *Pause acquisition*.



The curve freezes (while the process of data acquisition is still running in background), until you click on *Restart acquisition*.



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

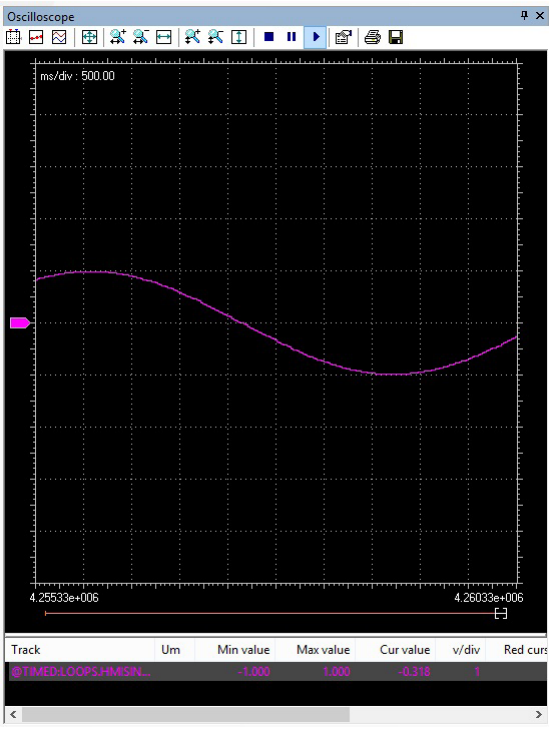


In this case, when you click on *Restart acquisition*, the evolution of the value of the variable is plotted from scratch.

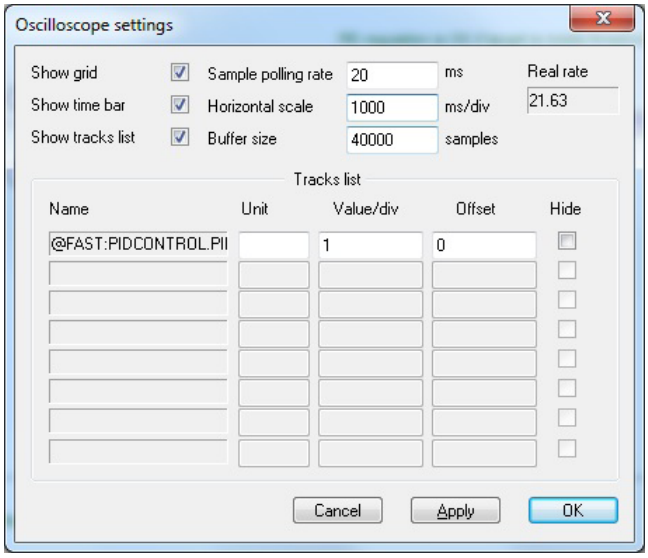
9.2.5.2 SETTING THE SCALE OF THE AXES

When you open the Oscilloscope, Application applies a default scale to the axes. However, if you want to set a different scale, you may follow this procedure:

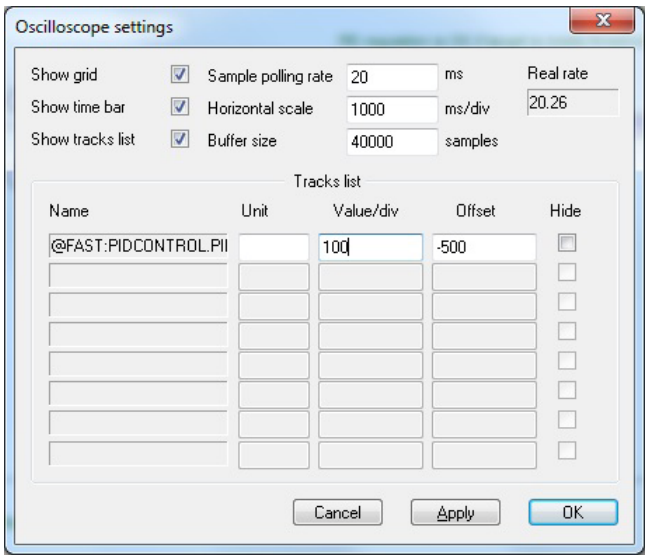
1) Open the graph properties by clicking on the corresponding item in the toolbar.



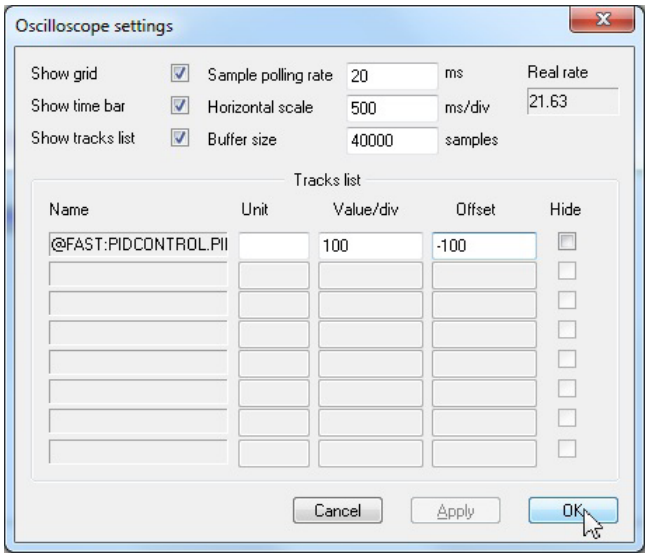
2) Set the scale of the horizontal axis, which is common to all the tracks.

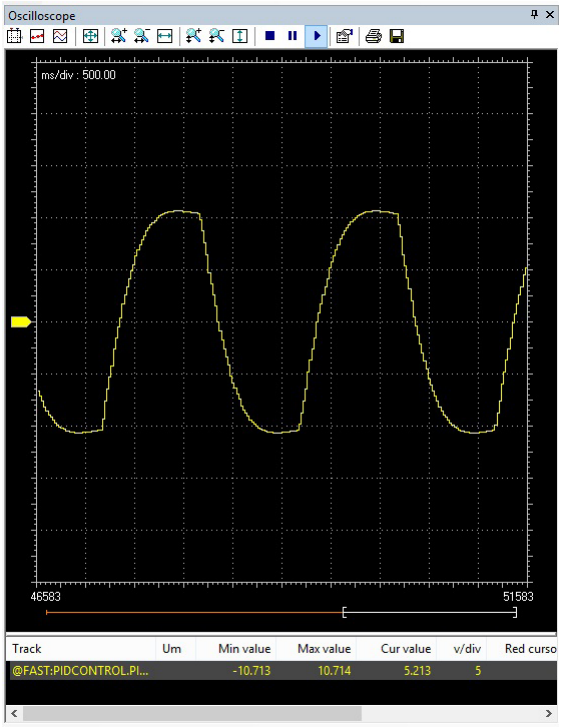


3) For each variable, you may specify a distinct scale for the vertical axis.

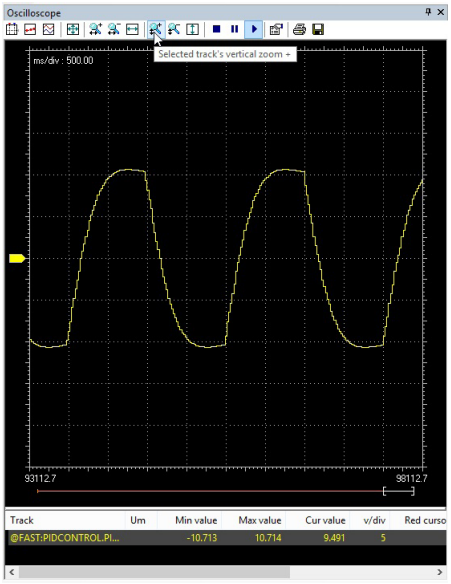


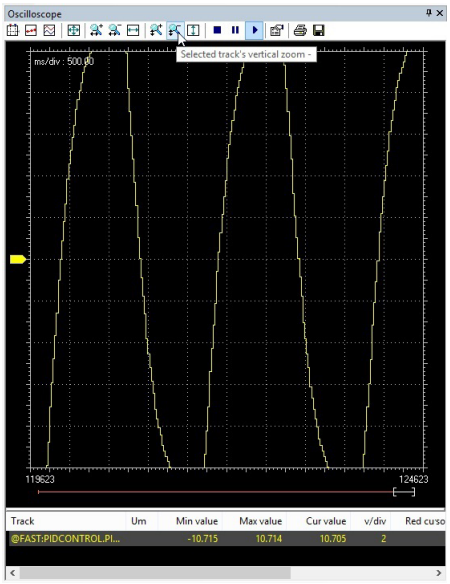
4) Confirm your settings. The graph adapts to reflect the new scale.



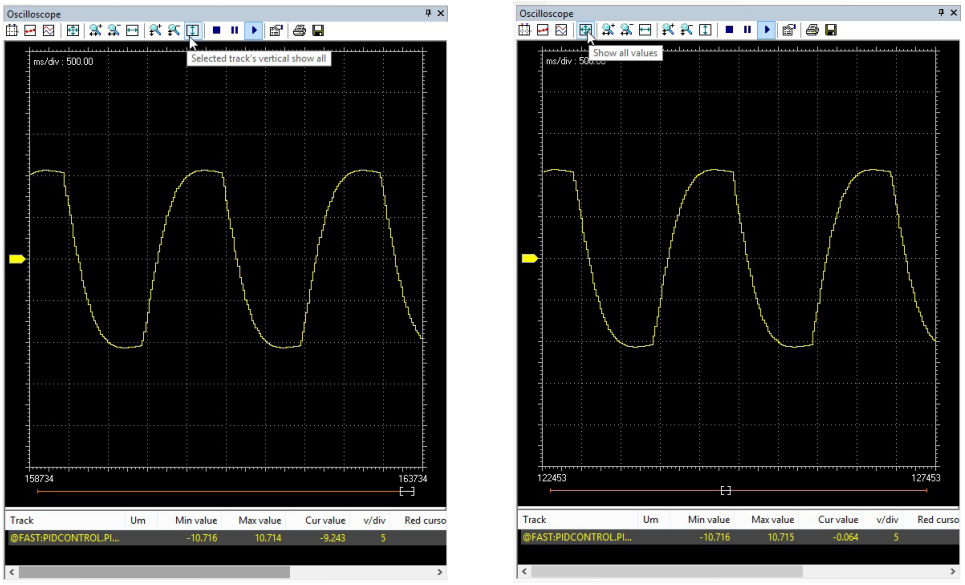


You can also zoom in and out with respect to both the horizontal and the vertical axes.



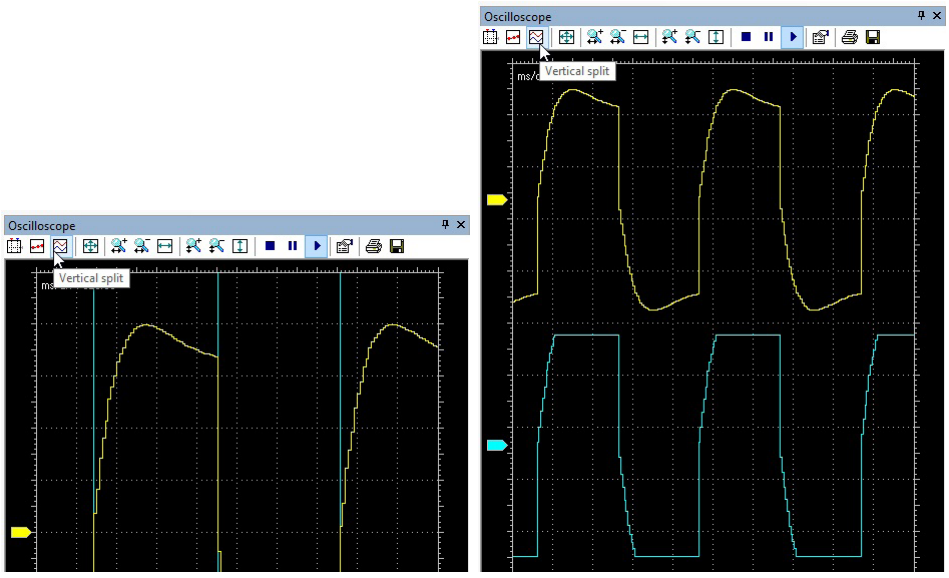


Finally, you may also quickly adapt the scale of the horizontal axis, the vertical axis, or both to include all the samples, by clicking on the corresponding item of the toolbar.



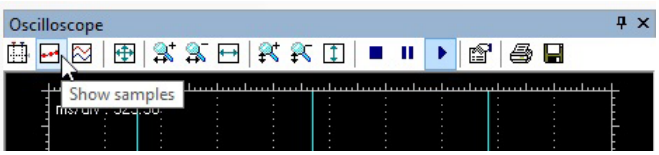
9.2.5.3 VERTICAL SPLIT

When you are watching the evolution of two or more variables, you may want to split the respective tracks. For this purpose, click on the *Vertical split* item in the *Oscilloscope* toolbar.

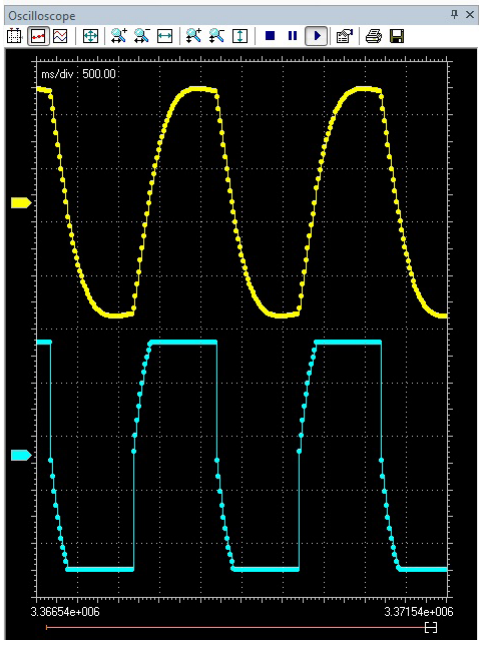


9.2.5.4 VIEWING SAMPLES

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

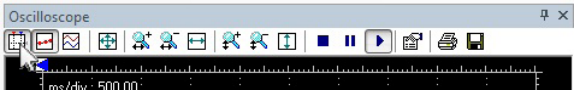


You can click on the same item again, in order to go back to the default view mode.

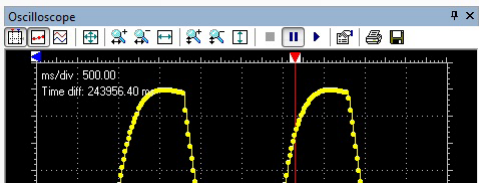


9.2.5.5 TAKING MEASURES

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 on the *Show measure bars* item in the *Oscilloscope* toolbar.

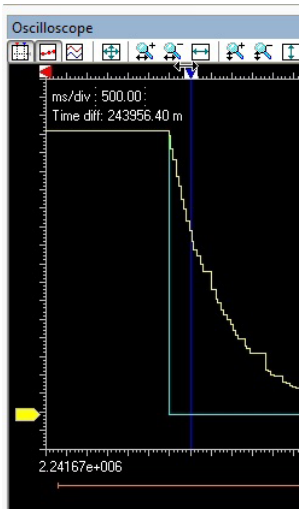


If you want to measure a time interval between two events, you just have to move one bar to the point in the graph that corresponds to the first event and the other to the point that corresponds to the second one.

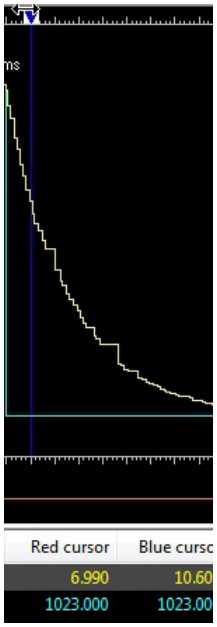


The time interval between the two bars is shown in the top left corner of the chart.

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.



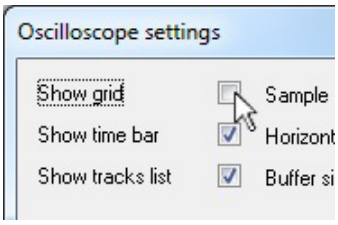
In the table below the chart, you can now read the values of all the variables at that particular moment.



9.2.5.6 OSCILLOSCOPE SETTINGS

You can further customize the appearance of the Oscilloscope by clicking on 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*.

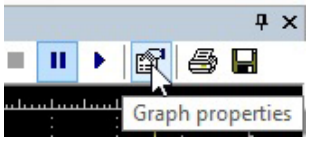


9.2.6 CHANGING THE POLLING RATE

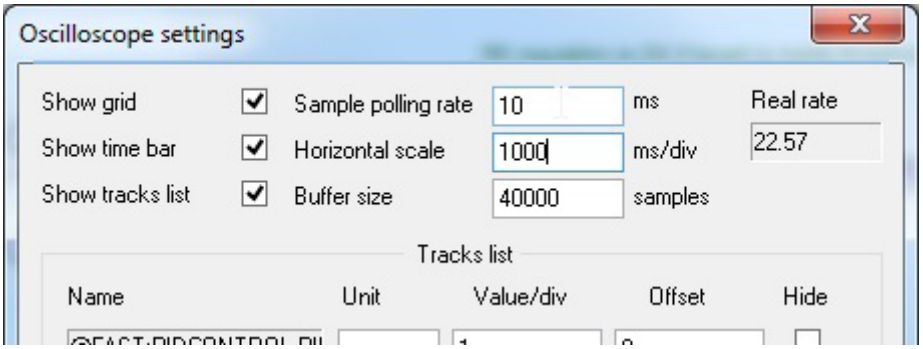
Application periodically sends queries to the target device, in order to read the data to be plotted in the Oscilloscope.

The polling rate can be configured by following this procedure:

1) Click on the *Graph properties* item in the toolbar.

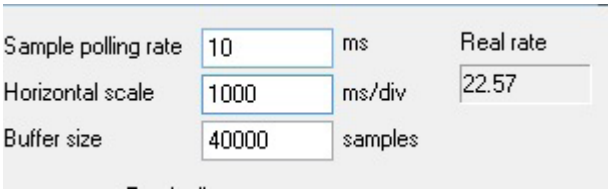


2) In the window that pops up edit the *Sampling polling rate*.



3) Confirm your decision.

Note that the actual rate depends on the performance of the target device (in particular, on the performance of its communication task). You can read the actual rate in the *Oscilloscope settings* window.



9.2.7 SAVING AND PRINTING THE GRAPH

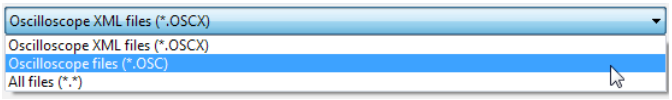
Application allows you to persist the acquisition either by saving the data to a file or by printing a view of the data plotted in the Oscilloscope.

9.2.7.1 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.

1) You may want to stop acquisition before saving data to a file.
2) Click on the *Save tracks data into file* in the *Oscilloscope* toolbar.

3) Choose between the available output file format: `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 tool, provided separately from Application.

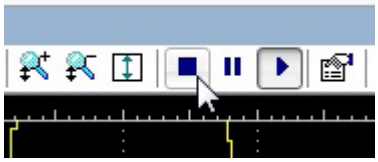


4) Choose a file name and a destination directory, then confirm the operation.

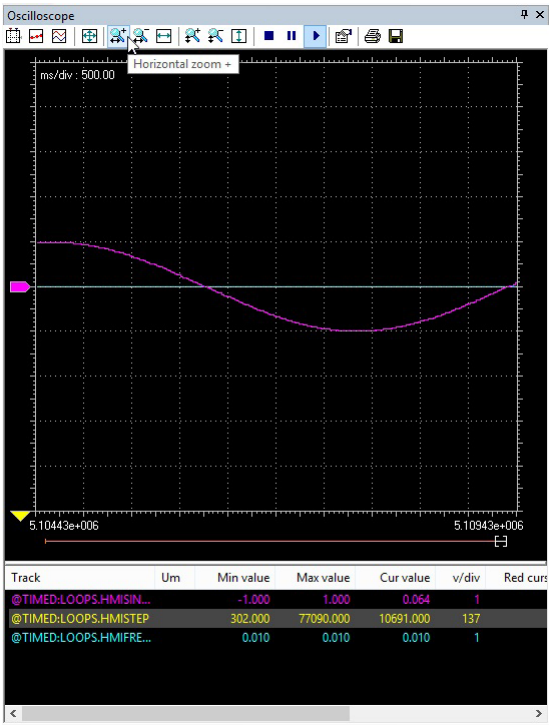
9.2.7.2 PRINTING THE GRAPH

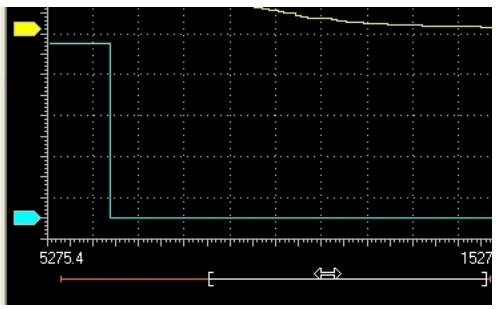
Follow this procedure to print a view of the data plotted in the Oscilloscope:

1) Either suspend or stop the acquisition.

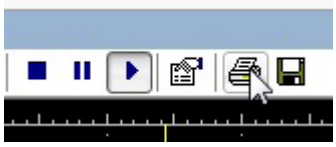


2) Move the time slide bar and adjust the zoom, in order to include in the view the elements you want to print.





3) Click on the *Print graph* item.



9.3 EDIT AND DEBUG MODE

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.

Application 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.

When all the conditions above are not met, the debug mode automatically switches off and Application enters in edit mode.

The status bar shows whether the debug mode is active or not.

Note that you cannot enter the debug mode if the connection status differs from *Connected*.

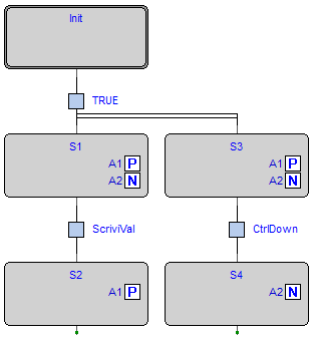
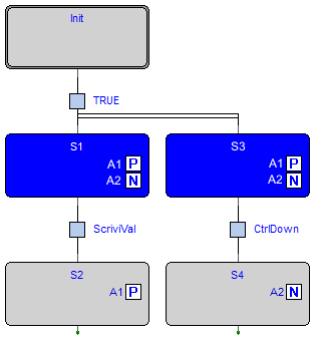
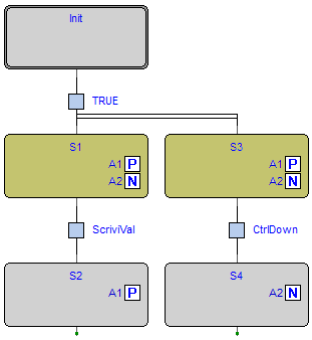
9.4 LIVE DEBUG

Application 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 switch on and off the live debug mode, you may click *Debug>Live debug mode* .

9.4.1 SFC ANIMATION

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
		

In the left column, a portion of an SFC network is shown, diagram animation being off.

In the middle column the same portion of network is displayed when the live debug mode is active. The picture in the middle column shows that steps `S1` and `S3` are currently active, whereas `Init`, `S2`, and `S4` are inactive.

In the right column 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.

Note that the SFC animation manager tests periodically the state of all steps, the user not being allowed to edit the sampling period. Therefore, it may happen that a step remains active for a slot of time too short to be displayed.

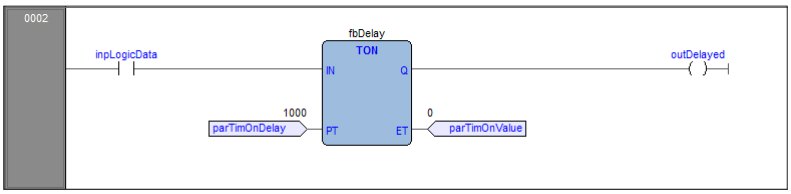
The fact that a step is never highlighted does not imply that its action is not executed, it may simply mean that the sampling rate is too slow to detect the execution.

9.4.1.1 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 code. 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.

9.4.2 LD ANIMATION

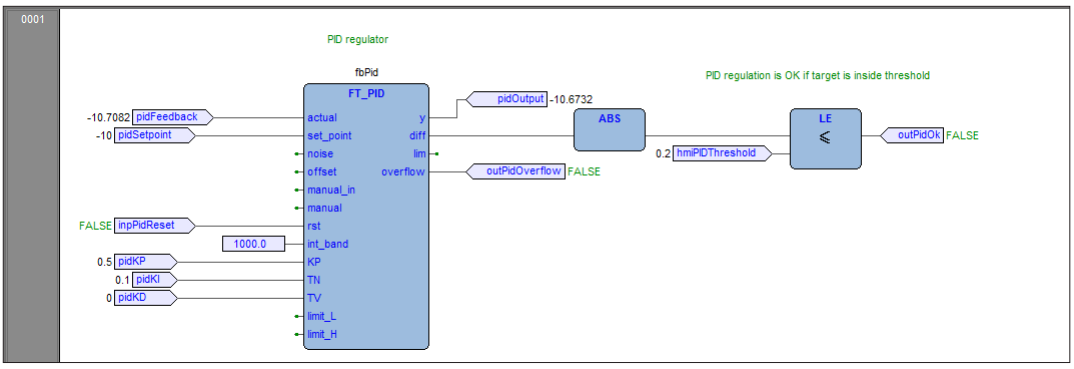
In live debug mode, Ladder Diagram schemes are animated by highlighting the contacts and coils whose value is true (in the example, i1 and i2).



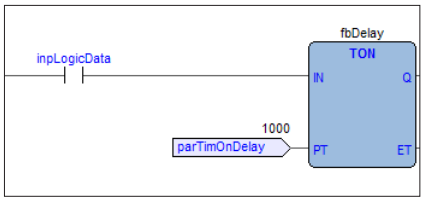
Note that the LD animation manager tests periodically the state of all the elements. It may happen that an element remains true for a slot of time too short to be displayed on the video. The fact that an element is never highlighted does not imply that its value never becomes true (the sampling rate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

9.4.3 FBD ANIMATION

In live debug mode, Application displays the values of all the visible variables directly in the graphical source code editor.



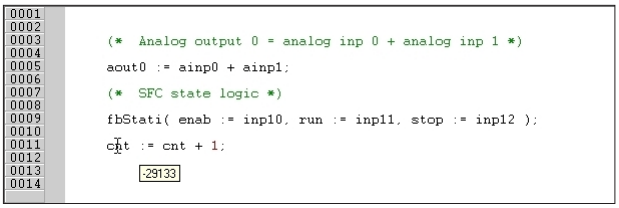
This works for both FBD and LD programming language.



Note that, once again, this tool is asynchronous.

9.4.4 IL AND ST ANIMATION

The live debug mode also applies to textual source code editors (the ones for IL and ST). You can quickly watch the values of a variable by hovering with the mouse over it.



9.5 TRIGGERS

9.5.1 TRIGGER WINDOW

The *Trigger window* tool allows you to select a set of variables and to have them updated synchronously in a special pop-up window.

9.5.1.1 PRE-CONDITIONS TO OPEN A TRIGGER WINDOW

No need for special compilation

Application debugging tools operate at run-time. Thus, unlike other programming languages such as C++, the compiler does not need to be told whether or not to support trigger windows: given a PLC code, the compiler's output is unique, and there is no distinction between debug and release version.

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 memory is available, otherwise an error message appears.

Incompatibility with graphic trigger windows

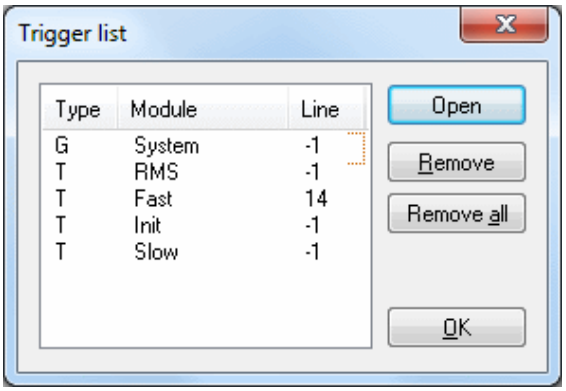
A graphic trigger window takes the whole free space of the application code sector. Therefore, once such a debugging tool has been started, it is not possible to add any trigger window, and an error message appears if you attempt to start a new window. Once the graphic trigger window is eventually closed,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

Note that all the trigger windows existing before the starting of a graphic trigger window keep working normally. You are simply not allowed to add new ones.

9.5.1.2 TRIGGER WINDOW TOOLBAR

Trigger window icons are part of the *Debug* toolbar and are enabled only if Application is in debug mode.

Button	Command	Description
	Set/Remove trigger	In order to actually start a trigger window, select the point of the PLC code where to insert the relative trigger and then press 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 press this button again.
	Graphic trace	This button operates exactly as the above *Set/Remove trigger*, except for that it opens a graphic trigger window. It can be used likewise also to remove a graphic trigger window. Shortcut key: pressing *Shift + F9* is equivalent to clicking on *Set/Remove trigger* button.
	Remove all triggers	Pressing this key causes all the existing trigger windows and the graphic trigger window to be removed simultaneously. Shortcut key: pressing *Ctrl+Shift+F9* is equivalent to clicking on this button.
	Trigger list	This key opens a dialog listing all the existing trigger windows. Shortcut key: pressing *Ctrl+I* is equivalent to clicking on this button.

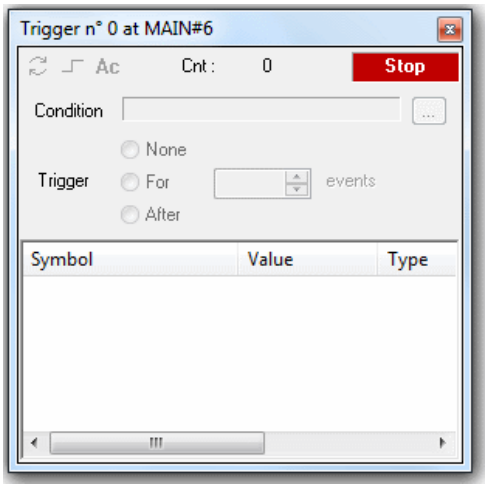


Each record refers to a trigger window, either graphic or textual. The following table explains the meaning of each field.

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 actually 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*.

9.5.1.3 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 shown 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, etc.)
	`BID` = block identifier

Controls section

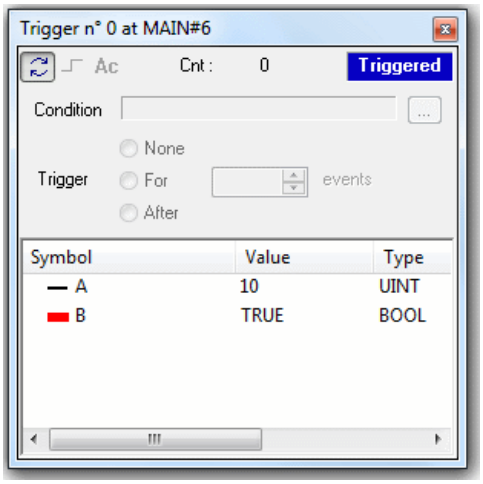
This dialog box allows the user to better control the refresh of the trigger window to get more information on the code under scope. A detailed description of the function of each control is given in the *Trigger window* controls section (see Paragraph 9.5.2.11).

All controls except *Ac*, the *Accumulator display* button, 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 four fields: the name of the variable, its value, its type, and its location (`@task:ModuleName`) read from memory during the last refresh.



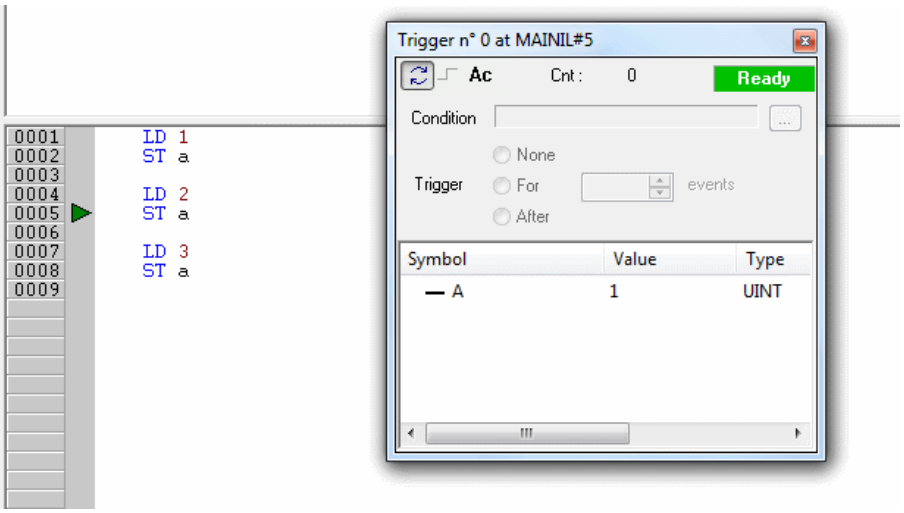
9.5.1.4 TRIGGER WINDOW: DRAG AND DROP INFORMATION

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.

9.5.1.5 REFRESH OF THE VALUES

Let us consider the following example.



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.

Note that 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 above 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.

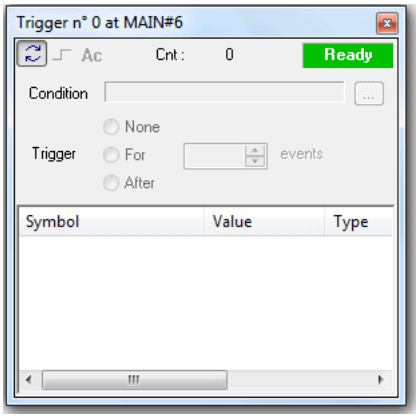
9.5.1.6 TRIGGER WINDOW CONTROLS

This paragraph deals with the trigger window controls, which allows you to better supervise the working of this debugging tool, to get more information on the code under scope.

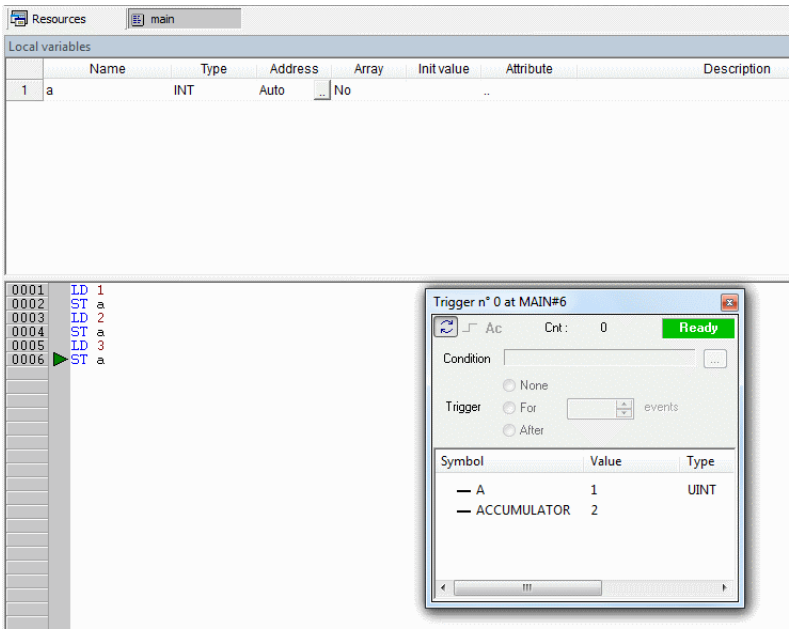
Trigger window controls act in a well-defined way on the behavior of the window, regardless for the type of the module (either IL or FBD) where the related trigger has been inserted.

All controls except the *Accumulator display* are not accessible until at least one variable is dragged into the *Variables* window.

Window controls are made accessible to users through the grey top half of the debug window.



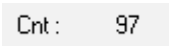
Button	Command	Description
	Start/Stop	This control is used to start a triggering session. If system is triggering you can click this button to force stop. Otherwise session automatically stops when conditions are reached. At this point you can press this button to start another triggering session.
	Single step execution	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. Specified condition is considered. After the single step trigger is done, triggering session automatically stops.
	Accumulator display	This control adds the *Accumulator* to the list of variables already dragged into the trigger window. A new row is added at the bottom of the table of variables, containing the string *Accumulator* in column *Symbol*, the accumulator's value in column *Value*, *Type* is not specified and *Location* is set to global as shown in the following figure.



In order to remove the accumulator from the table, click its name in *Symbol* column, and press the *Del* key.

This control can be very useful if a trigger was inserted before a ST statement, because it allows you to know what value is being written in the destination variable, during the current execution of the task. You can get the same result by moving the trigger to an instruction following the one marked by the green arrowhead.

Trigger counter



This read-only control counts how many times the debug window manager has been triggered, since the window was installed.

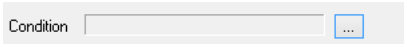
The window manager automatically resets this counter every time a new triggering session is started.

Trigger state

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

Ready	The trigger has not occurred during the current task execution.
Triggered	The trigger has occurred during the current task execution.
Stop	System is not triggering. Triggering has not been started yet or it has been stopped by user or an 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 re-

freshed 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 the user 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.

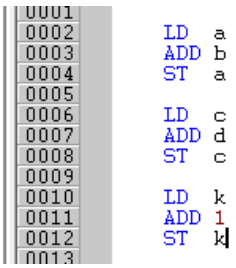
9.5.2 DEBUGGING WITH TRIGGER WINDOWS

9.5.2.1 INTRODUCTION

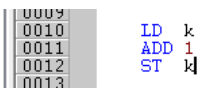
The trigger window tool allows the user to select a set of variables and to have their values displayed and updated synchronously in a pop-up window. Unlike the *Watch* window, trigger windows refresh simultaneously all the variables they contain, every time they are triggered.

9.5.2.2 OPENING A TRIGGER WINDOW FROM AN IL MODULE

Let us assume that you have an IL module, also containing the following instructions.



Let us also 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.



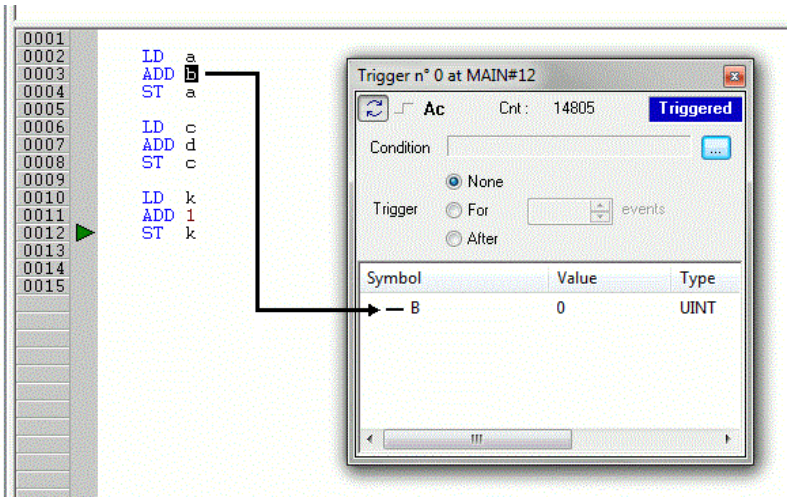
Then you can click *Debug>Set/Remove trigger*.

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.

9.5.2.3 ADDING A VARIABLE TO A TRIGGER WINDOW FROM AN IL MODULE

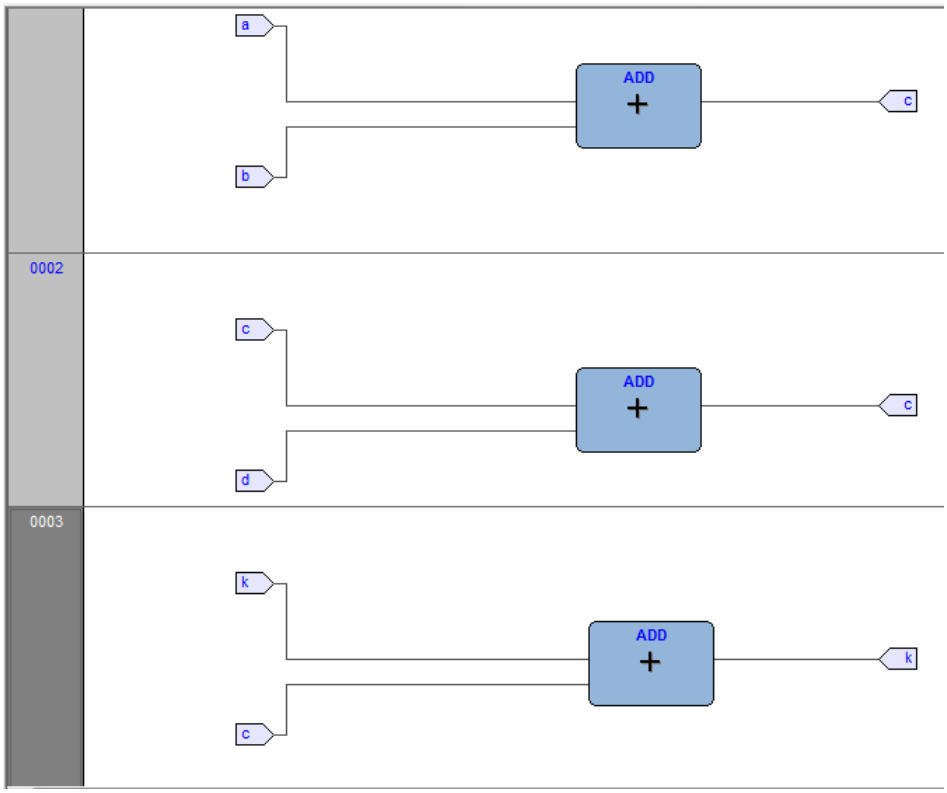
In order to watch the value of a variable, you need to add it to the trigger window. To this purpose, select a variable by double-clicking it, and then drag it into the *Variables* window, that is the lower white box in the pop-up window. The variable's name now appears in the *Symbol* column.



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

9.5.2.4 OPENING A TRIGGER WINDOW FROM AN FBD MULE

Let us assume that you have an FBD module, also containing the following instructions.



Let us also assume that you want to know the values of C, D, and K, just before the ST

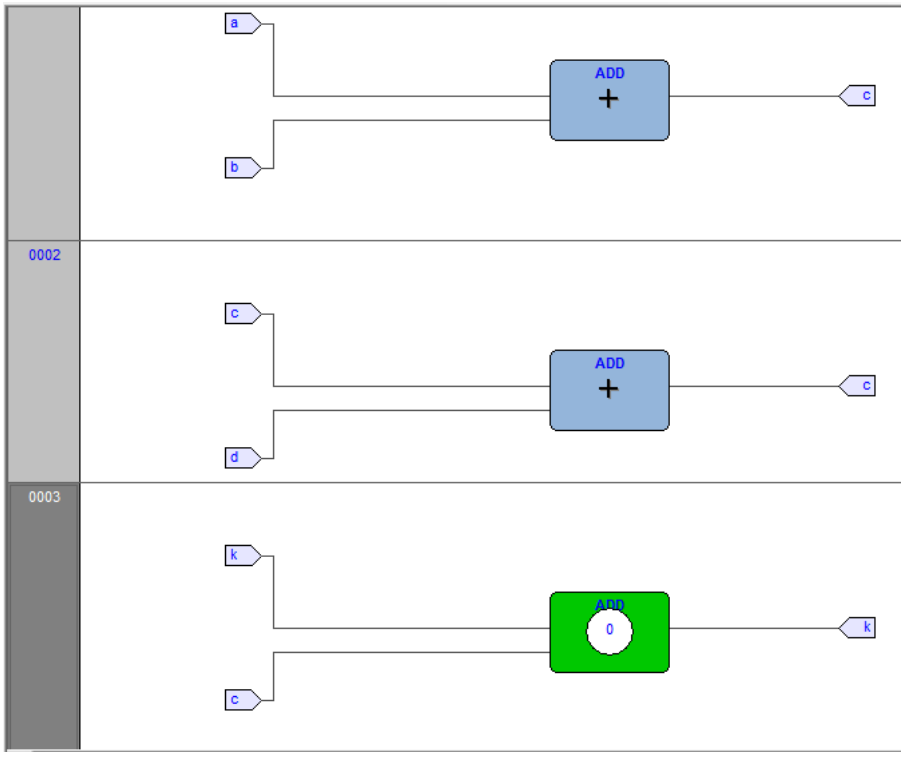
k instruction is executed.

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 above figure, you must move the cursor to network 3, and click the ADD block.

You can click *Debug>Set/Remove 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 actually place the trigger before the first statement of its IL equivalent code.

9.5.2.5 ADDING A VARIABLE TO A TRIGGER WINDOW FROM AN FBD MODULE

In order to watch the value of a variable, you need to add it to the trigger window. Let us assume that you want to inspect the value of variable k of the FBD code in the figure below.

To this purpose, click *Edit>Watch mode*.

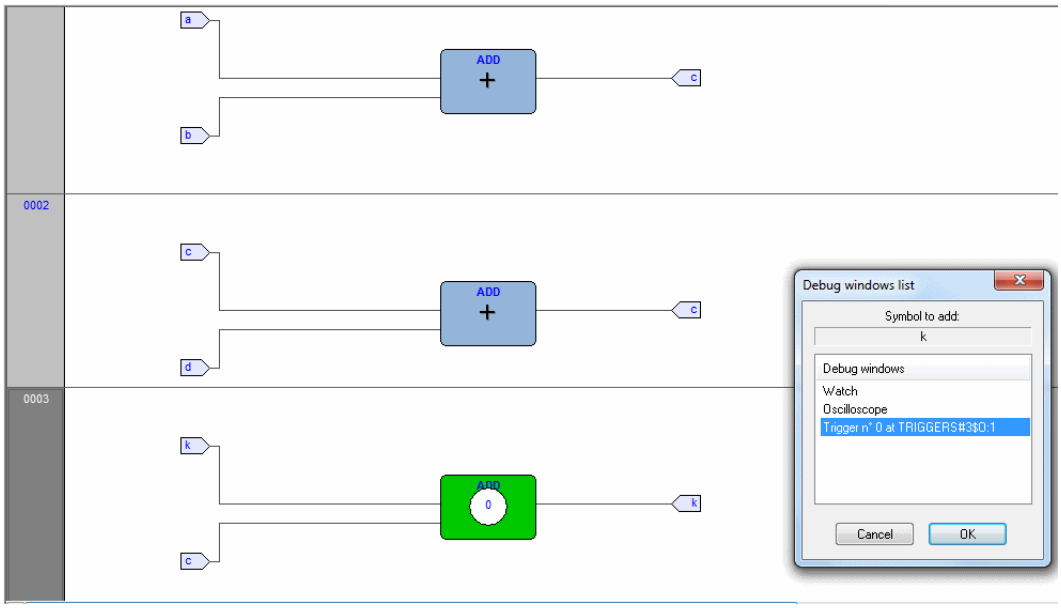
The cursor will become as follows.

Now you can click the block representing the variable you wish to be shown in the trigger

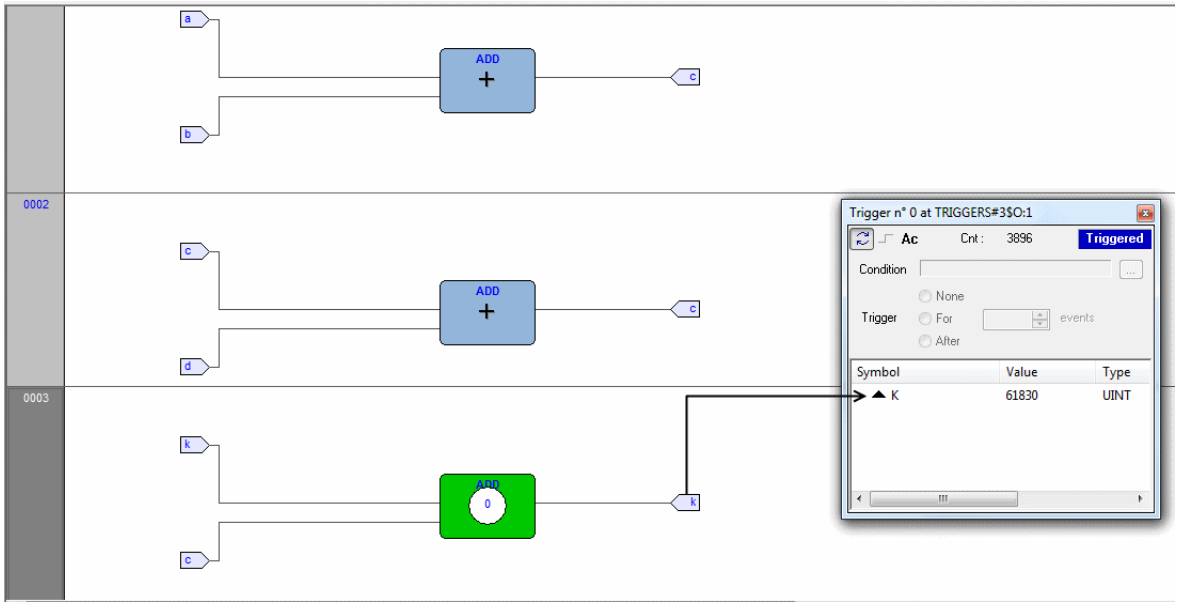
window.

In the example we are considering, 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 press *OK*. The name of the variable is now printed in the *Symbol* column.

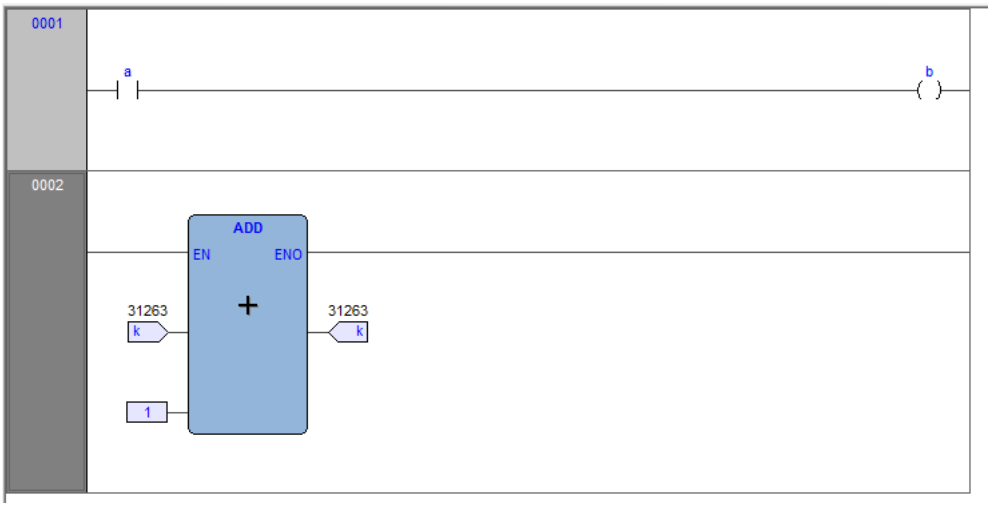


The same procedure applies to all the variables you wish to inspect.

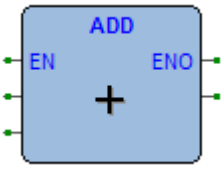
Once you have added to the *Graphic watch* window all the variables you want to observe, you can click *Edit>Insert/Move mode*, so as to let the cursor take back its original shape.

9.5.2.6 OPENING A TRIGGER WINDOW FROM AN LD MODULE

Let us assume that you have an LD module, also containing the following instructions.



You can place a trigger on a block such as follows.



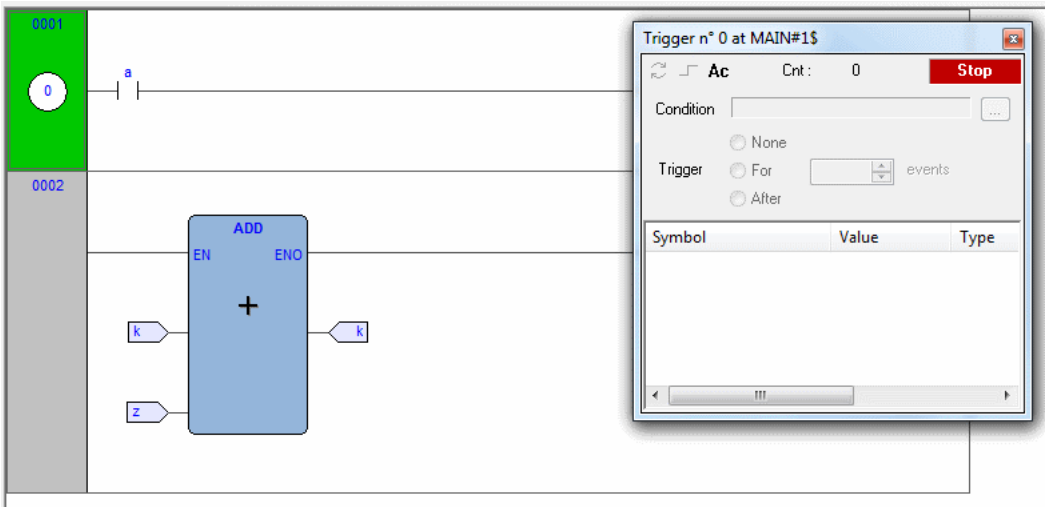
In this case, the same rules apply as to insert a trigger in an FBD module on a contact

or a coil

In this case, follow the SE instructions. Let us 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 grey raised button containing the network number turns to green, and a white circle with the number of the trigger inside appears in the middle of the button, while the related trigger window pops up.



Unlike the other languages supported by Application, 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.

9.5.2.7 ADDING A VARIABLE TO A TRIGGER WINDOW FROM AN LD MODULE

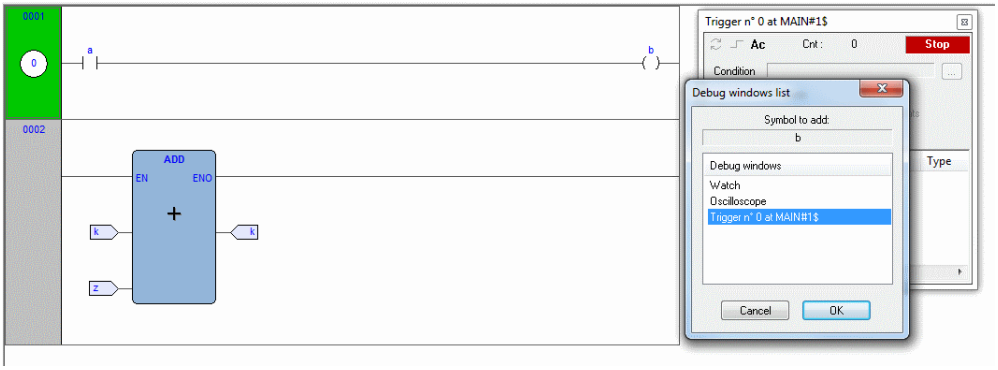
In order to watch the value of a variable, you need to add it to the trigger window. Let us assume that you want to inspect the value of variable `b` in the LD code represented in the figure below.

To this purpose, click *Edit>Watch mode*.

The cursor will become as follows.

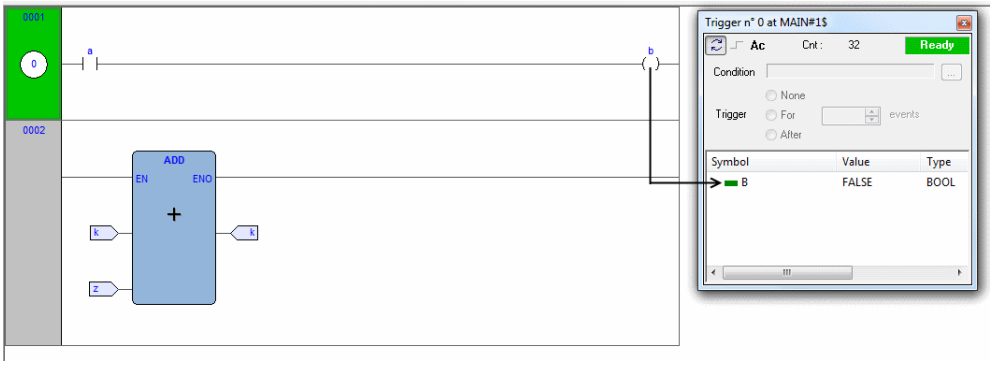
Now you can click the item representing the variable you wish to be shown in the trigger window.

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.



In order to display variable `B` in the trigger window, select its reference in the *Debug window* column, then press *OK*.

The name of the variable is now printed in the *Symbol* column.

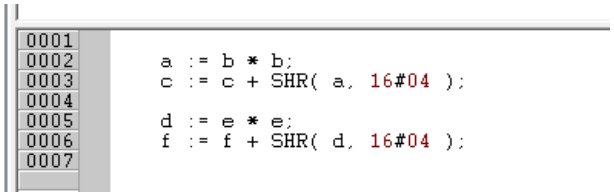


The same procedure applies to all the variables you wish to inspect.

Once you have added to the *Graphic watch* window all the variables you want to observe, you can click *Edit>Insert/Move mode* , so as to restore the original shape of the cursor.

9.5.2.8 OPENING A TRIGGER WINDOW FROM AN ST MODULE

Let us assume that you have an ST module, also containing the following instructions.



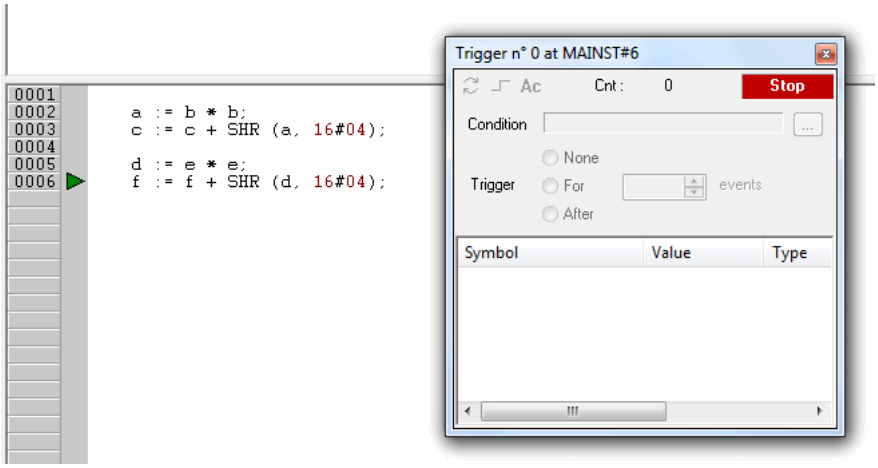
Let us also assume that you want to know the value of `e`, `d`, and `f`, just before the instruction

```
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* .

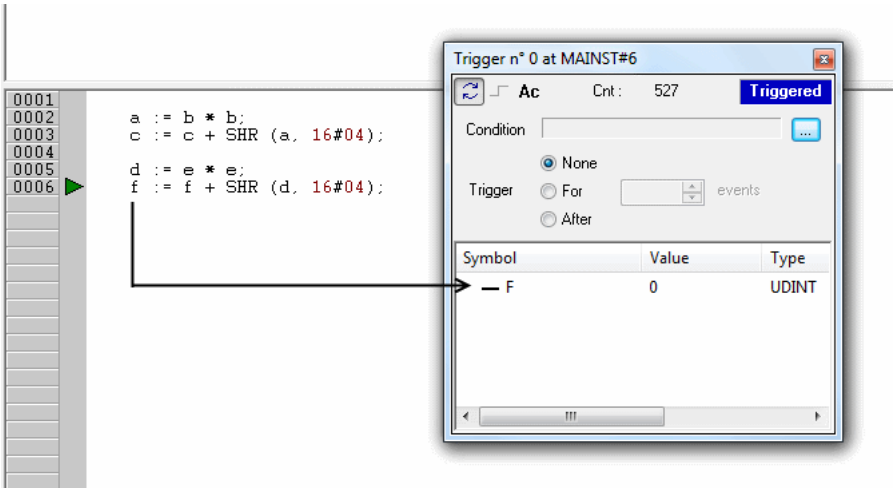
In both cases, a green arrowhead appears next to the line number, and the related trigger window pops up.



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`, etc..

9.5.2.9 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. To this purpose, 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 inspect.

9.5.2.10 REMOVING A VARIABLE FROM THE TRIGGER WINDOW

If you want a variable not to be displayed any more in the trigger window, select it by clicking its name once, then press the *Del* key.

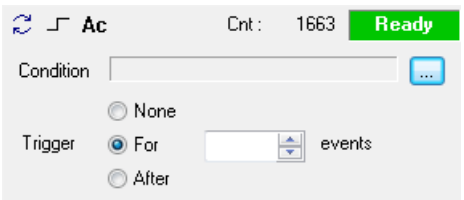
9.5.2.11 USING CONTROLS

This paragraph deals with trigger windows controls, which allow you to better supervise the working of this debugging tool to get more information on the code under scope. The main purpose of trigger window controls is to let you define more limiting conditions, so that variables in *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 single time the processor reaches the relative trigger.

Enabling controls

When you set a trigger, all the elements in the *Control* window look disabled.

As a matter of fact, you cannot access any of the controls, except the *Accumulator* display, until at least one variable is dragged into the *Debug* window. When this happens triggering automatically starts and the *Controls* window changes as follows.



Triggering can be started/stopped with the apposite 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 set the counter to *1* and select *For*.

If you want the values to be refreshed the first *X* times the window is triggered, set the counter to *X* and select *For*.

If you want the values to be refreshed after *Y* times the window is triggered, set the counter to *Y* and select *After*.

Triggers and conditions settings become the actual settings when the triggering is (re) started.

Watching the accumulator

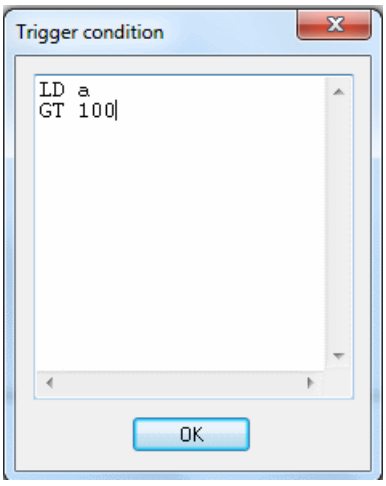
As stated in the Refresh of values section (see Paragraph 9.5.1.5), when you insert a trigger on an instruction line, you establish that the variables in the relative debugging window will be updated every time the processor reaches that location, before the instruction itself is executed. In some cases, for example when a trigger is placed before a ST statement, it can be useful to know the value of the accumulator. This allows you to forecast the outcome of the instruction that will be executed after all the variables in the trigger window have been updated. To add the accumulator to the trigger window, click on the *Accumulator display* button.

Defining a condition

This control enables users 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.

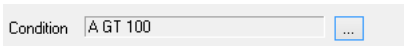
If you want to put a restriction on the refreshment mechanism, you can specify a condition by clicking on the apposite button.

When you do so, 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 the *OK* button to install it, or press the *Esc* button 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, press again the above mentioned button.

The text window appears, containing the text you originally wrote, which you can now edit.

To completely remove a user-defined condition, delete the whole IL code in the text window, then click *OK*.

After the execution of the condition code, the accumulator 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. No new variables can be declared in the condition window.

9.5.2.12 CLOSING A TRIGGER WINDOW AND REMOVING A TRIGGER

This web page deals with what you can do when you finish a debug session with a trigger window. You can choose between the following options.

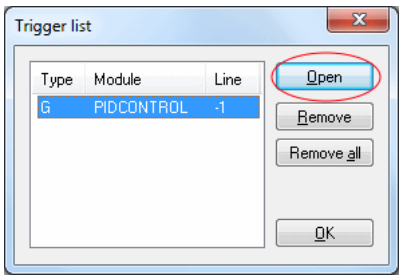
- Closing the trigger window.
- Removing the trigger.
- Removing all the triggers.

Notice that the actions listed above produce very different results.

Closing the trigger window

If you have finished watching a set of variables by means of 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 just hide the interface window, while the window manager and the relative trigger keep working.

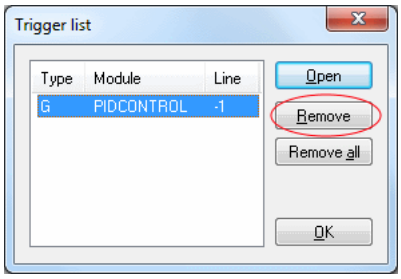
As a matter of fact, if later you want to resume debugging with a trigger window that you previously hid, you just need to open the *Trigger list* window, to select the record referred to that trigger window, and to click the *Open* button.



The interface window appears with value of variables and trigger counter updated, as if it had not been closed.

Removing a trigger

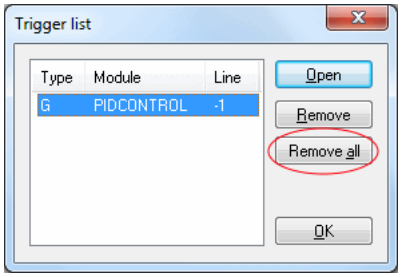
If you choose this option, you completely remove the code both of the window manager and of its trigger. To this purpose, just open the *Trigger list* window, select the record referred to the trigger window you want to eliminate, and click the *Remove* button.



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 press the *Set/Remove 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 on the *Remove all* button.



9.6 GRAPHIC TRIGGERS

9.6.1 GRAPHIC TRIGGER WINDOW

The graphic trigger window tool allows you to select a set of variables and 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 (i.e. the instruction - if IL, ST - or the block - if FBD, LD) where you placed the trigger.

9.6.1.1 PRE-CONDITIONS TO OPEN A GRAPHIC TRIGGER WINDOW

No need for special compilation

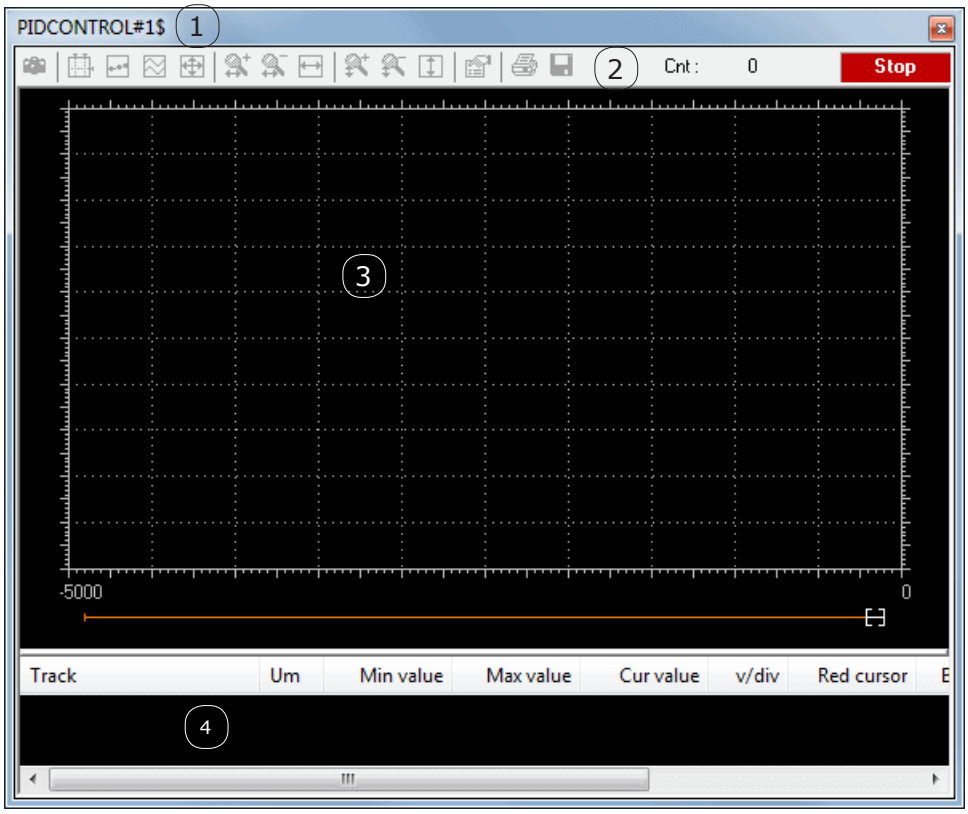
All the Application debugging tools operate at run-time. Thus, unlike other programming languages such as C++, the compiler does not need to be told whether or not to support trigger windows: given a PLC code, the compiler's output is unique, and there is no distinction between debug and release version.

Memory availability

A graphic trigger window takes all the free memory space in the application code sector. Obviously, in order to start up a trigger window, it is necessary that a sufficient amount of memory is available, otherwise an error message appears.

9.6.1.2 GRAPHIC TRIGGER WINDOW INTERFACE

Setting a graphic trigger causes a pop-up window to appear, which is called *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 shown 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, etc.)
	`BID` = block identifier

The Controls bar

This dialog box allows you to better 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 (see Paragraph 9.6.1.5).

The Chart area

The *Chart* area includes six items:

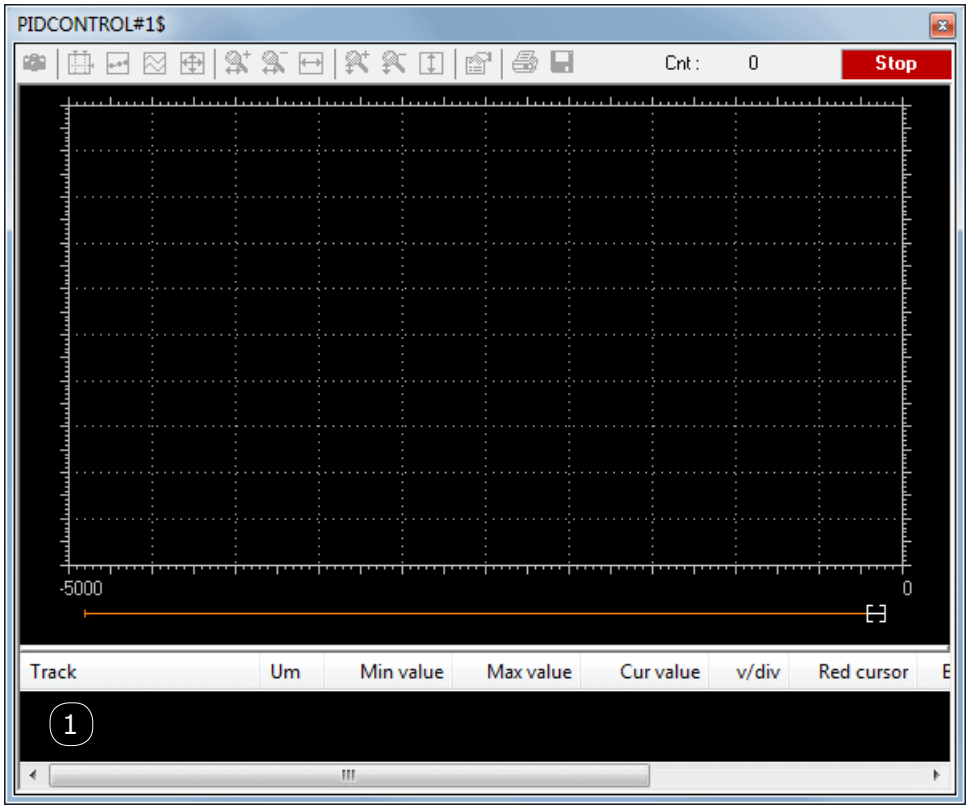
1) Plot: area containing the actual plot of the curve of the dragged-in variables.
2) Samples to acquire: number of samples to be collected by the graphic trigger window manager.
3) Horizontal cursor: cursor identifying a horizontal line. The value of each variable at the intersection with this line is reported in the column *horz cursor*.
4) 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*.
5) Red cursor: same as blue cursor.
6) 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. Every row has several fields, which are described in detail in the Drag and drop information section.

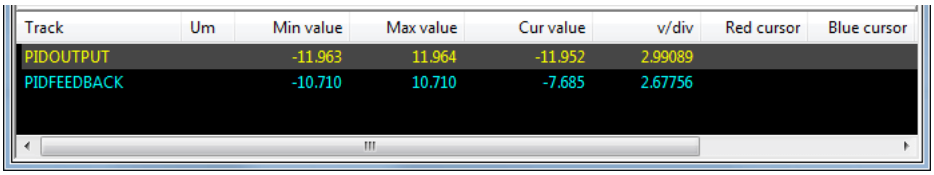
9.6.1.3 GRAPHIC TRIGGER WINDOW: DRAG AND DROP INFORMATION

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



1. *Variables window*

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 shown in the picture below.



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	Current value of the variable.
v/div	How many engineering units are represented by a unit of the y-axis (i.e. the space between two ticks on the vertical axis).
Blue cursor	Value of the variable at the intersection with the line identified by the blue cursor.
Red cursor	Value of the variable at the intersection with the line identified by the red cursor.
Horz cursor	Value of the variable at the intersection with the line identified by the horizontal cursor.

Note that 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, or function, or function block.

9.6.1.4 SAMPLING OF VARIABLES

Let us consider the following example.

The value of the variables is sampled 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 sampled when triggers also satisfy further limiting conditions that you define.

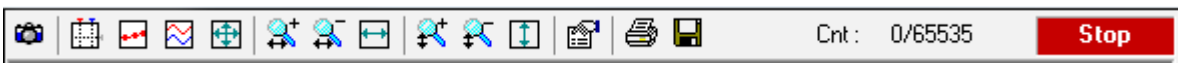
The value of the variables in the column *Track* is read from memory just before the marked instruction and immediately after the previous instruction.

9.6.1.5 GRAPHIC TRIGGER WINDOW CONTROLS

This paragraph deals with controls of the *Graphic trigger* window. Controls allow you to specify in detail when Application is supposed to sample the variables added to the *Variables* window.

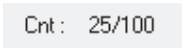
Graphic trigger window controls act in a well-defined way on the behavior of the window, regardless for the type of the module (IL, ST, FBD or LD) where the related trigger has been inserted.

Window controls are made accessible to users through the *Controls* bar of the debug window.



Button	Command	Description
	Start graphic trace	When you push this button down, you let acquisition start. Now, if acquisition is running and you release this button, you stop the sample collection process, and you reset all the data you have acquired so far.
	Enable/Disable cursors	The two cursors (red cursor, blue cursor) may be seen and moved along their axis as long as this button is pressed. Release this button if you want to hide simultaneously 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.
	Split variables	When pressed, 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 setrecord set.
	Horizontal Zoom In and *Zoom Out*	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 will be placed on the left margin, and last will be placed on the right margin of the graphic window.
	Vertical Zoom In and *Zoom Out*	*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 will be placed near top margin and low value sample will be placed on the bottom margin of the graphic window.
	Graphic trigger window properties	Pushing this button causes a tabs dialog box to appear, which allows you to set general user options affecting the action of the graphic trigger window. Since the options you can set are quite numerous, they are dealt with in a section apart. Click here to access this section.
	Print chart	Push this button to print both the *Chart area* and the *Variables* window.
	Save chart	Press this button to save the chart.

Trigger counter



This read-only control displays two numbers with the following format: 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

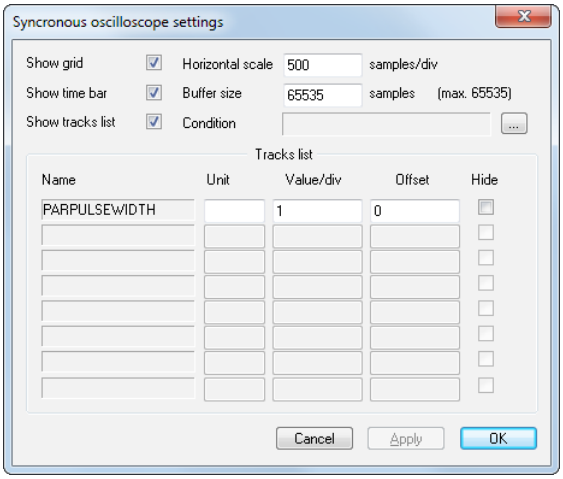
This read-only control shows you the state of the *Debug* window. It can assume the following values.

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.

9.6.1.6 GRAPHIC TRIGGER WINDOW OPTIONS

In order to open the options tab, you must click the *Properties* button in the *Controls* bar. When you do this, the following dialog box appears.

General

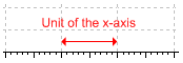


Control

Control	Description
Show grid	Tick 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 checked.

Control	Description
Show tracks list	The *Variables* window is shown as long as this box is checked, 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. By unit of the x-axis the space is meant between two vertical lines of the background grid.
Buffer size	Number of samples to acquire. When you open the option tab, after having dragged-in all the variables you want to watch, you can read a default number in this field, representing the maximum number of samples you can collect for each variable. You can therefore type a number which is less or equal to the default one.

Tracks

This tab allows you to define some graphic properties of the plot of each variable. To select a variable, click its name in the *Track list* column.

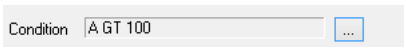
Control	Description
Unit	Unit of measurement, printed in the table of the *Variables* window.
Value/div	Δ value per unit of the y-axis. By unit of the y-axis is meant the space between two horizontal lines of the background grid.
Hide	Check this flag to hide selected track on the graph.

Push *Apply* to make your changes effective, or push *OK* to apply your changes and to close the options tab.

User-defined condition

If you define a condition by using this control, the sampling process does not start until that condition is satisfied. Note that, unlike trigger windows, once data acquisition begins, samples are taken every time the window manager is triggered, regardless of the user condition being still true or not.

After you enter a condition, the control displays its simplified expression.



9.6.2 DEBUGGING WITH THE GRAPHIC TRIGGER WINDOW

The graphic trigger window tool allows you to select a set of variables and to have them sampled synchronously and their curve displayed in a special pop-up window.

9.6.2.1 OPENING THE GRAPHIC TRIGGER WINDOW FROM AN IL MODULE

Let us assume that you have an IL module, also containing the following instructions.

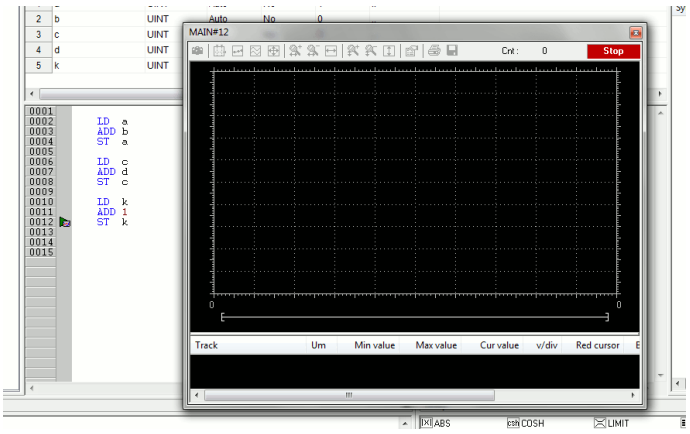
```
0001
0002    LD  a
0003    ADD b
0004    ST  a
0005
0006    LD  c
0007    ADD d
0008    ST  c
0009
0010    LD  k
0011    ADD 1
0012    ST  k
0013
```

Let us also 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/Remuve grafic trigger*.

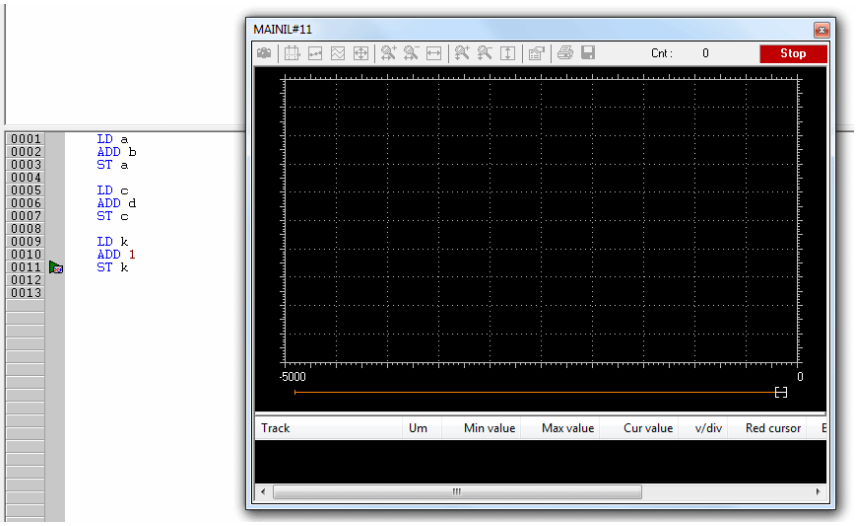
A green arrowhead appears next to the line number, and the graphic trigger window pops up.



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.

9.6.2.2 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. To this purpose, select a variable, by double clicking it, and then drag it into the *Variables* window. The variable now appears in the *Track* column.

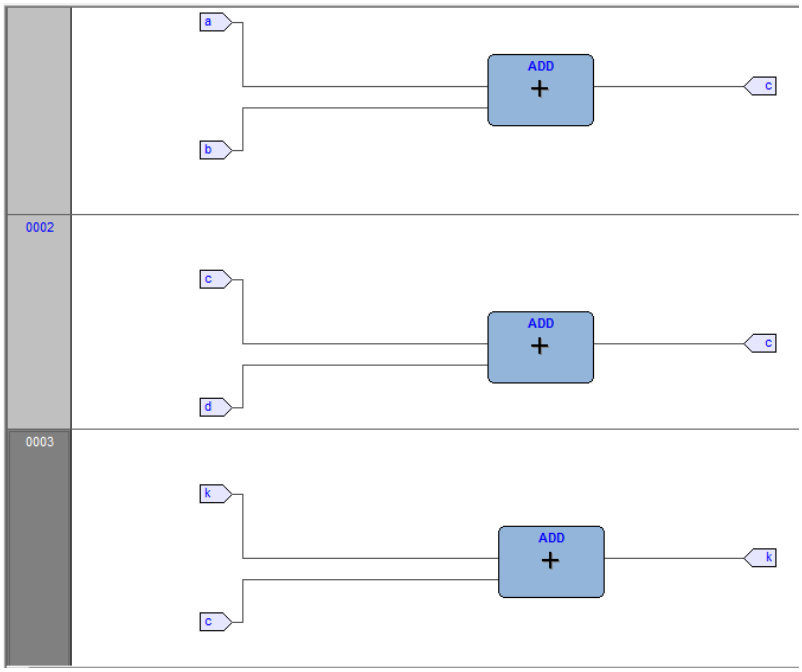


The same procedure applies to all the variables you wish to inspect.

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

9.6.2.3 OPENING THE GRAPHIC TRIGGER WINDOW FORM AN FBD MODULE

Let us assume that you have an FBD module, also containing the following instructions.



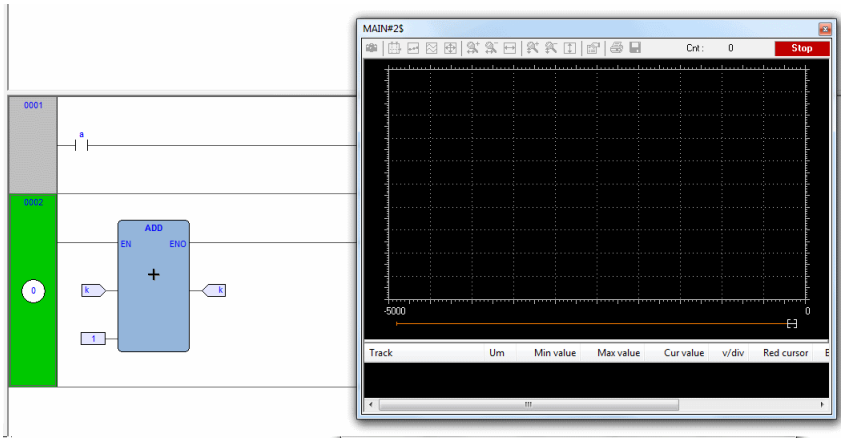
Let us also assume that you want to know the values of `c`, `d`, and `k`, just before the `ST k` instruction is executed.

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 above figure, you must move the cursor to network 3, and click the `ADD` block.

Now click *Debug>Add/Remuve grafic trigger*.

This causes the colour 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, 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 actually place the trigger before the first statement of its IL equivalent code.

9.6.2.4 ADDING A VARIABLE TO THE GRAPHIC TRIGGER WINDOW FROM AN FBD MODULE

In order to watch the diagram of a variable, you need to add it to the trigger window. Let us assume that you want to see the plot of the variable *k* of the FBD code in the figure below.

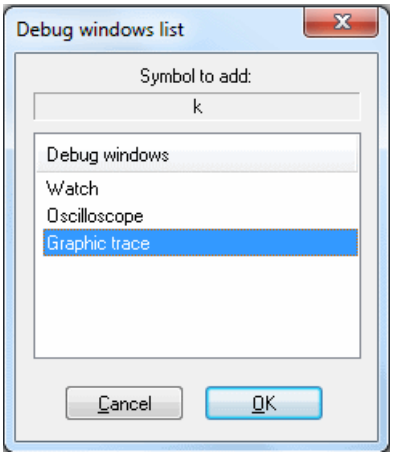
To this purpose, click *Edit>Watch mode*.

The cursor will become as follows.

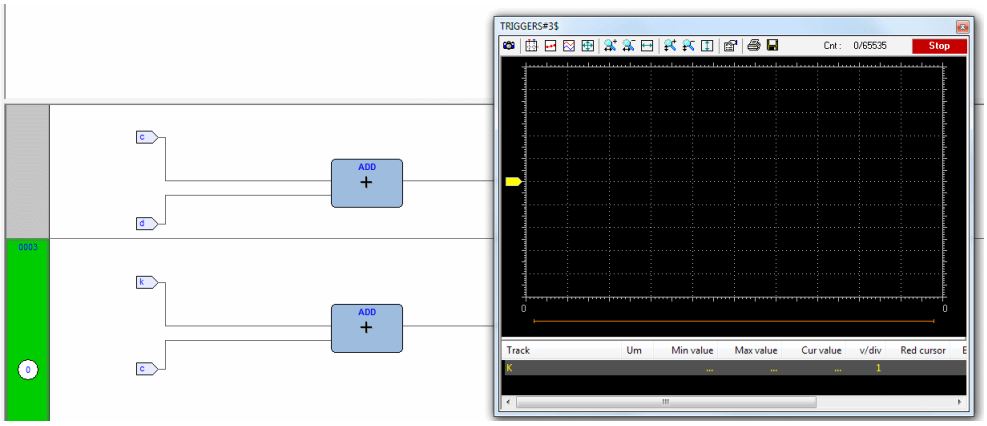
Now you can click the block representing the variable you wish to be shown in the graphic trigger window.

In the example we are considering, 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 press *OK*. The name of the variable is now printed in the *Track* column.



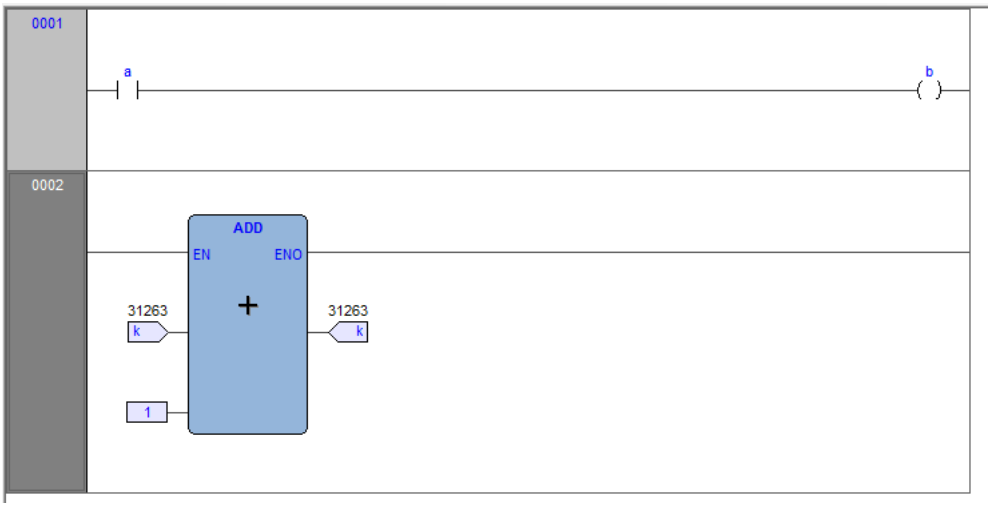
The same procedure applies to all the variables you wish to inspect.

Once you have added to the *Graphic watch* window all the variables you want to observe, you can click *Edit>Insert/Move mode*, in order to restore the original cursor.

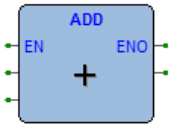
Once the first variable is dropped into a graphic trace, the *Graphic properties* window is automatically shown and allows the user to setup sampling and visualization properties.

9.6.2.5 OPENING THE GRAPHIC TRIGGER WINDOW FROM AN LD MODULE

Let us assume that you have an LD module, also containing the following instructions.



You can place a trigger on a block such as follows.



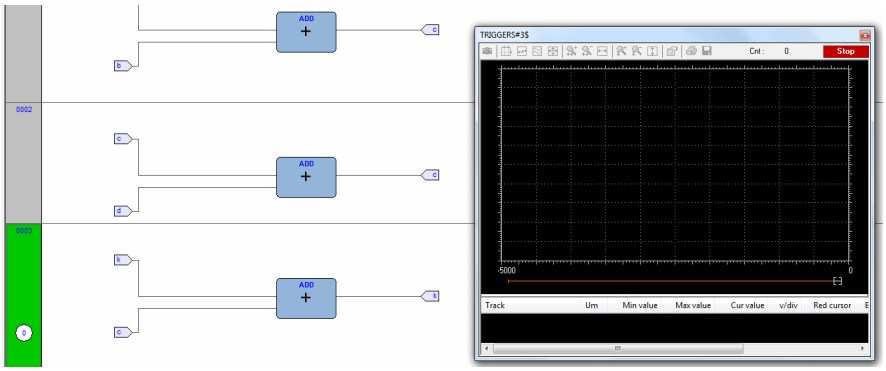
In this case, the same rules apply as to insert the graphic trigger in an FBD module on a contact

or coil

In this case, follow the instructions. Let us also 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 nr. 1, then click *Debug>Add/Remuve grafic trigger*

This causes the grey raised button containing the network number to turn to green, a white circle with a number inside to appear in the middle of the button, and the graphic trigger window to pop up.



Note that unlike the other languages supported by Application, 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.

9.6.2.6 ADDING A VARIABLE TO THE GRAPHIC TRIGGER WINDOW FROM AN LD MODULE

In order to watch the diagram of a variable, you need to add it to the *Graphic trigger* window. Let us assume that you want to see the plot of the variable `b` in the LD code represented in the figure below.

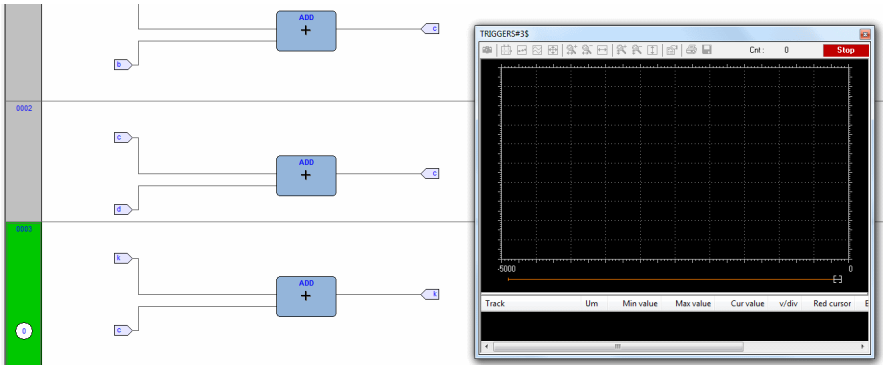
To this purpose, click *Edit>Watch mode*.

The cursor will become as follows.

Now you can click the item representing the variable you wish to be shown 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 press *OK*. The name of the variable is now printed in the *Track* column.



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

Once you have added to the *Graphic watch* window all the variables you want to observe, you can click *Edit>Insert/Move mode*, so as to restore the original shape of the cursor.

Once the first variable is dropped into a graphic trace, the *Graphic properties* window

is automatically shown and allows the user to setup sampling and visualization properties.

9.6.2.7 OPENING THE GRAPHIC TRIGGER WINDOW FROM AN ST MODULE

Let us assume that you have an ST module, also 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
```

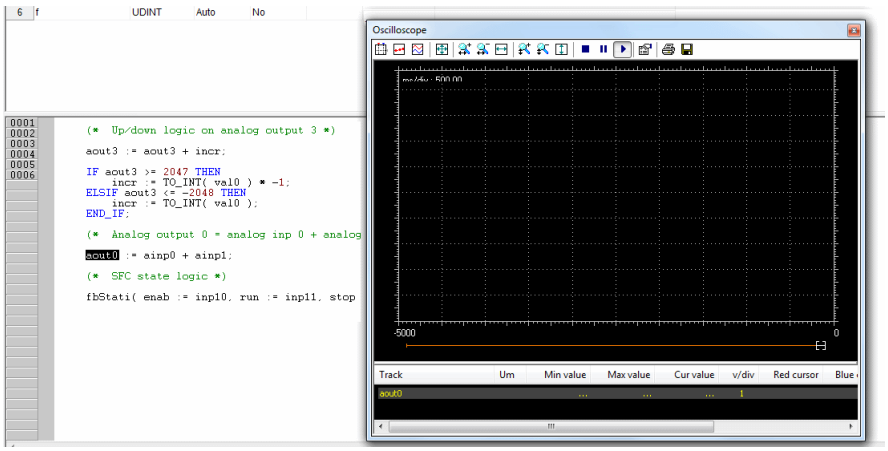
Let us also assume that you want to know the value of `e`, `d`, and `f`, just before the instruction

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

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

Then click *Debug>Add/Remuve grafic trigger*.

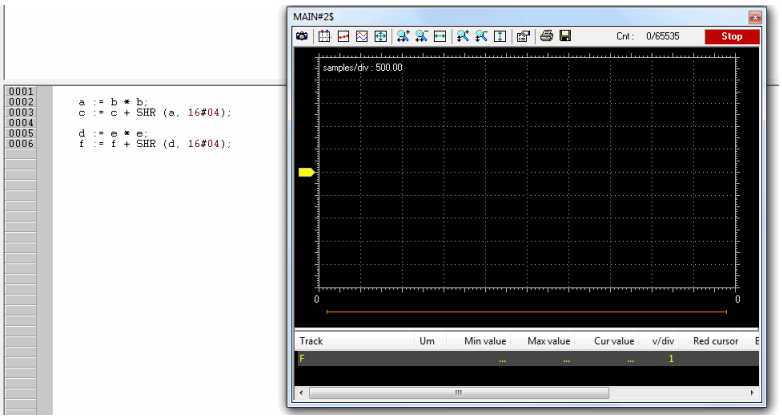
A green arrowhead appears next to the line number, and the *Graphic trigger* window pops up.



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`, etc.

9.6.2.8 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. To this purpose, 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 inspect.

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

9.6.2.9 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 *Del* key.

9.6.2.10 USING CONTROLS

This paragraph deals with graphic trigger window controls, which allow you to better supervise the working of this debugging tool, so as to get more information on the code under scope.

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 acquisition* button.

If you defined a user condition, which is currently false, data acquisition does not start, even though you press the apposite button.

On the contrary, once the condition becomes true, data acquisition starts and continues until the *Start graphic trace acquisition* button is released, regardless for the condition being or not still true.

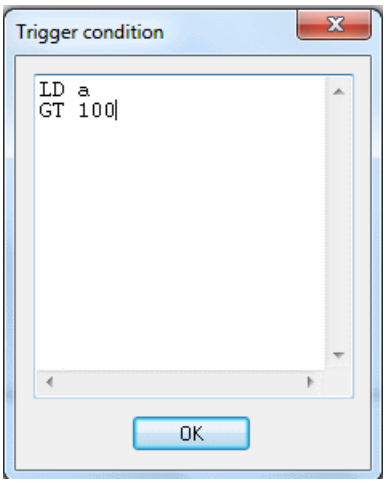
if you release the *Start graphic trace acquisition* button before all the required samples have been acquired, the acquisition process stops and all the collected data get lost.

Defining a condition

This control enables users to set a condition on when to start acquisition. By default, this condition is set to true, and acquisition begins as soon as you press 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 press 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 the *OK* button to install it, or press the *Esc* button to cancel. The collection of samples will not start until the *Start graphic trace acquisition* button is pressed and the user-defined condition is true. A simplified expression of the condition now appears in the control.

To modify it, press again the relevant button.

The text window appears, containing the text you originally wrote, which you can now edit.

To completely remove a user-defined condition, press again on the above mentioned button, delete the whole IL code in the text window, then click *OK*.

After the execution of the condition code, the accumulator 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, Application 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.

9.6.2.11 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.
- Removing the trigger.

- Removing all the triggers.

Closing the graphic trigger window

If you have finished plotting the diagram of a set of variables by means of 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 just hide the *Interface* window, while the window manager and the relative trigger keep working.

As a matter of fact, if later you want 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. To this purpose:

- 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 press 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 on the *Remove all triggers* button.

10. APPLICATION REFERENCE

10.1 MENUS REFERENCE

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

10.1.1 FILE MENU

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

10.1.2 EDIT MENU

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

10.1.3 VIEW MENU

Command	Icon	Key	Description
Toolbar			
Main Toolbar			Shows or hides the *Main* toolbar.
Status bar			Shows or hides the *Status* bar.
Debug bar		*Ctrl+B*	Shows or hides the *Debug* toolbar.
FBD bar		*Ctrl+D*	Shows or hides the *FBD* toolbar.
LD bar		*Ctrl+A*	Shows or hides the *LD* toolbar.
SFC bar		*Ctrl+Q*	Shows or hides the *SFC* toolbar.
Project bar		*Ctrl+J*	Shows or hides the *Project* toolbar.
Network		*Ctrl+N*	Shows or hides the *Network* toolbar.
Document bar		*Ctrl+M*	Shows or hides the *Document* bar.
Tool windows			
Workspace		*Ctrl+W*	Shows or hides the *Workspace* window (also called *Project* window).
Library		*Ctrl+L*	Shows or hides the *Libraries* window.
Output		*Ctrl+R*	Shows or hides the *Output* window.
Oscilloscope		*Ctrl+K*	Shows or hides the *Oscilloscope* window.
Watch window		*Ctrl+T*	Shows or hides the *Watch* window.
Force I/O bar			Shows or hides the *Force I/O* bar.
PLC run-time status			Shows or hides the *PLC run-time* window.
Cross Reference window			Not implemented yet.
Full screen		*Ctrl+U*	Expands the currently active document window to fill entire screen. (*Esc* to exit from this mode).
Grid			Shows or hides a dotted grid in the background of graphical source code editors.
Show comments for objects			Shows or hides comments for individual objects, not only for networks. (Only for LD editor).

10.1.4 PROJECT MENU

Command	Icon	Key	Description
New object			
New program			Creates a new program. A dialog is prompted in order to specify the new program properties.
New function block			Creates a new function block. A dialog is prompted in order to specify the new function block properties.
New function			Creates a new function block. A dialog is prompted in order to specify the new function properties.
New variable			
Automatic			Creates a new automatic variable. A dialog is prompted in order to specify the new variable properties.
Mapped variable			Creates a new mapped variable. A dialog is prompted in order to specify the new variable properties.
Constant			Creates a new constant. A dialog is prompted in order to specify the new constant properties.
Retain			Creates a new retain variable. A dialog is prompted in order to specify the new variable properties.
Copy object			Copies the object currently selected in the *Workspace*.
Paste object			Pastes the previously copied object.
Duplicate object			Duplicates the object currently selected in the *Workspace*, and asks you to type the name of the copy.
Delete object			Deletes the currently selected object.
View PLC object properties		*Alt+Enter*	Shows properties and description of the currently selected object.
Object browser			Opens the *Object* browser, which lets you navigate between objects.
Compile		*F7*	Launches the Application compiler.
Recompile all		*Ctrl+Alt+F7*	Recompiles the project.
Generate redistributable source module			Generates an RSM file.
Import object from library			Lets you import a Application object from a library.
Export object to library			Lets you export a Application object to a library.
Library manager			Opens the *Library* manager.
Refresh all libraries			Reloads all libraries linked to the project.
Macros			
New macro			Creates a new macro. A dialog is prompted in order to specify the new macro properties.

Command	Icon	Key	Description
Copy macro			Copies the selected macro creating a new one.
Delete macro			Deletes the selected macro.
Properties			Shows the selected macro properties.
Select target...			Lets you to select a new target for the project.
Refresh current target			Lets you update the target file for the same version of the target.
Options...			Opens the project options dialog.

10.1.5 ONLINE MENU

Command	Icon	Key	Description
Set up communication...			Lets you set the properties of the connection to the target.
Connect			Application tries to establish a connection to the target.
Download code		*F5*	Application checks if any changes have been applied since last compilation, if so compiles the project and then downloads the source code to the target.
Download options			Lets you set the properties of the source code downloaded to the target.
Force image upload			If the target device is connected, lets you upload the img file.
Force debug symbols upload			If the target device is connected, lets you upload the debug symbols file.
Halt			Stops the PLC execution.
Cold restart			Restarts the PLC execution and both retain and non-retain variables will be reset.
Warm restart			Restarts the PLC execution and non-retain variables will be reset.
Hot restart			Restarts the PLC execution without any reset on variables.
Reboot target			Reboots the target.
Read all logs again			Reloads all remote logs from target.

10.1.6 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.
Insert 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.
Insert new item into a debug window		*Shift+F10*	Inserts a new item into a *debug* window.
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.
Remove all triggers		*Ctrl+ Shift+F9*	Removes all the active triggers.
Trigger list		*Ctrl+I*	Lists all the active triggers.
Run			Restarts program after a breakpoint is hit.
Add/Remove breakpoint		*F12*	Adds or removes a breakpoint.
Remove all breakpoints			Removes all the active breakpoints.
Breakpoint list			Lists all the active breakpoints.
Change current instance			Changes the current function block instance (live debug mode).

10.1.7 SCHEME FBD MENU

Command	Icon	Key	Description
Network			
New			
Top			Adds a blank network at the top of the active document.
Bottom			Adds a blank network at the bottom of the active document.
Before			Adds a blank network before the selected network in the active document.
After			Adds a blank network after the selected network in the active document.
Label			Assigns a label to the selected network, so that it can be indicated as the target of a jump instruction.
Object			
New			
Function			Opens the object browser in order to choose a function to be added to the current active document.
Function block		shift+B	Opens the object browser in order to choose a function block to be added to the current active document.
Variable		shift+V	Opens the object browser in order to choose a variable to be added to the current active document.
Constant		shift+K	Opens the object browser in order to choose a constant to be added to the current active document.
Return		shift+R	Adds a return statement into the selected network.
Jump to label		shift+J	Adds a jump statement into the selected network.
Operator			Opens the object browser in order to choose an operator to be added to the current active document.
Comment		shift+M	Adds a comment into the selected network.
Instance name			Opens the object browser in order to choose an operator to be added to the current active document.
Open source			Opens the editor by which the selected object was created, and displays the relevant source code: - if the object is a program, or a function, or a function block, this command opens its source code; - if the object is a variable or a parameter, this command opens the corresponding variable editor; - if the object is a standard function or an operator, this command has no functionality.
Auto connect			Toggle auto-connection mode, in order to connect automatically two blocks when they are close enough.
Delete invalid connection		Ctrl+M	Removes all invalid connections, represented by a red line in the active scheme.

Command	Icon	Key	Description
Increment pins		*Ctrl+'+'*	Adds additional pins to the selected block in order to increase standard ones.
Decrement pins		*Ctrl+'-'*	Removes pins added by the *Increment pins* command.
Enable EN/ENO pins			Adds the *enable in*/*enable out* pins to the selected block. The code implementing the selected block will be executed only when the *enable in* signal is true. The *enable out* signal simply repeats the value of *enable in*, allowing you either to enable or to disable a set of blocks in cascade.
Object properties			Shows some properties of the selected block.

10.1.8 SCHEME LD MENU

Command	Icon	Key	Description
Network			
New			
Top			Adds a blank network at the top of the active document.
Bottom			Adds a blank network at the bottom of the active document.
Before			Adds a blank network before the selected network in the active document.
After			Adds a blank network after the selected network in the active document.
Label			Assigns a label to the selected network in order to be used as target of a jump instruction.
Object			
New			
Parallel contact before		Shift+P	Adds a contact parallel before the selected one into the selected network.
Parallel contact after			Adds a contact parallel after the selected one into the selected network.
Serie contact before			Adds a contact in series before the selected one into the selected network.
Serie contact after		Shift+C	Adds a contact in series after the selected one into the selected network.
Coil		Shift+O	Adds a Coil into the selected network.
Block		Shift+B	Opens the object browser in order to choose which block should be added to the current active document.
Constant		Shift+K	Opens the object browser in order to choose a constant to be added to the current active document.
Return		Shift+R	Adds a Return statement into the selected network.
Jump		Shift+J	Adds a jump statement into the selected network.
Variable		Shift+V	Opens the object browser in order to choose a variable to be added to the current active document.
Expression		Shift+E	Adds an expression into the selected network.
New branch			Creates new branch after the current position.
Comment		Shift+M	Adds a comment into the selected network.
Instance name			Lets you assign a name to an instance of the selected function block.

Command	Icon	Key	Description
Open source			Opens the editor by which the selected object was created, and displays the relevant source code: - if the object is a program, or a function, or a function block, this command opens its source code; - if the object is a variable or a parameter, this command opens the corresponding variable editor; - if the object is a standard function or an operator, this command has no functionality.
Open object		*O*	Changes the selected object into an open contact object.
Negated object		*C*	Changes the selected object into a negated contact object.
Positive object		*P*	Changes the selected object into a positive contact object.
Negative object		*N*	Changes the selected object into a negative contact object.
Set coil		*S*	Changes the selected coil into a set coil.
Reset coil		*R*	Changes the selected coil into a reset coil.
Increment pins		*Ctrl+'+'*	Adds additional pins to the selected block in order to increase standard ones.
Decrement pins		*Ctrl+'-'*	Removes pins added by the *Increment pins* command.
Enable EN/ENO pins		*E*	Adds the *enable in*/*enable out* pins to the selected block. The code implementing the selected block will be executed only when the *enable in* signal is true. The *enable out* signal simply repeats the value of *enable in*, allowing you either to enable or to disable a set of blocks in cascade.
Set output line			Set selected pin as the output line of the block.
Object properties			Shows some properties of the selected block.

10.1.9 SCHEME SFC MENU

Command	Icon	Key	Description
Object			
New			
Step			Adds new step into the selected network.
Transition			Adds new transition into the selected network.
Jump			Adds new jump into the selected network.
Modify			
Add pin to divergent transition			Adds a divergent pin to the selected transition.
Remove pin from divergent transition			Removes a divergent pin to the selected transition.
Add pin to convergent transition			Adds a convergent pin to the selected transition.
Remove pin from convergent transition			Removes a convergent pin to the selected transition.
Add pin to simultaneous divergent transition			Adds a simultaneous divergent pin to the selected transition.
Remove pin from simultaneous divergent transition			Removes a simultaneous divergent pin to the selected transition.
Add pin to simultaneous convergent transition			Adds a simultaneous convergent pin to the selected transition.
Remove pin from simultaneous convergent transition			Removes a simultaneous convergent pin to the selected transition.
Add space before rightmost pin			Adds a space before the rightmost pin.
Remove space before rightmost pin			Removes a space before the rightmost pin.
Code Object			
New Action			Adds an action in the active document.
New Transition code			Adds a transition in the active document.
Auto connect			Toggle auto-connection mode, in order to connect automatically two blocks when they are close enough.
Delete invalid connection		*Ctrl+M*	Removes all invalid connections, represented by a red line in the active scheme.

10.1.10 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		Del	Deletes the variable in the selected row of the currently active table.
Create multiple			Lets you to create a set of multiple variables.
Group			Opens a dialog box which lets you create and delete groups of variables.

10.1.11 WINDOW MENU

Command	Icon	Key	Description
Cascade			Displaces all 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 all open documents.

10.1.12 HELP MENU

Command	Icon	Key	Description
Index			Lists all 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.

10.2 TOOLBARS REFERENCE

In the following tables you can see the list of all Application's toolbars. The buttons making up each toolbar are always the same, whatever the currently active document. However, some of them may produce no effect, if there is no logical relation to the active document.

10.2.1 MAIN TOOLBAR

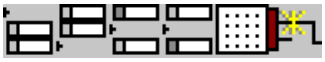
10.2.2 FBD TOOLBAR

10.2.3 LD TOOLBAR

10.2.4 SFC TOOLBAR

10.2.5 PROJECT TOOLBAR

10.2.6 NETWORK TOOLBAR



10.2.7 DEBUG TOOLBAR

11. LANGUAGE REFERENCE

All Application 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, Application implements some extensions:

- Pointers
- Macros.

11.1 COMMON ELEMENTS

By common elements textual and graphic elements are means which are common to 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 etc.) are managed by Application through specific editors, forms and tables.
Application does not allow to edit directly the source code related to the above mentioned common elements.
The following paragraphs are meant to be a language specification. To correctly manage common elements refer to the Application user guide.

11.1.1 BASIC ELEMENTS

11.1.1.1 CHARACTER SET

Textual documents and textual elements of graphic languages are written by using the standard ASCII character set.

11.1.1.2 COMMENTS

User comments are delimited at the beginning and end by the special character combinations "(*" and "*)", respectively. Comments are permitted 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, e.g., `(* (* NESTED *) *)`, is treated as an error.

11.1.2 ELEMENTARY DATA TYPES

A number of elementary (i.e. pre-defined) data types is made available by Application, all compliant with IEC 61131-3 standard.

Elementary data types, keyword for each data type, number of bits per data element, and range of values for each elementary data type are described in the following table.

Keyword	Data type	Bits	Range
BOOL	Boolean	See note	0 to 1
SINT	Short integer	8	-128 to 127
USINT	Unsigned short integer	8	0 to 255
INT	Integer	16	-32768 to 32767

Keyword	Data type	Bits	Range
UINT	Unsigned integer	16	0 to 65536
DINT	Double integer	32	-2^{31} to $2^{31}-1$
UDINT	Unsigned long integer	32	0 to 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 to +3.40E+38
STRING	String of characters	-	-

Note: the actual implementation of the BOOL data type depends on the processor of the target device, e.g. it is 1 bit long for devices that have a bit-addressable area.

11.1.3 DERIVED DATA TYPES

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.

11.1.3.1 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
```

11.1.3.2 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. Note that, 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. In order 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.

11.1.3.3 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 a concrete example consider the following declaration:

```
TYPE
int_0_to_100 : INT (0..100);
END_TYPE
```

11.1.3.4 STRUCTURES

A STRUCT declaration specifies that data elements of that type shall contain sub-elements 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
```

11.1.4 LITERALS

11.1.4.1 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. Based numbers do not contain any leading sign (+ or -).

Boolean data are represented by the keywords `FALSE` or `TRUE`.

Numerical literal features and examples are shown in the table below.

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`

11.1.4.2 CHARACTER STRING LITERALS

A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character (').

The three-character combination of the dollar sign ($) followed by two hexadecimal digits shall be interpreted as the hexadecimal representation of the eight-bit character code.

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

Example	Explanation
'"'	String of length one containing the *double quote* character
'RL'	String of length two containing CR and LF characters
'$0A'	String of length one containing the LF character

Two-character combinations beginning with the dollar sign shall be interpreted as shown in the following table when they occur in character strings.

Combination	Interpretation when printed
$$	Dollar sign
$'	Single quote
$L or $l	Line feed
$N or $n	Newline
$P or $p	Form feed (page)
$R or $r	Carriage return
$T or $t	Tab

11.1.5 VARIABLES

11.1.5.1 FOREWORD

Variables provide a means of identifying data objects whose contents may change, e.g., 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) (i.e., each program, function, or function block) contains at its beginning at least one declaration part, consisting 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.

11.1.5.2 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
```

11.1.5.3 KEYWORDS AND SCOPE

Keyword	Variable usage
VAR	Internal to organization unit.
VAR_INPUT	Externally supplied.
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.

Such 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 this variables to all type of POU, without using any keyword, you must enable this option in the code generation tab of the project options (see Paragraph 4.6.2).

There is an error if:

- any program organization unit 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 program organization unit (the "containing element") is used in a VAR_EXTERNAL declaration (without the CONSTANT qualifier) of any element contained within the containing element.

11.1.5.4 QUALIFIERS

Qualifier	Description
CONST	The attribute CONST indicates that the variables within the structuring elements are constants, i.e. 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, i.e. they keep their value even after the target device is reset or switched off.

11.1.5.5 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 easily 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.

11.1.5.6 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	See the next paragraph.
N	Index of the last element, the array having length N + 1.

11.1.5.7 LOCATION

Variables can be represented symbolically, i.e. 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 programmable controller's input, output, or memory structure.

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

Size prefix	Description
D	Double word (32 bits) size

3) index.index

This sequence of unsigned integers, separated by dots, specifies the actual 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 7th element of memory datablock 4.
%IX0.4	First bit of the first byte of the 5th element of input set 0.

Note that the absolute position depends on the size of the datablock elements, not on the size prefix. As a matter of fact, %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 7th 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 Application 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 datablock 4.
- Variable Counters is an array of 10 independent variables of type unsigned integer.

11.1.5.8 DECLARING VARIABLES IN APPLICATION

Whatever the PLC language you are using, Application allows you to disregard the syntax above, as it supplies the Local variables editor, the Global variables editor, and the Parameters editor, which provide a friendly interface to declare all kinds of variables.

11.1.6 PROGRAM ORGANIZATION UNITS

Program organization units are functions, function blocks, and programs. Program Organization Units can be delivered by the manufacturer, or programmed by the user 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 another program organization unit of the same type.

11.1.6.1 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, i.e., invocation of a function with the same arguments (input variables `VAR_INPUT` and in-out variables `VAR_IN_OUT`) always yields the same values (output variables `VAR_OUTPUT`, in-out variables `VAR_IN_OUT` and function result).

Declaration syntax

The declaration of a function must be performed as follows:

```
FUNCTION FunctionName : RetDataType
VAR_INPUT
    declaration of input variables (see the relevant section)
END_VAR
VAR
    declaration of local variables (see the relevant section)
END_VAR
    Function body
END_FUNCTION
```

Keyword	Description
`FunctionName`	Name of the function being declared.
`RetDataType`	Data type of the value to be returned by the function.
`Function body`	Specifies the operations to be performed upon the input variables in order to assign values dependent on the function's semantics to a variable with the same name as the function, which represents the function result. It can be written in any of the languages supported by Application.

Declaring functions in Application

Whatever the PLC language you are using, Application allows you to disregard the syntax above, as it supplies a friendly interface for using functions.

11.1.6.2 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; therefore, 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, i.e., 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 invoked 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 Application.

Declaring functions in Application

Whatever the PLC language you are using, Application allows you to disregard the syntax above, as it supplies a friendly interface for using function blocks.

11.1.6.3 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 Application.

Writing programs in Application

Whatever the PLC language you are using, Application allows you to disregard the syntax above, as it supplies a friendly interface for writing programs.

11.1.7 IEC 61131-3 STANDARD FUNCTIONS

This paragraph is a reference of all IEC 61131-3 standard functions available in Application, along with a few others, which may be considered as Application's extensions to the standard.

These functions are common to the whole set of programming languages and can therefore be used in any Programmable Organization Unit (POU).

A function specified in this paragraph to be extensible (Ext.) is allowed to have a variable number of inputs.

Type 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`). Application provides a more convenient set of overloaded type conversion functions, relieving the developer 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_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_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_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_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_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_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_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_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 *)`

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_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_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 *)`

Numerical functions

The availability of the following functions depends on the target device. Please refer to your hardware supplier for details.

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 *)`

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 *)`

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 *)`

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 *)`

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 *)`

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 *)`

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
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 *)`

ASIN	
Description	Arc sine. Computes the principal arc sine of input #0; result is expressed in radians
Number of operands	1
Input data type	LREAL where available, REAL otherwise
Output data type	LREAL where available, REAL otherwise
Examples	`OUT := ASIN( 0.0 ); (* OUT = 0.0 *)` `OUT := ASIN( 1.0 ); (* OUT = PI / 2 *)`

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 *)`

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 *)`

ADD	
Description	Arithmetic addition. Computes the sum of the two inputs.
Number of operands	2
Input data type	Any numerical type, Any numerical type
Output data type	Same as Inputs
Examples	`OUT := ADD( 20, 40 ); (* OUT = 60 *)`

MUL	
Description	Arithmetic multiplication. Multiplies the two inputs.
Number of operands	2
Input data type	Any numerical type, Any numerical type
Output data type	Same as Inputs
Examples	`OUT := MUL( 10, 10 ); (* OUT = 100 *)`

SUB	
Description	Arithmetic subtraction. Subtracts input #1 from input #0
Number of operands	2
Input data type	Any numerical type, Any numerical type
Output data type	Same as Inputs
Examples	`OUT := SUB( 10, 3 ); (* OUT = 7 *)`

DIV	
Description	Arithmetic division. Divides input #0 by input #1
Number of operands	2
Input data type	Any numerical type, Any numerical type
Output data type	Same as Inputs
Examples	`OUT := DIV( 20, 2 ); (* OUT = 10 *)`

MOD	
Description	Module. Computes input #0 module input #1
Number of operands	2
Input data type	Any numerical type, Any numerical type
Output data type	Same as Inputs
Examples	`OUT := MOD( 10, 3 ); (* OUT = 1 *)`

POW	
Description	Exponentiation. Raises Base to the power Expo
Number of operands	2
Input data type	LREAL where available, REAL otherwise; 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 *)`

ATAN2*	
Description	Arc tangent (with 2 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; 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 *)`

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 *)`

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 *)`

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 *)`

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 *)`

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 *)`

*: function provided as extension to the IEC 61131-3 standard.

Bit string functions

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, 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, Any numerical type
Output data type	Same as Input #0
Examples	`OUT := SHR( IN := 16#1000CAFE, 24 );` `(* OUT = 16#00000010 *)`

ROL	
Description	Input #0 left-shifted of Input #1 bits, circular.
Number of operands	2
Input data type	Any numerical 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, Any numerical type
Output data type	Same as Input #0
Examples	`OUT := ROR( IN := 16#1000CAFE, 16 );` `(* OUT = 16#CAFE1000 *)`

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, 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 *)`

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, 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, 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 *)`

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 *)`

Selection functions

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 *)`

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

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, Any numerical 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 *)`

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

Comparison functions

Comparison functions can be also used to compare strings if this feature is supported by target device.

GT	
Description	Greater than. Returns TRUE if Input #0 > Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any but BOOL, Any but BOOL
Output data type	BOOL
Examples	`OUT := GT( 0, 20 ); (* OUT = FALSE *)` `OUT := GT( 'AZ', 'ABC' ); (* OUT = TRUE *)`

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, Any but BOOL
Output data type	BOOL
Examples	`OUT := GE( 20, 20 ); (* OUT = TRUE *)` `OUT := GE( 'AZ', 'ABC' ); (* OUT = FALSE *)`

EQ	
Description	Equal to. Returns TRUE if Input #0 = Input #1, otherwise FALSE.
Number of operands	2
Input data type	Any, Any
Output data type	BOOL
Examples	`OUT := EQ( TRUE, FALSE );  (* OUT = FALSE *)` `OUT := EQ( '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, Any but BOOL
Output data type	BOOL
Examples	`OUT := LT( 0, 20 ); (* OUT = TRUE *)` `OUT := LT( 'AZ', 'ABC' ); (* OUT = FALSE *)`

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, Any but BOOL
Output data type	BOOL
Examples	`OUT := LE( 20, 20 ); (* OUT = TRUE *)` `OUT := LE( '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, Any
Output data type	BOOL
Examples	`OUT := NE( TRUE, FALSE ); (* OUT = TRUE *)` `OUT := NE( 'AZ', 'ABC' ); (* OUT = TRUE *)`

String functions

The availability of the following functions depends on the target device. Please refer to your hardware supplier for details.

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

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' *)`

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' *)`

11.2 INSTRUCTION LIST (IL)

This section defines the semantics of the IL (Instruction List) language.

11.2.1 SYNTAX AND SEMANTICS

11.2.1.1 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

Let us parse a small piece of code:

```
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

  By accumulator a register is meant containing the value of the currently evaluated 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
  ```

  and 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 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 shown below, referred to the instruction

  ```
  accumulator := accumulator AND (%MX1.3 OR %MX1.4)
  ```

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).

11.2.2 STANDARD OPERATORS

Standard operators with their allowed modifiers and operands are as listed below.

Operator	Modifiers	Supported operand types: Acc_type, Op_type	Semantics
LD	N	Any, Any	Sets the accumulator equal to operand.
ST	N	Any, Any	Stores the accumulator into operand location.
S		BOOL, BOOL	Sets operand to TRUE if accumulator is TRUE.
R		BOOL, BOOL	Sets operand to FALSE if accumulator is TRUE.
AND	N, (	Any but REAL, Any but REAL	Logical or bitwise AND
OR	N, (	Any but REAL, Any but REAL	Logical or bitwise OR
XOR	N, (	Any but REAL, Any but REAL	Logical or bitwise XOR
NOT		Any but REAL	Logical or bitwise NOT
ADD	(	Any but BOOL	Addition
SUB	(	Any but BOOL	Subtraction
MUL	(	Any but BOOL	Multiplication
DIV	(	Any but BOOL	Division
MOD	(	Any but BOOL	Modulo-division
GT	(	Any but BOOL	Comparison:
GE	(	Any but BOOL	Comparison: =
EQ	(	Any but BOOL	Comparison: =
NE	(	Any but BOOL	Comparison:
LE	(	Any but BOOL	Comparison:
LT	(	Any but BOOL	Comparison:
JMP	C, N	Label	Jumps to label
CAL	C, N	FB instance name	Calls function block
RET	C, N		Returns from called program, function, or function block.
)			Evaluates deferred operation.

11.2.3 CALLING FUNCTIONS AND FUNCTION BLOCKS

11.2.3.1 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
```

Note that 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 2nd), if required, are given in the operand field, separated by commas, in the order of their declaration. For example, operator `MUX` in the table above 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.

1) Assignments to `VAR_INPUT` arguments may be empty, constants, or variables.
2) 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.

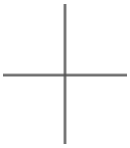
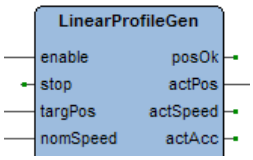
11.3 FUNCTION BLOCK DIAGRAM (FBD)

This section defines the semantics of the FBD (Function Block Diagram) language.

11.3.1 REPRESENTATION OF LINES AND BLOCKS

The graphic language elements are drawn using graphic or semi graphic elements, as shown in the table below.

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.

Feature	Example
Lines	
Line crossing with connection	
Blocks with connecting lines and unconnected pins	

11.3.2 DIRECTION OF FLOW IN NETWORKS

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 block(s) so connected.

11.3.3 EVALUATION OF NETWORKS

11.3.3.1 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:

1) No element of a network is evaluated until the states of all of its inputs have been evaluated.
2) The evaluation of a network element is not complete until the states of all of its outputs have been evaluated.
3) As stated when describing the FBD editor, a network number is automatically as-

signed 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 means of the execution control elements.

11.3.3.2 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

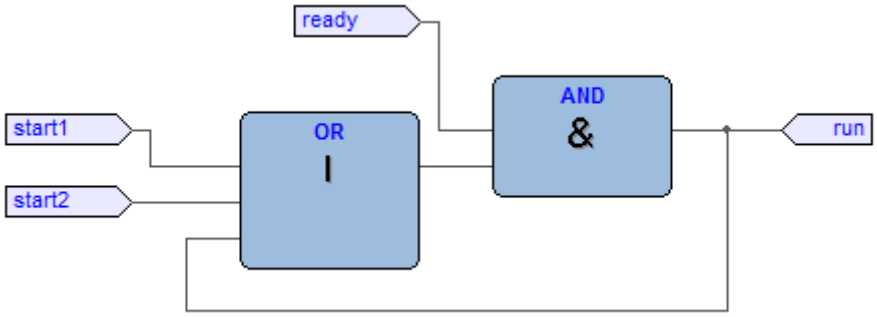
A feedback path is said to exist in a network when the output of a function or function block is used as the input to a function or function block which precedes it in the network; the associated variable is called a feedback variable.

Feedback paths can be utilized subject to the following rules:

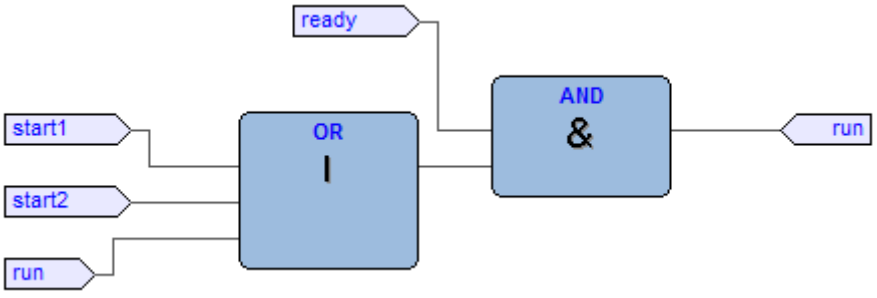
1) Feedback variables must be initialized, and the initial value is used during the first evaluation of the network. Look at the *Global variables* editor, the Local variables editor, or the Parameters editor to know how to initialize the respective item.
2) Once the element with a feedback variable 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 example shown below.

Explicit loop



Implicit loop



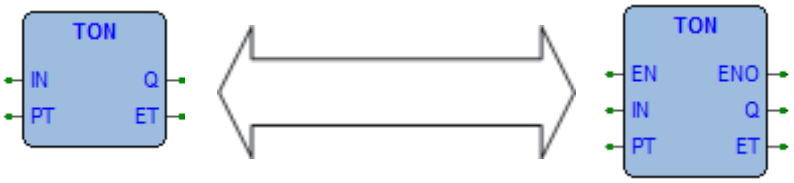
11.3.4 EXECUTION CONTROL ELEMENTS

11.3.4.1 EN/ENO SIGNALS

Additional Boolean EN (Enable) input and ENO (Enable Out) characterize Application blocks, according to the declarations

EN	ENO
VAR_INPUT EN: BOOL := 1; END_VAR	VAR_OUTPUT ENO: BOOL; END_VAR

See the *Modifying* properties of blocks section 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:

1) 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.
2) Otherwise, the value of ENO is set to TRUE by the programmable controller system, and the operations defined by the block body are executed.

11.3.4.2 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.

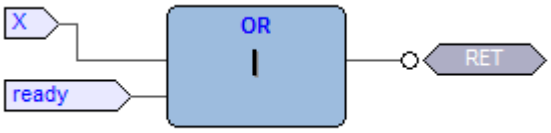
The target of a jump is a network label within the program organization unit within which the jump occurs.

Symbol / Example	Explanation
	Unconditional Jump
	Conditional Jump

Symbol / Example	Explanation
	Example: Jump Condition Network

11.3.4.3 CONDITIONAL RETURNS

- Conditional returns from functions and function blocks are implemented using a `RETURN` construction as shown in the table below. 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

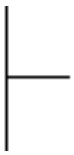
11.4 LADDER DIAGRAM (LD)

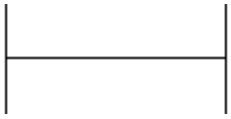
This section defines the semantics of the LD (Ladder Diagram) language.

11.4.1 POWER RAILS

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 Application implementation and it is always shown.

The two power rails are always connected with an horizontal line named signal link. All LD elements should be placed and connected to the signal link.

Description	Symbol
Left power rail (with attached horizontal link)	
Right power rail (with attached horizontal link)	

Description	Symbol
Power rails connected by the signal link	

11.4.2 LINK ELEMENTS AND STATES

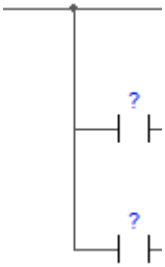
Link elements may be horizontal or vertical. The state of the link elements shall be denoted "ON" or "OFF", corresponding to the literal Boolean values 1 or 0, respectively. The term link state shall be synonymous with the term power flow.

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	

11.4.3 CONTACTS

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.	

11.4.4 COILS

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 shown 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.	

Name	Description	Symbol
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.	-(R)-
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.	-(P)-
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.	-(N)-

11.4.5 OPERATORS, FUNCTIONS AND FUNCTION BLOCKS

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 shown on each block to allow for power flow through the block as shown in the following figure.

11.5 STRUCTURED TEXT (ST)

This section defines the semantics of the ST (Structured Text) language.

11.5.1 EXPRESSIONS

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. Application does not set any constraint on the maximum length of expressions.

Expressions are composed of operators and operands.

11.5.1.1 OPERANDS

An operand can be a literal, a variable, a function invocation, or another expression.

11.5.1.2 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.

11.5.1.3 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, etc., 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.

11.5.1.4 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

11.5.2 STATEMENTS IN ST

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;
- Application does not set any constraint on the maximum length of statements.

ST statements can be divided into classes, according to their semantics.

11.5.2.1 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

```
a := b ;
```

assignment

```
pCV := pCV + 1 ;
```

assignment

```
c := SIN( x );
```

assignment with function invocation

```
FUNCTION SIMPLE_FUN : REAL
variables declaration
...
function body
...
SIMPLE_FUN := a * b - c ;
END_FUNCTION
```

assigning the output value to a function

11.5.2.2 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 (e.g., as the result of the evaluation of an IF statement).

Syntax

1) Function:

```
dst_var := function_name( arg1, arg2 , ... , argN );
```

2) Function block with formal argument list:

```
instance_name(    var_in1 := arg1 ,
                  var_in2 := arg2 ,
                  ... ,
                  var_inN := argN );
```

3) Function block with assignment of arguments:

```
instance_name.var_in1 := arg1;
...
instance_name.var_inN := argN;
instance_name();
```

4) Function and function block control statement:

```
RETURN;
```

Examples

```
CMD_TMR( IN := %IX5,
 PT:= 300 ) ;
```

FB invocation with formal argument list:

```
IN := %IX5 ;
PT:= 300 ;
CMD_TMR() ;
```

FB invocation with assignment of arguments:

```
a := CMD_TMR.Q;
```

FB output usage:

```
RETURN ;
```

early exit from function or function block.

11.5.2.3 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.

Application does not set any constraint on the maximum allowed number of selections in CASE statements.

Syntax

Note that square brackets include optional code, while braces include repeatable portions of code.

1) IF:

```
IF expression1 THEN
stat_list
[ { ELSIF expression2 THEN
stat_list } ]
ELSE
stat_list
END_IF ;
```

2) 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 ;
```

11.5.2.4 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 (e.g., 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; this value defaults to 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

Note that square brackets include optional code, while braces include repeatable portions of code.

1) FOR:

```
FOR control_var := init_val TO end_val [ BY increm_val ] DO
stat_list
END_FOR ;
```

2) WHILE:

```
WHILE expression DO
stat_list
END_WHILE ;
```

3) 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 ;
```

11.6 SEQUENTIAL FUNCTION CHART (SFC)

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 the purpose of 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

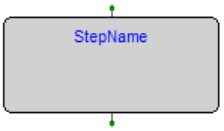
The SFC elements provide a means of partitioning a PLC program organization unit into a set of steps and transitions interconnected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition.

11.6.1 STEPS

11.6.1.1 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 link(s) into the step can be represented graphically by a vertical line attached to the top of the step. The directed link(s) 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)

Application does not set any constraint on the maximum number of steps per SFC.

Step flag

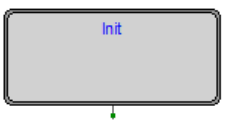
The step flag (active or inactive state of a step) can be represented by the logic value of a Boolean variable `***_x`, where `***` is the step name. This Boolean variable has the value `TRUE` when the corresponding step is active, and `FALSE` when it is inactive. The scope of step names and step flags is local to the program organization unit where the steps appear.

Representation	Description
`Step Name_x`	Step flag = `TRUE` when `Step Name_x` is active = `FALSE` otherwise

11.6.1.2 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, i.e., 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 shown below. For system initialization, the default initial state is `FALSE` for ordinary steps and `TRUE` for initial steps.

Application cannot compile an SFC network not containing exactly one initial step.

Representation	Description
	Initial step (graphical representation with direct links)

11.6.1.3 ACTIONS

An action can be:

- a collection of instructions in the IL language;
- a collection of networks in the FBD language;
- a collection of rungs in the LD language;
- a collection of statements in the ST language;
- a sequential function chart (SFC) organized as defined in this section.

Zero or more actions can be associated with each step. Actions are declared via one of the textual structuring elements listed in the following table.

Structuring element	Description

`STEP StepName :`
`(* Step body *)`
`END_STEP`
	Step (textual form)

`INITIAL_STEP StepName :`
`(* Step body *)`
`END_STEP`
	Initial step (textual form)

Such a structuring element exists in the *lsc* file for every step having at least one associated action.

11.6.1.4 ACTION QUALIFIERS

The time when an action associated to a step is executed depends on its action qualifier.

Application implements the following action qualifiers.

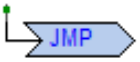
Qualifier	Description	Meaning
N	Non-stored (null qualifier).	The action is executed as long as the step remains active.
P	Pulse.	The action is executed only once per step activation, regardless of the number of cycles the step remains active.

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.

11.6.1.5 JUMPS

Direct links flow only downwards. Therefore, if you want to return to a upper step from a lower one, you cannot draw a logical wire from the latter to the former. A special type of block exists, called Jump, which lets you implement such a transition.

A Jump block is logically equivalent to a step, as they have to always be separated by a transition. The only effect of a Jump is to activate the step flag of the preceding step and to activate the flag of the step it points to.

Representation	Description
	Jump (logical link to the destination step)

11.6.2 TRANSITIONS

11.6.2.1 DEFINITION

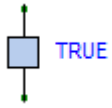
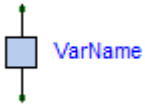
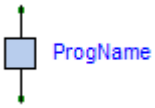
A transition represents the condition whereby control passes from one or more steps preceding the transition to one or more successor steps along the corresponding directed link. The transition is represented by a small grey square across the vertical directed link.

The direction of evolution following the directed links is from the bottom of the predecessor step(s) to the top of the successor step(s).

11.6.2.2 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 or not the transition is cleared.
	By writing a piece of code, in any of the languages supported by Application, 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.

11.6.3 RULES OF EVOLUTION

Introduction

The initial situation of a 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 is 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.

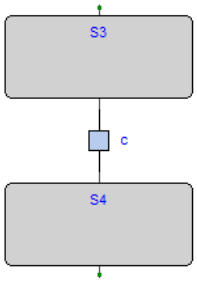
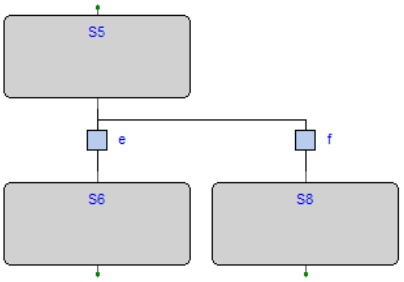
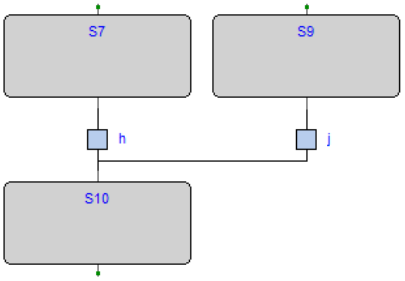
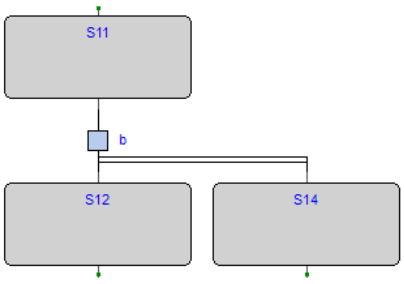
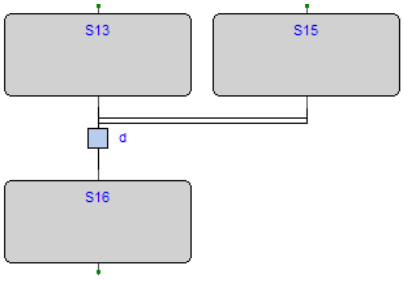
When the clearing of a transition leads to the activation of several steps at the same time, the sequences which these steps belong to are called simultaneous sequences. After their

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.

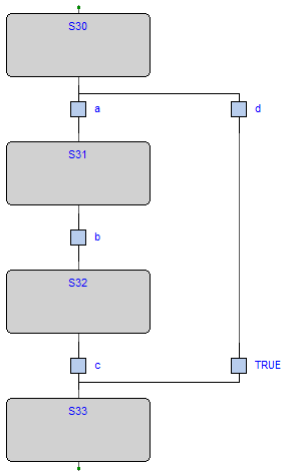
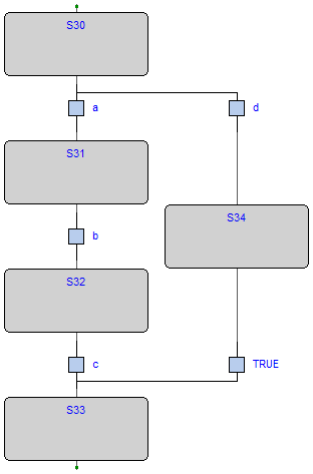
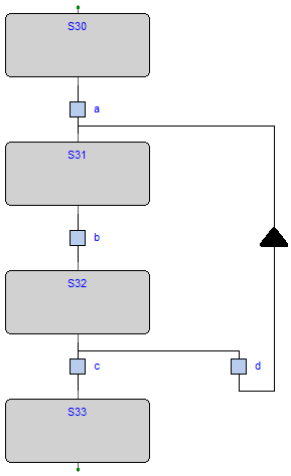
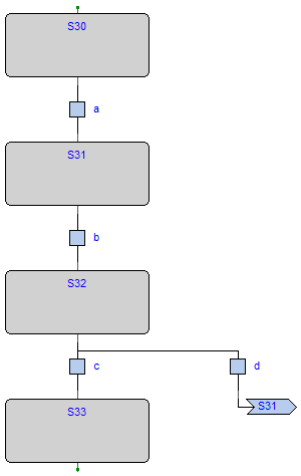
The clearing time of a transition may theoretically be considered as short as one may wish, but it can never be zero. In practice, the clearing time will be imposed by the PLC implementation: several transitions which can be cleared simultaneously will be cleared simultaneously, within the timing constraints 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 condition(s) 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`.
	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`, etc., 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.

11.6.4 SFC CONTROL FLAGS

Application provides some control flags for SFC program or function blocks.

To enable this feature, please refer to paragraph 4.6.2.

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 will be named `Main_HOLD_SFC` and `Main_RESET_SFC`.

Another couple of actions is available for every SFC action, which also are contained in a SFC POU.

For example, if the above program *Main* contains a SFC action named *Execute*, the control flags of this action will be `Main_Execute_HOLD_SFC` and `Main_Execute_RESET_SFC`.

These flags functionalities are described in details on next paragraphs.

11.6.4.1 HOLD FLAG

Following the main characteristics of the <POU name>_HOLD_SFC flag:

- 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.

11.6.4.2 RESET FLAG

Following the main characteristics of the <POU name>_RESET_SFC flag:

- 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 his own state becomes FALSE after his reset action has been executed. It is therefore not necessary to bring the <POU name>_RESET_SFC value back to FALSE.

11.6.4.3 FLAGS VISIBILITY

The <POU name>_HOLD_SFC and <POU name>_RESET_SFC flags are automatically generated from the code compiler and they belongs to the local variables of the POU which are referred to.

Application 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.

11.6.5 CHECK A SFC POU FROM OTHER PROGRAMS

To allow the managing of a SFC POU from other programs Application 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 cases above, Application compiler will use the variables declared among the VAR_INPUT or VAR_GLOBAL ones and not those automatically generated (therefore they will be 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
```

11.6.5.1 SFC MACRO LIBRARY

Application makes available to user a library, called *SFCControl.pll*, to allow the manage of the SFC states trough commands instead of variable settings.

This library is composed by macros usable only in ST language.

11.6.5.2 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 a 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 *).
```

11.7 APPLICATION LANGUAGE EXTENSIONS

Application features a few extensions to the IEC 61131-3 standard, in order to further enrich the language and to adapt to different coding styles.

11.7.1 MACROS

Application 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
```

Note that the parameter list may eventually be empty, thus distinguishing between object-like macros, which do not take parameters, and function-like macros, which do 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 actual 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 );
```

11.7.2 POINTERS

Pointers are a special kind of 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 *)
```

Beware that careless use of pointers is potentially dangerous: indeed, pointers can point to any arbitrary location, which can cause undesirable effects.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Ensure that all variables are initialized to an appropriate value before their first use as pointers.
- Write programming instructions to test the validity of operands intended to be used as memory pointers.
- Do not attempt to access memory element outside the defined bounds of the allocated memory.

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

11.7.3 WAITING STATEMENT

Application 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 will be 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 (see paragraph 4.6.2 for more details).

12. ERRORS REFERENCE

12.1 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 isn't supported by the target system.
A4099	Auto vars space exhausted	The total allocation space requested by all local variables exceeds the space available on the target system.
A4100	Retentive vars space exhausted	The total allocation space requested by all local retentive variables exceeds the space available on the target system.
A4101	Bit vars space exhausted	The total allocation space requested by all local bit (boolean) variables exceeds the space available on the target system.
A4102	Invalid ++ in data block	The variable indicated is associated with an index that is not available in the relative data block.
A4103	Data block not found	The variable indicated is associated with a data block that doesn't exist (isn't 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 isn't 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 isn't defined.
A4110	Invalid data type	The data type used in the variable definition doesn't 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 isn't 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 then 10% of free data	
A4118	Less then 10% of free retain data	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
A4119	Less then 10% of free bit data	
A4120	Variable exceeds data block space	
A4121	Element not found	
A4123	Invalid access to private member	
A4129	Not a structured type	
A4130	Not a function block instance	
A4131	Incompatible external declaration	
A4133	Not a variable	
A4134	Index exceeds array size	
A4135	Invalid index data type	
A4136	Missing index(es)	
A4137	Function block instance required	
A4138	Simple variable required	
A4139	Too many indexes	
A4140	Not a structure instance	
A4141	Not an array	
A4143	Not a pointer	
A4144	Double pointer indirection not allowed	
A4145	To be implemented	
A4146	Bit datatype not allowed	
A4147	Unable to calculate variable offset	
A4148	Complex variables cannot have process image	
A4149	Cannot use directly represented variables with process image in function blocks (not implemented)	
A4150	Function block instance not allowed	
A4151	Structure not allowed	
A4152	16-bit variables must be aligned to a 16-bit boundary	
A4153	32-bit variables must be aligned to a 32-bit boundary	
A4154	Temporary string variable allocation error. Instruction shall be split.	
A4155	Ext/aux auto vars space exhausted	
A4156	Ambiguous enum value, <enum># prefix required	
B0001	Data block not found	The variable indicated is associated with a data block that doesn't exist (isn't defined) in the target system.
B0002	Error on create file	The indicated file can't be created due to a file system error or to a missing source file.
C0001	Parser not initialized	Internal compiler error.
C0002	Invalid token	Invalid word for the current language syntax

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
C0003	Invalid file specification	Internal compiler error.
C0004	Can't open file	The indicated file can't 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 can't be used for declaration purposes because is a keyword of the language.
C0010	Invalid element	The indicated word isn't a valid one for the language syntax.
C0011	Aborted by user	
C0032	Too many parameters in macro call	
C0033	Invalid number of parameters in macro call	
C0034	Too many macro calls nested	
C4097	Invalid variable type	The data type indicated isn't allowed.
C4098	Invalid location prefix	The address string of the indicated variable isn't correct, '%' missing.
C4099	Invalid location specification	The address string of the indicated variable isn't correct, the data access type indication isn't 'I', 'Q' or 'M'.
C4100	Invalid location type	The address string of the indicated variable isn't correct, the data type indication isn't 'X', 'B', 'W', 'D', 'R' or 'L'.
C4101	Invalid location index specification	The address string of the indicated variable isn't correct, the index isn't 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 isn't indicated in the correct way (e.g.: contains invalid characters, negative numbers etc.).
C4105	Constant not initialized	Every constant need to have an initial value.
C4106	Invalid string size	
C4107	Initialization exceeding string size	
C4108	Invalid repetition in initialization	
C4109	Invalid data type for initialization	
C4353	Duplicate label	The indicated label has already been defined in the current POU (program, function or function block).
C4354	Constant not admitted	The operation indicated doesn't 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).

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
C4610	Invalid octal constant	A constant value with 8# prefix must contain only octal digits (between 0 and 7).
C4611	Invalid hexadecimal constant	A constant value with 16# prefix must contain only hexadecimal digits (between 0 and 9 and between A and F).
C4612	Invalid decimal constant	A decimal constant must contain only digits between 0 and 9, a leading sign + or -, a decimal separator '.' Or a exponent indicator 'e' or 'E'.
C4613	Invalid time constant	A constant value with t# prefix must contain a time indication in decimal notation and a time unit between 'ms, 's' or 'm'.
C4614	Invalid constant string	
C4864	Duplicate function name	The indicated function name has already been used for another application object.
C4865	Invalid function type	The data type returned by the indicated function is not correct.
C5120	Duplicate program name	The indicated program name has already been used for another application object.
C5376	Duplicate function block name	The indicated function block name has already been used for another application object.
C5632	Invalid pragma	
C5633	Invalid pragma value	
C5889	Duplicate macro name	
C5890	Duplicate macro parameter name	
C6144	Invalid resource definition: two or more tasks have the same ID	
C16385	Invalid init value	
C16386	Invalid initialization definition	
C16387	Invalid array delimiters (brackets)	
C16388	Empty init value	
C16389	Empty array init value	
C16390	Invalid repeated init value	
C16391	Not implemented	
C16392	Missing array delimiters (brackets)	
C16393	Missing comma	
C16394	Not implemented	
C16395	Invalid (incomplete) string	
D12289	Can't allocate database	The memory space needed for parameter's database exceeds the space available on the target system. If possible, remove unused parameter's records, menus etc.
D12290	Can't allocate database record	The memory space needed for parameter's database exceeds the space available on the target system. If possible, remove unused parameter's records, menus etc.
D12291	Database variable not found	Internal compiler error.
D12292	Invalid expression or expression syntax error	The database expression that has the result indicated isn't correct, contains syntax errors or invalid operators.

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
D12293	Invalid parameter reference in expression	The database expression that has the result indicated contains a parameter (as operand) that isn't the same to which the expression refers to. The expression can use only PLC variables (including the variables associated with parameters) and the value of the parameter that is exchanged at the moment. For example: pDELTA = DELTA / pRATIO + pOFFSET is correct because the parameter exchanged is DELTA and it's the only parameter value used in the expression. The expression: pDELTA = DELTA / pRATIO + OFFSET isn't correct because the parameter OFFSET used in the expression isn't currently exchanged
D12294	Recursive expression	The database expression that has the result indicated calls itself by means of some operand used that contains the current expression result.
D12295	Unresolved variable in expression	The database expression that has the result indicated uses an operand that isn't defined in the whole PLC project.
D12296	Unresolved expression result	Internal compiler error.
D12297	Invalid result type for expression	The parameter that is the result of the expression has a data type invalid (such as enumerative) or not defined.
D12298	Invalid operand in expression	The database expression that has the result indicated uses an invalid operand.
D12299	Invalid variable type for expression	The variable that is the result of the expression has a data type invalid (such as enumerative) or not defined.
D12300	Assembler error	Internal compiler error.
D12301	Can't allocate database code	The code space needed for the expression is exhausted. Is necessary to remove some expressions from the parameter's database.
D12302	Invalid operation in expression	The database expression that has the result indicated uses an invalid operand.
F1025	Invalid network	The indicated FBD or LD network contains a connection error (the errors are normally indicated by red connections).
F1026	Unconnected pin	The indicated block (operator, function, contact or coil) has an unconnected pin.
F1027	Invalid connection (incomplete, more than a source etc.)	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 doesn't 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	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
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 isn't 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 doesn't 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 doesn't 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	Can't allocate code buffer	There isn't enough memory on the PC to create the image of the code of the target system.
G0012	Function not defined	The indicated function doesn't exist in the current project.
G0013	Cyclic declaration of function blocks	The indicated function block call itself directly or by means of other functions.
G0014	Incompatible external declaration	The external variable declaration of the current function block doesn't 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 doesn't refer to any of the global variables of the project (e.g.: there isn't a global variable with the same name).
G0017	Program is not assigned to a task	The indicated program hasn't been assigned to a task in the target system.
G0018	Task not found in resources	The indicated task isn't 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 isn't 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) isn't correct or isn't supported by the compiler.
G0022	Function block with process image variables can't be used in event tasks	
G0023	Process image variables can't be used in event tasks	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G0024	Accumulator undefined	
G0025	Invalid index	
G0026	Only constant index allowed	
G0027	Illegal reference to the address of a register	
G0028	Less then 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	
G0059	Operand extension	
G0060	Division by zero	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G0061	Illegal comparison	
G0062	Function block must be instanciated	
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 boolean	
G0076	Implicit type conversion of real to boolean	
G0077	Implicit type conversion of real to integer	
G0078	Arithmetic operations require numerical operands	
G0079	Bitwise logical operations require bitstring/integer operands	
G0080	Comparison operations require elementary (i.e., 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	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
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	
G0095	Improper init value	
G0096	Init value object not found	
G0097	Invalid assignment to pointer	
G0513	Invalid operator	The operator indicated is not allowed for the indicated operation.
G0514	Operation not implemented	The operator indicated isn't supported by the current target system.
G0515	Real data not supported	The target system in use doesn't support floating point operations.
G0516	Destination may be too small to store current result	The variable destination of the store/assignment operation has a data type smaller than the one of the accumulator. Data may be lost in the operation. For example, if the accumulator contains 340 and the destination operand is of SINT type, the assignment operation will loose data. If the operation is under the programmer's control an appropriate type conversion function (TO_SINT, TO_INT, TO_DINT etc.) can be used to eliminate the warning message.
G0517	Long data not supported	The target system in use doesn't support long data operations.
G0518	Accumulator extension	The variable destination of the store/assignment operation has a data type bigger than the one of the accumulator. An extension operation has been performed automatically by the compiler. To eliminate this warning message use the appropriate type conversion function (TO_SINT, TO_INT, TO_DINT etc.).
G0519	Assembler error	Internal compiler error.
G0520	Negation allowed only on boolean	The 'N' modifier used for some IL operators (LDN, STN, ANDN etc.) can't be used with operators having type other than boolean.
G0521	Operation allowed with boolean types	The IL operator indicated (typically 'S' or 'R') can't be used when the accumulator has a type other than BOOL.
G0522	Instruction has constant result	The indicated operation has a result that is constant (ex. multiply by 0, AND with FALSE).
G0523	Instruction is a NOP	The operation indicated has no influence on the value of the accumulator (ex. multiply by 1, AND with TRUE).

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G0524	Unbalanced parentheses	The number of opened parentheses doesn't match with the number of the closed parentheses in the indicated code block.
G0525	Operation not allowed on boolean	The indicated operation can't be performed on boolean operands (ex. the arithmetic operations).
G0526	Can't perform modulo with long values	The current target system doesn't allow the modulo operation with long data types.
G0527	Division by 0	The indicated division operation has the constant value 0 as denominator.
G0528	Negation without condition	The indicated operation (JMP or RET) has the negation modifier 'N' without the conditional evaluation modifier 'C'. Use JMPCN instead of JMPN or RETCN instead of RETN.
G0529	Initial value not defined	Internal compiler error.
G0530	Invalid initial value	The initial value of the variable isn't indicated correctly.
G0531	Invalid accumulator type	The accumulator has a data type not allowed for the indicated operation (ex. MUX operator with REAL accumulator).
G0532	Code generator internal error	Internal compiler error.
G0533	Invalid operator value	The operator has a value not acceptable for the indicated operation (ex. SHL with constant value bigger than 32).
G0534	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G0535	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G0536	Only constant index allowed	The use of variable as index for the indicated array is not supported by the compiler. This error is typically issued with boolean (bit) arrays.
G0537	Indexing of boolean constants not allowed	The use of variable as index for the indicated array is not supported by the compiler. This error is typically issued with boolean (bit) arrays.
G0538	Return not allowed from programs	The RET operator isn't allowed in PROGRAM blocks.
G0539	Function block must be instantiated	A function block can't be invoked directly with a CAL instruction. It must be instantiated before its use eg. must be a variable with data type corresponding to the function block instead.
G0540	Operation not allowed with real types	The indicated operation can't be executed on REAL data types. Instructions of this kind are logical and bitwise operations.
G0541	Accumulator conversion	This warning informs that the data type of the accumulator has been automatically converted by the compiler. This operation is typically executed when the accumulator and the operand used in a arithmetic operation have different data types.
G0542	Real accumulator must be reloaded	Some target-specific implementations with software floating point emulation require that each store operation shall be preceded by a new load operation or a arithmetic sequence.
G0543	Real accumulator not stored	Some target-specific implementations with software floating point emulation require that when the floating point stack has been loaded, the same shall be unloaded at the end of arithmetic sequence.

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G0544	Long real data not supported	The long real data type LREAL isn't supported by the compiler.
G0769	Invalid operator	The operator indicated is not allowed for the indicated operation.
G0770	Operation not implemented	The operator indicated isn't supported by the current target system.
G0771	Assembler error	Internal compiler error.
G0772	Long real data not supported	The long real data type LREAL isn't supported by the compiler.
G0773	Long data not supported	The long data type LINT isn't supported by the compiler.
G0774	Negation of a non-boolean parameter	The negation modifier 'N' can't be used in operations with data types different than boolean.
G0775	Operation not allowed on boolean	The indicated operation can't be performed on boolean operands (ex. the arithmetic operations).
G0776	Accumulator extension	The variable destination of the store/assignment operation has a data type bigger than the one of the accumulator. An extension operation has been performed automatically by the compiler. To eliminate this warning message use the appropriate type conversion function (TO_SINT, TO_INT, TO_DINT etc.).
G0777	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G0778	Destination may be too small to store current result	The variable destination of the store/assignment operation has a data type smaller than the one of the accumulator. Data may be lost in the operation. For example, if the accumulator contains 340 and the destination operand is of SINT type, the assignment operation will loose data. If the operation is under the programmer's control an appropriate type conversion function (TO_SINT, TO_INT, TO_DINT etc.) can be used to eliminate the warning message.
G0779	Division by zero	The indicated division operation has the constant value 0 as denominator.
G0780	Operation allowed on real parameters only	The indicated operation can't be executed on REAL data types. Instructions of this kind are logical and bitwise operations.
G0781	Illegal comparison	The indicated comparison operation is executed between non homogeneous data types.
G0782	Negation without condition	The indicated operation (JMP or RET) has the negation modifier 'N' without the conditional evaluation modifier 'C'. Use JMPCN instead of JMPN or RETCN instead of RETN.
G0783	Boolean parameter required	The IL operator indicated (typically 'S' or 'R') can't be used when the accumulator has a type other than BOOL.
G0784	Operand extension	The data type of the operand has been extended to the data type of the accumulator. Then the operation is executed. The operand extension take place whenever the operand data type is smaller than the accumulator data type.
G0785	Does not support float accumulator	The accumulator has REAL data type and it's not allowed for the indicated operation (typically MUX operation).
G0786	Does not support boolean accumulator	The accumulator has boolean data type and isn't allowed for the indicated operation (ex. MUX operator).

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G0787	Comparison of unsigned type and signed type	The compare operation indicated is performed using operators that have signed and unsigned data type. Undesired or uncontrolled result may be possible.
G0788	Illegal conversion	Internal compiler error.
G0789	Conversion may result in loss or corruption of data	Error code not used.
G0790	Illegal negation of a real parameter	Error code not used.
G0791	Writing a real value into an integer var / param	The parameter passed to the function is of REAL type instead of an integer data type as required by the function input variables definition.
G0792	Writing an integer value into a real var / param	The parameter passed to the function is of an integer data type instead of the REAL type as required by the function input variables definition.
G0793	Writing a signed value into an unsigned var / param	The assignment operation is performed on an unsigned data type variable but the accumulator data type has a signed data type. Undesired result may be possible.
G0794	Writing an unsigned value into a signed var / param	The assignment operation is performed on an unsigned data type variable but the accumulator data type has a signed data type. Undesired result may be possible.
G0795	Unbalanced parentheses	The number of opened parentheses doesn't match with the number of the closed parentheses in the indicated code block.
G0796	Error while extending parameters	Internal compiler error.
G0797	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G0798	Using a boolean index to access an element of array	The indicated array access is incorrect because the index variable used has a boolean data type.
G0799	Return not allowed from programs	The RET operator isn't allowed in PROGRAM blocks.
G0800	Boolean accumulator required	The indicated SEL operator requires that the accumulator has the boolean data type.
G0801	Operators have mismatching type	The selection performed by MUX and SEL operators shall be done between elements that have homogeneous data types.
G0802	Function block must be instantiated	A function block can't be invoked directly with a CAL instruction. It must be instantiated before its use eg. must be a variable with data type corresponding to the function block instead.
G1537	Using a boolean index to access an element of array	
G1538	Does not support boolean accumulator	
G1539	Does not support float accumulator	
G1540	Error while extending operand(s)	
G1541	Writing a signed value into an unsigned variable	
G1542	Writing an unsigned value into a signed variable	
G1543	Writing a real value into an integer variable	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G1544	Writing an integer value into a real variable	
G1545	Converting a string into a number	
G1546	Converting a number into a string	
G1547	FPU stack full	
G1548	FPU stack empty	
G1549	FPU stack size error	
G1550	Illegal access to variable through function	
G1551	Illegal reference to address of variable accessible through function	
G1552	Invalid access through function	
G1553	Two variables with the same handle	
G1554	Invalid index for variable accessible through function	
G1555	Invalid instruction with non-empty FPU stack	
G1556	Function result of type string requires store to variable	
G8193	Type definition of unknown data type	
G8194	Type definition has exceeding array dimensions	
G8195	Cyclic definition of data type	
G8196	Double pointers are not supported	
G8197	No enumerative elements	
G8199	Invalid or undefined initialization constant	
G10241	Too many initializers for variable	
G10242	Too less initializers for variable	
G10243	Constant without init values	
P2048	Can't open parameters file	The source file for parameters (with PPC extension) can't be opened because of is missing or is locked by the PC's file system.
P2049	Symbol table file not created	The symbol allocation file (with SYM extension) can't be written because of disk write protection or insufficient disk space.
P2050	Can't create parameters file	The parameters file (with PAR extension) can't be written because of disk write protection or insufficient disk space.
P2051	Can't create directory	The directory for the new project can't be created. The problem arises when there is a disk write protection or when the new directory indicated for the project is more than one level deep form an existing disk directory. The compiler creates only one new directory level (the one with the name of the project) starting from an existing directory.

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
P2052	Can't open source project	The source project indicated for creating the new project doesn't exist, is incomplete or is locked by the file system.
P2053	Save project error	The new project can't be saved due to disk write protection, non existing destination directory or file system lock.
P2054	Generic file error	A non specific error occurred during file operations.
P2055	Can't copy file	The indicated file can't be copied because of missing source file, disk write protection or destination file existing and protected.
P2056	Can't save file	The indicated file can't be saved because of disk write protection or destination file existing and protected.
P2057	Object already exist in project	The indicated object (variable, function, function block or program) is contained in the last loaded library but there is already another object with the same name in the current project.
P2058	Can't open library file	The indicated library file doesn't exits or can't be opened due to file system locking.
P2059	Listing file not created	
P2060	Cannot create PLC application binary file	
P2061	Can't open template project	
P2062	Support for processor isn't 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	Can't open auxiliary source file	
P2071	Can't 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)	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
P2077	Cannot load library! Project private copy of this library was missing and the original path to the library is invalid: library has been dropped	
P2078	PLC variables export file not created	
P2079	Debug symbols package (for following download to the target device) not created	
P2080	Source code package (for following download to the target device) not created	
P2081	Invalid task definition	
P2083	Invalid or incoherent task period	
P2084	Broken library link	
S1281	Generic ST error	
S1282	Too many expressions nested	
S1283	No iteration to exit from	
S1284	Missing END_IF	
S1285	Invalid ST statement	
S1286	Invalid assignment	
S1287	Missing;	
S1288	Invalid expression	
S1289	Invalid expression or missing DO	
S1290	Missing END_WHILE	
S1291	Missing END_FOR	
S1292	Missing END_REPEAT	
S1293	Invalid expression or missing THEN	
S1294	Invalid expression or missing TO	
S1295	Invalid expression or missing BY	
S1296	Invalid statement or missing UNTIL	
S1297	Invalid assignment, := expected	
S1298	Invalid address expression	
S1299	Invalid size expression	
S1300	Function return value ignored	
S1301	Invalid parameter passing	
S1302	Function parameter not defined	
S1303	Useless expression	
S1304	Unbalanced parentheses	
S1305	Unknown function	
S1306	Invalid function parameter(s) specification	
S1307	Function parameter doesn't exist	

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
S1308	Multiple assignment not allowed (in accordance with IEC 61131-3)	
S1309	ST preprocessor buffer overflow	
S1310	Function block invocation of a non-function block instance	
S1311	Missing END_WAITING	
S1312	Syntax error	
S1537	Generic SFC error	
S1538	Initial step missing	
S1539	Output connection missing	
S1540	The output pin must be connected to a transition	
S1541	Every output pin of a transition must be connected to a step/jump block	
S1542	Transition expected	
S1543	Step or jump expected	
S1544	Could not find the associate program code	
S1545	Could not find the condition code	
S1546	Unknown-type transition	
S1547	Invalid destination	
S1548	Duplicates action. Same SFC action cannot be used in more than one step	
T8193	Communication timeout	The communication with the target system failed because there is non 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.

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
T8200	Communication device unavailable	The compiler tried to communicate with the target system but the communication channel is not available. If the problem 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 isn't 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	
T8212	Memory read error	
T8213	Memory write error	
T8214	Not enough space available on the target device for the PLC application binary	