

# SoMachine HVAC Software

## Programming Guide

09/2014



---

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

If you have any suggestions for improvements or amendments or have found errors in this publication, please notify us.

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

All pertinent state, regional, and local safety regulations must be observed when installing and using this product. For reasons of safety and to help ensure compliance with documented system data, only the manufacturer should perform repairs to components.

When devices are used for applications with technical safety requirements, the relevant instructions must be followed.

Failure to use Schneider Electric software or approved software with our hardware products may result in injury, harm, or improper operating results.

Failure to observe this information can result in injury or equipment damage.

© 2014 Schneider Electric. All rights reserved.

---

## TABLE OF CONTENTS



---

PART 1. CONNECTION .....	7
PART 2. APPLICATION .....	59
PART 3. USER INTERFACE .....	301
PART 4. SIMULATION .....	446

---

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



---

## ABOUT THE BOOK



---

### Document Scope

This document is aimed at designers and developers and requires a knowledge of one or more IEC61131-3 standard programming languages and is designed to provide a first-level overview of the installation, functions and use of **SoMachine HVAC**.

### Validity Note

This document is valid for **SoMachine HVAC**.

### Related Documents

Title of Documentation	Reference Document Code
Modicon M171 Performance Logic Controllers Hardware User Manual	EIO0000002030 (ENG)
Modicon M171 Optimized Logic Controllers Hardware User Manual	EIO0000002032 (ENG)
SoMachine HVAC software Quick Start	EIO0000002035 (ENG)

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

[www.schneider-electric.com](http://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.<sup>(1)</sup>
- 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.**



## Contents

<b>1.</b>	<b>Basic concepts</b>	13
1.1	Entry point and container	13
1.2	Composite applications and Field I/O	13
1.3	Distributed applications and Binding I/O	13
<b>2.</b>	<b>Using the environment</b>	15
2.1	The workspace	15
2.1.1	The main window	15
2.1.2	The output window	16
2.1.3	The project window	17
2.1.4	The catalog window	18
2.2	Layout customization	20
2.3	Toolbars and docking windows	20
2.3.1	Showing/hiding	20
2.3.2	Moving toolbars	20
2.3.3	Moving docking windows	21
<b>3.</b>	<b>Managing projects</b>	23
3.1	Creating a new project and main page	23
3.2	Saving the project	24
3.3	Managing existing projects	24
3.3.1	Opening an existing project	24
3.3.2	Closing the project	24
3.4	Building projects	25
3.5	Distributing projects	25
3.5.1	Distributing to other developers	25
3.5.2	Distributing to users or installers	25
<b>4.</b>	<b>Managing project elements</b>	27
4.1	M171 Perf. Display	27
4.1.1	PLC	28
4.1.2	HMI	28
4.1.3	CANopen	29
4.1.4	RS485	32
4.1.5	Ethernet	33
4.2	M171 Perf. Blind	34
4.3	Generic Modbus	35
4.3.1	Modbus messages	35
4.4	Modbus Custom	37



4.4.1	Creating a new Modbus custom device	37
4.4.2	Editing an existing Modbus custom device	37
4.4.3	Deleting a Modbus custom device	38
4.4.4	Using a Modbus custom device	38
<b>4.5</b>	<b>CAN custom</b>	<b>40</b>
4.5.1	Importing a new CAN custom device	40
4.5.2	Deleting a CAN Custom device	41
4.5.3	Using a CAN custom device	41
<b>4.6</b>	<b>Display for M171 Perf.</b>	<b>43</b>
4.6.1	CANopen	43
<b>4.7</b>	<b>M171 Perf. Flush Mounting</b>	<b>46</b>
4.7.1	PLC	46
4.7.2	HMI	46
4.7.3	Providing HMI pages	47
4.7.4	CANopen	47
4.7.5	RS485	49
4.7.6	Ethernet	49
<b>4.8</b>	<b>Modicon M171 Perf. Expansion 27 I/Os</b>	<b>49</b>
4.8.1	Using Modicon M171 Perf. Expansion 27 I/Os as CANopen field slave	50
4.8.2	Using Modicon M171 Perf. Expansion 27 I/Os as RS485 field slave	50
<b>4.9</b>	<b>Virtual channels assignment criteria</b>	<b>51</b>
4.9.1	CANopen network - virtual SDO servers	51
4.9.2	Ethernet - TCP Slave Channels	52
4.9.3	CANopen field - virtual master channels	52
<b>5.</b>	<b>Technical reference</b>	<b>53</b>
<b>5.1</b>	<b>CANopen protocol</b>	<b>53</b>
5.1.1	Overview	53
5.1.2	Physical structure of a CANopen network	53
5.1.3	COB and COB-ID	53
5.1.4	The object Dictionary	53
5.1.5	The Service Data Objects (SDO)	54
5.1.6	The Process Data Objects (PDO)	54
5.1.7	PDO transmission modes	54
5.1.8	The Emergency Object	55
5.1.9	SYNC Object and Time Stamp Object	55
5.1.10	Error Control: Node guarding	55
5.1.11	Error control: Heartbeat	55
5.1.12	The Network Behavior	55
5.1.13	The Boot-up Message	56
5.1.14	The CANopen Device Profiles	56
<b>5.2</b>	<b>Modbus protocol</b>	<b>56</b>



## SoMachine HVAC - Connection

5.2.1	Overview	56
5.2.2	Data types	56
5.2.3	Function codes	57
5.2.4	Error detection and CRC	57
5.2.5	Protocol versions	57



## 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](http://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.<sup>(1)</sup>
- 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. BASIC CONCEPTS

### 1.1 ENTRY POINT AND CONTAINER

Device is an important piece in the SoMachine HVAC software suite.

It is designed to be the “entry point” to create and manage complex projects, made of different devices and sub-projects; its main purpose is to keep all the pieces together and to simplify the task of linking the various elements (software components or physical devices).

For example, with Connection you can create a project (that can be seen as a big “workspace”) that consists in two or more devices physically linked together on the same network, that have both a PLC and HMI project, that act as a master and exchange data with remote slaves, and moreover exchange data between each other, in a peer-to-peer relationship.

At the end of the developing process, Connection will create a single file containing ALL the PLC programs, HMI pages, parameters and settings of ALL the devices; then, using Device, you can distribute and deploy your complex application in the product environment with a single click.

Connection can also be seen as the starting point from which all the other tools of the Suite can be launched, opening their respective documents and projects: Application, UserInterface, and Device.

### 1.2 COMPOSITE APPLICATIONS AND FIELD I/O

Rich and advanced devices (such as M171 Perf. Display) can both run a PLC program and show HMI pages on the same hardware.

With Connection you can create the two separate sub-projects by launching the corresponding program (Application for PLC and UserInterface for HMI) and keep them together in a single folder on disk.

If the device can act as a master (that is the case of M171 Perf. Display), it can exchange data on a local bus talking with one or more slaves, with a protocol like Modbus or CANopen; with Connection you can describe those master/slave networks, by inserting the slaves into the project and connecting their remote objects to local PLC variables, making the PLC program aware of them.

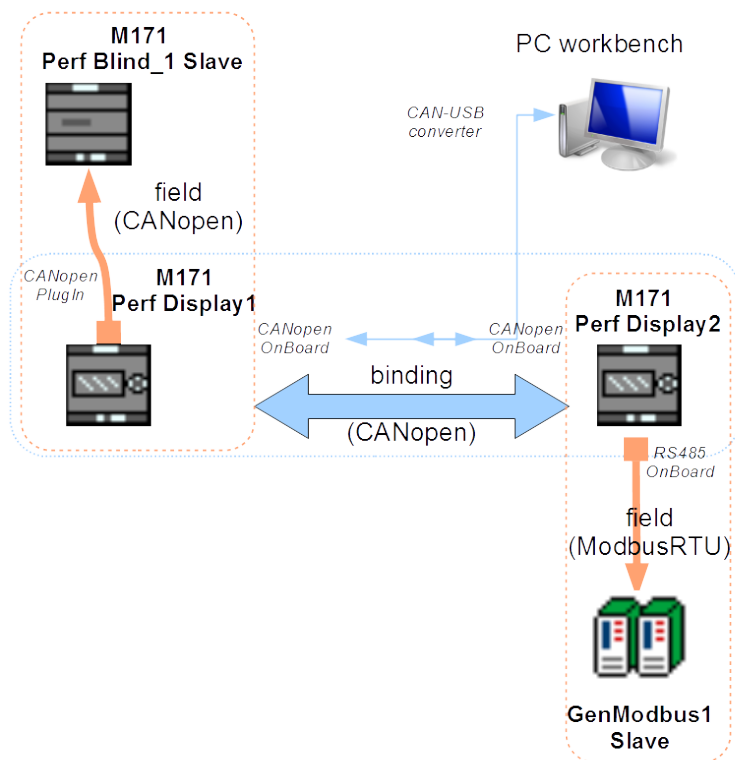
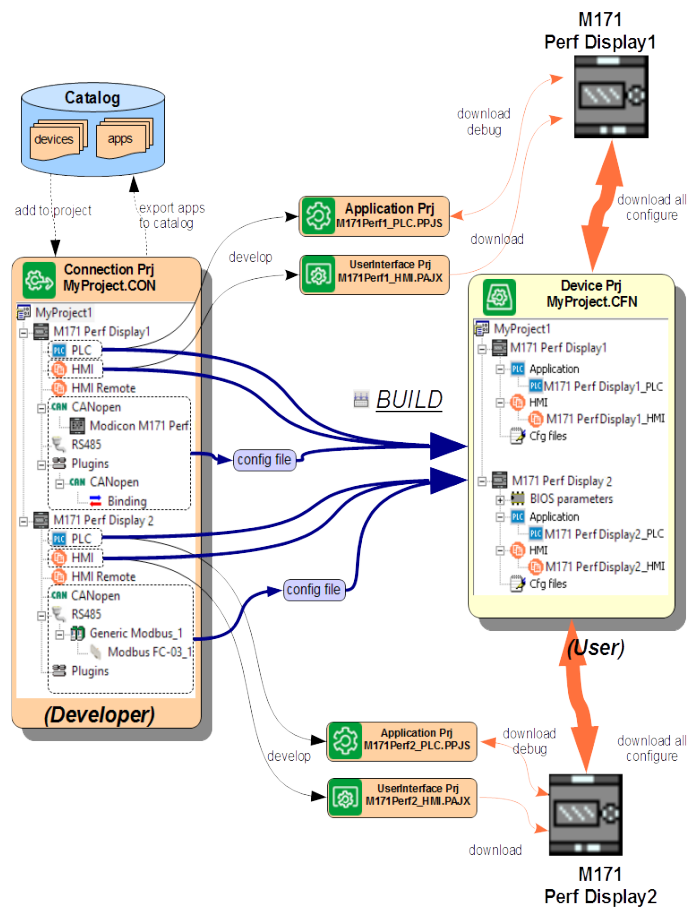
We call this architecture “Field I/O”.

### 1.3 DISTRIBUTED APPLICATIONS AND BINDING I/O

Sometimes a single application on a single device is not enough to solve complex applications; sometimes it is necessary to create two or more applications that will act together, communicating and exchanging data on a network to take decisions and cooperate.

This scenario is different from “Field I/O” because there is no master or slave, but a group of devices of the same kind (like a group of M171 Perf. Display) that talk to each other in a peer-to-peer way on a common network, exchanging objects (parameters and values.)

We call this architecture “Binding I/O”, because the various elements are bound to each other to operate together.

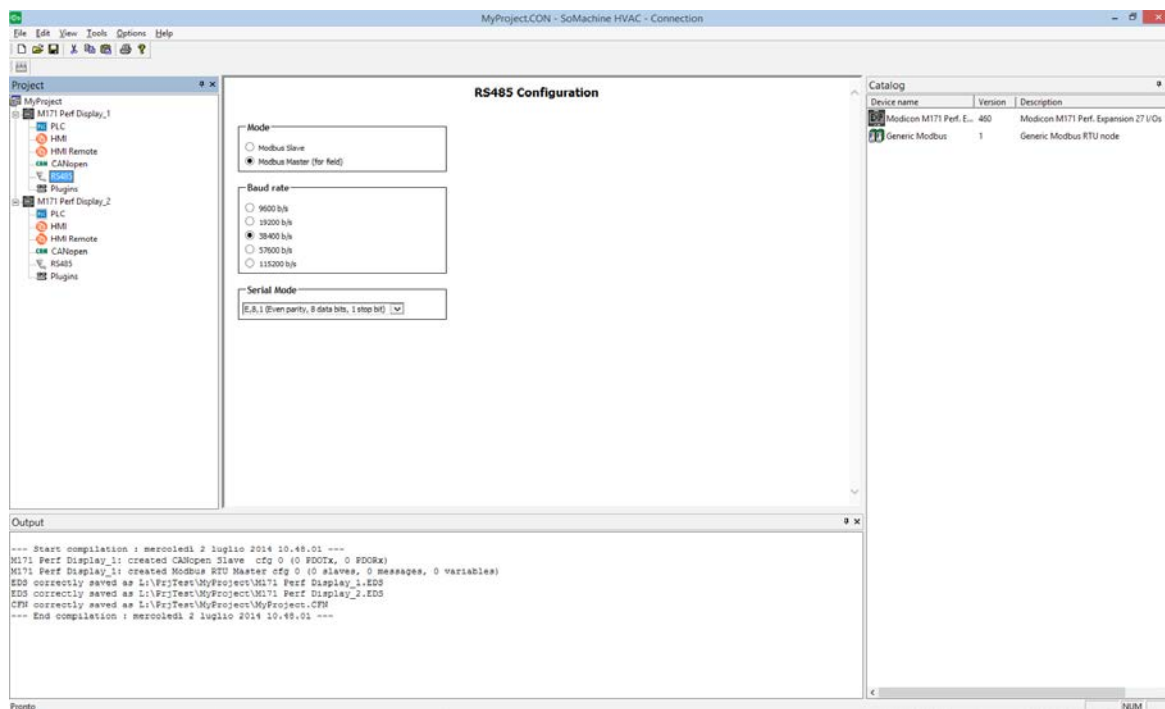




## 2. USING THE ENVIRONMENT

### 2.1 THE WORKSPACE

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



The following paragraphs give an overview of these elements.

#### 2.1.1 THE MAIN WINDOW

The *Main window* is the central part of the program window, that is surrounded by toolbars and docking windows.

It shows information and configuration pages in a graphical and user-friendly form; the current page is always determined by the selected item in *Project* window.

For example, in the previous image you can see that the *RS485* node is selected (and highlighted) in the *Project* tree and so the *Main window* shows the *RS485 Configuration* page.

To change the current selected item and so the current page, just do a single click in the *Project* tree.



M171 Perf Display Configuration											
General		SDO Set			PDO Tx - Input			PDO Rx - Output			
Assign		UnAssign									
#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size			
1	6000	1	1	0	0	Read Input 1h to 8h	BOOL	1			
2	6000	1	1	1	0	Read Input 1h to 8h	BOOL	1			
3	6000	1	1	2	0	Read Input 1h to 8h	BOOL	1			
4	6000	1	1	3	0	Read Input 1h to 8h	BOOL	1			
5	6000	1	1	4	0	Read Input 1h to 8h	BOOL	1			
6	6000	1	1	5	0	Read Input 1h to 8h	BOOL	1			
7	6000	1	1	6	0	Read Input 1h to 8h	BOOL	1			
8	6000	1	1	7	0	Read Input 1h to 8h	BOOL	1			
9	6000	2	1	8	0	Read Input 9h to 16h	BOOL	1			
10	6000	2	1	9	0	Read Input 9h to 16h	BOOL	1			
11	6000	2	1	10	0	Read Input 9h to 16h	BOOL	1			
12	6000	2	1	11	0	Read Input 9h to 16h	BOOL	1			
13	6401	1	2	0	0	Analogue Input 1	INT	16			
14	6401	2	2	16	0	Analogue Input 2	INT	16			
15	6401	3	2	32	0	Analogue Input 3	INT	16			
16	6401	4	2	48	0	Analogue Input 4	INT	16			
17	6401	5	3	0	0	Analogue Input 5	INT	16			
18	6401	6	3	16	0	Analogue Input 6	INT	16			
19	2230	0	5	0	0	Counter	UDINT	32			
20	2232	0	5	32	0	Fequency	UDINT	32			

### 2.1.2 THE OUTPUT WINDOW

The *Output* window is the place where Connection prints its output messages: errors, information, debug information, and compilation results.

```

Output
--- Start compilation : mercoledi 2 luglio 2014 10.56.37 ---
M171 Perf Display_1: created CANopen Slave  cfg 0 (0 PDOTx, 0 PDORx)
M171 Perf Display_1: created Modbus RTU Master  cfg 0 (0 slaves, 0 messages, 0 variables)
EDS correctly saved as L:\PrjTest\MyProject\M171 Perf Display_1.EDS
EDS correctly saved as L:\PrjTest\MyProject\M171 Perf Display_2.EDS
CFN correctly saved as L:\PrjTest\MyProject\MyProject.CFN
--- End compilation : mercoledi 2 luglio 2014 10.56.38 ---
    
```

In some situations (for example compilation errors) you can double-click on the error in the output window and you will be brought just at the source of the error, that will be highlighted with a red box.



# SoMachine HVAC - Connection

**Modicon M171 Perf. Expansion 27 I/Os Configuration**

General | SDO Set | PDO Tx - Input | PDO Rx - Output

**Network settings**

Node number (1..122)	0
Node Guard Period (ms)	200
Life time Factor	3
Boot time elapsed (ms)	2000
Node heartbeat producer time (ms)	0
Node heartbeat consumer time (ms)	0
Master heartbeat consumer time (ms)	0
Identity object check	<input type="checkbox"/>

**PDO Tx communication settings**

- USER DEFINED Mode
- SYNC Mode
- EVENT Mode
- CYCLIC Mode  ms

**PDO Rx communication settings**

- USER DEFINED Mode
- SYNC Mode
- EVENT Mode

Output

```
--- Start compilation : mercoledi 2 luglio 2014 10.58.49 ---  
ERROR: (BuildCfg CANopenMaster) Invalid node number: 0 (must be in 1..122)
```

## 2.1.3 THE PROJECT WINDOW

**Project**

- MyProject
  - M171 Perf Display\_1
    - PLC
    - HMI
    - HMI Remote
    - CANopen
      - Modicon M171 Perf. Expansion 27 I/Os\_1
      - RS485
      - Plugins
  - M171 Perf Display\_2
    - PLC
    - HMI
    - HMI Remote
    - CANopen
      - Binding
    - RS485
      - Generic Modbus\_1
        - Modbus FC-03\_1
        - Modbus FC-03\_2
        - Modbus FC-04\_1
      - Plugins



The *Project window* shows the elements of the current project in the form a tree, making easy to see the master/slave and parent/child relationship between them.

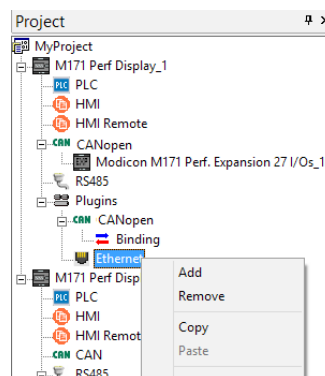
Click on the + and - icons next to each item (or press to corresponding keys) to expand or collapse each item; or press the \* key to expand all children of the current item in depth.

Left-clicking an item opens its configuration page in the *Main window* (if there is one), and shows in the *Catalog window* all objects that can be inserted under the current item (if there are).

Right-clicking an item selects it and opens its context menu (if there is one), showing some operations you can do on the current tree item, like *Add/Remove/Copy/Paste* and so on.

Pressing the *Delete* key also triggers the *Remove* command.

A single left-click of the item name (or the *F2* key) triggers the in-place rename of the object (if it supports it).



## 2.1.4 THE CATALOG WINDOW

Catalog	
Device name	Version
M171 Perf Display	423
M171 Perf Blind	477
Display for M171 Perf	476
M171 Perf Flush Mounting	489

This window shows a list of objects that can be inserted in the project under the currently selected item in the *Project window*; selecting a different item in *Project window* refreshes this list.

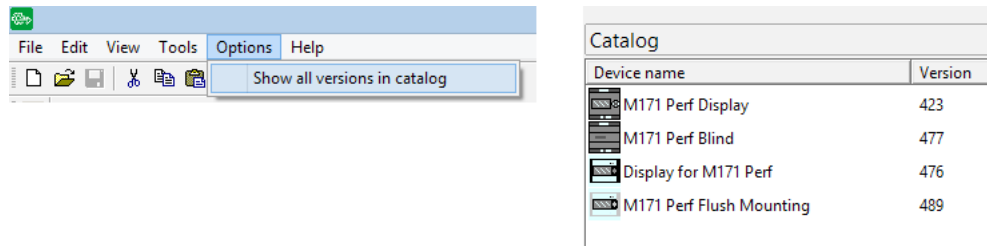
By default, only the “major” version number of each device is shown, and the highest minor version number is implicitly selected; for example, if three different versions of the same device are present in the catalog (10.0, 10.1, 10.2), the *Catalog* will show only the 10 (without the minor version) but will select the 10.2 (the highest).

This behavior can be changed by selecting the *Show all versions in catalog* option in

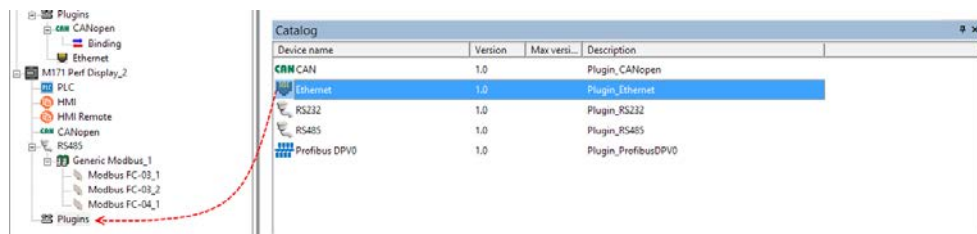


## SoMachine HVAC - Connection

the *Options* menu in the menu bar; if you activate this option ALL the available versions (even the older ones) will be shown in the *Catalog* and you will have the chance to manually select and add in the project older versions of each device.

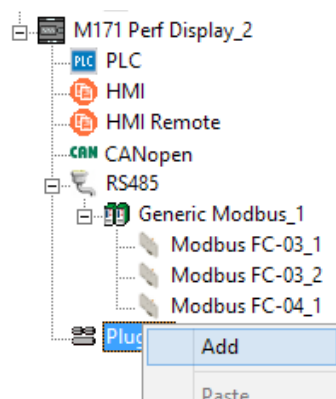


To add an object, drag and drop it from the *Catalog* window to the *Project* window, over the currently selected item (a + icon will appear); it will be added as last child.



Another way to add an object is to right-click an item in the *Project* window and choose *Add*; a pop-up window will appear, showing the same list of the *Catalog* window. In this way you can add an object without having the *Catalog* visible, useful for example if you are working with a very small screen.

This window also has a *Show all versions* option, that behaves like the flag in the *Options* menu described before.





Device name	Version	Max versi...	Description
CAN CANopen	1.0		Plugin_CANopen
Ethernet	1.0		Plugin_Ethernet
RS232	1.0		Plugin_RS232
RS485	1.0		Plugin_RS485
Profibus DPV0	1.0		Plugin_ProfibusDPV0

## 2.2 LAYOUT CUSTOMIZATION

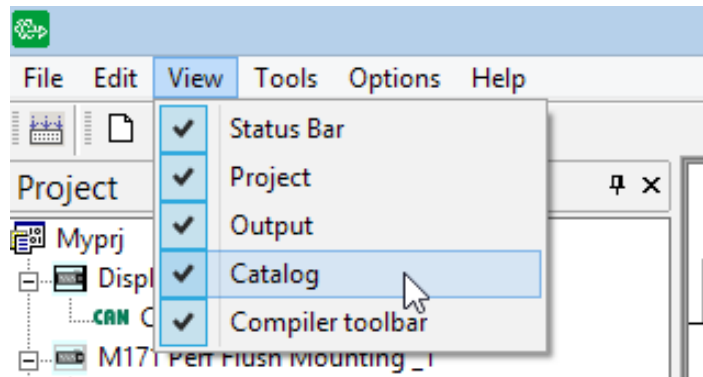
The layout of Connection workspace can be freely customized in order to suit your needs.

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

## 2.3 TOOLBARS AND DOCKING WINDOWS

### 2.3.1 SHOWING/HIDING

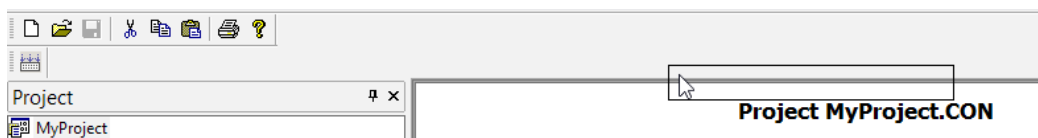
To show (or hide) a toolbar, open the *View* menu and select the desired toolbar or docking window (for example, the *Catalog* dock window).



The element is then shown or hidden.

### 2.3.2 MOVING TOOLBARS

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



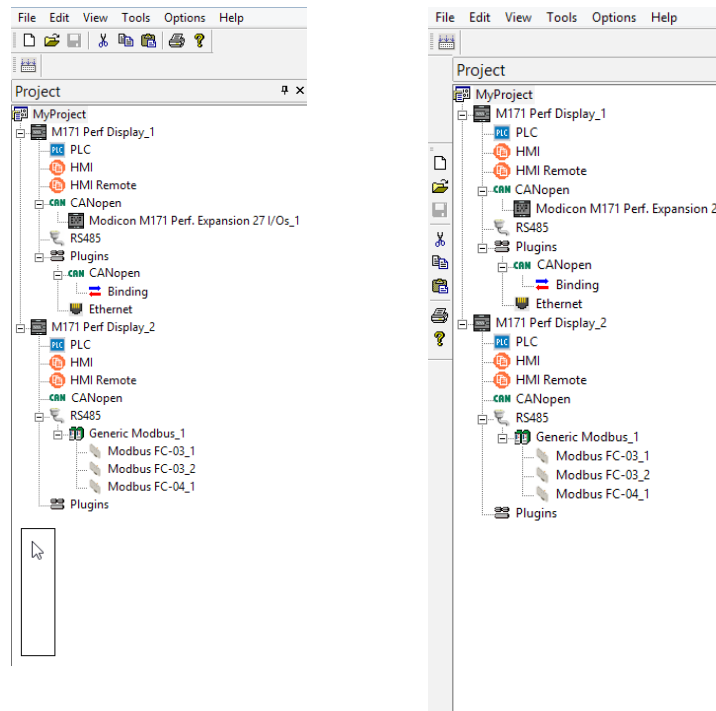
The toolbar shows up in the new position.

You can change the shape of the toolbar, from horizontal to vertical, either by pressing the





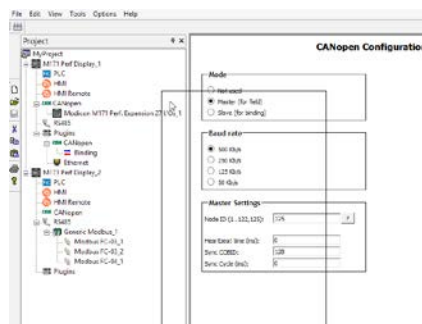
*Shift* key or by moving the toolbar next to the vertical border of any window.



You can also make the toolbar float, either by pressing the *CTRL* key or by moving the toolbar away from any window border.

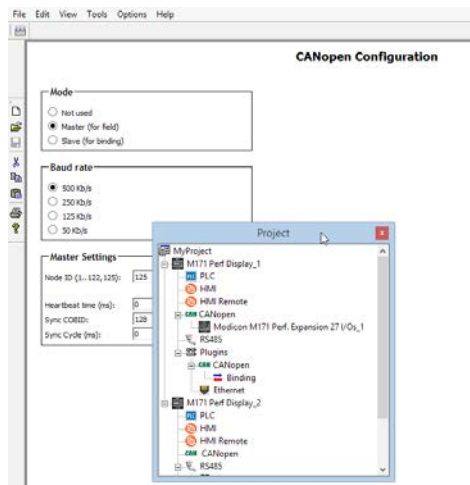
## 2.3.3 MOVING DOCKING WINDOWS

In order to move a docking window, click on its name (at the top of the window) and then drag and drop it to the destination.





You can make the tool window float, by double-clicking on its name, or by pressing the *CTRL* key, or by moving the tool window away from the main window borders.



A tool window can be resized by clicking-and-dragging on its border until the desired size is reached.



## 3. MANAGING PROJECTS

### 3.1 CREATING A NEW PROJECT AND MAIN PAGE

When you open Connection, you are presented with the *Main page*.

In the *General* tab you can open the last recently opened projects (shown in upper section) or insert a new device in the project, selecting it in the lower panel.

Here you can see all the "top level" devices that you can add, and this window shows the same content of the *Catalog window* when the root item is selected in the *Project window*; therefore it follows the same behavior with respect to the *Show all versions in catalog* flag.

With a just a click in the list, a new device is inserted in the project tree, ready to be configured and programmed.

**Project Untitled**

General
Networks list

**Most recent projects**

- L:\PrjTest\MyProject\MyProject.CON
- L:\PrjTest\MyProject1\MyProject1.CON
- L:\PrjTest\TestSESuite\TestSESuite.CON
- L:\PrjTest\TestEli\TestEli.CON

Add new device to project		
	M171 Perf Display	423
	M171 Perf Blind	477

In the second tab of the *Main page*, named *Networks list*, you can manage a list of all the "virtual networks" to be used in your project with the devices that will be connected with Binding I/O.

For each network you have to choose a name, the protocol to use (CANopen or Ethernet/ModbusTCP) and symbolic color to show as a small circle in the project tree; each device connected to the same network will be shown with the same color.

While by default you already have two predefined networks (one CANopen and one Ethernet) you can add any number of other networks, to build complex scenarios.

**Project MyProject.CON**

General
Networks list

Add    Remove

Name	Type	Color	Description
CANopen1	CANopen		CANopen Network 1
Ethernet1	Ethernet		Ethernet Network 1

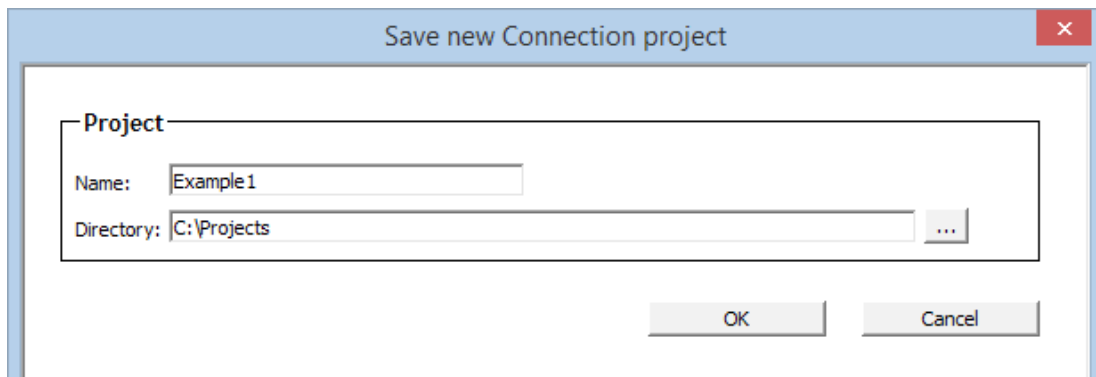


## 3.2 SAVING THE PROJECT

To save the project, you can select the corresponding item of the menu *File* or the *Main* toolbar.

The Connection project is a single file that has *.CON* extension; it links other sub-components (like PLC application or HMI pages) that typically reside in the same containing folder.

If you are saving a new project (that is still *Untitled*), you are presented with a dialog that asks you the new name for the project and the directory where to save it; the program will create a folder of the chosen *Name* under the chosen *Directory*, and will save a file named like *Name.CON* under it.



In the above example, the folder *C:\Projects\Example1\* will be created and the project will be saved as *C:\Projects\Example1\Example1.CON*.

If you want to save the project with another name, you can choose the command *File / Save as...* and specify a new name and location for the *.CON* file.

**IMPORTANT:** only the *.CON* project file is saved, no folder is created nor the linked components (PLC or HMI) are copied or moved.

## 3.3 MANAGING EXISTING PROJECTS

### 3.3.1 OPENING AN EXISTING PROJECT

To open an existing project, click *Open* in the *File* menu of Connection's main window, or in the *Main* toolbar. This will open a dialog box, which lets you browse to the directory containing the project and select the relative project file.

Otherwise, you can select one of recently opened projects from the *File* menu or in the *Main page*.

### 3.3.2 CLOSING THE PROJECT

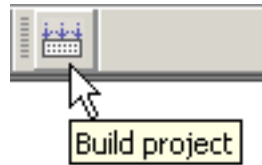
You can terminate the working session either by:

- starting a new project, with the *File / New* command, or the button in the toolbar;
- explicitly closing the current project with *File / Close* command;
- by exiting Connection.

In all cases, when there are changes not yet saved to file, the program asks you to choose between saving and discarding them.



## 3.4 BUILDING PROJECTS



When you press the *Build project* button in the toolbar (or the *F7* key), Connection will examine the current project and will:

- Print in the output window any error it found in the checking process; you can then double-click the error to see the source position.
- Generate specific configuration files for each device (for example *CONNEX.PAR* for M171 Perf. Display, with Field and Binding configuration settings).
- Generate a single *.CFN* file to be used in Device; this file will contain all the sub-components of the current project (devices configurations, PLC applications and HMI pages) all contained inside the *CFN*, in a redistributable form; this file will have the same name of the *.CON* project and will reside in the same folder.

Choosing the *Tools / Open with Device* command, Device will be launched with the generated *CFN* file opened.

**IMPORTANT:** before executing compilation, please make sure that all the PLC and HMI sub-project have been built with the respective tools (Application and UserInterface). In fact Connection will include in the *CFN* the last compilation output of each sub-component, so you have to build them **BEFORE** compiling the Connection project.

**NOTE:** be sure to eliminate all compiler errors and warnings before downloading your application to your controller or other device.

## 3.5 DISTRIBUTING PROJECTS

This topic should be discussed in two different parts:

### 3.5.1 DISTRIBUTING TO OTHER DEVELOPERS

To distribute the full Connection project to other developers (for example for further development or debugging) you can give the entire folder containing the *.CON* file, that has been created by Connection with the first *Save* command.

In this way all the sub-components created by Connection (PLC, HMI, *CFN* file) are all contained inside, and since the file paths are maintained as relatives the project can be moved around; so other developers can open the Connection project anywhere and work normally.

One important exception is for *.CON* projects that link external components, for example external PLC projects (on an external directory, or taken from catalog); in this scenario you will have to distribute all the external components manually, because they are not self-contained in the main project folder.

### 3.5.2 DISTRIBUTING TO USERS OR INSTALLERS

In this scenario, it is enough to distribute the *CFN* file (Device document) created by Connection; you will be able to download everything (PLC, HMI, config files, parameters values) only using Device with a single click.

One important exception is for *.CON* projects that link external components from the catalog (PLC and HMI), in this case the produced *CFN* file will not include them; they must be distributed manually.





## 4. MANAGING PROJECT ELEMENTS

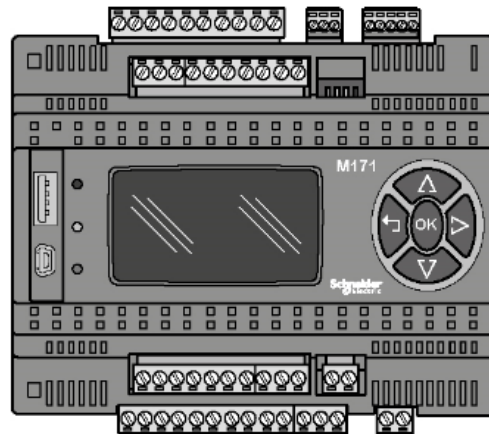
### 4.1 M171 PERF. DISPLAY

M171 Perf. Display is one of the top-level devices that you can insert in the project. On its main page you can change its name and see a picture of it.

**General**

Name:

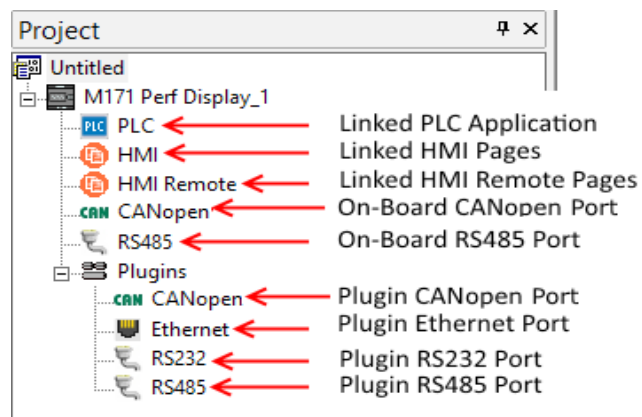
Version:



Dip-switch setting:



It can run both a PLC application and HMI pages on the same CPU and has a lot of connectivity capabilities, in terms of on-board connectors and may optional plug-ins.



Follows detailed description of each element.



## 4.1.1 PLC

This tree item lets you create or associate a PLC project to the M171 Perf. Display; the associated page shows the relative path of the associated *PPJS* file (Application project).

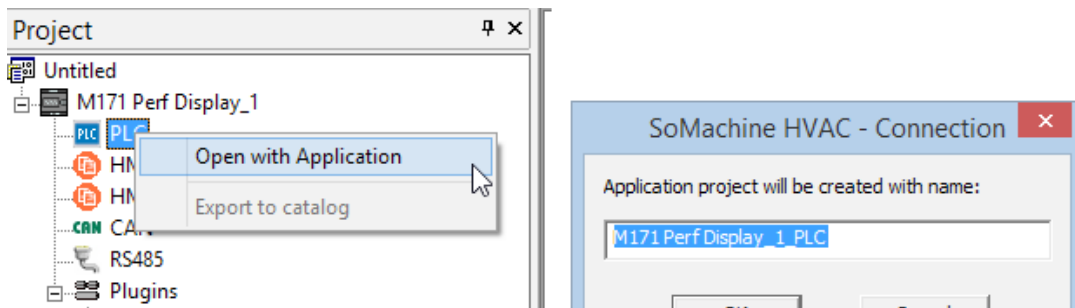
### PLC Configuration

**General**

From Project     From Catalog

PLC Project:

If you do a right-click on the *PLC*, a pop-up menu will appear with the command *Open with Application*; if the device has no associated project, you will be prompted for the name to give to the new application (by default, the name of the device with the *\_PLC* suffix).



Otherwise if a PLC project has already been associated, Application will be launched and the existing PLC project opened.

If you want to manually associate an existing PLC project to the device, you can choose between a project on the disk in a particular folder or choosing from the local catalog of applications.

If a PLC project has been associated, the *Export to catalog* command in the pop-up menu will be enabled, allowing you to export the application in the catalog for further reuse.

## 4.1.2 HMI

This tree item lets you create or associate a HMI project to the M171 Perf. Display; the associated page shows the relative path of the associated *PAJX* file (UserInterface project).

### HMI Configuration

**General**

From Project     From Catalog   

HMI Project:

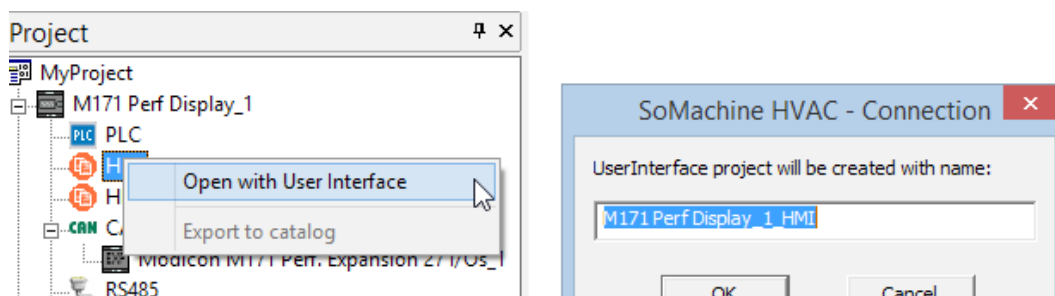
Name	ID	Protocol	Address
Device_0	0	Local	





## SoMachine HVAC - Connection

If you do a right-click on the *HMI*, a pop-up menu will appear with the command *Open with UserInterface*; if the device has no associated project, you will be prompted for the name to give to the new application (by default, the name of the device with the *\_HMI* suffix).



Otherwise if a HMI project has already been associated, *UserInterface* will be launched and the existing HMI project opened.

If you want to manually associate an existing HMI project to the device, you can choose between a project on the disk in a particular folder or choosing from the local catalog of applications.

If a HMI project has been associated, the *Export to catalog* command in the pop-up menu will be enabled, allowing you to export the application in the catalog for further reuse.

### 4.1.2.1 RETRIEVING REMOTE DATA FROM LOCAL HMI PAGES

If in your HMI pages project you have imported one or more parameter map, you can configure the real address of the remote device here.

In fact by default any parameter map is considered as "local", and if you want to view in your page any parameter of a remote device you have to insert here (and so outside and independently from *UserInterface*) the used protocol (Modbus RTU, Modbus TCP or CANopen) and address.

In this way you can design the HMI pages in *UserInterface* as they were "local" and then later change the real address of the remote device without even recompiling the *PAJX* project (the change is made only in *Connection*).

To load or update the list of remote devices (parameter maps) inserted in the *UserInterface* project, press the *Reload device list* button; please remember to build the *PAJX* project with *UserInterface* to have an updated list before doing this.

### 4.1.3 CANOPEN

M171 Perf. Display has one on-board CANopen port, plus another one available as an external plug-in. Each port can be configured as *Not used* (disabled), *Master* (field), *Slave* (binding).

#### 4.1.3.1 FIELD

When you configure the CANopen port as *Master* the M171 Perf. Display will act as a CANopen master on this port, so you can attach any number of CANopen slave devices here and exchange data with Field I/O.



**Mode**

Not used  
 Master (for field)  
 Slave (for binding)

**Baud rate**

500 Kb/s  
 250 Kb/s  
 125 Kb/s  
 50 Kb/s

**Master Settings**

Node ID (1..127):

Heartbeat time (ms):

Sync COBID:

Sync Cycle (ms):

For a CANopen master port, you have to configure (see 5.1 for further information):

- Baud rate used in this CANopen network (in Kb/s).
- Node ID for the master (1..127), by default is 127.
- Heartbeat time in ms, by default 0 (heartbeat producer disabled): it is the master producer heartbeat time.
- The SYNC COBID to use, by default 128.
- The period for the SYNC cycle in ms, by default 0 (sync disabled).

Example of possible slaves are the Modicon M171 Perf. Expansion 27 I/Os module (see 4.5) or generic custom devices imported from their EDS files (CAN custom, see 4.4).

After you added and configured the various CANopen slaves, you can establish the “link” between the remote objects of the slave and the internal PLC variables to read or write.

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

- Status variables, created with Application (not BIOS).
- Field variables, created with Application.

### 4.1.3.2 BINDING

When you configure the CANopen port as *Slave* the M171 Perf. Display will act as a CANopen slave on this port, so you can exchange data with Binding I/O with other M171 Perf. Display devices on the CANopen network.

#### Configuring the port

**Mode**

Not used  
 Master (for field)  
 Slave (for binding)

**Baud rate**

500 Kb/s  
 250 Kb/s  
 125 Kb/s  
 50 Kb/s

**Slave Settings**

Node ID (1..127):

Network:



## SoMachine HVAC - Connection

For a CANopen slave port, you have to configure:

- Baud rate used in this CANopen network (in Kb/s).
- Node ID for the slave (1..127), by default is 127.
- The "virtual network" where this M171 Perf. Display is attached; in the tree will appear a small colored circle of same color of the chosen network (same color means same network).

There are two devices on the same CANopen network, named CANopen1 (see red circle indicator)

### The *Binding* object

When you configure a CANopen port as *Slave*, you can add under it a *Binding* object: add it if a device wants to READ objects from other ones, while it not needed if the device only SEND objects on the network.

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

- EEPROM parameters, created with Application (not BIOS).
- Status variables, created with Application (not BIOS).

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

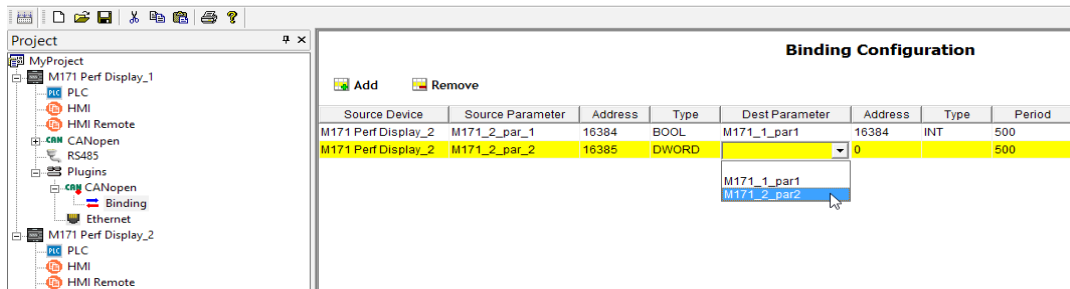
To do this click the *Add* button, a window showing all the "public" objects from all other devices on this same network will appear; here you can apply search filters and choose which objects to read from (multi-selection is also supported).

Source Device	Source Parameter	Address	Type	Dest Parameter	Address	Type	Period
M171 Perf Display_2	16384	par_1	(BOOL)				
M171 Perf Display_2	16385	par_2	(DWORD)				



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

**IMPORTANT:** please remember to rebuild the PLC project with Application to see an updated list of public objects here.



In the above example:

- *M171 Perf Display\_1* will read from *M171 Perf Display\_2* the *M171\_2\_par1* object and will put it in its local *M171\_1\_par1* object.
- *M171 Perf Display\_1* will read from *M171 Perf Display\_2* the *M171\_2\_par2* object and will put it in its local *M171\_1\_par2* object.

In the *Period* you can configure in detail the single period for each parameter; the object will be updated every "period" ms.

## 4.1.4 RS485

M171 Perf. Display has one on-board RS485 port, plus another one available as an external plug-in. Each port can be configured as *Not used* (disabled) or *Master* (field).

### 4.1.4.1 FIELD

When you configure the RS485 port as *Master* the M171 Perf. Display will act as a ModbusRTU master on this port, so you can attach any number of Modbus slave devices here and exchange data with Field I/O.

#### RS485 Configuration

**Mode**

Not used  
 Modbus Master (for field)

---

**Baud rate**

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

---

**Serial Mode**

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

For a Modbus master port, you have to configure:

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



Example of possible slaves are the Modicon M171 Perf. Expansion 27 I/Os module (see 4.5), *Generic Modbus* devices (see 4.2), or custom devices created with the ModbusCustomEditor tool (see 4.3).

After you added and configured the various Modbus slaves, you can establish the “link” between the remote objects of the slave and the internal PLC variables to read or write.

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

- Status variables, created with Application (not BIOS).
- Field variables, created with Application.

## 4.1.5 ETHERNET

M171 Perf. Display can have one Ethernet port, available as an external plug-in. The port always acts a Modbus TCP slave, and additionally can be configured also as *Master* (binding).

### 4.1.5.1 BINDING

#### Configuring the port

##### Ethernet Configuration

**General**

Enable Modbus Master (for binding)

IP Address:

**Settings**

Network:

For an Ethernet port, you have to configure:

- if it acts also a *Master* (otherwise only *Slave* is implied);
- its IP address;
- the “virtual network” where this M171 Perf. Display is attached; in the tree will appear a small colored circle of same color of the chosen network (same color means same network).

Name	Type	Color	Description



## The *Binding* object

When you configure a Ethernet port as *Master*, you can add under it a *Binding* object: add it if a device wants to READ objects from other ones, while it not needed if the device only SEND objects on the network (in this case you do not even need to activate the *Master* feature).

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

- EEPROM parameters, created with Application (not BIOS).
- Status variables, created with Application (not BIOS).

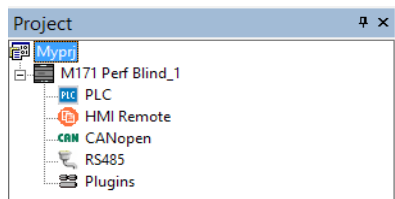
The configuration page for the *Binding* object in Modbus TCP is the same of CANopen, so see 4.1.3.2 for a description and usage of this page.

Because the interface is the same between the two protocols, you can focus on designing your distributed application without knowing the specific communication protocol details.

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

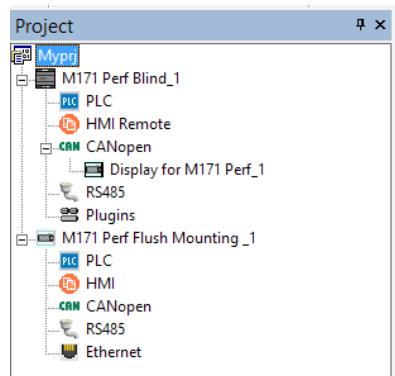
## 4.2 M171 PERF. BLIND

M171 Perf. Blind is a top-level device that has the same characteristics and network behaviour of a M171 Perf. Display device but does not support local HMI. In fact it has no on-board display to show its own pages. M171 Perf. Blind supports HMI Remote so its pages can be downloaded and shown by Display for M171 Perf. or M171 Perf. Flush Mounting keyboards.



Please refer to 4.1 - M171 Perf. Display chapter for a full description of all M171 Perf. Blind features.

### Usage example



In this scenario M171 Perf. Blind\_1 device has a PLC project and has an HMI Remote project that makes available M171 Perf. Blind pages for linked keyboards.

M171 Perf. Blind HMI Remote pages can be remoted and shown by Display for M171



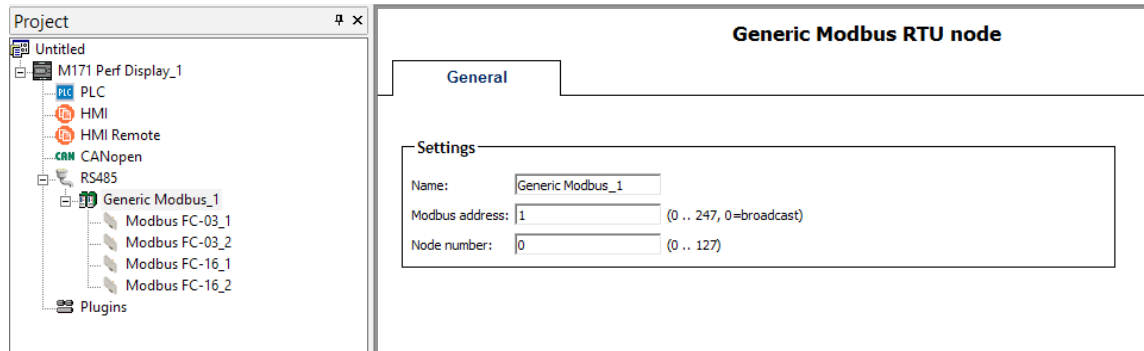
Perf.\_1 via CANopen field and by M171 Perf. Flush Mounting\_1 via Ethernet network.

## 4.3 GENERIC MODBUS

The *Generic Modbus* object is a generic Modbus slave that can be inserted under the RS485 port of the M171 Perf. Display, when configured as *Modbus master*.

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

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



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

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

### 4.3.1 MODBUS MESSAGES

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

The following messages are supported:

- Function 2 (Read discrete inputs, 0x2): reads one or more read-only digital input (1-bit objects).
- Function 3 (Read holding registers, 0x3): reads one or more read-write register (16-bit objects).
- Function 4 (Read input registers, 0x4): reads one or more read-only register (16-bit objects).
- Function 15 (Write multiple coils, 0xF): writes one or more digital output (1-bit objects).
- Function 16 (Write multiple registers, 0x10): writes one or more register (16-bit objects).

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



## 4.3.1.1 GENERAL TAB

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

General
Inputs

**Settings**

Start address:  (1 .. 65536)

Polling time:  ms (0 = write on variation)

Time out:  ms

Wait before send:  ms

- Start address: address of the first modbus object to read or write (1..65536).
- Polling time: the message will be processed with this period (ms); for writing operations, 0 means to write it only on variation of the value, for reading operations 0 means maximum speed.
- Timeout: the operation will be unsuccessful when this time-out expires (ms).
- Wait before send: this is an additional timeout, to be used with slow slaves that do not answer if the messages are sent too fast.

## 4.3.1.2 REGISTERS TAB

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

General
Holding Reg.

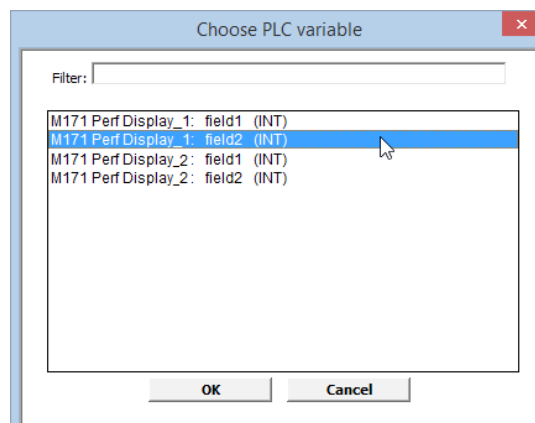
Add
 Remove
 Assign
 UnAssign

#	Name	ObjType	Label	DataBlock	Description
1	Register	WORD	field1	IW11.0	
2	Register	WORD	field2	IW11.1	
3	Register	WORD	field3	IW11.2	
4	Register	WORD	field4	IW11.3	

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

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

**IMPORTANT:** please remember to rebuild the PLC project with Application to see an updated list of PLC variables here.







## 4.4 MODBUS CUSTOM

*Modbus custom* devices can be created and edit directly by the user.

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

### 4.4.1 CREATING A NEW MODBUS CUSTOM DEVICE

To create a new *Modbus custom* device, choose *Tools / Run ModbusCustomEditor*; the external *ModbusCustomEditor* tool will be launched, with a new empty document.

#	Address	Label	Type	Read only	Modbus type	Description
1	1	temperature	INT	False	Holding Register (16 bit)	Temperature value
2	2	pressure	UINT	False	Holding Register (16 bit)	Pressure value
3	3	speed	INT	True	Input Register (16 bit)	Speed reading
4	10	digInput1	BOOL	True	Discrete Input	First digital input
5	20	digOutput1	BOOL	False	Coil	First digital output

Here you can configure:

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

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

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

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

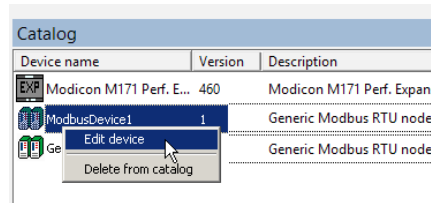
### 4.4.2 EDITING AN EXISTING MODBUS CUSTOM DEVICE

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

- Run the *ModbusCustomEditor* with the *Tools / Run ModbusCustomEditor* command, and then manually open the *PCT* file with the standard *File / Open* command.



- When the device you want to edit is visible in the *Catalog window* (for example when a RS485 node is selected and is in *Master mode*), you can right-click on it and choose the *Edit device* command; the *ModbusCustomEditor* will be launched and the selected device opened.



IMPORTANT: when the *ModbusCustomEditor* is running, *Connection* is blocked waiting for it to be closed.

### 4.4.3 DELETING A MODBUS CUSTOM DEVICE

To delete an existing *Modbus custom* device, when the device is visible in the *Catalog window* do a right-click and choose *Delete from catalog* (see previous paragraph).

### 4.4.4 USING A MODBUS CUSTOM DEVICE

When you insert the *Modbus custom* device as a Modbus slave (for example under a RS485 port) and click on it on the tree, you will see a page with three tabs.

#### 4.4.4.1 GENERAL TAB

### ModbusDevice1 Configuration

General	Input	Output
<p><b>Settings</b></p> <p>Modbus address: <input type="text" value="1"/> (1 .. 247)</p> <p>Node number: <input type="text" value="0"/> (0 .. 127)</p> <p>Polling time: <input type="text" value="0"/> ms (0 = write on variation)</p> <p>TimeOut: <input type="text" value="1000"/> ms</p> <p>Wait before send: <input type="text" value="0"/> ms</p>		

In the *General* tab you can configure:

- Its *Modbus address* (in the range 1..247).
- Its *Node number* (in the range 0..127); this value is incremented automatically, and can be used in the PLC program to index the `SysMbMRtuNodeStatus[]` array, that contains diagnostic information about each slave node.
- *Polling time*: the Modbus messages will be processed with this period (ms); for writing operations, 0 means to write it only on variation of the value, for reading operations 0 means maximum speed.
- *Timeout*: the operation will be unsuccessful when this time-out expires (ms).



## SoMachine HVAC - Connection

- *Wait before send*: this is an additional timeout, to be used with slow slaves that do not answer if the messages are sent too fast.

Here you can notice that for *Modbus custom* the *Polling time*, *Timeout* and *Wait before send* are generic for the whole device, while for the *Generic Modbus* you can put specific different values for each single message. This is because with the *Modbus custom* the low-level Modbus messages are automatically calculated and you do not have to worry about them, but as a side-effect you can not "fine-tune" them, because these settings are global.

### 4.4.4.2 INPUT/OUTPUT TAB

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

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

IMPORTANT: please remember to rebuild the PLC project with Application to see an updated list of PLC variables here.

The screenshot shows the 'ModbusDevice1 Configuration' dialog box with the 'Input' tab selected. The dialog has three tabs: 'General', 'Input', and 'Output'. Below the tabs are buttons for 'Add', 'Remove', 'Assign', 'UnAssign', 'Up', and 'Down'. A table lists the configured Modbus parameters:

Parameter	Address	Type	Variable	Type	DataBlock
temperature1	1	INT			
temperature2	2	INT			

A 'Modbus Parameters' dialog box is open in the foreground, showing a list of available parameters:

- 1 temperature1 (INT)
- 2 temperature2 (INT)
- 3 pressure (UINT)
- 4 digitalInput (BOOL)
- 5 digitalOutput (BOOL)

The 'pressure (UINT)' parameter is selected. The dialog has 'OK' and 'Cancel' buttons at the bottom.

Connection will create the correct Modbus messages analyzing the sequence of addresses and types; if the addresses are consequent and the types are homogenous, different objects will be grouped in single messages to optimize the communication.

The maximum number of registers configured with the ModbusCustomEditor is also considered, along with the maximum number of registers per message of the master (that is 16 for the M171 Perf. Display).

The grouping and generation of the Modbus messages is totally automatic and recalculated at each compilation, so you do not have to know technical details of the Modbus protocol.



## 4.5 CAN CUSTOM

*CAN custom* device can be created and added to the Catalog by importing their *EDS* file. In this way you can use any third-party CANopen device as a slave, if it provides a standard-compliant *EDS* file (Electronic Data Sheet), that follows the DS306 CiA specification.

### 4.5.1 IMPORTING A NEW CAN CUSTOM DEVICE

To import a new *CAN custom* device, choose *Tools / Import from EDS* command. The *Import EDS* window will appear.

**Import EDS**

Source EDS: L:\Test\CANcustom1.eds Choose...

New Name: CANcustom1 1.0

Short Name: CANcustom1\_1p0

Has dynamic PDO mapping:

Vendor Name:

Product Name: CANcustom1 Revision: 0.0

Description: EDS sample device

Comments:

**Number of Objects**

Mandatory: 3 Optional: 9 Manufacturer: 0

**OK** **Cancel**

Here you have to configure:

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

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



## 4.5.2 DELETING A CAN CUSTOM DEVICE

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

## 4.5.3 USING A CAN CUSTOM DEVICE

When you insert a *CAN custom* device as a CANopen slave (for example under a CANopen port) and click on it on the tree, you will see the following page.

### 4.5.3.1 GENERAL TAB

**CANcustom1 Configuration**

General	SDO Set	PDO Tx - Input	PDO Rx - Output
---------	---------	----------------	-----------------

**Network settings**

Node number (1..127)	<input type="text" value="1"/>
Node Guard Period (ms)	<input type="text" value="200"/>
Life time Factor	<input type="text" value="3"/>
Boot time elapsed (ms)	<input type="text" value="0"/>
Node heartbeat producer time (ms)	<input type="text" value="0"/>
Node heartbeat consumer time (ms)	<input type="text" value="0"/>
Master heartbeat consumer time (ms)	<input type="text" value="0"/>
Identity object check	<input checked="" type="checkbox"/>

**PDO Tx communication settings**

USER DEFINED Mode

SYNC Mode

EVENT Mode

CYCLIC Mode  ms

**PDO Rx communication settings**

USER DEFINED Mode

SYNC Mode

EVENT Mode

In the *General* tab you can configure (see 5.1 for further information):

- *Node number (1..127)*.
- *Node guard period* in ms (default 200ms), 0 to disable node guard for this slave; if not zero is the interval of node guarding packets sent by the master to the slave.
- *Life time factor* (default 3x), 0 to disable node guard for this slave; if not zero, multiplied for the *Node guarding period* is the maximum amount of time the master will wait for the slave answer of the node guard.
- *Boot time elapsed*: this is the maximum amount of time in ms that the master will wait for the slave to become pre-operational at boot (default 10s), before signaling an error.
- *Node heartbeat producer time* in ms, default is 0 (heartbeat disabled); if not zero the master will enable the heartbeat error handling check for this node.
- *Node heartbeat consumer time* in ms, default is 0 (heartbeat disabled); it is the maximum amount of time the slave will wait for the heartbeat produced by the master, before timing out. This should be greater than the *Heartbeat time* of the master.



- *Master heartbeat consumer time* in ms, default is 0 (heartbeat disabled); it is the maximum amount of time the master will wait for the heartbeat sent by the slave, before timing out. This should be greater than the *Node heartbeat producer time*.
- *Identity object check*: when this option is enabled (the default) the master at boot will check the slave for his identity, verifying that the *Identity object* fields (object 0x1018) match with EDS default values (Vendor ID, Product code, Revision, Serial); if the option is not enabled, no check will be done (this is useful for example with slaves not totally CANopen-compliant or incorrect EDS files).
- *PDO Tx comm settings*: configure here the transmission mode for PDO Tx; depending on the device features (determined from EDS values), not all options may be available.
- *PDO Rx comm settings*: configure here the transmission mode for PDO Rx; depending on the device features (determined from EDS values), not all options may be available.

### 4.5.3.2 SDO SET TAB

**CANcustom1 Configuration**

General | **SDO Set** | PDO Tx - Input | PDO Rx - Output

Add Remove

#	Label	Index	SubIndex	Type	Value	Timeout
1	Transmission Type	1400	2	USINT	255	100

**Variables List**

Filter:

- 1005.0 COB-ID SYNC message (UDINT)
- 100c.0 Guard Time (UINT)
- 100d.0 Life Time Factor (USINT)
- 1014.0 COB-ID Emergency Message (UDINT)
- 1016.1 Consumer Heartbeat Time 1 (UDINT)
- 1017.0 Producer Heartbeat Time (UINT)
- 1400.1 COB-ID (UDINT)
- 1400.2 Transmission type (USINT)**
- 1600.0 Number of mapped objects (USINT)
- 1600.1 1st mapped Object (UDINT)
- 6200.1 Write Output 1h to 8h (USINT)
- 2.0 Dummy0002 (SINT)
- 3.0 Dummy0003 (INT)
- 4.0 Dummy0004 (DINT)

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

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

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

### 4.5.3.3 PDO TX AND PDO RX TABS

General | SDO Set | **PDO Tx - Input** | PDO Rx - Output

Add Remove Assign UnAssign

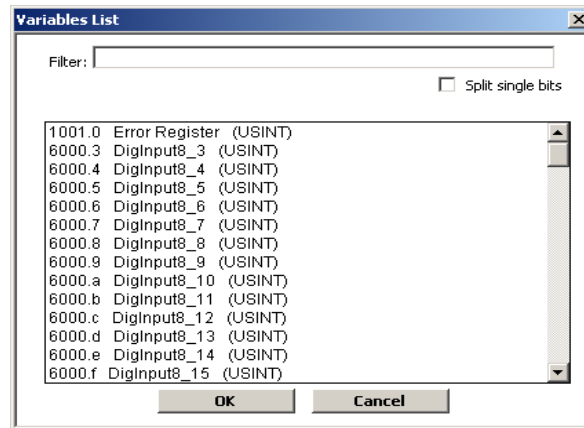
#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	DataBlock
1	6000	1	1	0	0	DigInput8_1	USINT	8		
2	6000	2	1	8	0	DigInput8_2	USINT	8		
3	6401	1	2	0	0	Analogue Input 1	INT	16		
4	6401	2	2	16	0	Analogue Input 2	INT	16		
5	6401	3	2	32	0	Analogue Input 3	INT	16		
6	6401	4	2	48	0	Analogue Input 4	INT	16		



## SoMachine HVAC - Connection

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

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



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

**IMPORTANT:** please note that the PDO mapping configuration you enter here is NOT sent to the device, its only purpose is to match an already configured PDO mapping on the device.

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

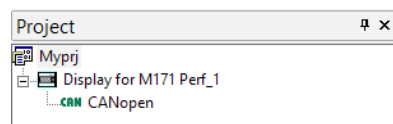
**IMPORTANT:** please remember to rebuild the PLC project with Application to see an updated list of PLC variables here.

### 4.6 DISPLAY FOR M171 PERF.

Display for M171 Perf. is a keyboard with a display. It is used to show HMI Remote pages that are made available by M171 Perf. Display or M171 Perf. Blind devices.

Display for M171 Perf. keyboard has not PLC, HMI and HMI Remote features and it has one on-board CANopen port.

Display for M171 Perf. can maintain on-board no more than one HMI set of remote device pages.



#### 4.6.1 CANOPEN

Display for M171 Perf. can be connected to M171 Perf. Display or to M171 Perf. Blind in field mode or in network mode.

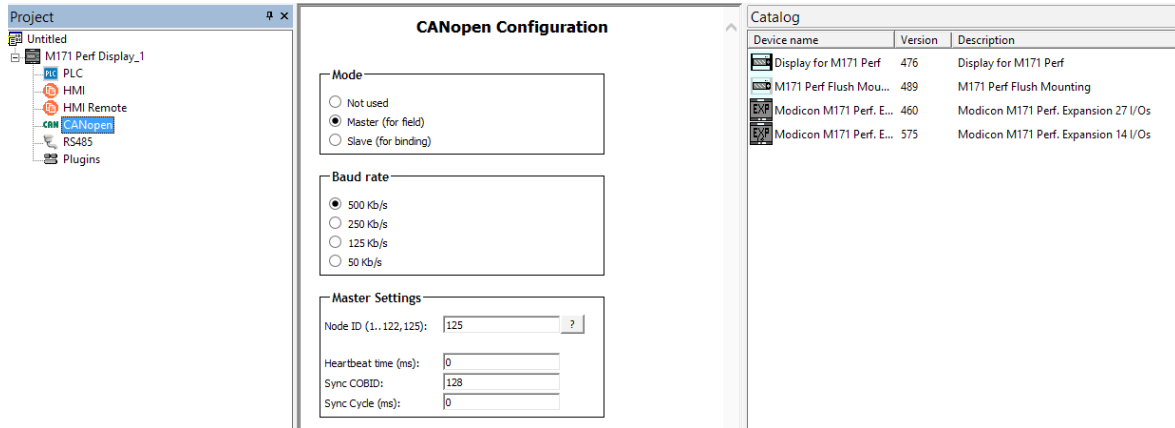


## 4.6.1.1 FIELD MODE

In this connection mode Display for M171 Perf. has to be considered a slave of M171 Perf. Display. Display for M171 Perf. will be able to show only the remote pages of the master device which is linked to.

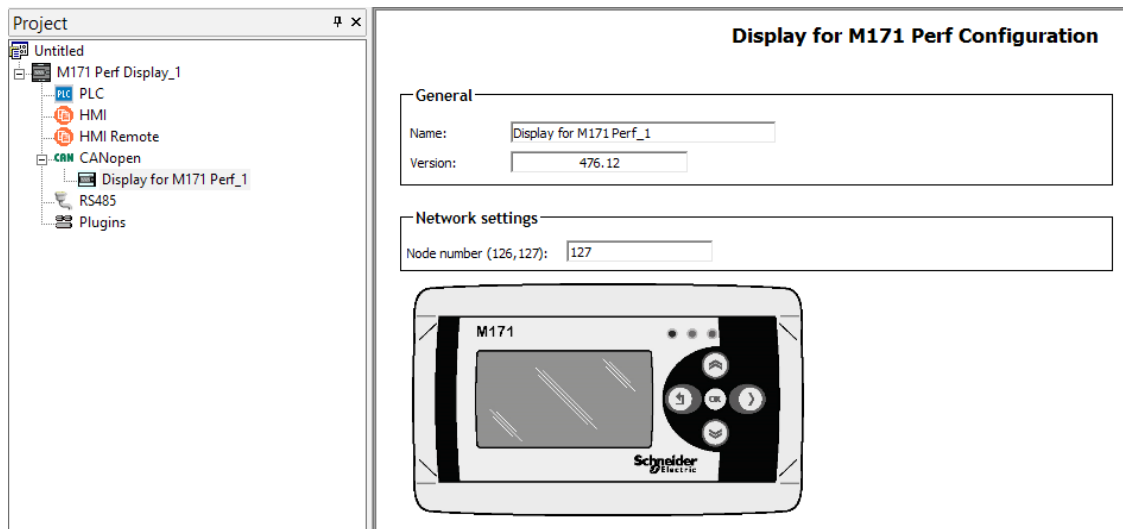
To configure network in this way select M171 Perf. Display or M171 Perf. Blind and add it as first level node; then configure its PLC, HMI, and HMI Remote projects normally.

The project associated to M171 Perf. Display\_1 - HMI Remote node will be shown by Display for M171 Perf. device.



Click on *CANopen* and select *Master* (for field) option in *Mode* tab and configure CANopen settings. Then Display for M171 Perf. device can be selected from *Catalog*, dragged and dropped over CANopen node.

Select Display for M171 Perf.\_1 device child node and adjust network settings.







As resulting output of the compiling process we see the output line:

```
M171 Perf. Display_1: added field CAN keyboard 'Display for M171 Perf._1'  
(with virtual master nodeID 124)
```

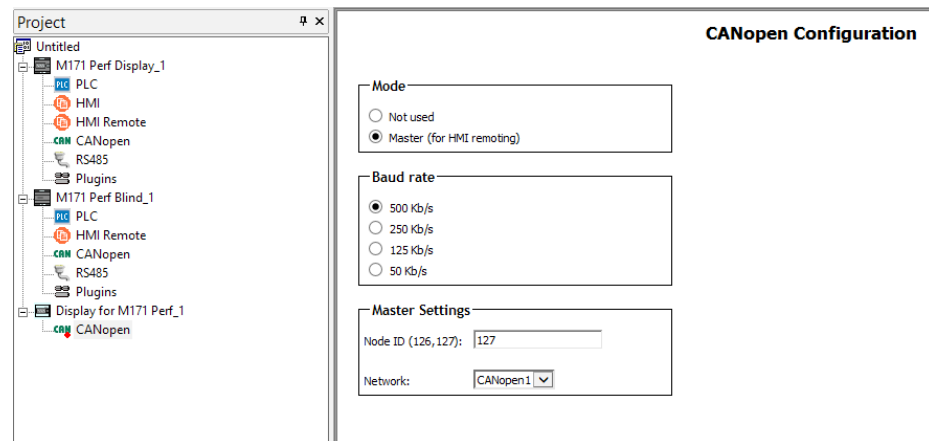
Display for M171 Perf.\_1 device communicates with M171 Perf. Display\_1 using this CAN nodeID. This node ID will be used for navigating remote pages.

## 4.6.1.2 NETWORK MODE

In this connection mode Display for M171 Perf. can be linked to one of the remote devices that are available on the network to navigate HMI Remote pages provided by other devices.

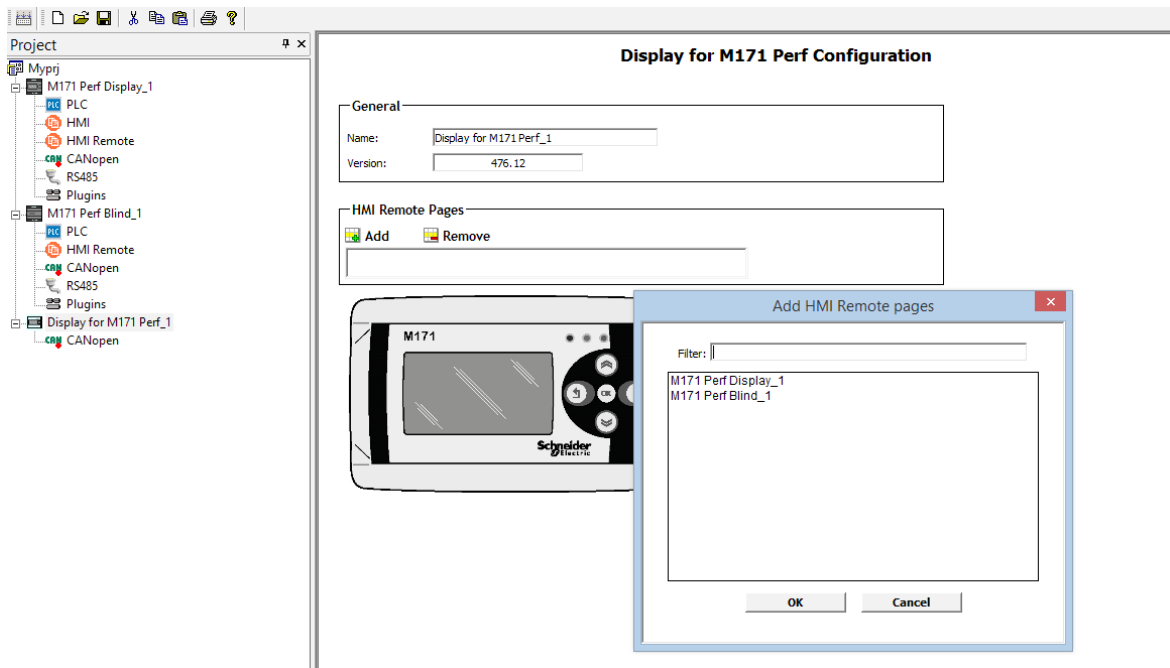
Using Connection it is possible to do so indicating one of the available HMI Remote device of the network. Let's see how with an example.

We have a CANOpen network with M171 Perf. Display\_1 and M171 Perf. Blind\_1, then add as first level node Display for M171 Perf.\_1 to the network taking it from *Catalog* panel. Click on *CANOpen node* of Display for M171 Perf.\_1 and select *Master* (for HMI remoting) node, assign univoque Node ID and select network *CANopen1*.



Once linked to the CANopen1 network it is possible to select the HMI remote pages to navigate with Display for M171 Perf.\_1.

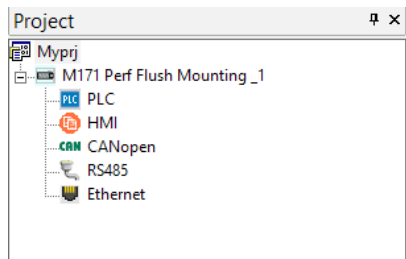
Click on the device node then click on *Add*. Window *Add HMI Remote pages* will be shown. It is possible to select to navigate pages of M171 Perf. Display\_1 or M171 Perf. Blind\_1 because they are on the same network of the keyboard and provide HMI Remote pages.



Click on *M171 Perf. Display\_1*, then click on *OK*. Selected device will be added to HMI Remote Pages. It is not possible to navigate more than one remote device at a time.

## 4.7 M171 PERF. FLUSH MOUNTING

M171 Perf. Flush Mounting is an advanced keyboard with display that can be used to navigate HMI Remote pages and offers more connectivity (CANopen, RS485, Ethernet) than Display for M171 Perf.. It can also run PLC and local HMI pages and it is also provided with probes.



### 4.7.1 PLC

M171 Perf. Flush Mounting can run PLC. This configuration step can be done in the same way of M171 Perf. Display and is fully described in section 4.1.1 - PLC.

### 4.7.2 HMI

M171 Perf. Flush Mounting can run local HMI project with its own pages. This configuration step can be done in the same way of M171 Perf. Display and is fully described in section 4.1.2 - HMI.



## 4.7.3 PROVIDING HMI PAGES

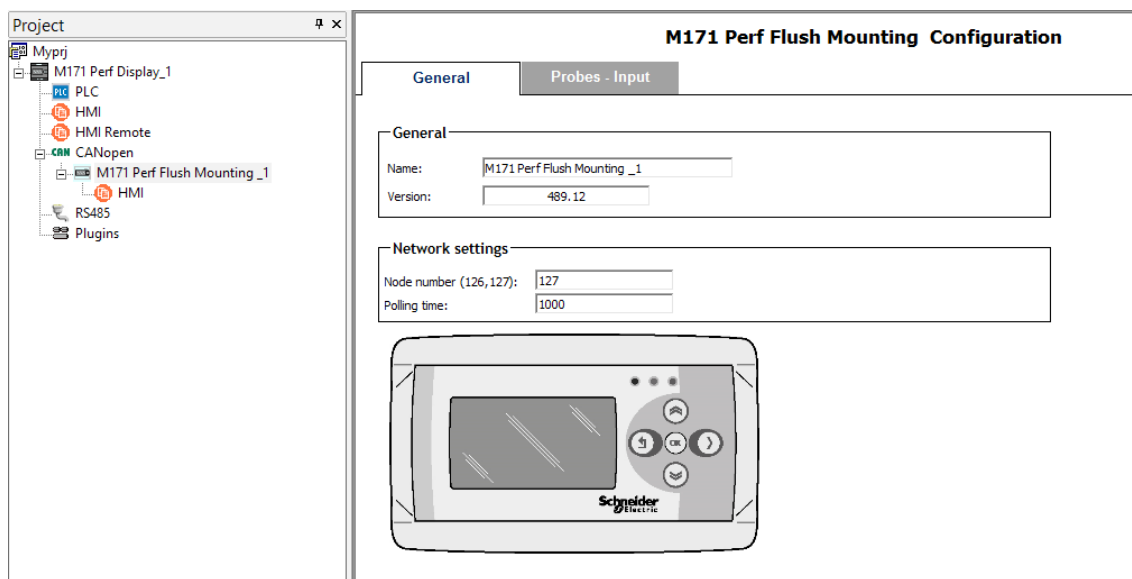
This feature is not supported by M171 Perf. Flush Mounting. No linked device can upload HMI pages from M171 Perf. Flush Mounting device.

## 4.7.4 CANOPEN

M171 Perf. Flush Mounting can be connected using CANopen in field mode or in network mode.

### 4.7.4.1 FIELD MODE

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



Select *M171 Perf. Flush Mounting\_1* child node and configure *Network* settings.

### Probes

M171 Perf. Display\_1 can access on board M171 Perf. Flush Mounting\_1 on-board probes. To do so select *Probes-Input* tab then it is possible to map a M171 Perf. Display\_1 parameter to let it obtain the value of an on-board M171 Perf. Displayprobe.

Choose one of the probe and click on *Assign* button. Take one of the M171 Perf. Display\_1 INT parameter and click *OK* button.



**M171 Perf Flush Mounting Configuration**

General		Probes - Input							
<span style="color: orange;">➤</span> Assign <span style="color: orange;">➤</span> UnAssign									
#	Idx	Sub	PDO	COBID	Object Name	Type	Size	Label	DataBlock
1	2090	0	1	0	AIL1 - Analogue input 1	INT	16		
2	2091	0	1	0	AIL2 - Analogue input 2	INT	16		
3	2092	0	1	0	AIL3 - Analogue input 3	INT	16		
4	2093	0	1	0	AIL4 - Analogue input 4	INT	16		

## HMI

It is possible to associate to a M171 Perf. Flush Mounting (configured as CANopen field slave) an HMI project with local pages. M171 Perf. Flush Mounting would be able to show its own target variables and parameters of the master CANopen which belongs to.

### 4.7.4.2 NETWORK MODE

In this connection mode M171 Perf. Flush Mounting can be linked to one of the remote devices that are available on the network to navigate HMI Remote pages provided by other devices.

Using Connection it is possible to do so by indicating one of the available HMI Remote device of the network. Let's see how with an example.

We have a CANopen network with M171 Perf. Display\_1 and M171 Perf. Blind\_1 then add as first level node *M171 Perf. Flush Mounting\_1* to the network taking it from *Catalog* panel.

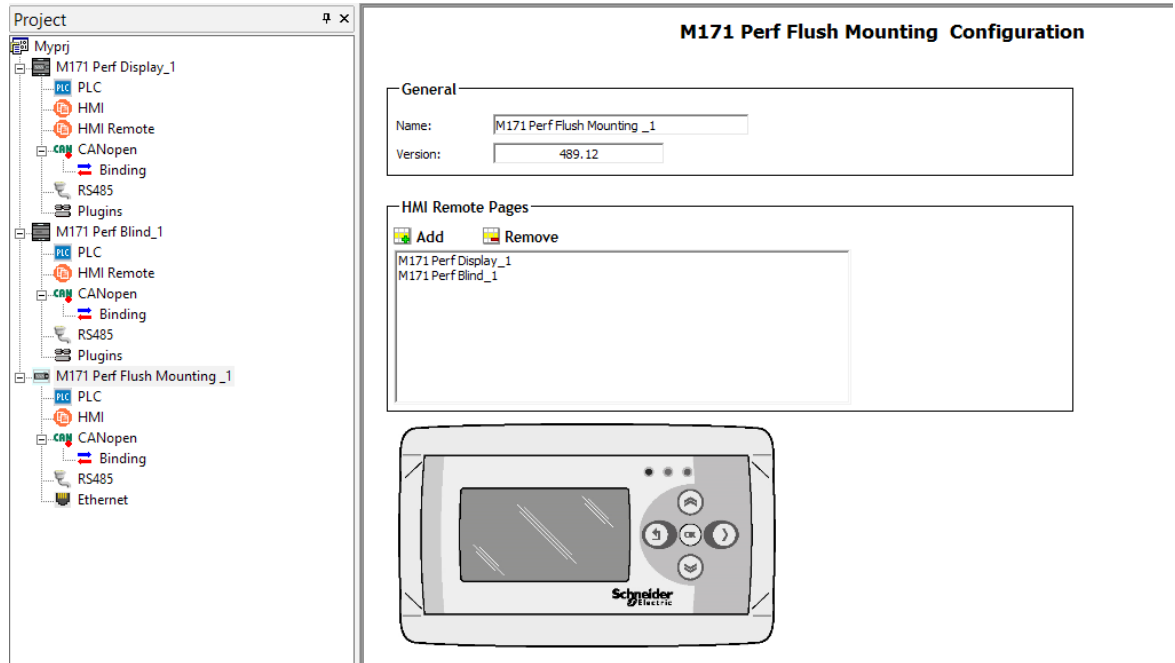
Click on *CANOpen* node of M171 Perf. Flush Mounting\_1 and select *Master* (for HMI remot-ing and binding) node, assign univoque *Node ID* and select network *CANOpen1*.

Binding of variables between M171 Perf. Flush Mounting\_1 and M171 Perf. Blind\_1 and M171 Perf. Display\_1 is allowed in a network of this type (see 4.1.3.2 for more details).



## HMI Remote pages

In CANopen network mode it is possible to configure M171 Perf. Flush Mounting in order to be linked to 0 to 10 remote devices that can provide HMI Remote pages to the keyboard.



To add HMI Remote pages select *M171 Perf. Flush Mounting\_1* node, then press *Add* on the *HMI Remote pages* box thus all available devices will be shown and the user can select the pages to navigate.

### 4.7.5 RS485

The usage of this communication feature is the same of M171 Perf. Display (see 4.1.4 - RS485 paragraph).

### 4.7.6 ETHERNET

M171 Perf. Flush Mounting is provided with on-board Ethernet. Ethernet configuration and features for this kind of device is similar to the configuration of the Ethernet plugin of M171 Perf. Display (see 4.1.5 - Ethernet).

## 4.8 MODICON M171 PERF. EXPANSION 27 I/OS

Modicon M171 Perf. Expansion 27 I/Os is a device that can be linked in a CANopen field or Modbus RTU field network whose master can be a M171 Perf. Display, a M171 Perf. Blind or a M171 Perf. Flush Mounting device.

Modicon M171 Perf. Expansion 27 I/Os main feature is to provide a lot of I/O signal to its field master device. I/O signals mapping can be configured by using Connection.



### 4.8.1 USING MODICON M171 PERF. EXPANSION 27 I/OS AS CANOPEN FIELD SLAVE

In this configuration sample we want to use Modicon M171 Perf. Expansion 27 I/Os as expansion of a M171 Perf. Display device. The same can be done for M171 Perf. Blind and M171 Perf. Flush Mounting.

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

#	Idx	Sub	PDO	Bit	COBID	Object Name	Type	Size	Label	DataBlock
1	6000	1	1	0	0	Read Input 1h to 8h	BOOL	1		
2	6000	1	1	1	0	Read Input 1h to 8h	BOOL	1		
3	6000	1	1	2	0	Read Input 1h to 8h	BOOL	1		
4	6000	1	1	3	0	Read Input 1h to 8h	BOOL	1		
5	6000	1	1	4	0	Read Input 1h to 8h	BOOL	1		
6	6000	1	1	5	0	Read Input 1h to 8h	BOOL	1		
7	6000	1	1	6	0	Read Input 1h to 8h	BOOL	1		
8	6000	1	1	7	0	Read Input 1h to 8h	BOOL	1		
9	6000	2	1	8	0	Read Input 9h to 16h	BOOL	1		
10	6000	2	1	9	0	Read Input 9h to 16h	BOOL	1		
11	6000	2	1	10	0	Read Input 9h to 16h	BOOL	1		
12	6000	2	1	11	0	Read Input 9h to 16h	BOOL	1		
13	6401	1	2	0	0	Analogue Input 1	INT	16		
14	6401	2	2	16	0	Analogue Input 2	INT	16		
15	6401	3	2	32	0	Analogue Input 3	INT	16		
16	6401	4	2	48	0	Analogue Input 4	INT	16		
17	6401	5	3	0	0	Analogue Input 5	INT	16		
18	6401	6	3	16	0	Analogue Input 6	INT	16		
19	2230	0	5	0	0	Counter	UDINT	32		
20	2232	0	5	32	0	Frequency	UDINT	32		

Modicon M171 Perf. Expansion 27 I/Os configuration is quite similar to CAN Custom configuration (see 4.5.3 - Using a CAN custom device) with dynamic PDO mapping feature disabled. Available Input/Output objects that can be mapped on M171 Perf. Display PLC variables via PDO are listed in *PDO TX-Input* and *PDO RX-Output*.

Connection knows the Modicon M171 Perf. Expansion 27 I/Os dictionary. Each object can be here linked to M171 Perf. Display\_1 PLC variable as it has been done in the above figure for *Analogue Input 1* signal.

### 4.8.2 USING MODICON M171 PERF. EXPANSION 27 I/OS AS RS485 FIELD SLAVE

In this configuration sample we want to use Modicon M171 Perf. Expansion 27 I/Os as expansion of a M171 Perf. Display device. The same can be done for M171 Perf. Blind and M171 Perf. Flush Mounting.

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



Parameter	Address	Type	Variable	Type	DataBlock
DIL1	1	BOOL			
DIL2	2	BOOL			
DIL3	3	BOOL			
DIL4	4	BOOL			
DIL5	5	BOOL			
DIL6	6	BOOL			
DIL7	7	BOOL			
DIL8	8	BOOL			
SW1	9	BOOL			
SW2	10	BOOL			
SW3	11	BOOL			
SW4	12	BOOL			
AIL1	8336	INT			
AIL2	8337	INT			
AIL3	8338	INT			
AIL4	8339	INT			
AIL5	8340	INT			
AIL6	8341	INT			
Counter	8752	UDINT			
Frequency	8754	UDINT			

Modicon M171 Perf. Expansion 27 I/Os\_1 configuration is quite similar to a Modbus Custom device configuration (see 4.4.4 - Using a Modbus custom device) in which it is possible to assign available Modicon M171 Perf. Expansion 27 I/Os dictionary I/O objects to M171 Perf. Display\_1 PLC variables.

Connection knows the Modicon M171 Perf. Expansion 27 I/Os dictionary. *Input* and *Output* objects can be added, removed, assigned, unassigned or changed in position. Only assigned objects will be requested by M171 Perf. Display\_1 device.

## 4.9 VIRTUAL CHANNELS ASSIGNMENT CRITERIA

This paragraph concerns the criteria used by Connection to assign virtual node IDs due to the network configuration.

### 4.9.1 CANOPEN NETWORK - VIRTUAL SDO SERVERS

When CANopen is in use on a M171 Perf. Display or M171 Perf. Blind device in *slave* mode (network for binding) three SDO servers are activated on it.

First SDO server is used to process requests that arrives to its physical node ID (the ID assigned by user in the configuration box). Supervisor PC should be connected using this node ID. CANopen physical node ID *addr* must be chosen in a range between 1 to 42.

Two other virtual SDO servers are opened on this device and are dedicated to the communication with keyboards (max 2 for each CANopen network). So the device is able to process requests addressed to these node IDs.

Virtual SDO servers node IDs are calculated with this criteria:

$$ch\_1 = 124 - 2 * ( addr - 1 )$$

$$ch\_2 = 123 - 2 * ( addr - 1 )$$

The first keyboard on the network communicates to the destination M171 Perf. Display device using *ch\_1*, channel *ch\_2* is dedicated to the second.

Example:

$$addr = 1 \rightarrow ch\_1 = 124, \quad ch\_2 = 123$$

$$addr = 2 \rightarrow ch\_1 = 122, \quad ch\_2 = 121$$



## 4.9.2 ETHERNET - TCP SLAVE CHANNELS

If Ethernet network communication is enabled on a M171 Perf. Display or M171 Perf. Blind device two TCP slave channels are always opened to support the communication with keyboards.

## 4.9.3 CANOPEN FIELD - VIRTUAL MASTER CHANNELS

When CANopen is in use on a M171 Perf. Display or M171 Perf. Blind device in *master* mode (field) three master channels are opened.

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

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

Virtual master node IDs have fixed values :

`ch_1` = 123

`ch_2` = 124





## 5. TECHNICAL REFERENCE

### 5.1 CANOPEN PROTOCOL

#### 5.1.1 OVERVIEW

CANopen realizes a communication model using the serial bus network Controller Area Network (CAN).

Developed originally for passenger cars, the CAN two-wire bus system is already in use in over one million industrial control devices, sensors and actuators.

CiA (CANopen in Automation) maintains the CANopen specifications, including device profiles for I/O modules (CiA DS-401), for electric drive systems (CiA DSP-402) and many more. The process of defining new profiles is continually performed. An independent test and certification process is available at CiA.

A number of CANopen implementations (OEM code) and many CANopen products are already available. CiA regularly publishes an up-to-date catalog of CANopen products and of certified ones.

#### 5.1.2 PHYSICAL STRUCTURE OF A CANOPEN NETWORK

The underlying CAN architecture defines the basic physical structure of the CANopen network. Therefore, a line (bus) topology is used; to avoid reflections of the signals, both ends of the network must be terminated. In addition, the maximum permissible branch line lengths for connection of the individual network nodes must be observed.

Additionally, for CANopen, two additional conditions must be fulfilled:

- all nodes must be configured to the same bit rate and
- no node-ID may exist twice.

Unfortunately there are no mechanisms automatically ensuring these conditions. The system integrator has to check the bit rate and node-ID of every single network node when wiring a network and adjust if necessary.

#### 5.1.3 COB AND COB-ID

CANbus, the physical layer of CANopen, can transmit short packages of data (called COB, Communication Object), that have a 11-bit ID or 29-bit ID (in version CAN 2.0 B); this ID of a CAN-frame is known as Communication Object Identifier, or COB-ID. In case of a transmission collision, the bus arbitration used in the CANbus allows the frame with the smallest ID to be transmitted first and without a delay. Thus giving a low code number for time critical functions helps ensure the lowest possible delay.

#### 5.1.4 THE OBJECT DICTIONARY

All device parameters are stored in an object dictionary. This object dictionary contains the description, data type and structure of the parameters as well as the address from others point of view. The address is being composed of a 16 bit index and a 8 bit sub-index; the sub-index refers to the elements of complex data types, like arrays and records.

There are a range of mandatory entries in the dictionary which helps ensure that all CANopen devices of a particular type show the same behavior. The object dictionary concept caters for optional device features which means a manufacturer does not have to provide certain extended functionality on his device, but if he wishes to do so he has to do it in a pre-defined fashion. Additionally, there is sufficient address space for truly manufacturer specific functionality.



## 5.1.5 THE SERVICE DATA OBJECTS (SDO)

Service Data Messages, in CANopen called Service Data Objects (SDO), are used for read and write access to all entries of the object dictionary of a device. Main usage of this type of messages is the device configuration; SDOs are typically transmitted asynchronously. The requirements towards transmission speed are not as high as for PDOs; the SDO message contains information to address data in the device object dictionary and the data itself.

## 5.1.6 THE PROCESS DATA OBJECTS (PDO)

Process Data Messages, in CANopen called Process Data Objects (PDO), are used to perform the real-time data transfer between different automation units. PDOs have to be transmitted quickly, without any protocol overhead and within a predefined structure.

The contents of the PDO is encoded in the PDO mapping entries. A PDO can contain up to 8 bytes or 64 single data elements from the object dictionary (in the case of 64, that are bit data); the data are described via its index, sub-index and length. The mapping parameter of a PDO resides also in the object dictionary.

The mapping for the PDO can be static or changeable. If the mapping can be changed, it is called dynamic PDO mapping; changing of mapping can be done in the state pre-operational (default) or operational.

## 5.1.7 PDO TRANSMISSION MODES

For the PDOs different transmission modes are distinguished:

- SYNC: PDO are transmitted according to the SYNC clock transmitted by the master.
- EVENT: PDO are transmitted when the value changes (asynchronous).
- CYCLIC: PDO transmission is periodic and timer-based.
- RTR: PDO are transmitted only on master request.

The communication parameters of a PDO reside in the object dictionary. The indices for PDOs are built like follow:

- PDO Tx:  $0x1800 + \text{PDO number}$ .
- PDO Rx:  $0x1400 + \text{PDO number}$ .

The range of the PDO numbers is 1..512. that means up to 512 receive PDOs (RPDO) and up to 512 transmit PDOs (TPDO) are possible for a device.

The communication parameter of PDOs are described with a structure: only sub-index 1 and 2 are mandatory.

Subindex 1 describes the used COB-ID of the PDO: a PDO communication channel between two devices is created by setting the TPDO COB-ID of the first device to the RPDO COB-ID of the second device. For PDOs a 1:1 and a 1:n communication is possible: that means there is always only one transmitter, but an unlimited number of receivers.

The transmission type (sub-index 2) describes the kind of transmission; transmission type 1 means PDO will be triggered with each SYNC Object. If this entry has the value 240, the PDO will be sent/received with each 240th SYNC. If the entry is 255, the transmission is EVENT or CYCLIC, depending on the event timer (see below).

The optional entry inhibit time (sub-index 3) defines a minimum time period between two PDO transmissions. This feature helps ensure that messages with lower priorities than the actual PDO can be transmitted in the case of continuous transmission of the actual PDO.

The optional entry event timer (sub-index 5) is only relevant for asynchronous Transmit PDOs: if this value is greater then zero, indicates the time to elapse for the CYCLIC; otherwise means EVENT (on variation).



### 5.1.8 THE EMERGENCY OBJECT

The Emergency Message (EMCY) is a service which signs internal device errors.

The EMCY is transmitted with highest priority; CANopen defines EMCY-Server and EMCY-Clients, the server transmits EMCYs and the clients receive them.

The EMCY telegram consists of 8 bytes: it contains an emergency error code, the contents of object and 5 byte of manufacturer specific error code.

### 5.1.9 SYNC OBJECT AND TIME STAMP OBJECT

The SYNC Object is a network wide system clock. It is the trigger for synchronous message transmission; the SYNC has a very high priority and contains no data in order to help minimize jitter. The SYNC COB-ID is by default 128, but can be configured.

The Time Stamp Object provides a common time reference; it is transmitted with high priority.

### 5.1.10 ERROR CONTROL: NODE GUARDING

The Node Guarding is the periodical monitoring of certain network nodes; each node can be checked by the master with a certain period called "Node guard period". If the node does not answer after a time calculated as the guard period x "Life time factor", the connection should be considered lost.

This feature is enabled for a slave when both parameters are not zero; please note that when it is enabled it has a big impact on network load.

### 5.1.11 ERROR CONTROL: HEARTBEAT

The Heartbeat is an error control service without need for remote frames: the Heartbeat producer transmits periodically a heartbeat message; one or more heartbeat consumer receive this message and monitor this indication.

Each heartbeat producer can use a certain period (heartbeat producer time); the heartbeat starts immediately if the heartbeat producer time is zero.

The heartbeat consumer has to monitor the heartbeat producer; it has an entry for each heartbeat producer in its own object dictionary. The heartbeat consumer time can be different for each heartbeat producer but should be greater than the heartbeat producer time.

Heartbeat has a big impact on network load, but in practice the half of the load of the node guarding.

### 5.1.12 THE NETWORK BEHAVIOR

Devices have four operative states: the *initialization*, the *pre-operational*, the *stopped* and the *operational* one; the difference between master and slave devices is the initiation of the state transitions.

The master controls the state transitions of each device in the network: after power-on a device goes in the *initialization*, and then in the *pre-operational* automatically; in this state reading and writing to its object dictionary via the service data object (SDO) is possible. Therefore the device can now be configured: this means setting of objects or changing of default values in the object dictionary like preparing PDO transmission.

Afterwards the device can be switched into the *operational* state via the command *Start Remote Node* in order to start PDO communication. PDO communication can be stopped by the network master by simply switching the remote node back to pre-operational by using the *Enter Pre-Operational State* command.



Via the *Stop Remote Node* command the master can force the slave(s) to the *stopped* state. In this state no services besides network and error control mechanism are available.

The command *Reset Communication* resets the communication on the node: all communication parameters will be set to their defaults.

The application will be reset by *Reset Node* command, that resets all application parameter and then calls *Reset Communication* command.

## 5.1.13 THE BOOT-UP MESSAGE

After a CANopen node has finished its own initialization and entered in the node state *pre-operational* it has to send the Boot-up Protocol Message; this message indicated that the slave is ready for work (e.g. configuration).

The master can wait for this message up to *Boot time elapsed* ms.

## 5.1.14 THE CANOPEN DEVICE PROFILES

A device profile defines a standard kind of device: for these standard devices a basic functionality has been specified, that every device has to implement. The CANopen Device Profiles help to ensure a minimum of identical behavior for the same kind of device in order to provide the greatest degree of interoperability and vendor independence possible.

Each device has to fulfill the requirements on the behavior; furthermore it has to support all mandatory objects: these objects are parameter and data for the device.

Additionally the manufacturer can decide about supported optional objects; all parameters and data, which are not covered by the standardized device profiles can be realized as manufacturer specific objects.

For example, two of the most commonly used Device Profiles are DS401 (Generic I/O Modules) and DS402 (Drives and Motion Control).

## 5.2 MODBUS PROTOCOL

### 5.2.1 OVERVIEW

Modbus is a serial communication protocol. In simple terms, it is a method used for transmitting information over serial lines between electronic devices. The device requesting the information is called the Modbus Master and the devices supplying information are Modbus Slaves. In a standard Modbus network, there is one Master and up to 247 Slaves, each with a unique Slave Address from 1 to 247; the Master can also write information to the Slaves.

Address 0 is used as broadcast address.

### 5.2.2 DATA TYPES

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

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

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



### 5.2.3 FUNCTION CODES

The Modbus protocol specifies different "function codes" for each Modbus message:

- 01 (0x01): Read Discrete Output Coils.
- 05 (0x05): Write single Discrete Output Coil.
- 15 (0x0F): Write multiple Discrete Output Coils.
- 02 (0x02): Read Discrete Input Contacts.
- 04 (0x04): Read Analog Input Registers.
- 03 (0x03): Read Analog Output Holding Registers.
- 06 (0x06): Write single Analog Output Holding Register.
- 16 (0x10): Write multiple Analog Output Holding Registers.

### 5.2.4 ERROR DETECTION AND CRC

CRC stands for Cyclic Redundancy check: it is two bytes added to the end of every Modbus message for error detection. Every byte in the message is used to calculate the CRC. The receiving device also calculates the CRC and compares it to the CRC from the sending device: if even one bit in the message is received incorrectly, the CRCs will be different and an error will result.

### 5.2.5 PROTOCOL VERSIONS

Versions of the Modbus protocol exist for serial port and for Ethernet and other networks that support the Internet protocol suite. There are many variants of Modbus protocols:

- Modbus RTU: This is used in serial communication (RS232 or RS485) and makes use of a compact, binary representation of the data for protocol communication. The RTU format follows the commands/data with a cyclic redundancy check checksum as an error check mechanism helps ensure the reliability of data. Modbus RTU is the most common implementation available for Modbus. A Modbus RTU message must be transmitted continuously without inter-character delay. Modbus messages are framed (separated) by idle (silent) periods.
- Modbus ASCII: This is used in serial communication and makes use of ASCII characters for protocol communication. The ASCII format uses a longitudinal redundancy check checksum. Modbus ASCII messages are framed by leading colon (':') and trailing new-line (CR/LF).
- Modbus TCP: This is a Modbus variant used for communications over TCP/IP networks. It does not require a checksum calculation as lower layer takes care of the same.





## Contents

<b>1.</b>	<b>Overview</b>	67
1.1	The workspace	67
1.1.1	The output window	68
1.1.2	The status bar	68
1.1.3	The document bar	68
1.1.4	The watch window	69
1.1.5	The library window	69
1.1.6	The workspace window	71
1.1.7	The source code editors	72
<b>2.</b>	<b>Using the environment</b>	73
2.1	Layout customization	73
2.2	Toolbars	73
2.2.1	Showing/hiding toolbars	73
2.2.2	Moving toolbars	73
2.3	Docking windows	75
2.3.1	Showing/hiding tool windows	75
2.3.2	Moving tool windows	76
2.4	Working with windows	77
2.4.1	The document bar	77
2.4.2	The window menu	78
2.5	Full screen mode	78
2.6	Environment options	79
<b>3.</b>	<b>Managing projects</b>	83
3.1	Creating a new project	83
3.2	Uploading the project from the target device	83
3.3	Saving the project	85
3.3.1	Persisting changes to the project	85
3.3.2	Saving to an alternative location	85
3.4	Managing existing projects	85
3.4.1	Opening an existing Application project	85
3.4.2	Editing the project	85
3.4.3	Closing the project	86
3.5	Distributing projects	86
3.6	Project options	87
3.6.1	Project info	87
3.6.2	Code generation	87
3.6.3	Build output	88



3.6.4	Download	89
3.6.5	Debug	89
3.6.6	Build events	90
3.7	Selecting the target device	90
3.8	Working with libraries	91
3.8.1	The library manager	91
3.8.2	Exporting to a library	92
3.8.3	Importing from a library or another source	93
3.8.4	Updating existing libraries	94
<b>4.</b>	<b>Managing project elements</b>	<b>95</b>
4.1	Program Organization Units	95
4.1.1	Creating a new Program Organization Unit	95
4.1.2	Editing POUs	96
4.1.3	Deleting POUs	98
4.1.4	Source code encryption	98
4.2	Variables	99
4.2.1	Global variables	99
4.2.2	Local variables	105
4.2.3	Create multiple	106
4.3	Tasks	107
4.3.1	Assigning a program to a task	107
4.3.2	Task configuration	108
4.4	Derived data types	108
4.4.1	Typedefs	108
4.4.2	Structures	110
4.4.3	Enumerations	112
4.4.4	Subranges	113
4.5	Browsing the project	115
4.5.1	object browser	115
4.5.2	Searching with the Find in project command	124
4.6	Working with Application extensions	126
<b>5.</b>	<b>Editing the source code</b>	<b>127</b>
5.1	Instruction List (IL) editor	127
5.1.1	Editing functions	127
5.1.2	Reference to PLC objects	127
5.1.3	Automatic error location	128
5.1.4	Bookmarks	128
5.2	Structured Text (ST) Editor	128
5.2.1	Creating and editing ST objects	128
5.2.2	Editing functions	128





5.2.3	Reference to PLC objects	129
5.2.4	Automatic error location	129
5.2.5	Bookmarks	129
<b>5.3</b>	<b>Ladder Diagram (LD) editor</b>	<b>129</b>
5.3.1	Creating a new LD document	130
5.3.2	Adding/Removing networks	130
5.3.3	Labeling networks	130
5.3.4	Inserting contacts	131
5.3.5	Inserting coils	132
5.3.6	Inserting blocks	132
5.3.7	Editing coils and contacts properties	132
5.3.8	Editing networks	133
5.3.9	Modifying properties of blocks	133
5.3.10	Getting information on a block	133
5.3.11	Automatic error retrieval	133
<b>5.4</b>	<b>Function Block Diagram (FBD) editor</b>	<b>134</b>
5.4.1	Creating a new FBD document	134
5.4.2	Adding/Removing networks	134
5.4.3	Labeling networks	134
5.4.4	Inserting and connecting blocks	135
5.4.5	Editing networks	136
5.4.6	Modifying properties of blocks	136
5.4.7	Getting information on a block	136
5.4.8	Automatic error retrieval	136
<b>5.5</b>	<b>Sequential Function Chart (SFC) Editor</b>	<b>137</b>
5.5.1	Creating a new SFC document	137
5.5.2	Inserting a new SFC element	137
5.5.3	Connecting SFC elements	137
5.5.4	Assigning an action to a step	137
5.5.5	Specifying a constant/a variable as the condition of a transition	139
5.5.6	Assigning conditional code to a transition	139
5.5.7	Specifying the destination of a jump	141
5.5.8	Editing SFC networks	141
<b>5.6</b>	<b>Variables editor</b>	<b>141</b>
5.6.1	Opening a variables editor	142
5.6.2	Creating a new variable	143
5.6.3	Editing variables	143
5.6.4	Deleting variables	146
5.6.5	Sorting variables	147
5.6.6	Copying variables	147
<b>6.</b>	<b>Compiling</b>	<b>149</b>
6.1	Compiling the project	149



6.1.1	Image file loading	149
6.2	Compiler output	150
6.2.1	Compiler errors	150
6.3	Command-line compiler	152
<b>7.</b>	<b>Launching the application</b>	<b>153</b>
7.1	Setting up the communication	153
7.1.1	Saving the last used communication port	155
7.2	On-line status	155
7.2.1	Connection status	155
7.2.2	Application status	155
7.3	Downloading the application	156
7.3.1	Controlling source code download	156
7.4	Simulation	158
7.5	Control the PLC execution	159
7.5.1	Halt	159
7.5.2	Cold restart	159
7.5.3	Warm restart	160
7.5.4	Hot restart	160
7.5.5	Reboot target	161
<b>8.</b>	<b>Debugging</b>	<b>163</b>
8.1	Watch window	163
8.1.1	Opening and closing the watch window	163
8.1.2	Adding items to the watch window	164
8.1.3	Removing a variable	167
8.1.4	Refreshment of values	167
8.1.5	Changing the format of data	168
8.1.6	Working with watch lists	169
8.1.7	Autosave watch list	171
8.2	Oscilloscope	171
8.2.1	Opening and closing the oscilloscope	172
8.2.2	Adding items to the oscilloscope	173
8.2.3	Removing a variable	175
8.2.4	Variables sampling	175
8.2.5	Controlling data acquisition and display	176
8.2.6	Changing the polling rate	182
8.2.7	Saving and printing the graph	183
8.3	Edit and debug mode	184
8.4	Live debug	185
8.4.1	SFC animation	186



8.4.2	LD animation	186
8.4.3	FBD animation	187
8.4.4	IL and ST animation	187
<b>8.5</b>	<b>Triggers</b>	<b>187</b>
8.5.1	Trigger window	187
8.5.2	Debugging with trigger windows	193
<b>8.6</b>	<b>Graphic triggers</b>	<b>204</b>
8.6.1	Graphic trigger window	204
8.6.2	Debugging with the graphic trigger window	210
<b>9.</b>	<b>Application reference</b>	<b>221</b>
<b>9.1</b>	<b>Menus reference</b>	<b>221</b>
9.1.1	File menu	221
9.1.2	Edit menu	222
9.1.3	View menu	222
9.1.4	Project menu	223
9.1.5	Debug menu	224
9.1.6	On-line menu	224
9.1.7	Scheme menu	225
9.1.8	Variables menu	226
9.1.9	Window menu	226
9.1.10	Help menu	227
<b>9.2</b>	<b>Toolbars reference</b>	<b>227</b>
9.2.1	Main toolbar	227
9.2.2	FBD toolbar	228
9.2.3	LD toolbar	229
9.2.4	SFC toolbar	230
9.2.5	Project toolbar	231
9.2.6	Network toolbar	232
9.2.7	Debug toolbar	232
<b>10.</b>	<b>Language reference</b>	<b>235</b>
<b>10.1</b>	<b>Common elements</b>	<b>235</b>
10.1.1	Basic elements	235
10.1.2	Elementary data types	235
10.1.3	Derived data types	236
10.1.4	Literals	238
10.1.5	Variables	239
10.1.6	Program Organization Units	242
10.1.7	IEC 61131-3 standard functions	245
<b>10.2</b>	<b>Instruction List (IL)</b>	<b>258</b>
10.2.1	Syntax and semantics	258
10.2.2	Standard operators	260



10.2.3	Calling Functions and Function blocks	260
<b>10.3</b>	<b>Function Block Diagram (FBD)</b>	<b>261</b>
10.3.1	Representation of lines and blocks	261
10.3.2	Direction of flow in networks	262
10.3.3	Evaluation of networks	262
10.3.4	Execution control elements	263
<b>10.4</b>	<b>Ladder Diagram (LD)</b>	<b>265</b>
10.4.1	Power rails	265
10.4.2	Link elements and states	265
10.4.3	Contacts	266
10.4.4	Coils	267
10.4.5	Operators, functions and function blocks	268
<b>10.5</b>	<b>Structured Text (ST)</b>	<b>268</b>
10.5.1	Expressions	268
10.5.2	Statements in ST	269
<b>10.6</b>	<b>Sequential Function Chart (SFC)</b>	<b>274</b>
10.6.1	Steps	274
10.6.2	Transitions	276
10.6.3	Rules of evolution	277
<b>10.7</b>	<b>Application Language Extensions</b>	<b>279</b>
10.7.1	Macros	279
10.7.2	Pointers	280
10.7.3	Waiting statement	281
<b>11.</b>	<b>ERRORS REFERENCE</b>	<b>283</b>
11.1	Compile time error messages	283



## 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](http://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.<sup>(1)</sup>
- 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. OVERVIEW

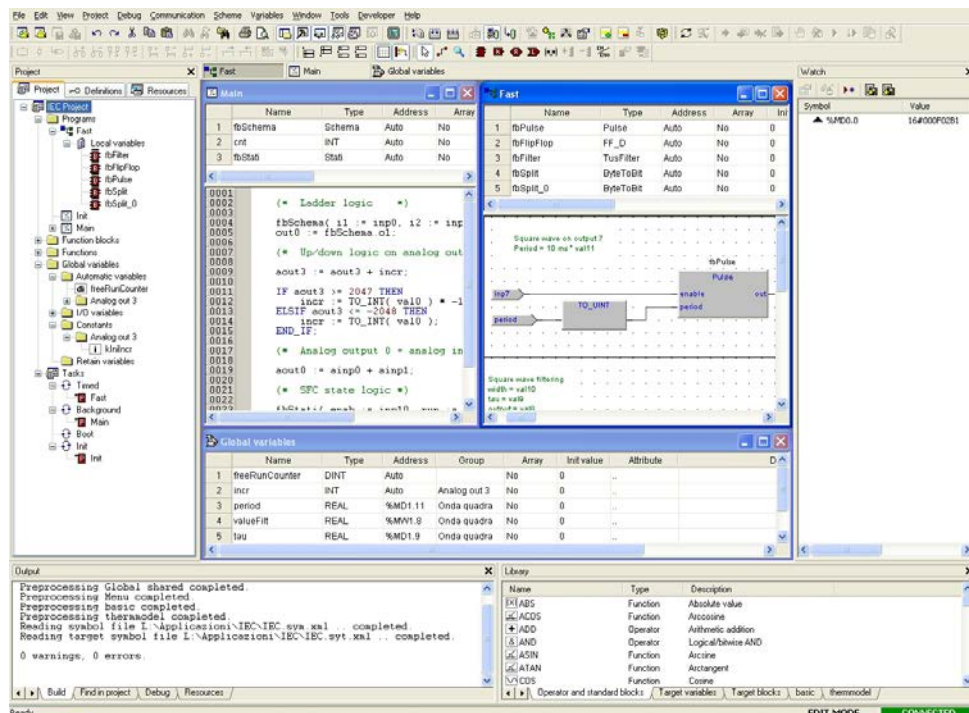
Application is an IEC61131-3 Integrated Development Environment supporting the whole range of languages defined in the standard.

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 5);
- 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 5);
- 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);
- communication system which allows the download of the application to the target environment (see Chapter 7);
- 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, helping to ensure that the information is accurate and reliable (see Chapter 8).

### 1.1 THE WORKSPACE

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

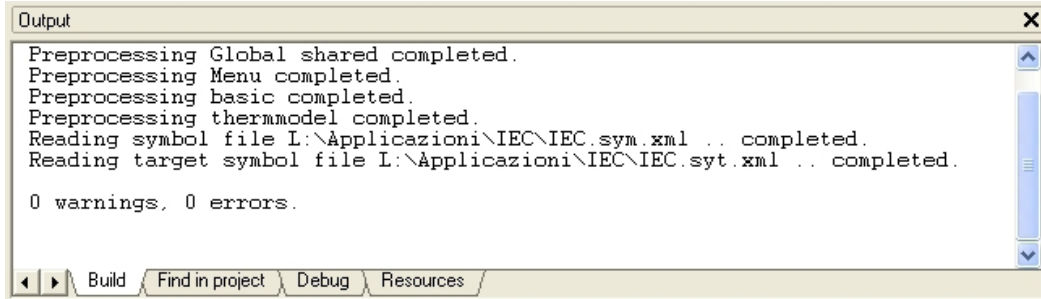




The following paragraphs give an overview of these elements.

## 1.1.1 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

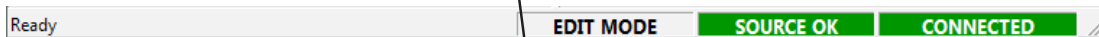
The *Debug* panel displays information about advanced debugging activities (for example, breakpoints). Depending on the target device you are interfacing with, Application can print on this output window every PLC run-time error (for example, division by zero), locating the exact position where the error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

### Resources

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

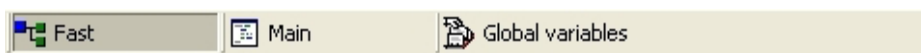
## 1.1.2 THE STATUS BAR

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.



## 1.1.3 THE DOCUMENT BAR

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

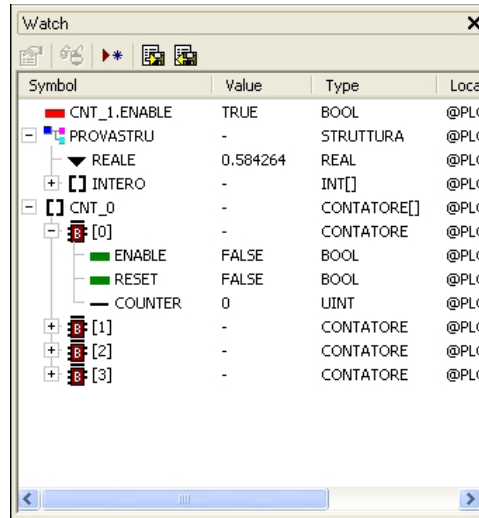






## 1.1.4 THE WATCH WINDOW

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 8.2), triggers, and the live debug mode (see Paragraph 8.2).



## 1.1.5 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 details* 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.).

### 1.1.5.1 OPERATORS AND STANDARD BLOCKS

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

Name	Type	Description
ABS	Function	Absolute value
ACOS	Function	Arccosine
ADD	Operator	Arithmetic addition
AND	Operator	Logical/bitwise AND
ASIN	Function	Arcsine
ATAN	Function	Arctangent

Operator and standard blocks | Target variables | basic | themmodel



## 1.1.5.2 TARGET VARIABLES

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

Name	Type	Address	Group	Description
<b>i</b> Ad_InPo	INT	%MW0.1	DEB - ANALOG-DIGITAL EN...	incremental position
<b>i</b> Ad_NuCi	INT	%MW0.12	DEB - ANALOG-DIGITAL EN...	DSP cycles without position increment
<b>di</b> Ad_PeSp	DINT	%MW0.10	DEB - ANALOG-DIGITAL EN...	calculated speed
<b>i</b> Ad_SeOf	INT	%MW0.9	DEB - ANALOG-DIGITAL EN...	sine channel offset
<b>di</b> Ad_ViPo	DINT	%MW0.2	DEB - ANALOG-DIGITAL EN...	virtual position
<b>di</b> Ad_ViPoIni	DINT	%MW0.218	DEB - ANALOG-DIGITAL EN...	

Operator and standard blocks | Target variables | basic | thermmodel

## 1.1.5.3 TARGET BLOCKS

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

Name	Type	Description
<b>F</b> sysMsgInterpMono	Function	Checks messages from single axis int...
<b>F</b> sysQuiesInterpMonoPlc	Function	Verifies the end of an interpolated sin...
<b>F</b> sysResetInterpMonoPlc	Function	Resets an interpolated single axis mo...
<b>F</b> sysSleep	Function	Puts the current task in the sleeping s...
<b>F</b> sysStartInterpMono	Function	Starts an interpolated single axis mov...
<b>F</b> sysStartInterpMonoPlc	Function	Starts an interpolated single axis mov...
<b>F</b> sysTargetInterpMonoPlc	Function	Verifies if the target of an interpolated ...
<b>F</b> sysWaitInterpMono	Function	Waits until the end of an interpolated ...

Operator and standard blocks | Target variables | Target blocks | basic

## 1.1.5.4 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 also Paragraph 3.7).

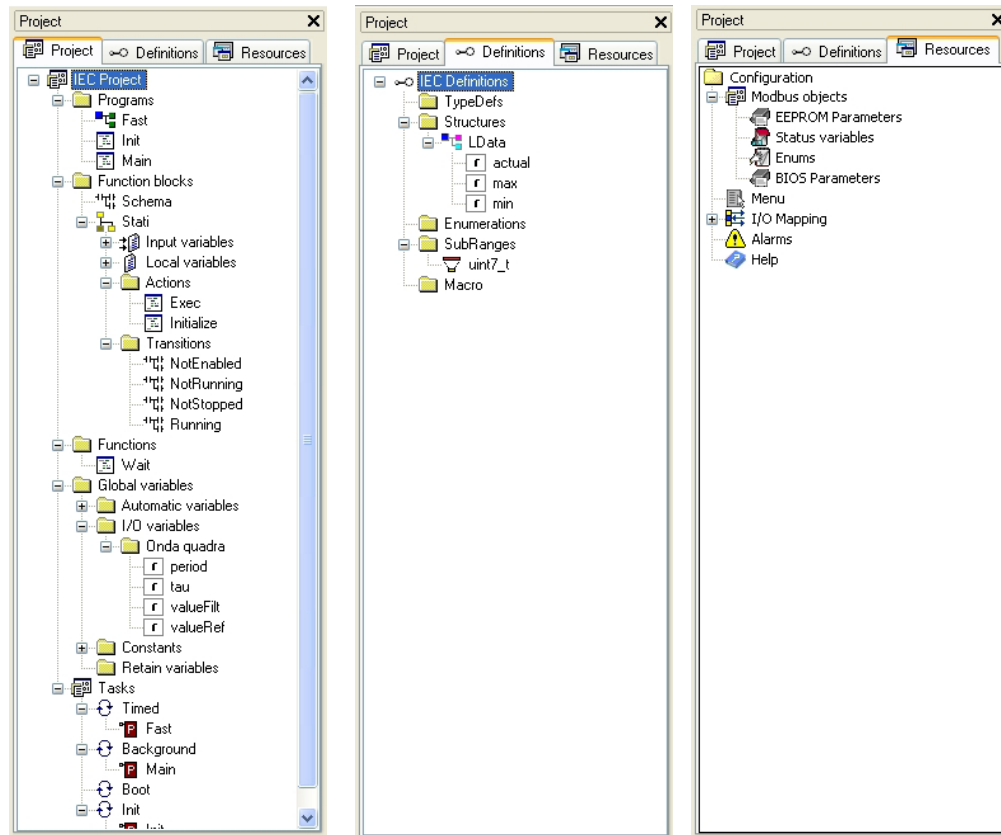
Name	Type	Description
<b>F</b> BitToByte	Function	Compose a byte from 8 bits
<b>F</b> BitToWord	Function	Compose a word from 16 bits
<b>B</b> ByteToBit	Function block	Split a byte into bits
<b>F</b> ByteToWord	Function	Compose a word from 2 bytes
<b>B</b> F_TRIG	Function block	Falling edge detector
<b>B</b> FF_D	Function block	D-type flip-flop

Operator and standard blocks | Target variables | basic | thermmodel



## 1.1.6 THE WORKSPACE WINDOW

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



### 1.1.6.1 PROJECT

The *Project* panel contains a set of folders:

- *Program, Function blocks, Functions*: each folder contains Program Organization Units (briefly, POUs - see Paragraph 4.1) of the type specified by the folder name.
- *Global variables*: it is further divided in *Variables, I/O Variables, Constants* and *Retain variables*. Each folder contains global variables of the type specified by the folder name (see Paragraph 4.2).
- *Tasks*: this item lists the system tasks and the programs assigned to each task (see Paragraph 4.3).

### 1.1.6.2 DEFINITIONS

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

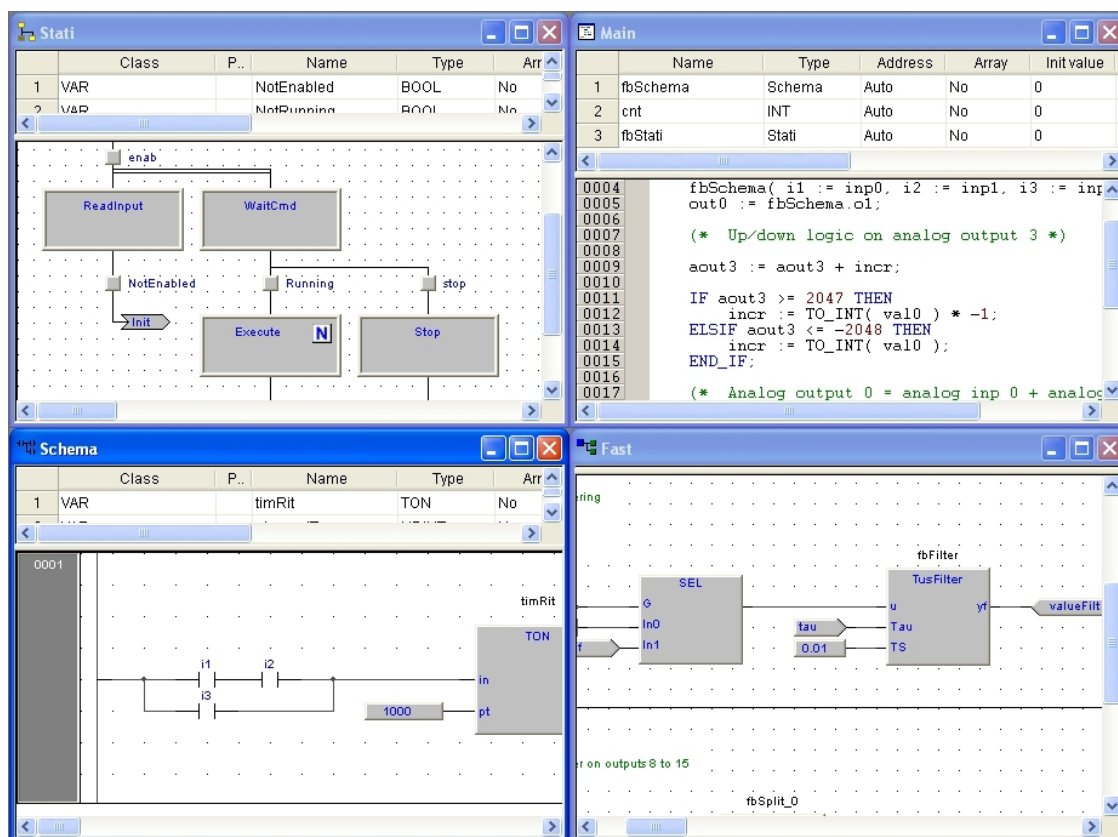
### 1.1.6.3 RESOURCES

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.



## 1.1.7 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 5).



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	freeRunCounter	DINT	Auto		No	0	..	
2	incr	INT	Auto	Analog out 3	No	0	..	
3	period	REAL	%MD1.11	Onda quadra	No	0	..	
4	valueFilt	REAL	%MW1.8	Onda quadra	No	0	..	
5	tau	REAL	%MD1.9	Onda quadra	No	0	..	
6	valueRef	REAL	%MD1.10	Onda quadra	No	0	..	
7	knIncr	INT	Auto	Analog out 3	No	50	CONSTANT	



## 2. 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.

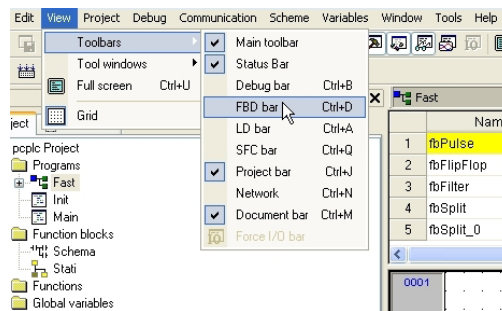
### 2.1 LAYOUT CUSTOMIZATION

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.

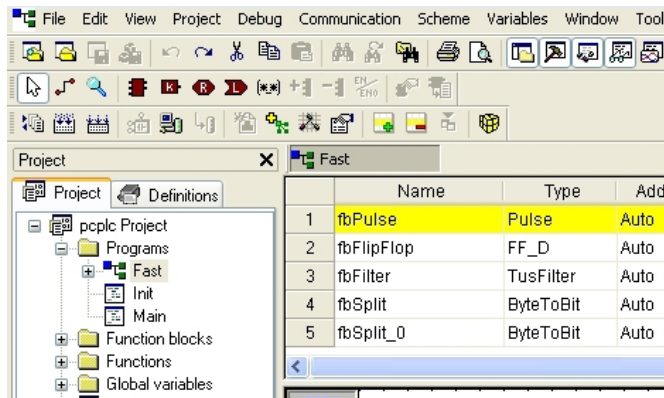
### 2.2 TOOLBARS

#### 2.2.1 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 *Function Block Diagram* bar).

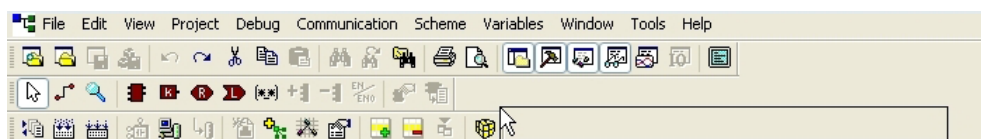


The toolbar is then shown (hidden).



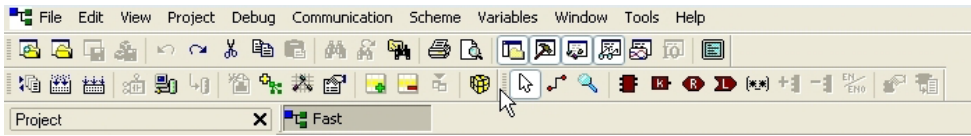
#### 2.2.2 MOVING TOOLBARS

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

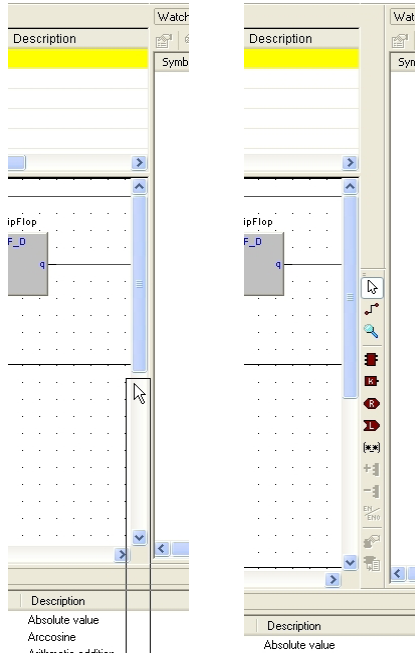




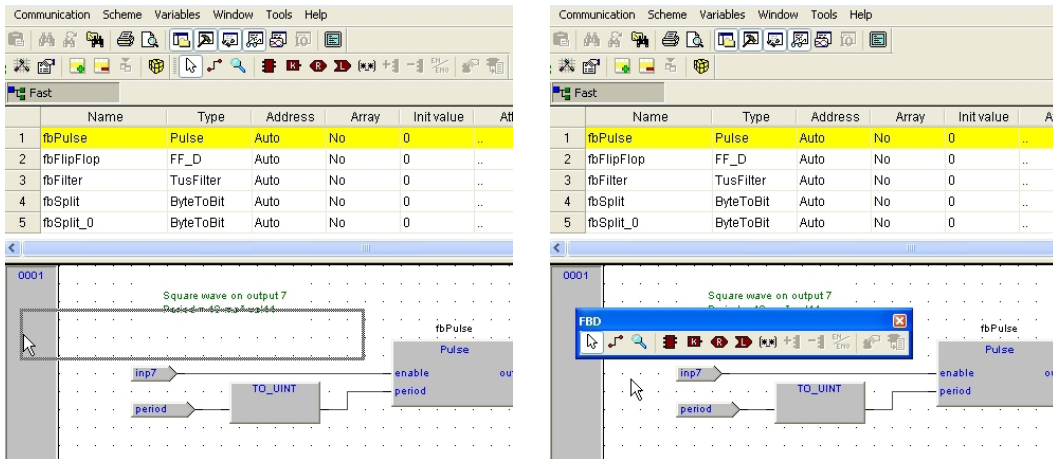
The toolbar shows up in the new position.



You can change the shape of the toolbar, from horizontal to vertical, either by pressing the *Shift* key or by moving the toolbar next to the vertical border of any window.



You can also make the toolbar float, either by pressing the *CTRL* key or by moving the toolbar away from any window border.

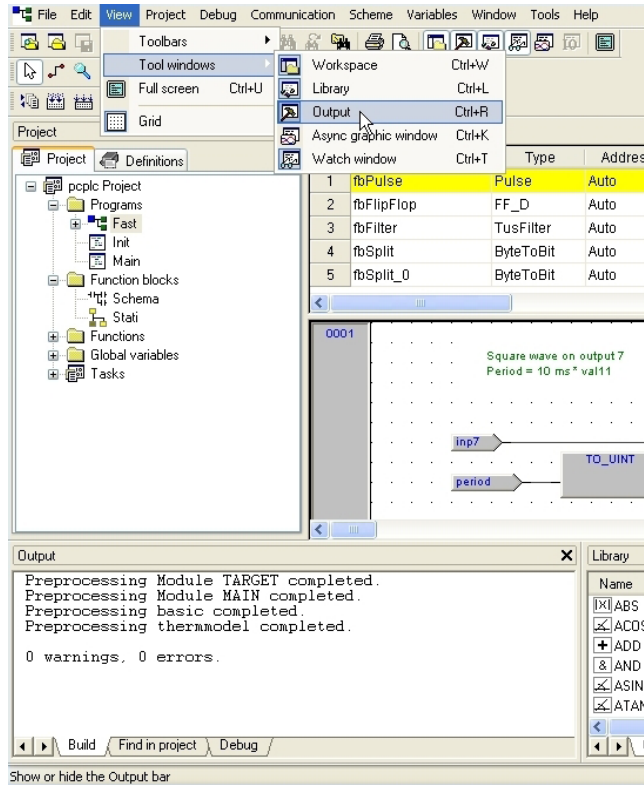




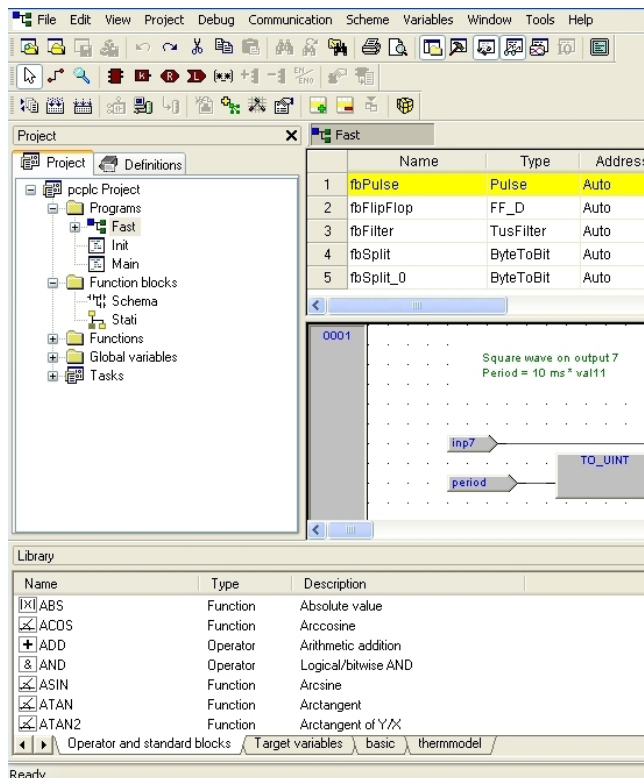
## 2.3 DOCKING WINDOWS

### 2.3.1 SHOWING/HIDING TOOL WINDOWS

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



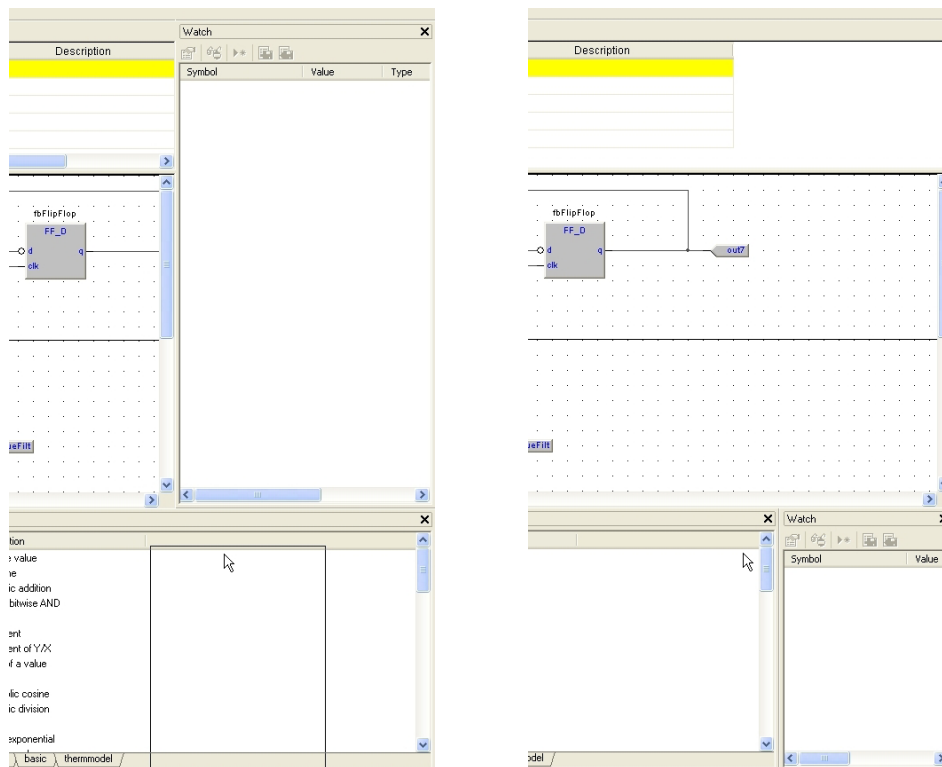
The tool window is then shown (hidden).



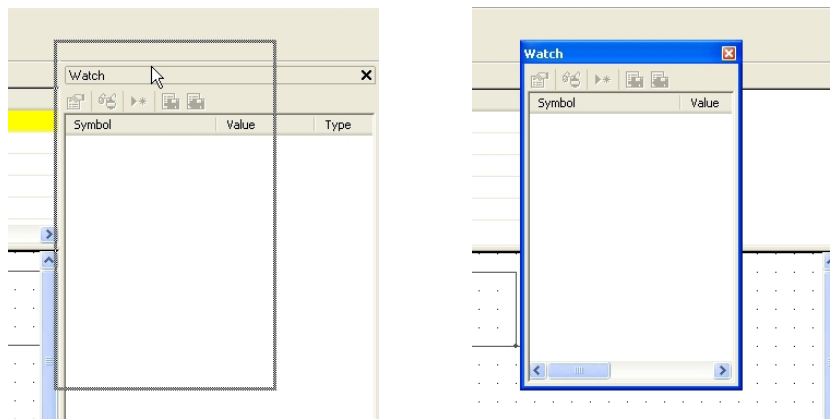


## 2.3.2 MOVING TOOL WINDOWS

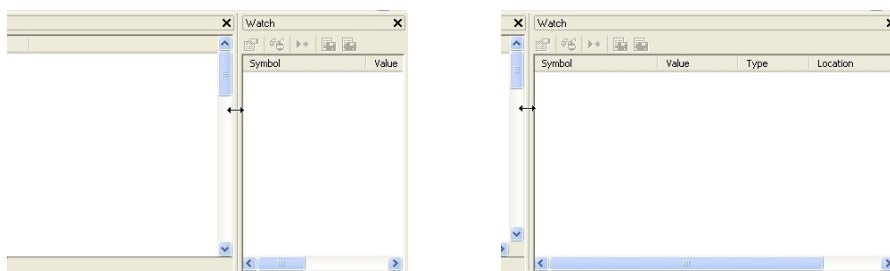
In order to move a tool window, click on its name (at the top of the window) and then drag and drop it to the destination.



You can make the tool window float, by double-clicking on its name, or by pressing the *CTRL* key, or by moving the tool window away from the main window borders.



A tool window can be resized by clicking-and-dragging on its border until the desired size is reached.



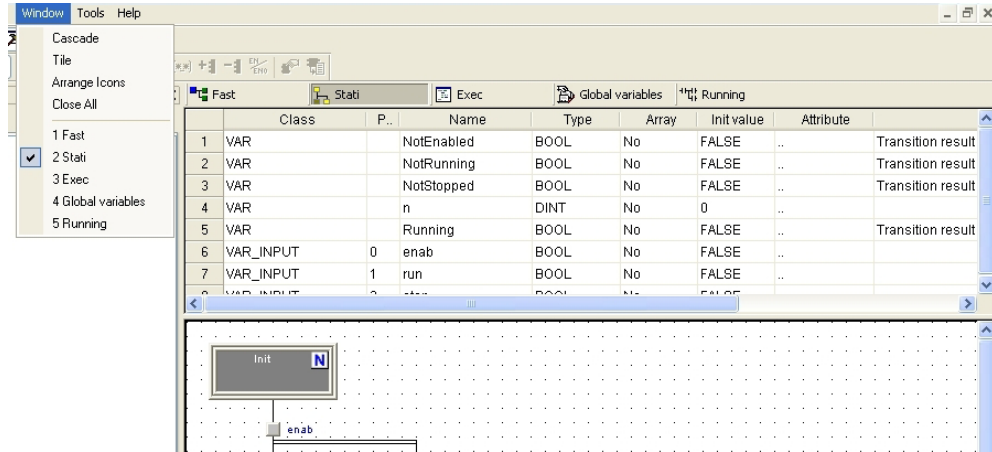




## 2.4 WORKING WITH WINDOWS

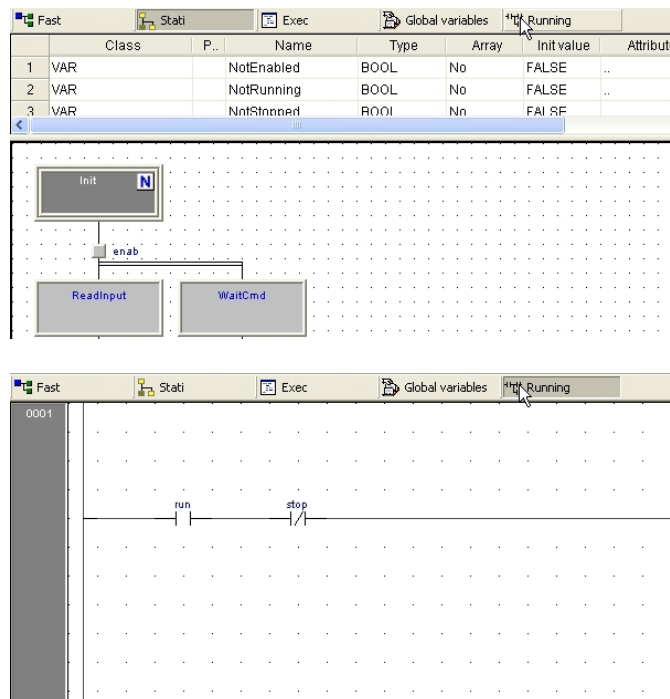
Application allows to open many source code editors so that the workspace could get rather messy.

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



### 2.4.1 THE DOCUMENT BAR

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

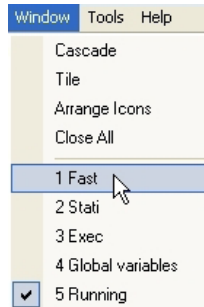


You can show or hide the *Document* bar with the menu option of the same name in the menu *View>Toolbars*.



## 2.4.2 THE WINDOW MENU

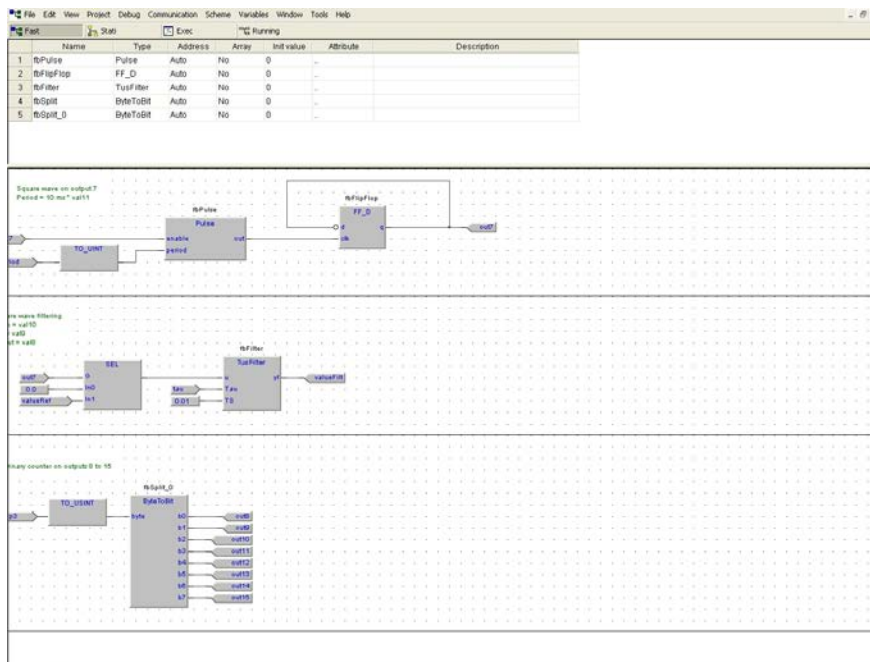
The *Window* menu is an alternative to the *Document* bar: it lists all the currently open editors and allows to switch between them.



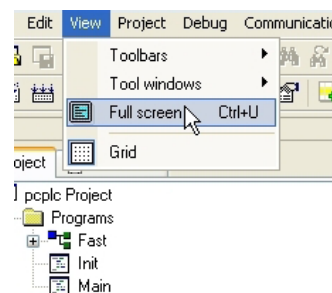
Moreover, this menu supplies a few commands to automate some basic tasks, such as closing all windows.

## 2.5 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 *Full screen* option of the menu *View* or with the corresponding command of the *Main* toolbar.





## 2.6 ENVIRONMENT OPTIONS

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

### General

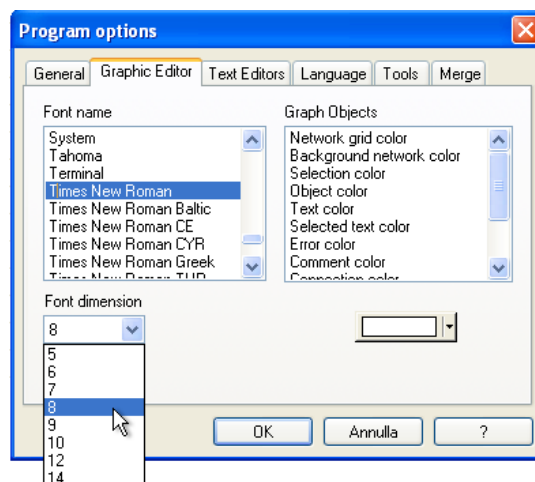
**Autosave:** if the *Enable Autosave* box is checked, Application periodically saves the whole project. You can specify the period of execution of this task by entering the number of minutes between two automatic savings in the *Autosave interval* text box.

### 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.

### Graphic Editor

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

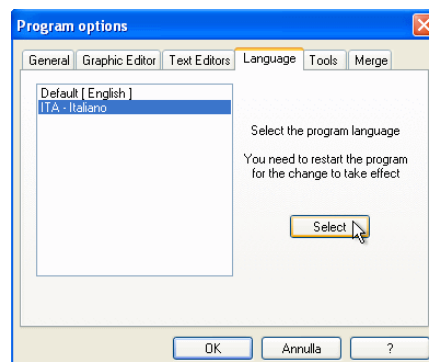


### Text Editors

#### 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.

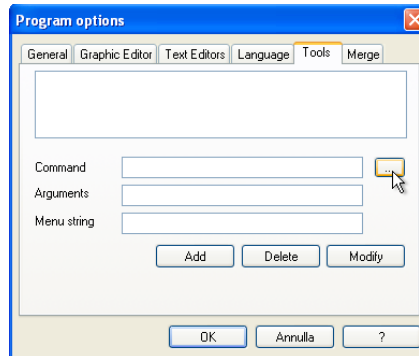


#### Tools

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.



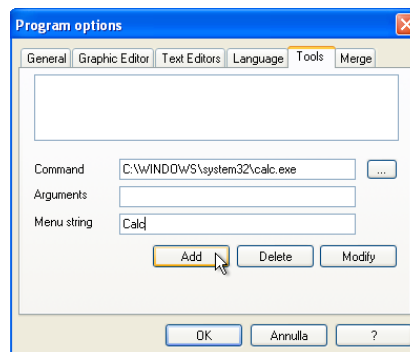
- 1) Type the full path of the executable file of the tool in the *Command* text box. Otherwise, you can specify the filename by selecting it from Windows Explorer, which you open by clicking the *Browse* button.



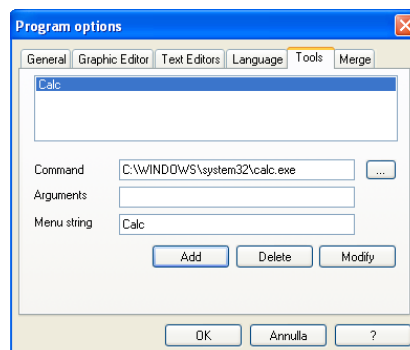
- 2) In the *Arguments* text box, type the arguments - if any - to be passed to the executable command mentioned at step 1. They must be separated by a space.
- 3) Enter in *Menu string* the name you want to give to the tool you are adding. This is the string that will be displayed in the *Tools* menu.
- 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:

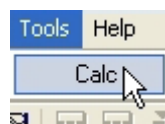
- Fill the fields of the dialog box as displayed.



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



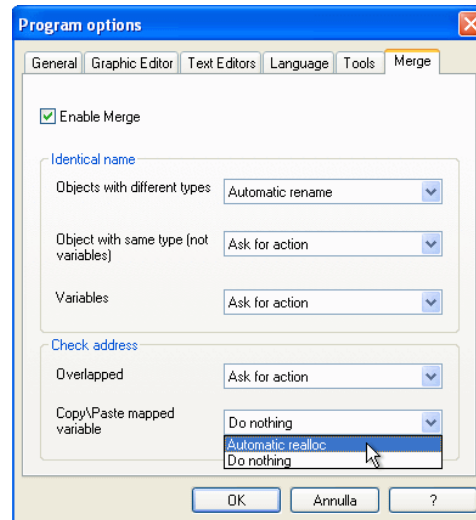
And in the *Tools* menu as well.





## Merge

Here you can set the merge function behavior.



See paragraph 3.8.3.2 for more details.





## 3. MANAGING PROJECTS

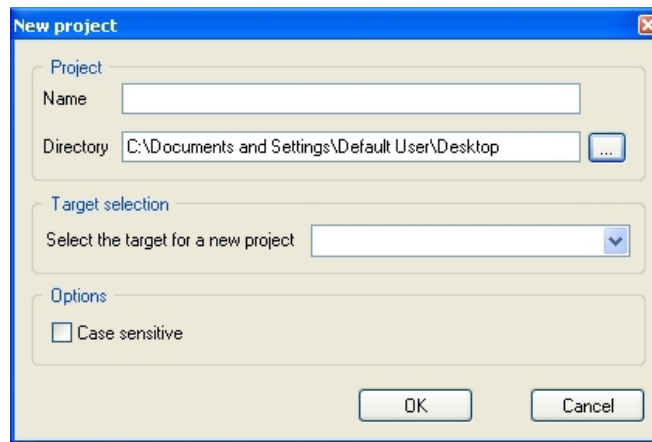
This chapter focuses on Application projects.

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.

### 3.1 CREATING A NEW PROJECT

To start a new project, click *New project* in the *File* menu of the Application main window. The same command is available in the *Main* toolbar and, if no project is open, in Application's welcome page. This causes the following dialog box to appear.



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 incorrect 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.

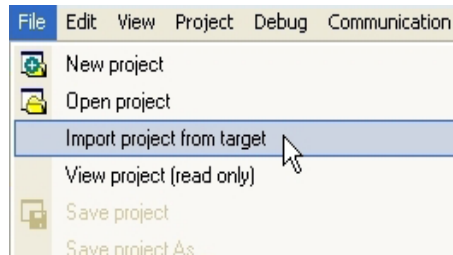
### 3.2 UPLOADING THE PROJECT FROM THE TARGET DEVICE

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



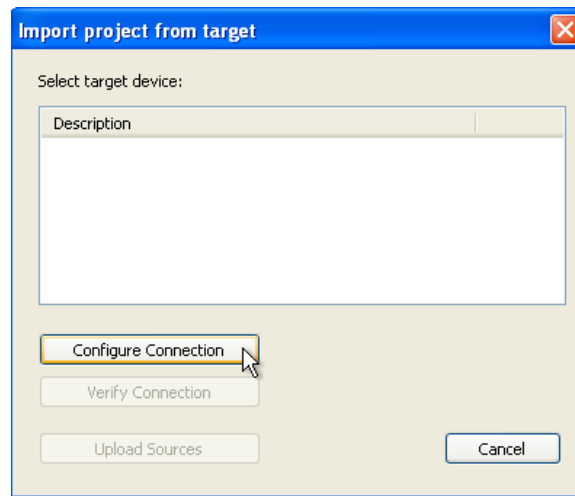
In order to upload the project from the target device, follow the procedure below:

1) Select the item *Import project from target* in the menu *File*.

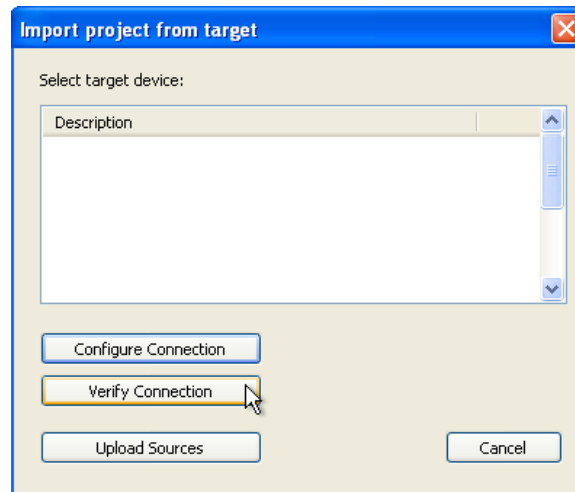


2) Select the target device you are connecting to, from the list shown in the *Target list* window.

3) Configure connection (see paragraph 7.1 for more details).



4) You may optionally test the connection with the target device.



Application tries to establish the connection and reports the test result.







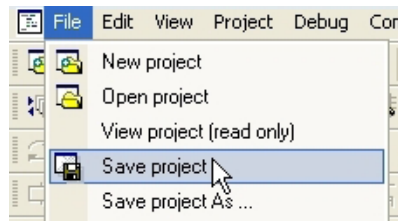
5) If the connection is available confirm the operation by clicking on the upload sources button. When the application upload completes successfully, the project is open for editing.

## 3.3 SAVING THE PROJECT

### 3.3.1 PERSISTING CHANGES TO THE PROJECT

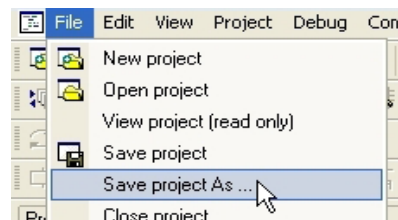
When you make any change to the project (for example, you add a new Program Organization Unit) you are required to save the project in order to persist that change.

To save the project, you can select the corresponding item of the menu *File* or the *Main* toolbar.



### 3.3.2 SAVING TO AN ALTERNATIVE LOCATION

When you do not want to (or cannot - for example, because the file is read-only) overwrite the project file, you may save the modified version of the project to an alternative location, by selecting *Save project as...* from the *File* menu.



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.

## 3.4 MANAGING EXISTING PROJECTS

### 3.4.1 OPENING AN EXISTING APPLICATION PROJECT

To open an existing project, click *Open project* in the *File* menu of Application's main window, or in the *Main* toolbar, 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.

### 3.4.2 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.

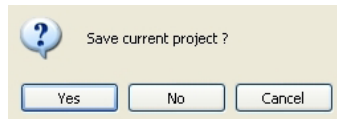


Note that Application prevents you from applying changes to elements of a project, when at least one of the following conditions holds:

- You cannot modify any object of the project if you are in debug mode.
- You cannot edit 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).

### 3.4.3 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 *Close project* from the *File* menu; Application shows the *Welcome page*, so that you can rapidly start a new working session.

## 3.5 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 target device available) 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 of the *Project* menu or toolbar.



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.

## 3.6 PROJECT OPTIONS

You can edit some significant project properties choosing *Options...* in the *Project* menu.

### 3.6.1 PROJECT INFO

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

Download	Debug	Build events
Project info	Code generation	Build output
Project:	<input type="text" value="PLC examp"/> (max 10 chars)	
Version:	<input type="text" value="1.5"/> (example: 1.0)	
Author:	<input type="text" value="John Doe"/>	
Note:	<input type="text"/>	

### 3.6.2 CODE GENERATION

Here you can edit some properties about code generation.

Download	Debug	Build events
Project info	Code generation	Build output
Case sensitivity (IEC default=no)	<input type="checkbox"/>	
Check FB external variables	<input checked="" type="checkbox"/>	
Print debug informations	<input type="checkbox"/>	
Allow only integer indexes for arrays	<input type="checkbox"/>	
Run-time check of array bounds	<input type="checkbox"/>	
Run-time check of division by zero	<input type="checkbox"/>	
Enable WAITING statement (standard extension)	<input type="checkbox"/>	
Disable warning emission	<input type="checkbox"/>	
Disabled warning codes:	<input type="text"/>	<input type="button" value="+"/> <input type="button" value="-"/>

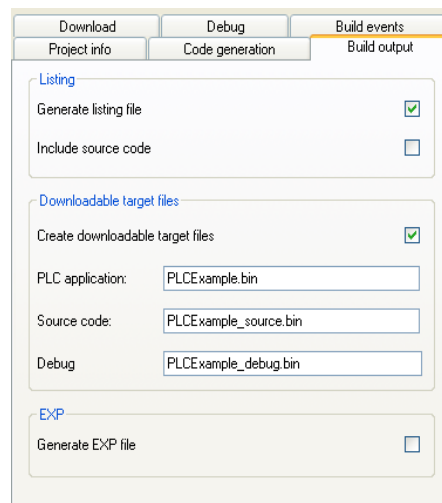
- *Case sensitivity*: you can set the project as case-sensitive checking this option. Note that, by default, this option is not active.
- *Check FB external variables*: if this option is checked you must declare all function blocks as external variables.



- *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.
- *Enable WAITING statement (standard extension)*: 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.
- *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.

### 3.6.3 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*

## 3.6.4 DOWNLOAD

Here you can edit some significant properties of the download behavior. See paragraph 7.3.1 for more information.

Project info	Code generation	Build output
Download	Debug	Build events
Source code		
Download time	On PLC application download	
Protect with password	<input checked="" type="checkbox"/>	
Password	<input type="text"/>	
Debug symbols		
Download time	On PLC application download	

## 3.6.5 DEBUG

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

Project info	Code generation	Build output
Download	Debug	Build events
Polling period for debug functions (ms)	20	
Number of displayed array elements without alert message	20	
Polling period between more variables (ms)	0	
Autosave watch list	<input type="checkbox"/>	

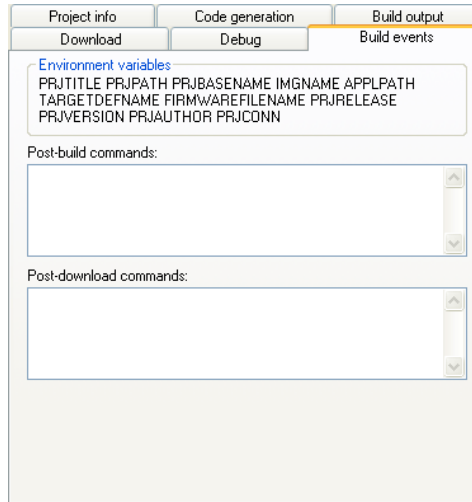
- *Polling period for debug function (ms)*: specifies the period how often functions are seen in what state they are.
- *Number of displayed array elements without alert message*: specifies the maximum number of array element to be added in watch window without being alerted.
- *Polling period between more variables (ms)*: specifies the period between variables before are seen in what state they are.



- *Autosave watch list*: if checked (by default no) the watch list status will be saved into a file, when the project is closed. See paragraph 9.1.7 for more details.

## 3.6.6 BUILD EVENTS

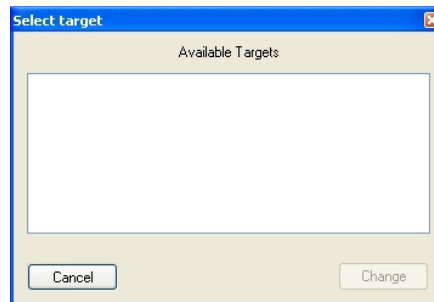
Here you can specify commands that run before the build starts or after the build finishes. You can also use a set of defined environment variables listed on the top of the window.



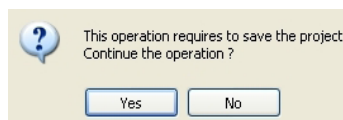
## 3.7 SELECTING THE TARGET DEVICE

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

- 1) Click *Select target* in the *Project* 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.

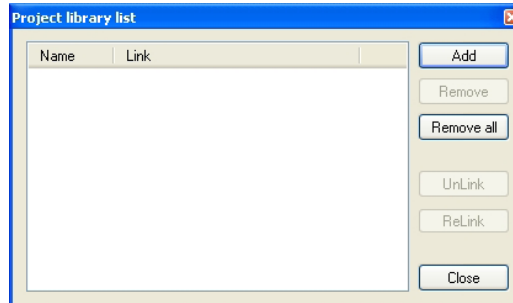
It also makes a backup copy of the project file(s) in a sub-directory inside the project directory, so that you can roll-back the operation by manually (i.e., using Windows Explorer) replacing the project file(s) with the backup copy.



## 3.8 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 *.p11*.

### 3.8.1 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 *Library manager* in the *Project* menu.

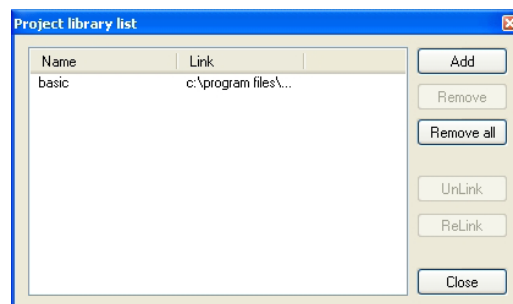
#### 3.8.1.1 INCLUDING A LIBRARY

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 *.p11* 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 *Library manager* in the *Project* menu, 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 *.p11* file of the library you want to open.
- 3) When you have found the *.p11* 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.



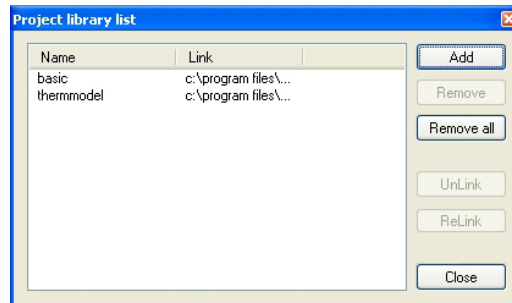
- 4) Repeat step 1, 2, and 3 for all the libraries you wish to include.
- 5) When you have finished including libraries, press either *OK* to confirm, or *Cancel* to quit.



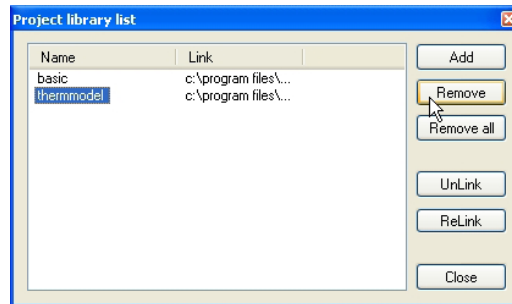
## 3.8.1.2 REMOVING A LIBRARY

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 *Library manager* in the *Project* 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.

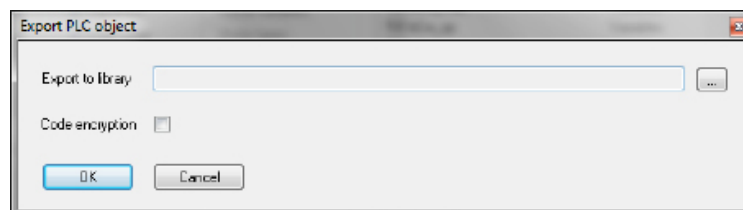


- 2) Click the *Remove* button, which causes the reference to the selected library to disappear from the *Project Library* list.
- 3) Repeat for all the libraries you wish to remove. Alternatively, if you want to remove all the libraries, you can press the *Remove all* button.
- 4) When you have finished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

## 3.8.2 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 *Export object to library* in the *Project* menu. This causes the following dialog box to appear.







- 3) Enter the destination library by specifying the location of its `.p11` 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 source code.
- 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 `.p11` file, Application creates the file, thus establishing a new library.

### 3.8.2.1 UNDOING EXPORT TO A LIBRARY

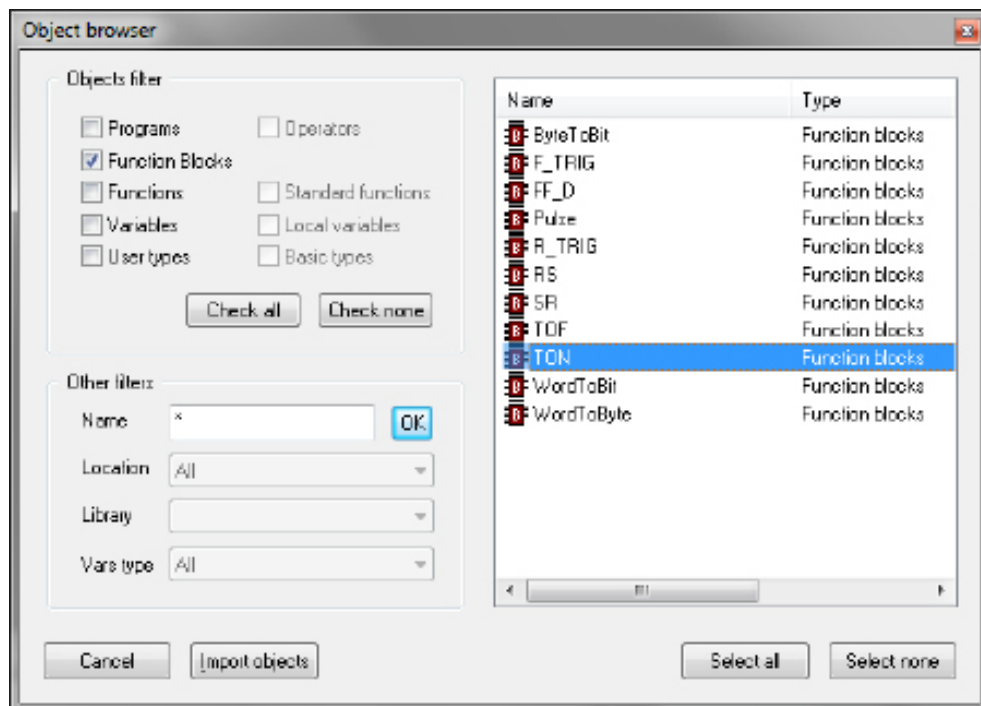
So far, it is not possible to undo export to a library. The only possibility to remove an object is to create another library containing all the objects of the current one, except the one you wish to delete.

### 3.8.3 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 *Import object from library* in the *Project* menu. This causes an explorer dialog box to appear, which lets you select the `.p11` file of the library you want to open.
- 2) When you have found the `.p11` 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.

### 3.8.3.1 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.

### 3.8.3.2 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 2.6 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
<b>Naming behavior</b>	If different types	X	X		X
	If same type but not variables	X	X	X	
	If both variables	X	X	X	
<b>Address behavior</b>	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.

### 3.8.4 UPDATING EXISTING LIBRARIES

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

- 1) Click *Refresh all libraries* object in the *Project* menu or click the item in the project bar.
- 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.



## 4. MANAGING PROJECT ELEMENTS

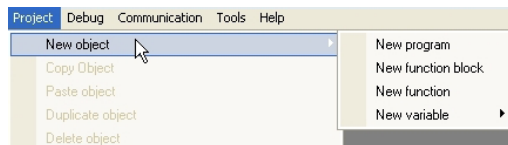
This chapter shows you how to deal with the elements which compose a project, namely: Program Organization Units (briefly, POUs), tasks, derived data types, and variables.

### 4.1 PROGRAM ORGANIZATION UNITS

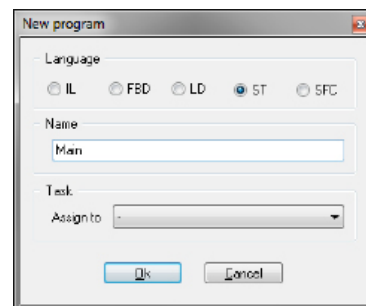
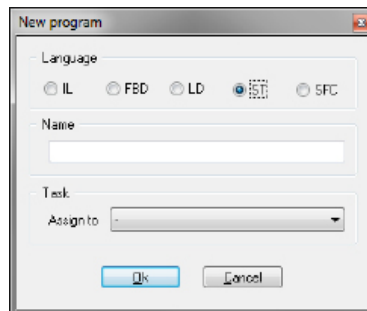
This paragraph shows you how to add new POUs to the project, how to edit and eventually remove them.

#### 4.1.1 CREATING A NEW PROGRAM ORGANIZATION UNIT

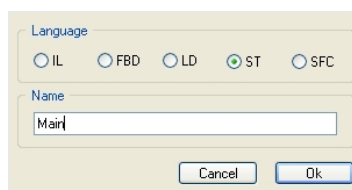
- 1) Select the *New object* item in the *Project* menu.



- 2) Specify what kind of POU you want to create by clicking one of the items in the sub-menu which pops up.
- 3) Select the language you will use to implement the POU.

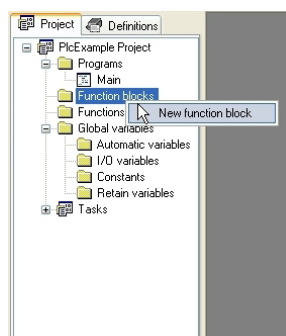


Enter the name of the new module.



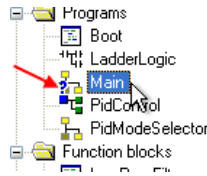
- 4) Confirm the operation by clicking on the *OK* button.

Alternatively, you can create a new POU of a specific type (program, function block, or function) by right-clicking on the correspondent item of the project tree.





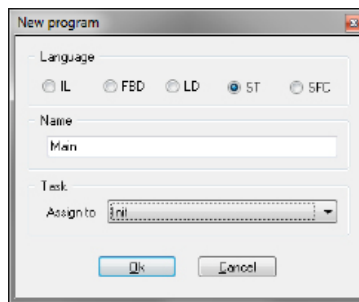
- After creating a new program, an alert icon (interrogation mark) appears below the new program icon.



This alert icon indicates that the program is not yet associated to a task. Refer to paragraph 4.3.1 to assign the program to the desired task.

### 4.1.1.1 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.

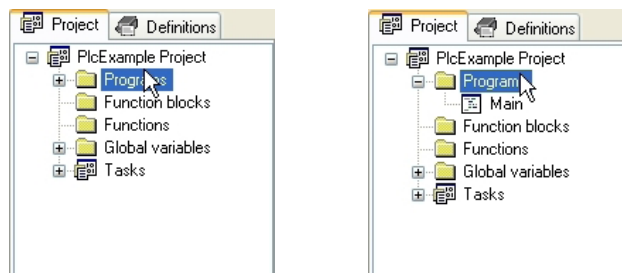


### 4.1.2 EDITING POUS

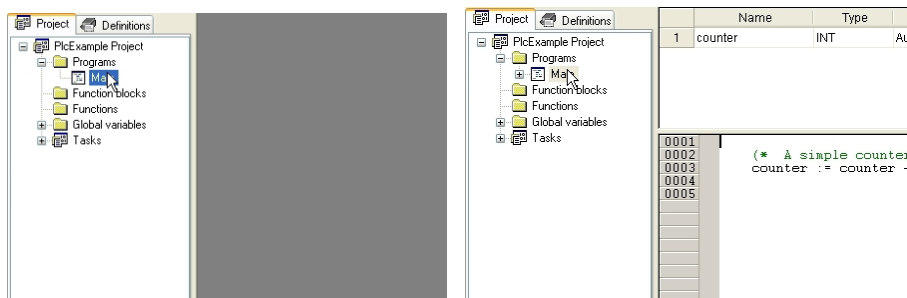
All the POUs of the project are listed in the *Programs*, *Function blocks*, and *Functions* folders in the *Project* tab of the *Workspace* bar.

The following procedure shows you how to edit the source code of an existing POU.

- Open the folder in the *Project* tab of the workspace that contains the object you want to edit by double-clicking the folder name.



- Double-click the name of the object you want to edit. The relative editor opens and lets you modify the source code of the POU.

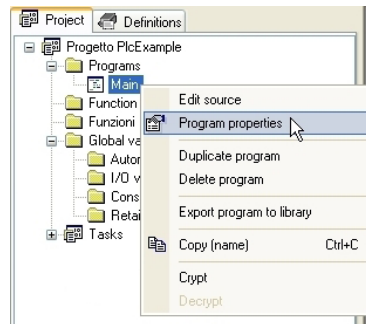




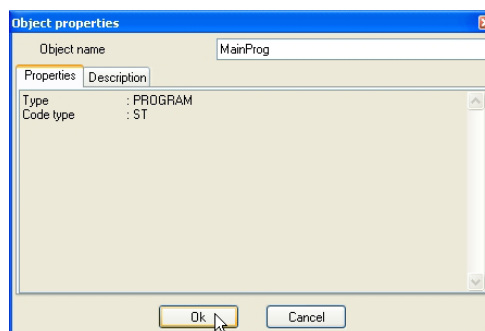
# SoMachine HVAC - Application

You may want to change the name of the POU:

- 1) Open the *Object properties* editor from the contextual menu which pops up when right-clicking the POU name in the project tree (alternatively, select the correspondent item in the *Project* menu).

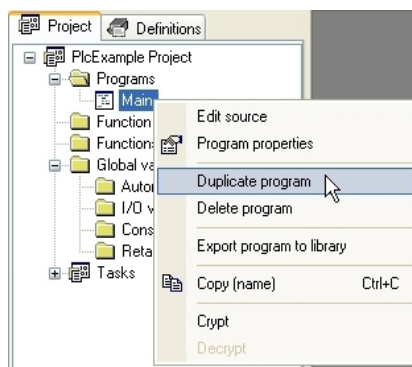


- 2) Change the object name and confirm.

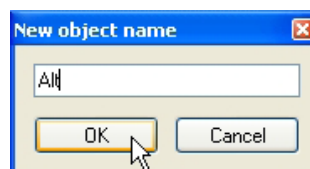


Finally, you can create a duplicate of the POU in this way:

- 1) Select *Duplicate* from the contextual menu (or the *Project* menu).



- 2) Enter the name of the new POU and confirm.

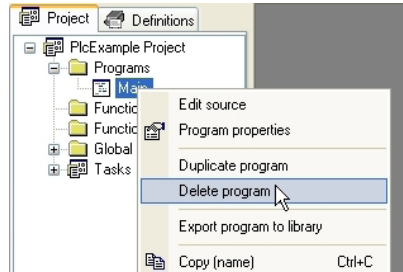




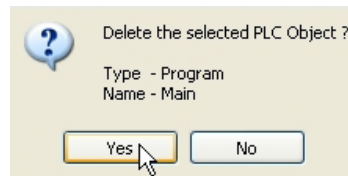
## 4.1.3 DELETING POU S

Follow this procedure to remove a POU from your project:

- 1) Open the folder in the *Project* tab of the workspace that contains the object you want to delete by double-clicking the folder name.
- 2) Right-click the name of the object you want to delete. A context menu appears referred to the selected object.



- 3) Click *Delete object* in the context menu, then press *Yes* to confirm your choice.



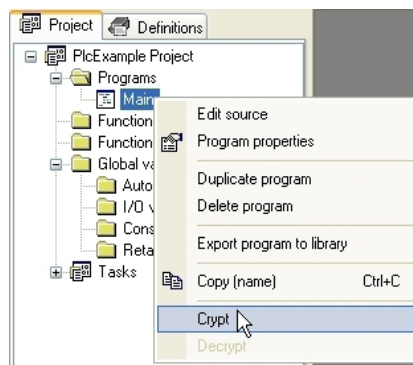
## 4.1.4 SOURCE CODE ENCRYPTION

You may want to hide the source code of one or more POU s.

Application lets you encrypt POU s and protect them with a password.

To encrypt a POU, perform the following steps:

- 1) Right-click the POU name in the project tree and choose *Crypt* from the contextual menu.

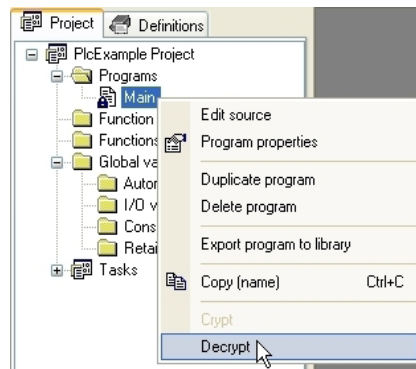


- 2) Enter the password twice (to avoid any problem which may arise from typos) and confirm the operation.



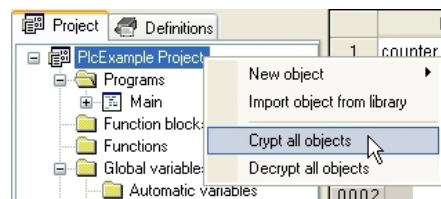


To decrypt a POU, right-click the POU name in the project tree and choose *Decrypt* from the contextual menu.



Application prompt you to enter the password.

You can choose to encrypt all the unencrypted POUs at once:



the same password applies to all objects.

## 4.2 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.

### 4.2.1 GLOBAL VARIABLES

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

#### 4.2.1.1 CLASSES OF GLOBAL VARIABLES

Global variables are listed in the project tree, in the *Global variables* folder, where they are further classified according to their properties as Automatic variables, Mapped variables, Constants, and Retain variables.

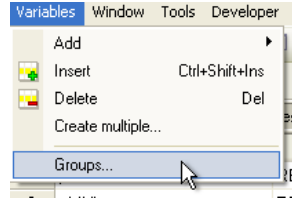
- Automatic variables include all the variables that the compiler automatically allocates to an appropriate location in the target device memory.
- Mapped variables, on the other way, do have an assigned address in the target device logical addressing system, which shall be specified by the developer.
- Constants list all the variables which the developer declared as having the `CONSTANT` attribute, so that they cannot be written.
- Retain variables list all the variables which the developer declared as having the `RETAIN` attribute, so that their values are stored in a persistent memory area of the target device.



## 4.2.1.2 GROUPS OF GLOBAL VARIABLES

You can further categorize the set of all global variables by grouping them according to application-specific criteria. In order to define a new group, follow this procedure:

- 1) Select *Group* from the *Variables* menu (note that this menu is available only if the *Global variables* editor is open).



- 2) Enter the name of the new variable group, then click *Add*.

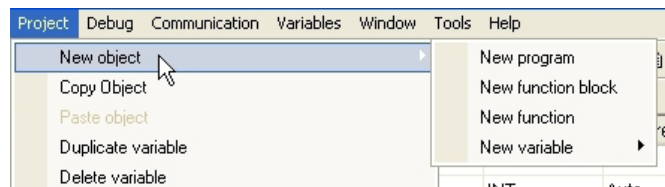


- 3) You can now use the variable group in the declaration of new global variables.

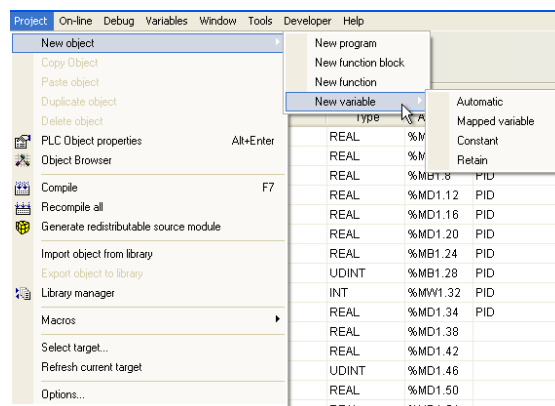
## 4.2.1.3 CREATING A NEW GLOBAL VARIABLE

Apply the following procedure to declare a new global variable:

- 1) Select *New object* in the *Project* menu.



- 2) Select *New variable* from the menu that shows up.

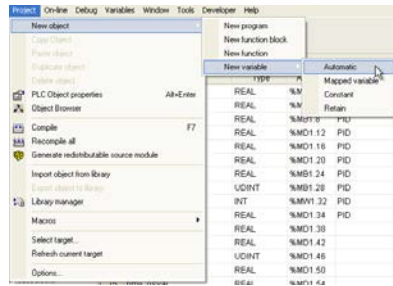




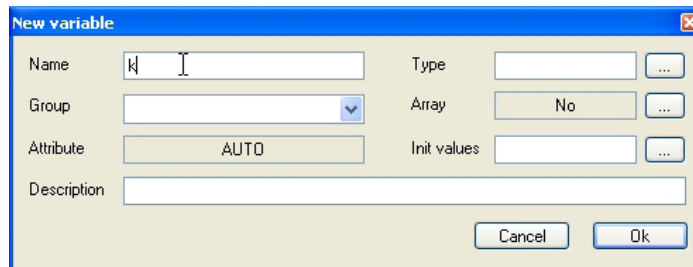


# SoMachine HVAC - Application

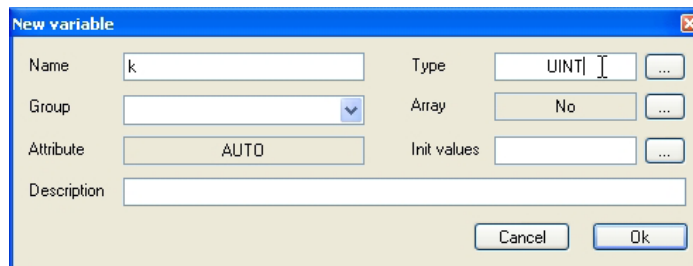
- 3) Choose the class of the variable you want to declare (Automatic variables, Mapped variables, Constants, or Retain variables).



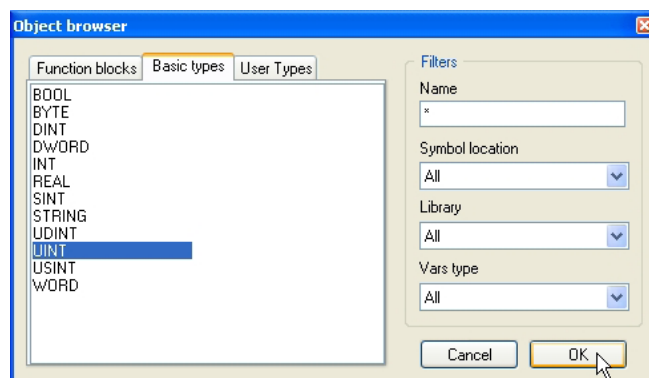
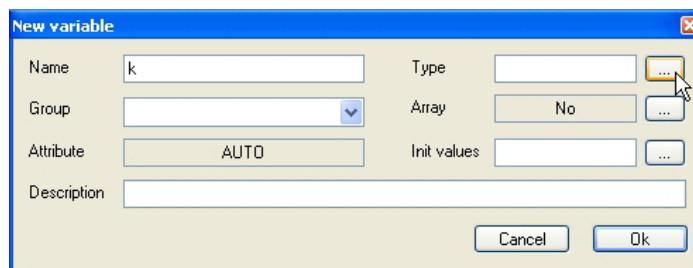
- 4) Enter the name of the variable (remember that some characters, such as '?', ':', '/', and so on, cannot be used: the variable name must be a valid IEC 61131-3 identifier).



- 5) Specify the type of the variable either by typing it

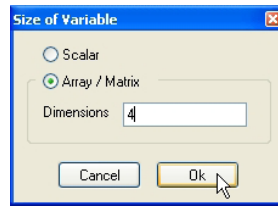


or by selecting it from the list that Application displays when you click on the *Browse* button.

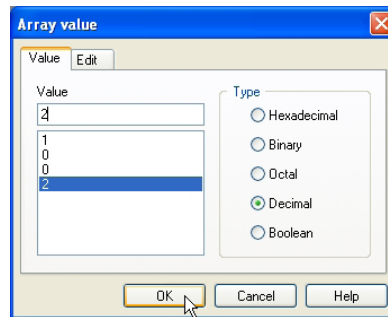
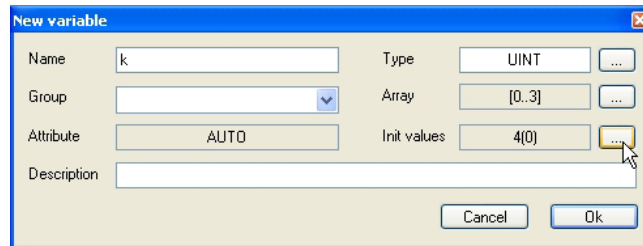




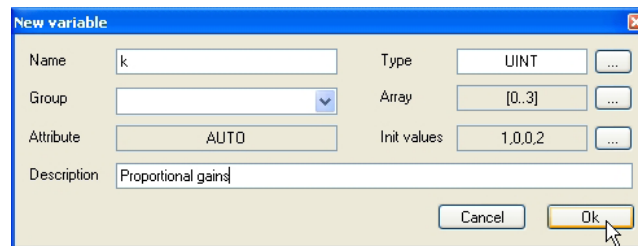
6) If you want to declare an array, you can specify its size.



7) You may optionally assign the initial value to the variable.

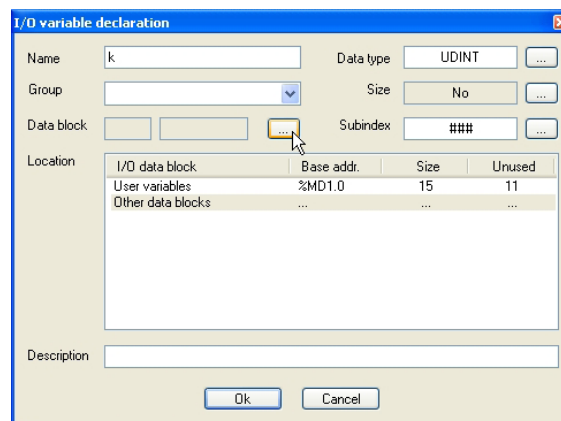


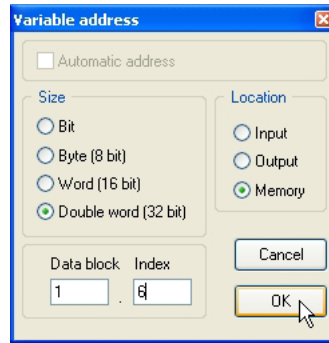
8) Finally, you can add a brief description and then confirm the operation.



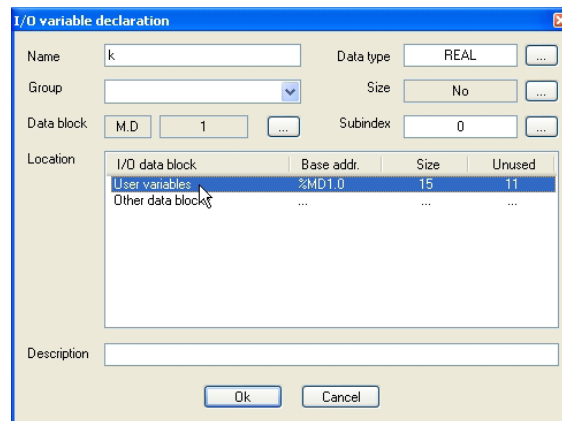
If you create a new mapped variable, you are required to specify the address of the variable during its definition. In order to do so, you may do one of the following actions:

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





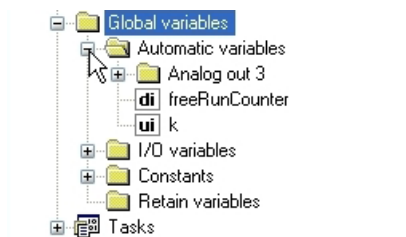
- Select from the list that Application shows you the memory area you want to use: the tool automatically chooses the address of the first free memory location of that area.



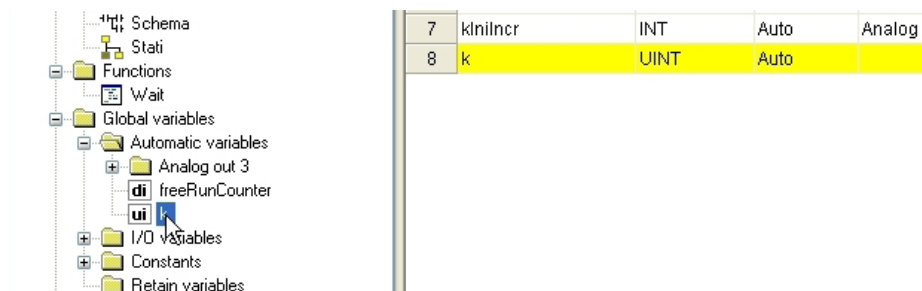
### 4.2.1.4 EDITING A GLOBAL VARIABLE

To edit the definition of an existing global variable:

- 1) Open the folder in the *Project* tab of the workspace that contains the variable you want to edit.



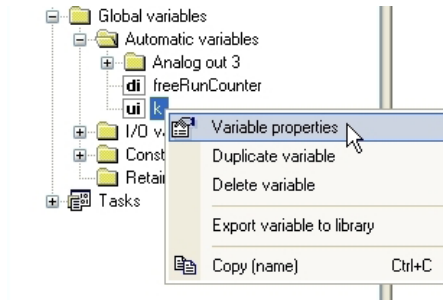
- 2) Double-click the name of the variable you want to edit: the global variables editor opens and lets you modify its definition.



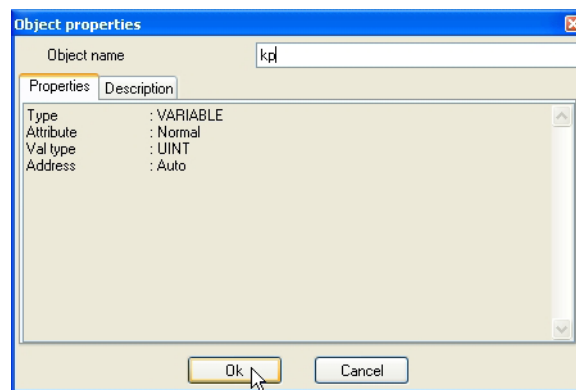


If you just want to change the name of the variable:

- 1) Open the *Variable properties* editor from the contextual menu which pops up when right-clicking the variable name in the project tree (alternatively, select the correspondent item in the *Project* menu).

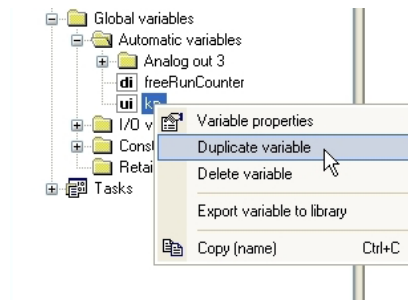


- 2) Change the variable name and confirm.

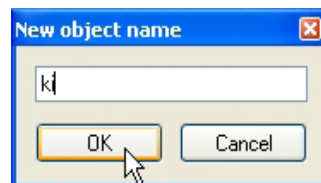


Finally, you can create a duplicate of the variable in this way:

- 1) Select *Duplicate variable* from the contextual menu (or the *Project* menu).



- 2) Enter the name of the new variable and confirm.

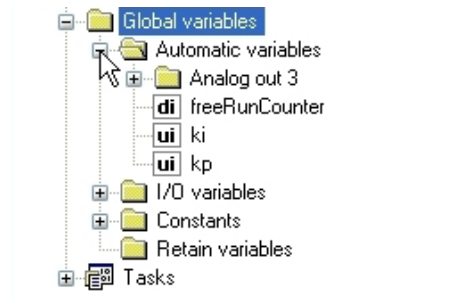




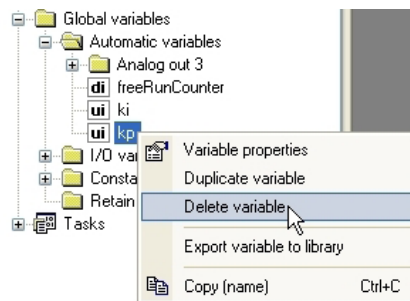
## 4.2.1.5 DELETING A GLOBAL VARIABLE

Follow this procedure to remove a global variable from your project:

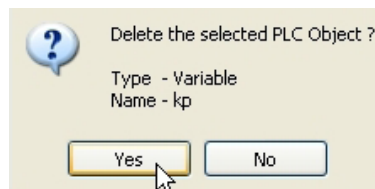
- 1) Open the folder in the *Project* tab of the workspace that contains the variable you want to delete.



- 2) Right-click the name of the variable you want to delete. A context menu appears referred to the selected variable.



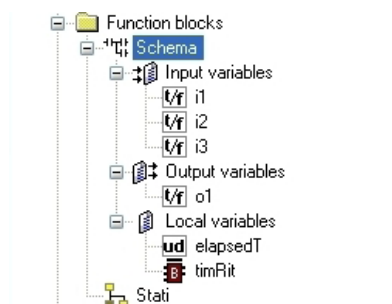
- 3) Click *Delete variable* in the context menu, then press *Yes* to confirm you choice.



## 4.2.2 LOCAL VARIABLES

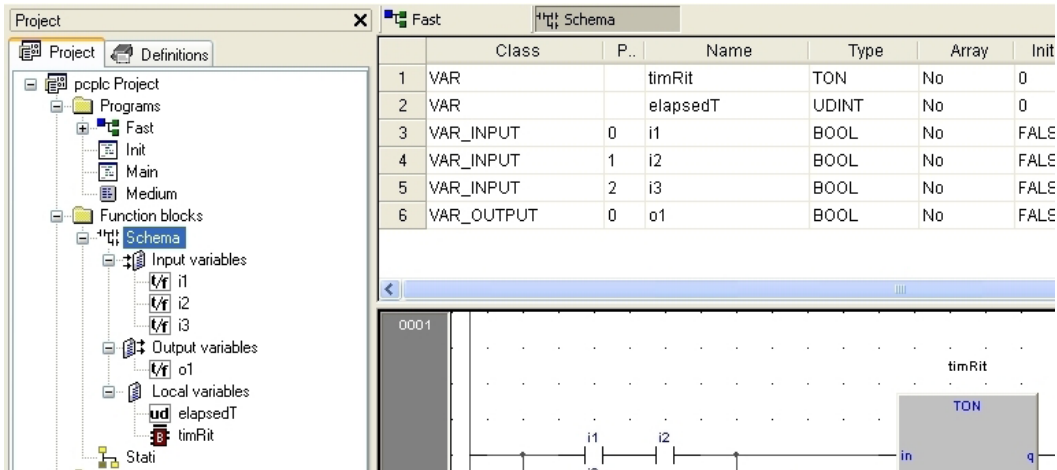
Local variables are declared within a POU (either program, or function, or function block), the module itself being the only project element which can refer to and access them.

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.

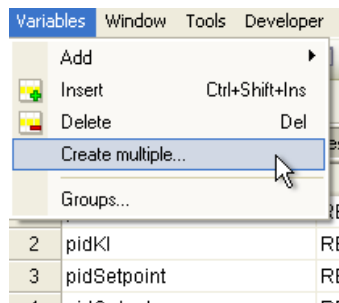


Refer to the corresponding section in this manual for details (see Paragraph 5.6.1.2).

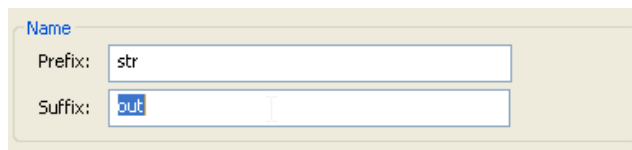
### 4.2.3 CREATE MULTIPLE

Follow this procedure to add multiple variables in one shot.

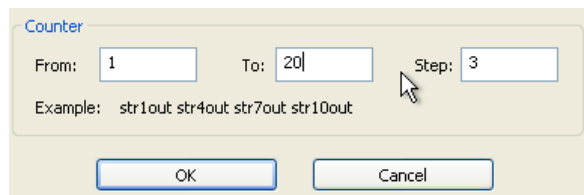
- 1) Select *Create Multiple* in the *Variables* menu.



- 2) Insert the prefix and the suffix to name the variables.



- 3) Select the type of the variables.
- 4) 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.



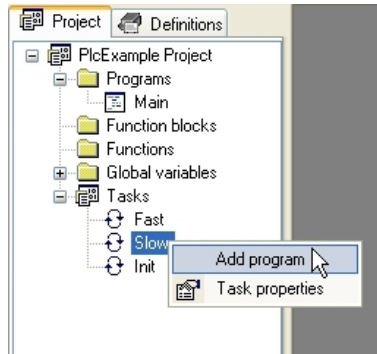


## 4.3 TASKS

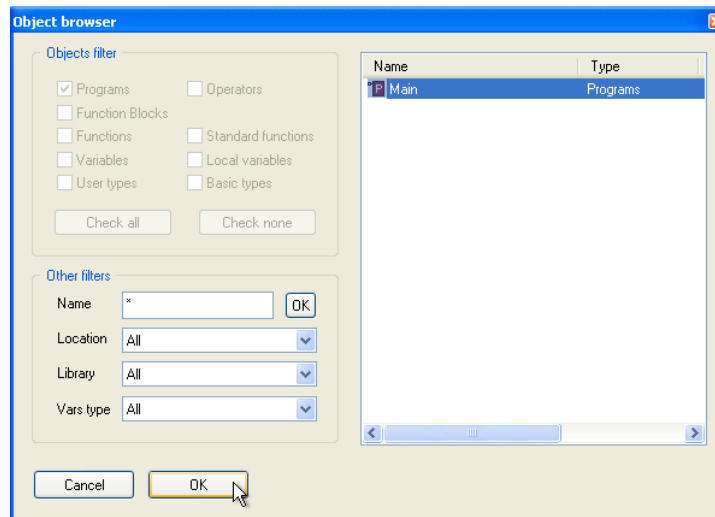
### 4.3.1 ASSIGNING A PROGRAM TO A TASK

Read the instructions below to know how to make a task execute a program.

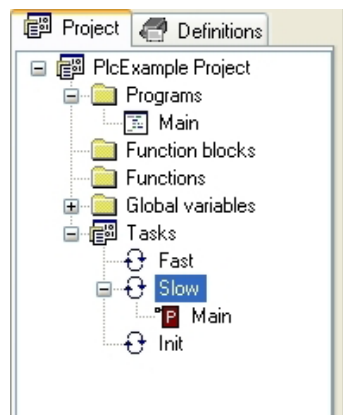
- 1) The tasks running on the target device are listed in the *Project* tab of the *Workspace* window. Right-click the name of the task you want to execute the program and choose *Add program* from the contextual menu.



- 2) Select the program you want the task to execute from the list which shows up and confirm your choice.

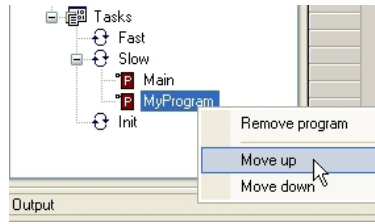


- 3) The program has been assigned to the task, as you can see in the project tree.





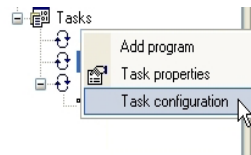
Note that you can assign more than a program to a task. From the contextual menu you can sort and, eventually, remove program assignments to tasks.



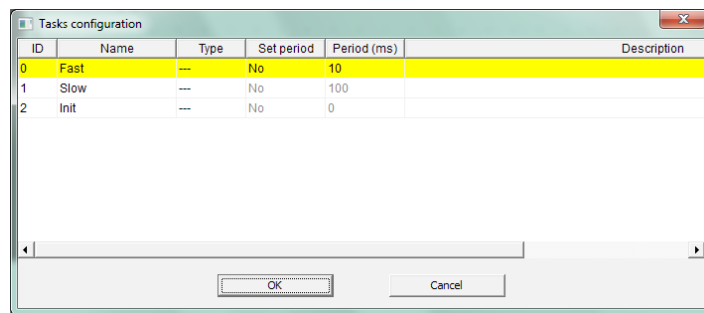
## 4.3.2 TASK CONFIGURATION

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

- 1) Double click on tasks item in the project tree or select the task configuration item in the contextual menu which pops up, if you right-click on a task.



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



## 4.4 DERIVED DATA TYPES

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

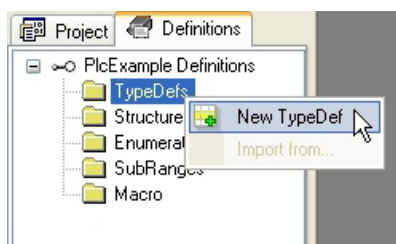
### 4.4.1 TYPEDEFS

The following paragraphs show you how to manage typedefs.

#### 4.4.1.1 CREATING A NEW TYPEDEF

In order to define a new typedef follow this procedure:

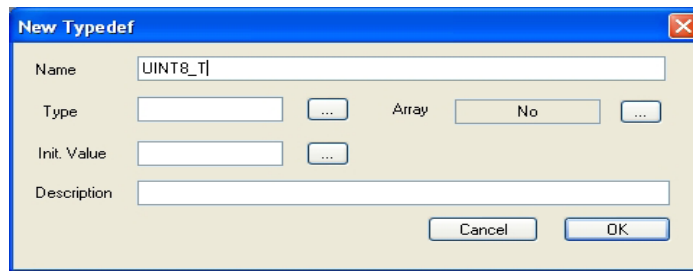
- 1) Right-click the *TypeDefs* folder and choose *New TypeDef* from the contextual menu.



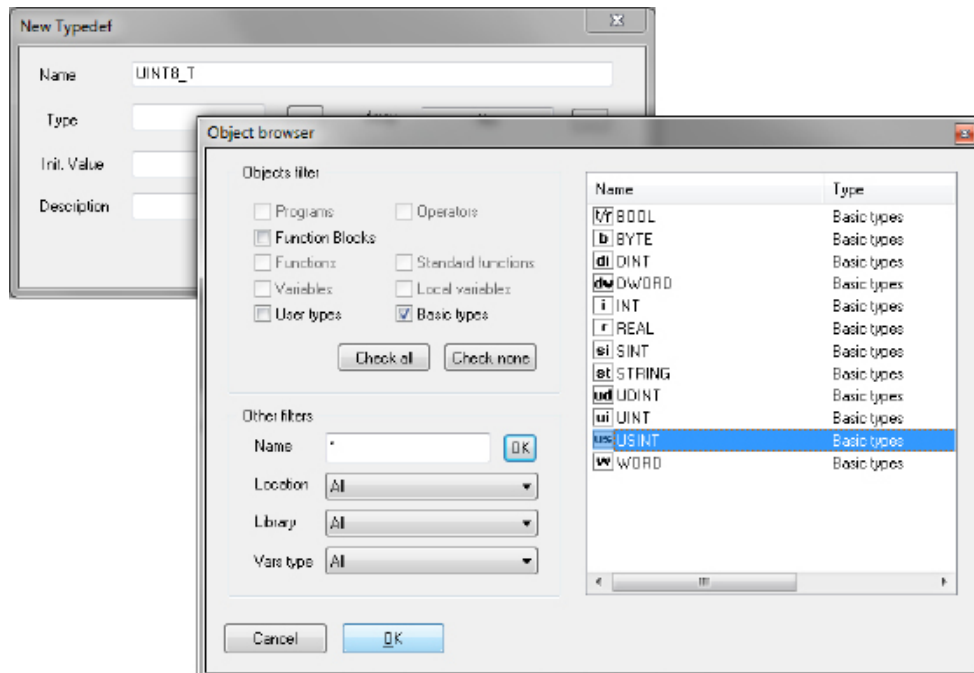
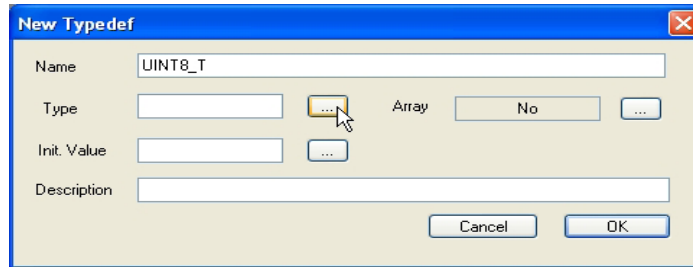




- 2) Type the name of the typedef.

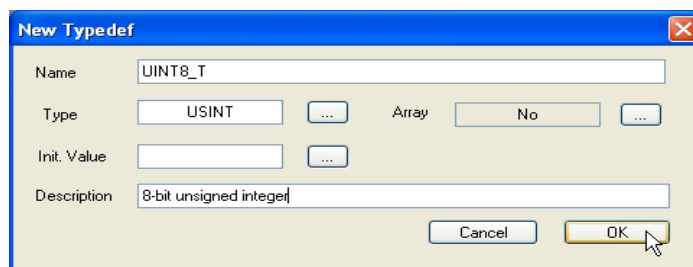


- 3) Select the type you are defining an alias for



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

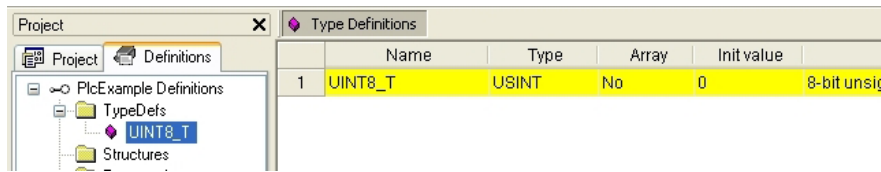
- 4) Enter a meaningful description (optional) and confirm the operation.





## 4.4.1.2 EDITING A TYPEDEF

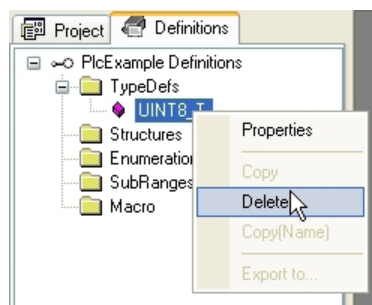
The typedefs of the project are listed under the *TypeDefs* folder. In order to edit a typedef you just have to double-click on its name.



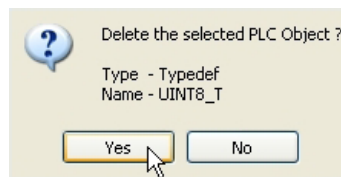
## 4.4.1.3 DELETING A TYPEDEF

To delete a typedef, follow this procedure:

- 1) Right-click the typedef name and choose *Delete* from the contextual menu.



- 2) Confirm your choice.



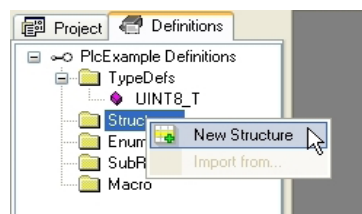
## 4.4.2 STRUCTURES

The following paragraphs show you how to manage structures.

### 4.4.2.1 CREATING A NEW STRUCTURE

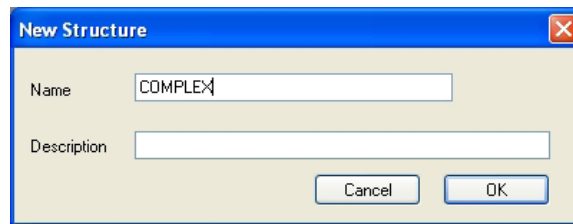
Follow this procedure to create a new structure:

- 1) Right-click the *Structures* folder and choose *New structure* from the contextual menu.

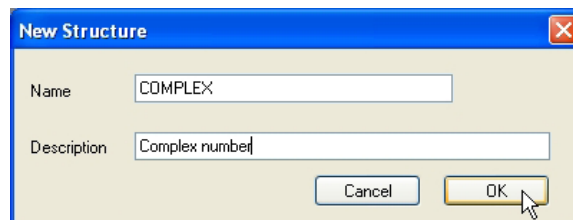




- 2) Type the name of the structure.

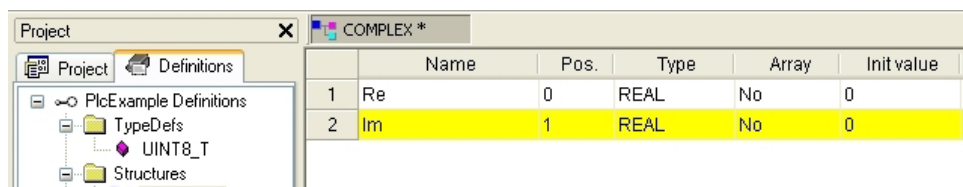


- 3) Enter a meaningful description and confirm the operation.



## 4.4.2.2 EDITING A STRUCTURE

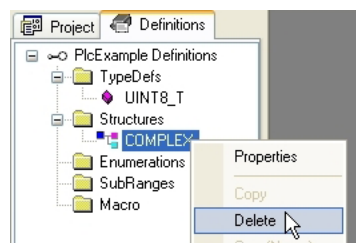
The structures of the project are listed under the *Structures* folder. In order to edit a structure (for example, to define its fields) you have to double-click on its name.



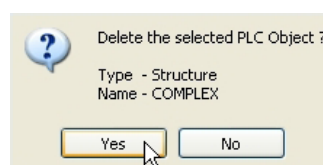
## 4.4.2.3 DELETING A STRUCTURE

Follow this procedure to delete a structure:

- 1) Right-click the structure name and choose *Delete* from the contextual menu.



- 2) Confirm your choice.





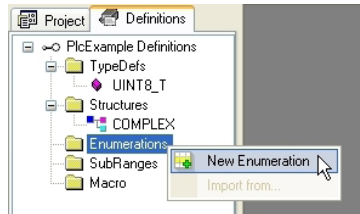
## 4.4.3 ENUMERATIONS

The following paragraphs show you how to manage enumerations.

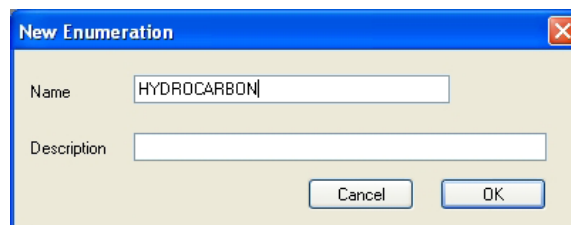
### 4.4.3.1 CREATING A NEW ENUMERATION

Follow this procedure to create a new enumeration:

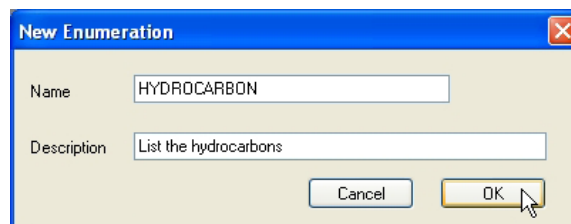
- 1) Right-click the *Enumerations* folder and choose *New enumeration* from the contextual menu.



- 2) Type the name of the enumeration.

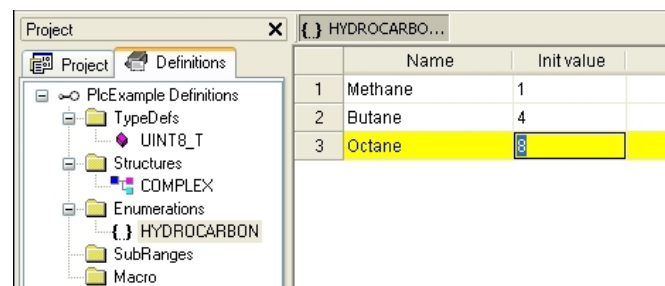


- 3) Enter a meaningful description and confirm the operation.



### 4.4.3.2 EDITING AN ENUMERATION

The enumerations of the project are listed under the *Enumerations* folder. In order to edit an enumeration (for example, to define its values) you have to double-click on its name.

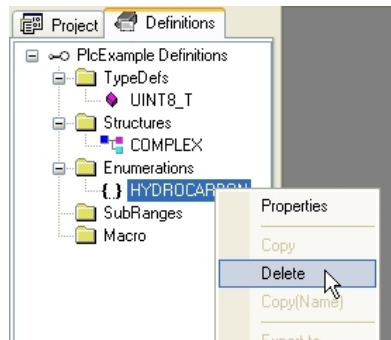




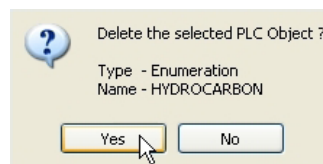
## 4.4.3.3 DELETING AN ENUMERATION

Follow this procedure to delete an enumeration:

- 1) Right-click the enumeration name and choose *Delete* from the contextual menu.



- 2) Confirm your choice.



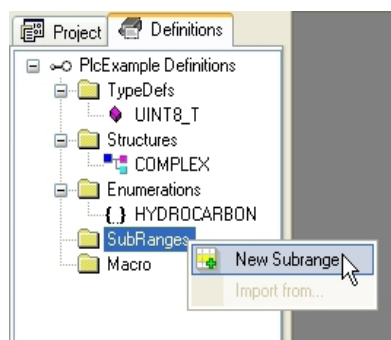
## 4.4.4 SUBRANGES

The following paragraphs show you how to manage subranges.

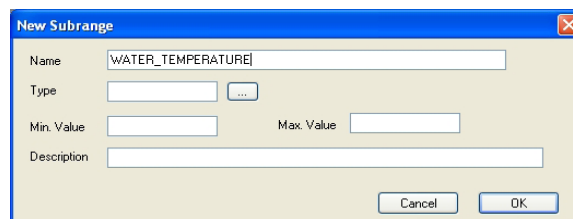
### 4.4.4.1 CREATING A NEW SUBRANGE

Follow this procedure to create a new subrange:

- 1) Right-click the *SubRanges* folder and choose *New Subrange* from the contextual menu.



- 2) Type the name of the subrange.





- 3) Select the basic type for the subrange.

Name	Type
rdi DINT	Basic types
i INT	Basic types
si SINT	Basic types
udi UDINT	Basic types
ui UINT	Basic types
usi USINT	Basic types

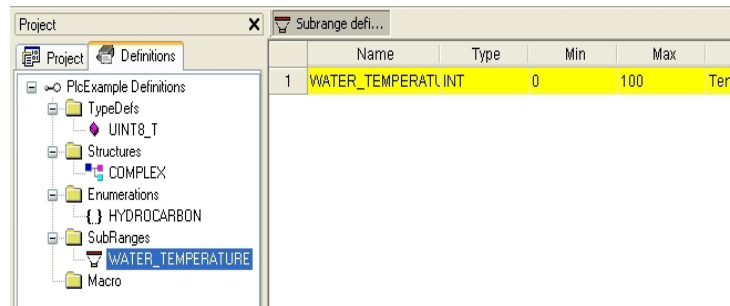
- 4) Enter minimum and maximum values of the subrange.

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



## 4.4.4.2 EDITING A SUBRANGE

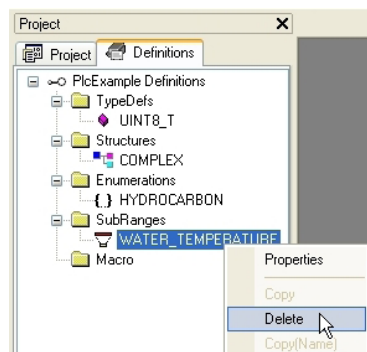
The subranges of the project are listed under the *Subranges* folder. In order to edit a subrange you just have to double-click on its name.



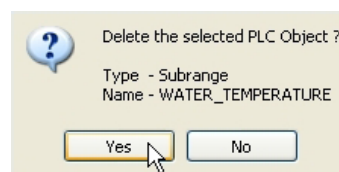
## 4.4.4.3 DELETING A SUBRANGE

Follow this procedure to delete a subrange:

- 1) Right-click the subrange name and choose *Delete* from the contextual menu.



- 2) Confirm your choice.

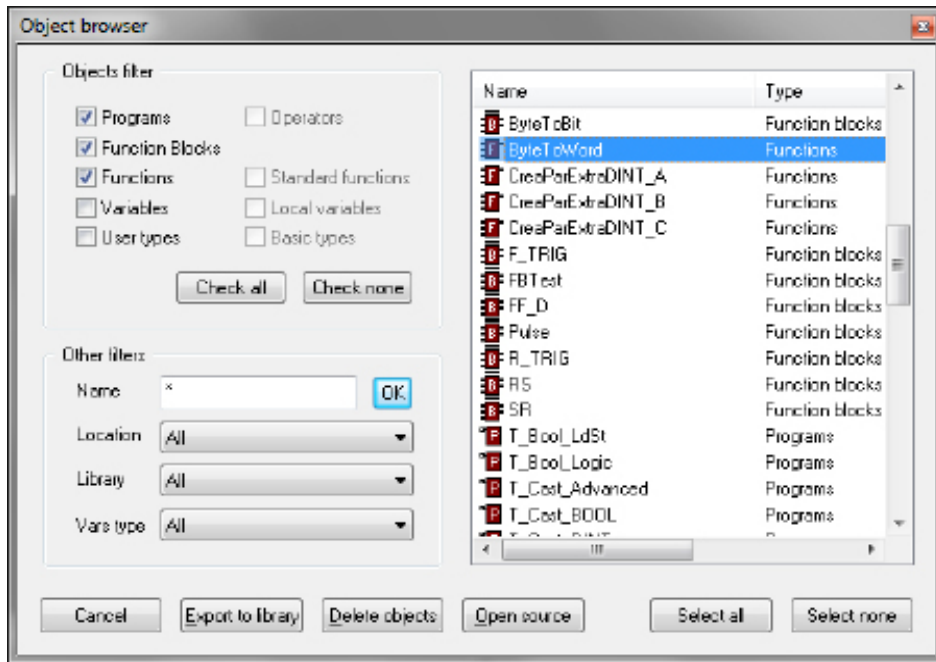


## 4.5 BROWSING THE PROJECT

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.

### 4.5.1 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 context are not the same.

*Object browser* can be opened in these three main ways:

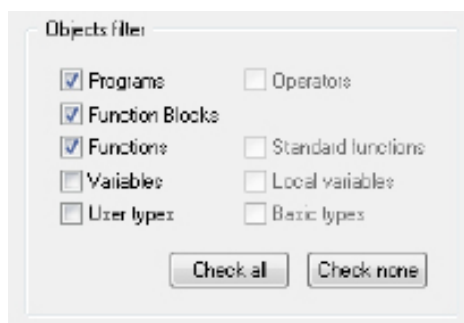
- *Browser mode*.
- *Import object mode*.
- *Select object mode*.

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

### 4.5.1.1 COMMON CHARACTERISTICS 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.





## SoMachine HVAC - Application

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

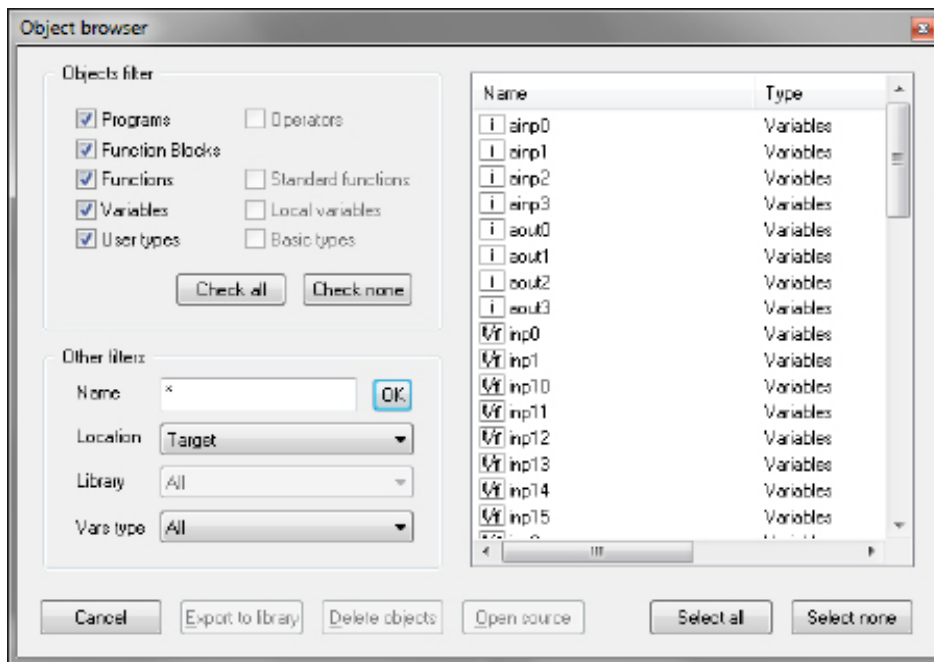
Selected objects can be also filtered by name, symbol location, specific library and var type.

Filters are all additive and are immediately applied after setting.

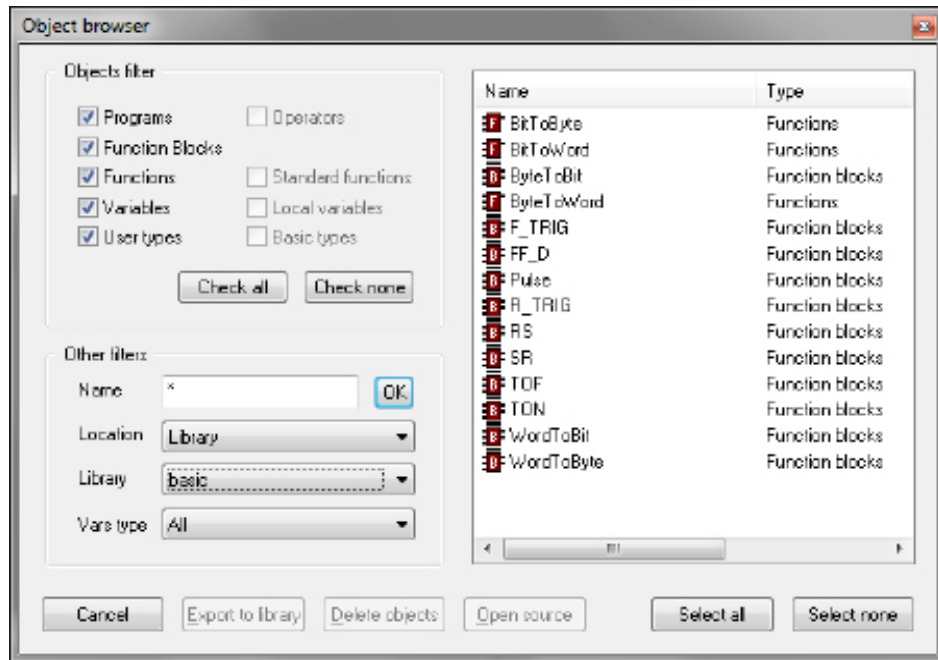
Name	
<b>Function</b>	Filters objects on the base of their name.
<b>Set of legal values</b>	All the strings of characters.
<b>Use</b>	Type a string to display the specific object whose name matches the string. Use the * wildcard if you want to display all the objects whose name contains the string in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



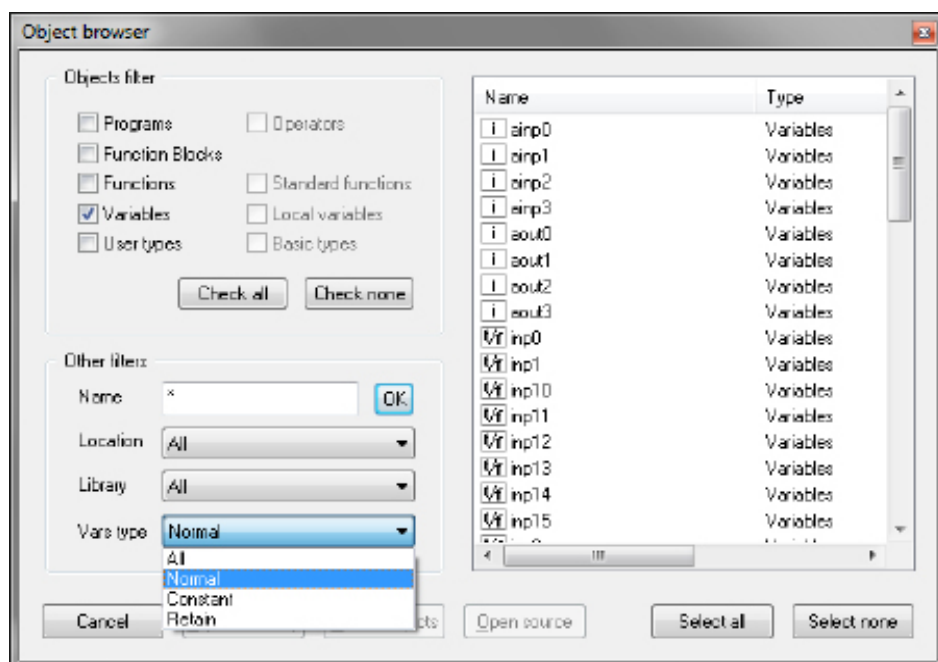
Symbol location	
<b>Function</b>	Filters objects on the base of their location.
<b>Set of legal values</b>	All, Project, Target, Library, Aux. Sources.
<b>Use</b>	All= Disables this filter. Project= Objects declared in the Application project. Target= Firmware objects. Library= Objects contained in a library. In this case, use simultaneously also the <i>Library</i> filter, described below. Aux sources= Shows aux sources only.
<b>Applies to</b>	All objects types.



Library	
<b>Function</b>	Completes the specification of a query on objects contained in libraries. The value of this control is relevant only if the <i>Symbol location</i> filter is set to <i>Library</i> .
<b>Set of legal values</b>	All, libraryname1, libraryname2, ...
<b>Use</b>	All= Shows objects contained in whatever library. LibrarynameN= Shows only the objects contained in the library named librarynameN.
<b>Applies to</b>	All objects types.

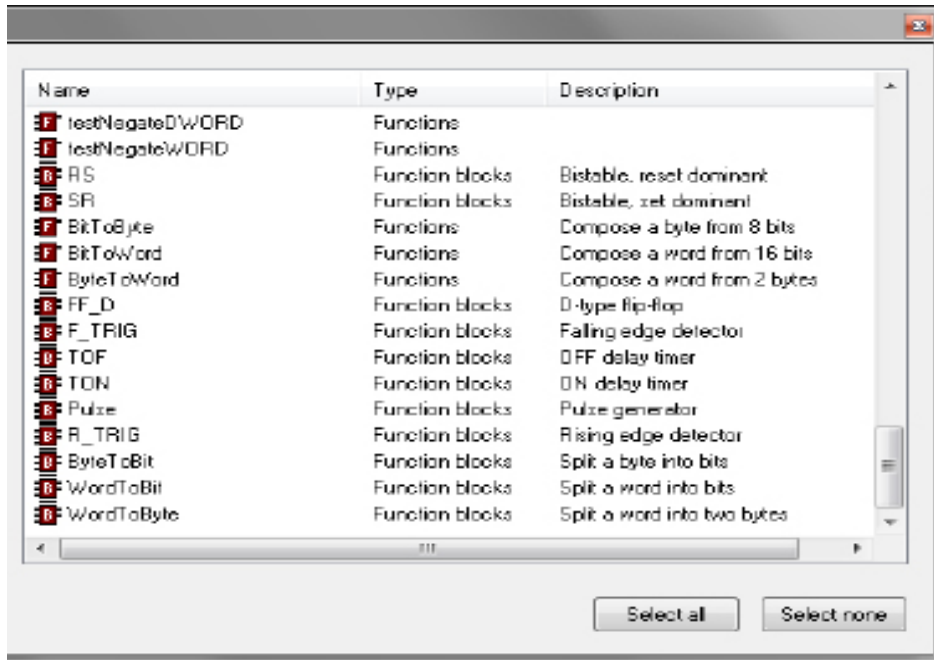


Vars Type	
<b>Function</b>	Filters global variables and system variables (also known as firmware variables) according to their type.
<b>Set of legal values</b>	All, Normal, Constant, Retain
<b>Use</b>	All= Shows all the global and system variables. Normal= Shows normal global variables only. Constant= Shows constants only. Retain= Shows retain variables only.
<b>Applies to</b>	Variables.





## Object list



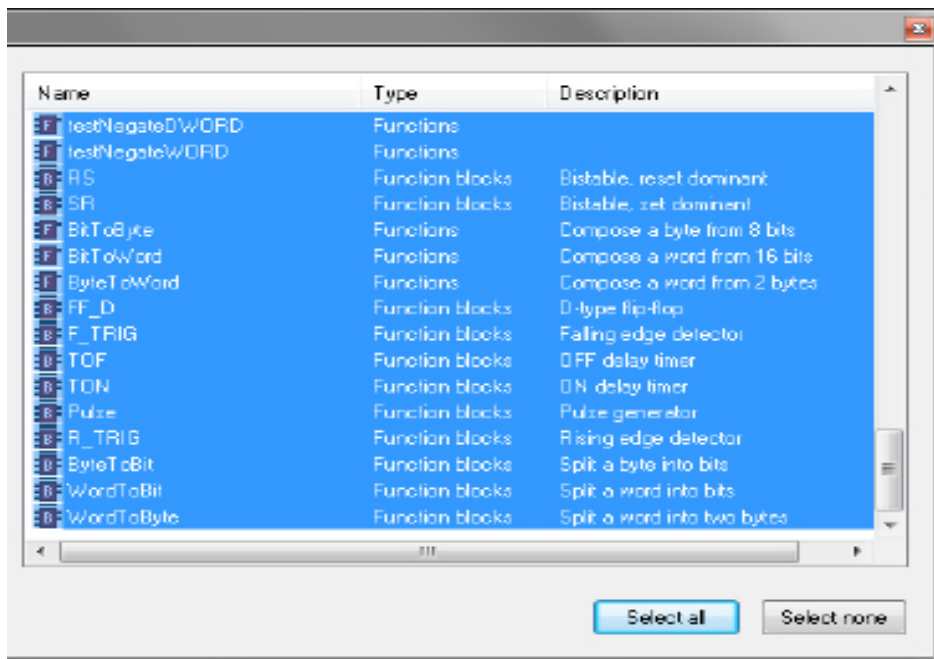
*Object list* shows all the filtered objects. List can be ordered in ascending or descending way by clicking on the header of the column. So it is possible to order items by *Name*, *Type*, or *Description*.

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 an 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.

## Close dialog

You have two options for closing the *Object browser*:

- Press the button near the right-end border of the caption bar.

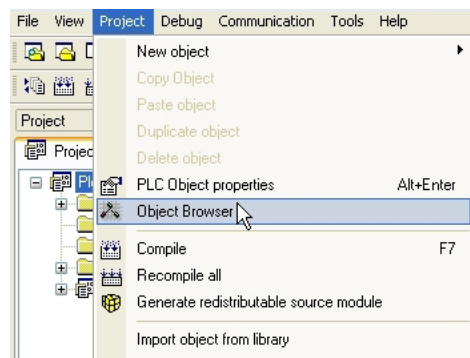


- Press the *Cancel/OK* button below the filter box.



### 4.5.1.2 USING OBJECT BROWSER AS A BROWSER

To use *Object browser* in this way click on *Object browser* in the *Project* menu. This causes the *Object browser* dialog box to appear, which lets you navigate between the objects of the currently open project.



## Available objects

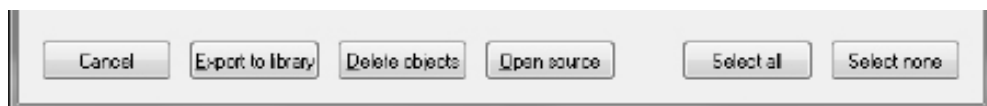
In this mode you can list objects of these types:

- 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	
<b>Function</b>	Opens the editor by which the selected object was created and displays the relevant source code.
<b>Use</b>	<p>If the object is a program, or a function, or a function block, this button opens the relevant source code editor.</p> <p>If the object is a variable, then this button opens the variable editor.</p> <p>Select the object whose editor you want to open, then click on the <i>Open source</i> button.</p>

Export to library	
<b>Function</b>	To export an object to a library.
<b>Use</b>	Select the objects you want to export, then press the <i>Export to library</i> button.

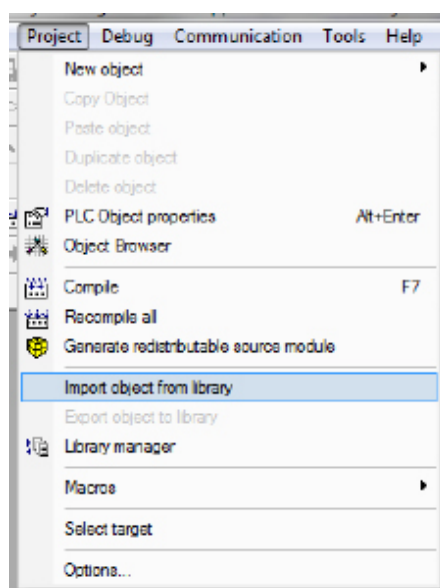
Delete objects	
<b>Function</b>	Allows you to delete an object.
<b>Use</b>	Select the object you want to delete, then press the <i>Delete object</i> button.

### Multi selection

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

### 4.5.1.3 USING OBJECT BROWSER FOR IMPORT

Object browser is also used to support objects importation in the project from a desired external library. Select *Import object from library* in the *Project* menu, then choose the desired library.





## Available objects

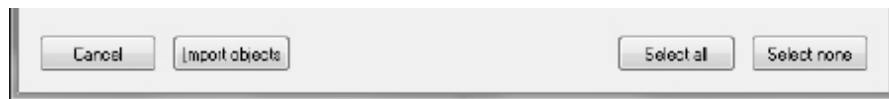
In this mode you can list objects of these types:

- 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.

### 4.5.1.4 USING OBJECT BROWSER FOR OBJECT SELECTION

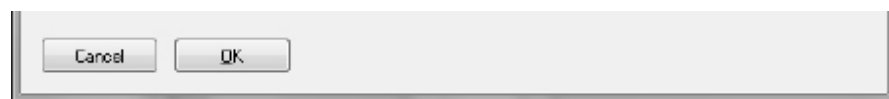
Object browser dialog is useful for many operations that requires the selection of a single PLC object. So Object browser can be used to select the program to add to a task, to select the type of a variable, to select an item to find in the project, etc..

## 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

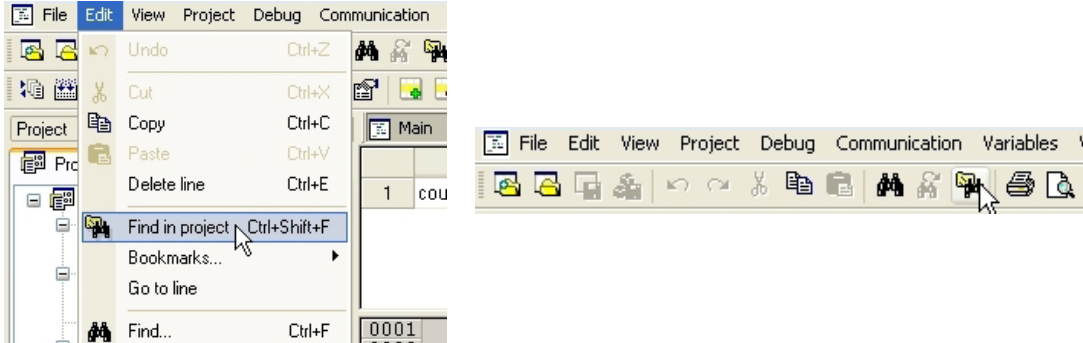
Multi selection is not allowed for this mode, *Select all* and *Select none* buttons are not visible.



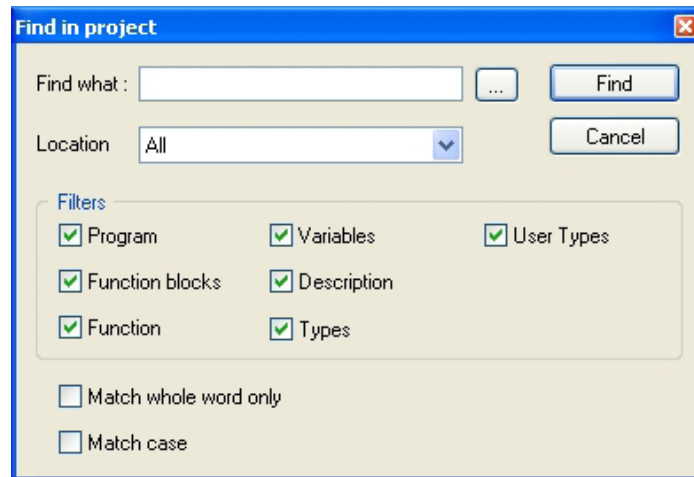
## 4.5.2 SEARCHING WITH THE FIND IN PROJECT COMMAND

The *Find in project* command retrieves all the instances of a specified character string in the project. Follow the procedure to use it correctly.

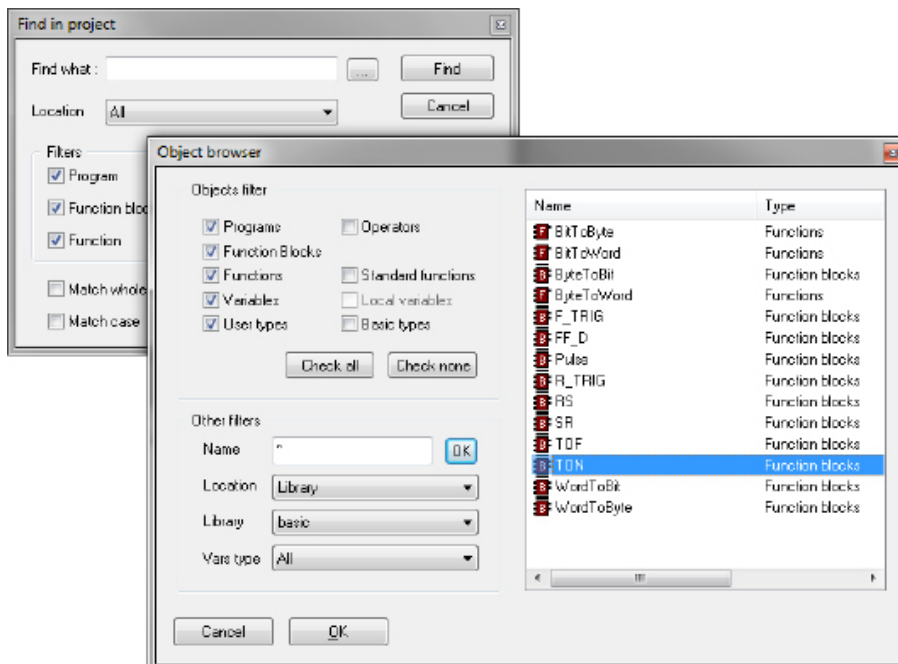
- 1) Click *Find in project...* in the *Edit* menu or in the *Main* toolbar.



This causes the following dialog box to appear.



- 2) In the *Find what* text box, type the name of the object you want to look for.

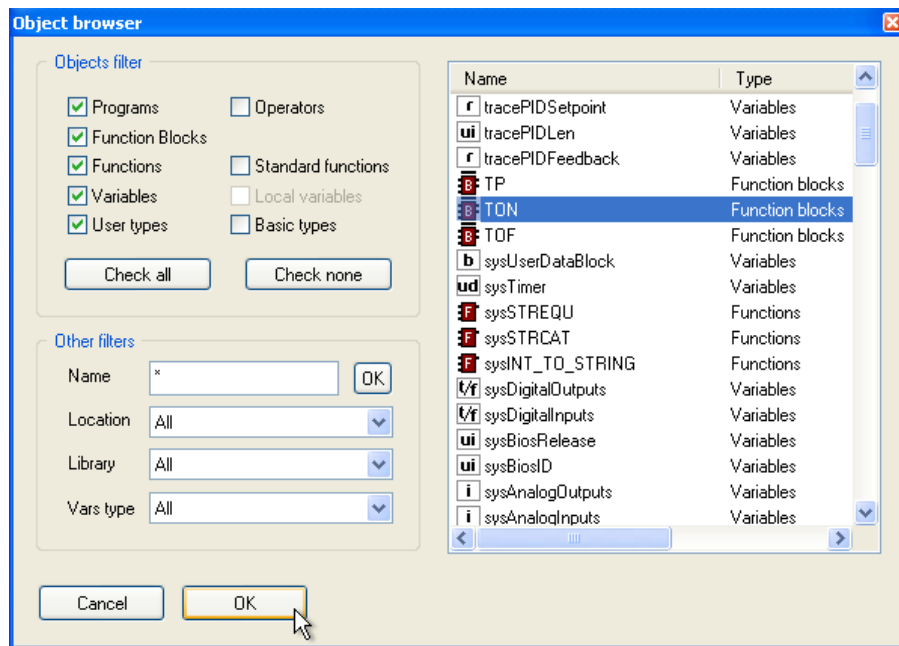
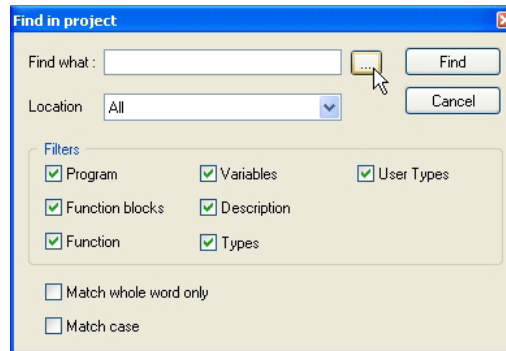




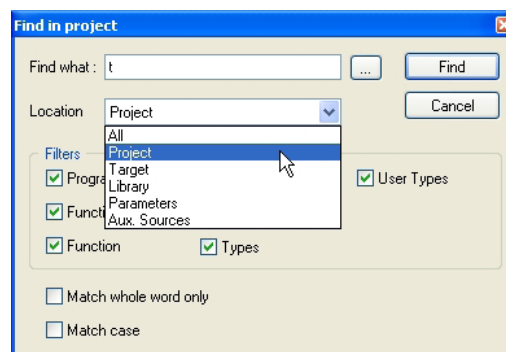


## SoMachine HVAC - Application

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.



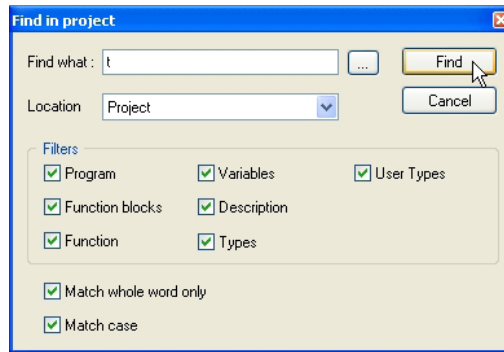
- 3) Select one of the values listed in the *Location* combo box, so as to specify a constraint on the location of the objects to be inspected.



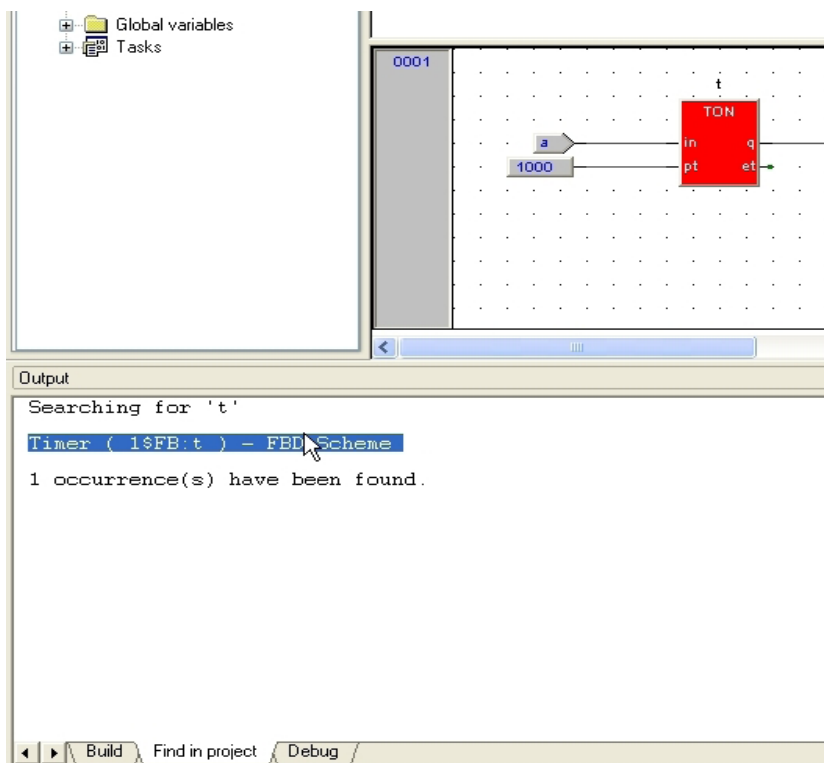
- 4) The frame named *Filters* contains 7 checkboxes, each of which, if ticked, enables research of the string among the object it refers to.
- 5) Tick *Match whole word only* if you want to compare your string to entire word only.
- 6) Tick *Match case* if you want your search to be case-sensitive.



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



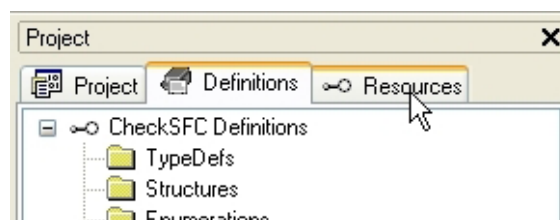
The results will be printed in the *Find in project* tab of the *Output* window.



## 4.6 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. 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.

This chapter focuses on all these editors.

### 5.1 INSTRUCTION LIST (IL) EDITOR

```
0004
0005      LD      sysIq
0006      MUL     sysIq
0007      ST      IqDW
0008      SHR     16#04
0009      ADD     addIqSq
0010      ST      addIqSq
0011
0012      LD      sysId
0013      MUL     sysId
0014      ST      IdDW
0015      SHR     16#04
0016      ADD     addIdSq
0017      ST      addIdSq
0018
```

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

#### 5.1.1 EDITING FUNCTIONS

The IL editor is endowed with functions common to most editors running on a Windows platform, namely:

- Text selection.
- *Cut*, *Copy*, and *Paste* operations.
- *Find and Replace* functions.
- Drag-and-drop of selected text.

Many of these functions are accessible through the *Edit* menu or through the *Main* toolbar.

#### 5.1.2 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.



## 5.1.3 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.

## 5.1.4 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.

### 5.1.4.1 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.



### 5.1.4.2 JUMPING TO A BOOKMARK

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

### 5.1.4.3 REMOVING A BOOKMARK

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

## 5.2 STRUCTURED TEXT (ST) EDITOR

```

0001
0002      IqDW := sysIq * sysIq ;
0003      addIqSq := addIqSq + SHR( IqDW, 16#04 ) ;
0004
0005      IdDW := sysId * sysId ;
0006      addIdSq := addIdSq + SHR( IdDW, 16#04 ) ;
0007
0008      IF a > b THEN
0009          a := c;
0010          n := a * b * c;
0011      END_IF;
0012
0013

```

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

### 5.2.1 CREATING AND EDITING ST OBJECTS

See the Creating and Editing POU's section (Paragraphs 4.1.1 and 4.1.2).

### 5.2.2 EDITING FUNCTIONS

The ST editor is endowed with functions common to most editors running on a Windows platform, namely:

- Text selection.
- *Cut*, *Copy*, and *Paste* operations.
- *Find and Replace* functions.
- Drag-and-drop of selected text.

Many of these functions are accessible through the *Edit* menu or through the *Main* toolbar.



## 5.2.3 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.

## 5.2.4 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.

## 5.2.5 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.

### 5.2.5.1 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.



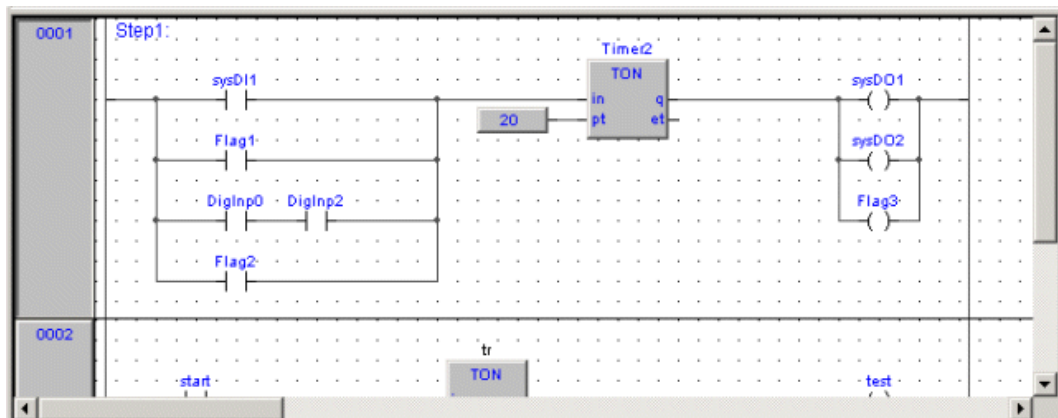
### 5.2.5.2 JUMPING TO A BOOKMARK

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

### 5.2.5.3 REMOVING A BOOKMARK

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

## 5.3 LADDER DIAGRAM (LD) EDITOR





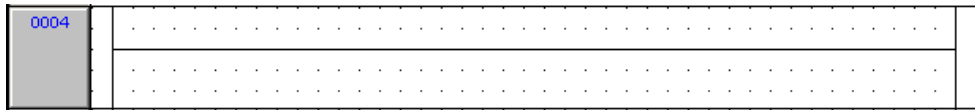
The LD editor allows you to code and modify POU's using LD (i.e. Ladder Diagram), one of the IEC-compliant languages.

### 5.3.1 CREATING A NEW LD DOCUMENT

See the Creating and Editing POU's section (Paragraphs 4.1.1 and 4.1.2).

### 5.3.2 ADDING/REMOVING NETWORKS

Every 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.

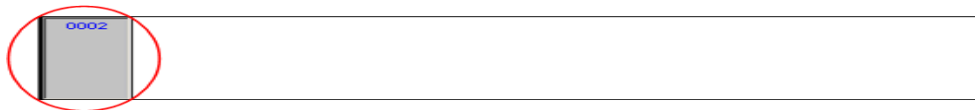
On the new LD network a horizontal line links the two power rails. It is called the "power link". On this link, all the LD elements (contacts, coils and blocks) are to be placed.

You can perform the following operations on networks:

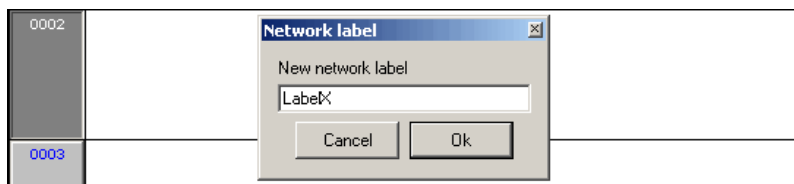
- To add a new blank network, click *Network>New* in the *Scheme* menu, or press one of the equivalent buttons in the *Network* toolbar.
- To assign a label to a selected network, give the *Network>Label* command from the *Scheme* menu. This enables jumping to the labeled network.
- To display a background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

### 5.3.3 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.

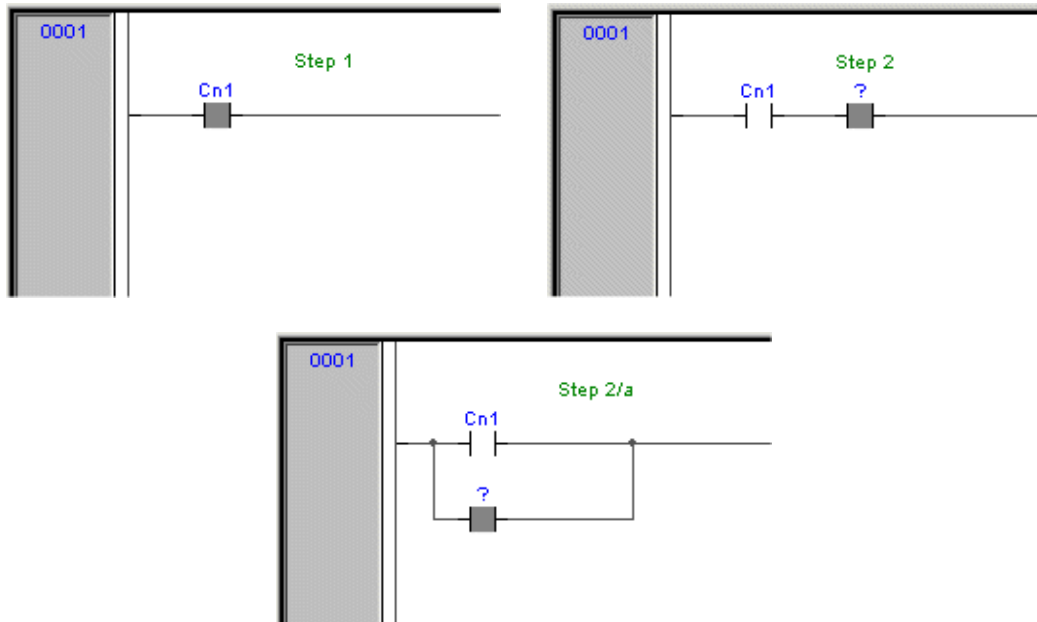




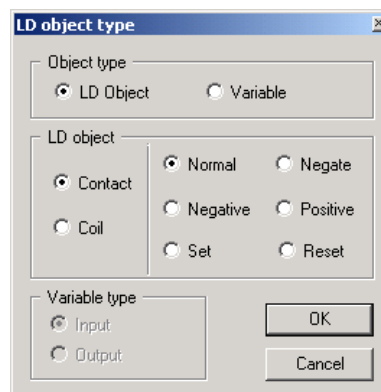
## 5.3.4 INSERTING CONTACTS

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

- Select a contact, a block or a connection. Select the insertion mode between serial or parallel (using the button on the *LD* toolbar or the *Scheme* menu). Insert the appropriate contact (using the button on the *LD* toolbar, the *Scheme>Object>New* or the pop-up menu option). For serial insertion, the new contact will be inserted on the 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/blocks can be selected before performing the insertion. The new contact will be inserted at the endpoints of the selection block.



- Drag a boolean variable to the desired place over a connection. For example, global variables can be taken from the *Workspace* window, whereas local variables can be selected from the local variables editor. The dialog box shown below will appear, requesting to define whether the variable should be inserted as a contact, coil or variable (like FBD schemes). Choose the appropriate contact type. Contacts inserted with drag and drop will always be inserted in series.

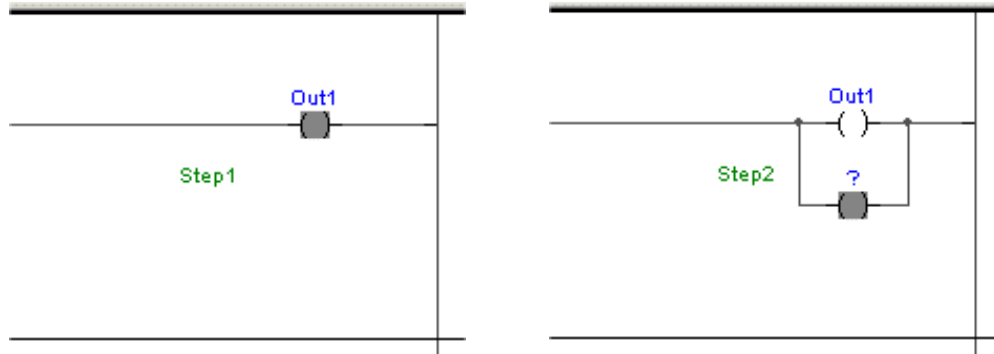




## 5.3.5 INSERTING COILS

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

- Press one of the coil buttons in the *LD* toolbar. The new coil will be inserted and linked to the right power rail. If other coils 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. For example, global variables can be taken from the *Workspace* window, whereas local variables can be selected from the local variables editor. A dialog box will appear, requesting to indicate whether the variable should be inserted as a contact, coil or variable. Choose the appropriate coil type.

## 5.3.6 INSERTING BLOCKS

Operators, functions and function blocks can be inserted into an LD network in the following modes:

- On the power link, as contacts and coils.
- Outside the power link (to do so, follow the indications as for the FBD blocks).

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

- Select a contact, connection or block then click *Object>New* in the *Scheme* menu.
- Select a contact, connection or block, then press the *New block* button in the *FBD* toolbar, 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 (from the *Workspace* window, the *Libraries* window or the local variables editor) over the desired connection.

The two upper pins will be connected to the power link. The *EN/ENO* pins should be activated before the insertion.

## 5.3.7 EDITING COILS AND CONTACTS PROPERTIES

The type of a contact (normal, negated) or a coil (normal, negated, set, reset) 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 with the right mouse button, then select *Properties*.

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





## 5.3.8 EDITING NETWORKS

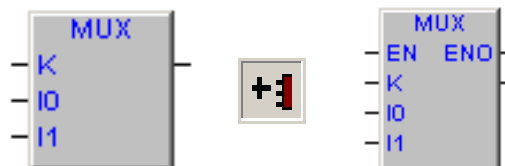
The LD editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

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

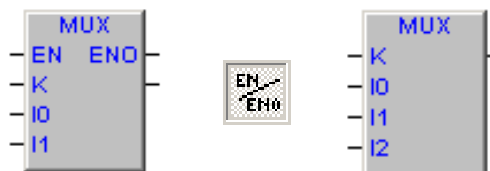
All the mentioned functions are accessible through the *Edit* menu or through the *Main* toolbar.

## 5.3.9 MODIFYING PROPERTIES OF BLOCKS

- Click *Increment pins +* in the *Scheme* menu, or press the *Inc pins* button in the *FBD* toolbar, to increment the number of input pins of some operators and embedded functions.



- Click *Enable EN/ENO pins* in the *Scheme* menu, or press the *EN/ENO* button in the *FBD* toolbar, to display the enable input and output pins.



- Click *Object . Instance name* in the *Scheme* menu, or press the *FBD properties* button in the *FBD* toolbar, to change the name of an instance of a function block.

## 5.3.10 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:

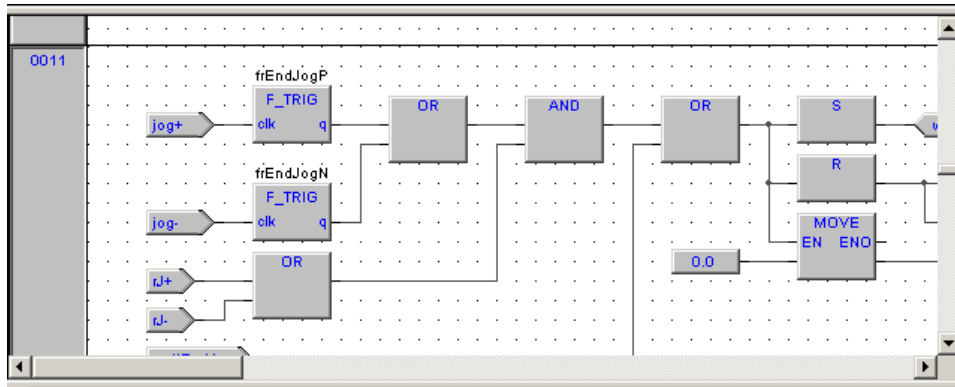
- Click *Object>Open source* in the *Scheme* menu, or press the *View source* button in the *FBD* toolbar, to open the source code of a block.
- Click *Object properties* in the *Scheme* menu, or press the *FBD properties* button in the *FBD* toolbar, to see properties and input/output pins of the selected block.

## 5.3.11 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.



## 5.4 FUNCTION BLOCK DIAGRAM (FBD) EDITOR



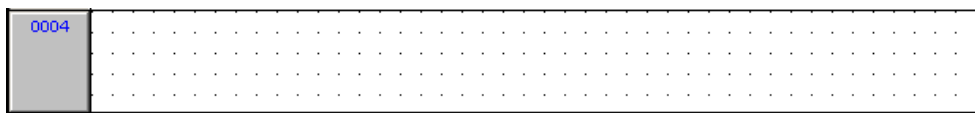
The FBD editor allows you to code and modify POUs using FBD (i.e. Function Block Diagram), one of the IEC-compliant languages.

### 5.4.1 CREATING A NEW FBD DOCUMENT

See the Creating and editing POUs section (Paragraphs 4.1.1 and 4.1.2).

### 5.4.2 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.

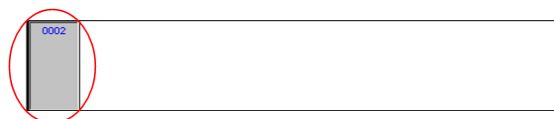


You can perform the following operations on networks:

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

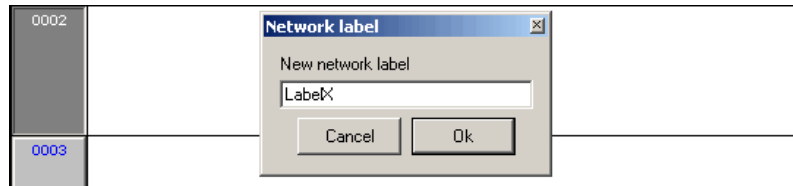
### 5.4.3 LABELING NETWORKS

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.

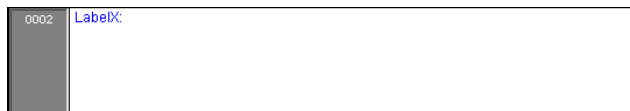




This causes a dialog box to appear, which lets you type the label you want to associate with the selected network.



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



## 5.4.4 INSERTING AND CONNECTING BLOCKS

This paragraph shows you how to build a network.

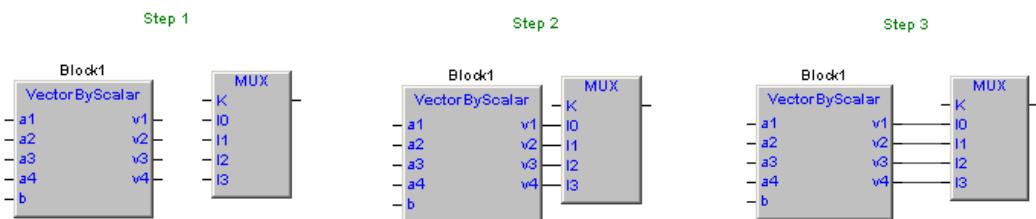
Add a block to the blank network, by applying one of the following options:

- Click *Object>New* in the *Scheme* menu.
- Press the *New block* button in the *FBD* toolbar, 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 *Connection mode* in the *Edit* menu, or press the *Connection* button in the *FBD* toolbar, 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 autoconnection mode by clicking *Autoconnect* in the *Scheme* menu, or by pressing the *Autoconnect* button in the *Network* toolbar. 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 *Delete invalid connection* in the *Scheme* menu, or type *Ctrl+B* on your keyboard.



## 5.4.5 EDITING NETWORKS

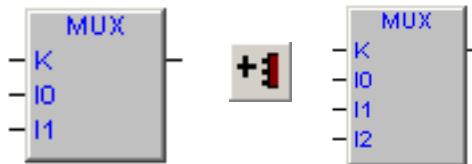
The FBD editor is endowed with functions common to most graphic applications running on a Windows platform, namely:

- 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.
- *Cut*, *Copy* and *Paste* operations of a single block as well as of a set of blocks.
- Drag-and-drop.

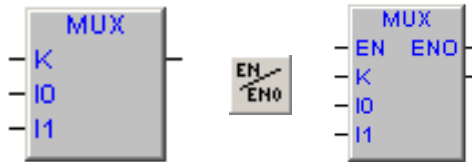
All the mentioned functions are accessible through the *Edit* menu or through the *Main* toolbar.

## 5.4.6 MODIFYING PROPERTIES OF BLOCKS

- Click *Increment pins +* in the *Scheme* menu, or press the *Inc pins* button in the *FBD* toolbar, to increment the number of input pins of some operators and embedded functions.



- Click *Enable EN/ENO pins* in the *Scheme* menu, or press the *EN/ENO* button in the *FBD* toolbar, to display the enable input and output pins.



- Click *Object>Instance name* in the *Scheme* menu, or press the *FBD properties* button in the *FBD* toolbar, to change the name of an instance of a function block.

## 5.4.7 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 *Object>Open source* in the *Scheme* menu, or press the *View source* button in the *FBD* toolbar, to open the source code of a block.
- Click *Object properties* in the *Scheme* menu, or press the *FBD properties* button in the *FBD* toolbar, to see properties and input/output pins of the selected block.

## 5.4.8 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.



## 5.5 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.

### 5.5.1 CREATING A NEW SFC DOCUMENT

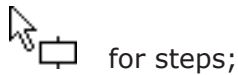
See the creating and editing POUs section (Paragraphs 4.1.1 and 4.1.2).

### 5.5.2 INSERTING A NEW SFC ELEMENT

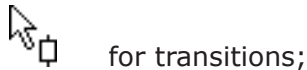
You can apply indifferently one of the following procedures:

- Click *Object>New* in the *Scheme* menu, then select the type of the new element (action, transition, or jump).
- Press the *New step*, *Add Transition* or *Add Jump* button in the *SFC* toolbar.

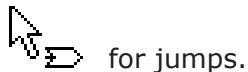
In either case, the mouse pointer changes to:



for steps;



for transitions;



for jumps.

### 5.5.3 CONNECTING SFC ELEMENTS

Follow this procedure to connect SFC blocks:

- Click *Connection mode* in the *Edit* menu, or press the *Connection* button in the *FBD* toolbar, 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 autoconnection mode by clicking *Autoconnect* in the *Scheme* menu, or by pressing the *Autoconnect* button in the *Network* toolbar. 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.

### 5.5.4 ASSIGNING AN ACTION TO A STEP

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

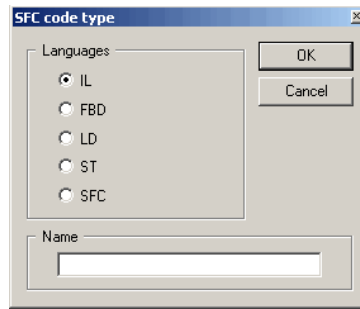
#### 5.5.4.1 WRITING THE CODE OF AN ACTION

To start implementing an action, you need to open an editor. Do it by applying one of the following procedures:

- Click *Code object>New action* in the *Scheme* menu.
- Right-click on the name of the SFC POU in the *Workspace* window. A context menu appears, from which you can select the *New Action* command.



In either case, Application displays a dialog box like the one shown below.

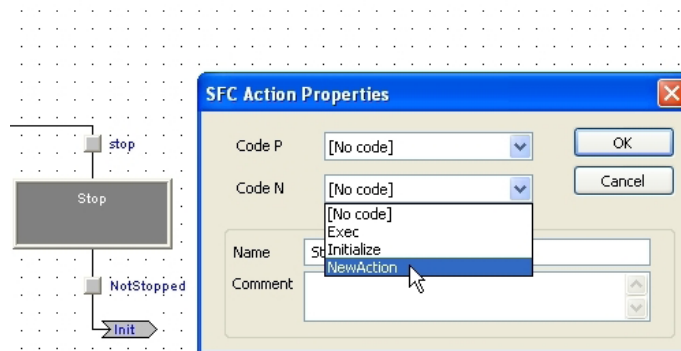


Select one of the languages and type the name of the new action in the text box at the bottom of the dialog box. Then either confirm by pressing *OK*, or quit by clicking *Cancel*. If you press *OK*, Application opens automatically the editor associated with the language you selected in the previous 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.

### 5.5.4.2 ASSIGNING AN ACTION TO A STEP

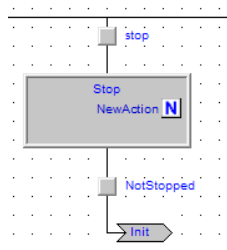
When you have finished writing the code, double-click the step you want to assign the new action to. This causes the following dialog box to appear.



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.



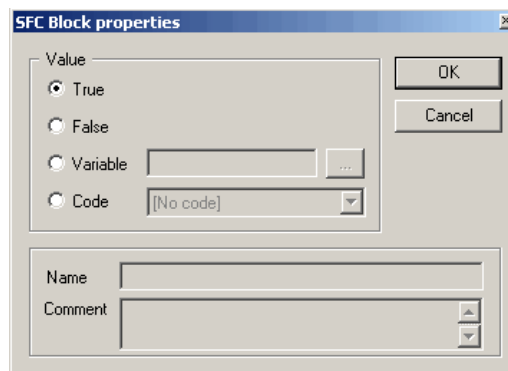


If later you need to edit the source code of the action, you can just double-click these letters. Alternatively, you can double-click the name of the action in the *Actions* folder of the *Workspace* window.

## 5.5.5 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.

## 5.5.6 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.

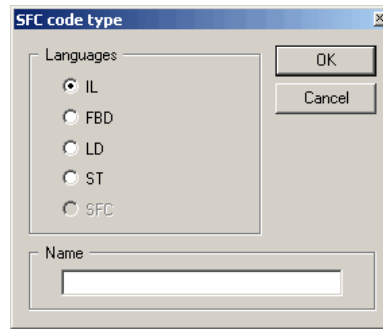
### 5.5.6.1 WRITING THE CODE OF A CONDITION

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

- Click *Code object>New transition* in the *Scheme* menu.
- Right-click on the name of the SFC POU in the *Workspace* window, then select the *New transition* command from the context menu that appears.



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



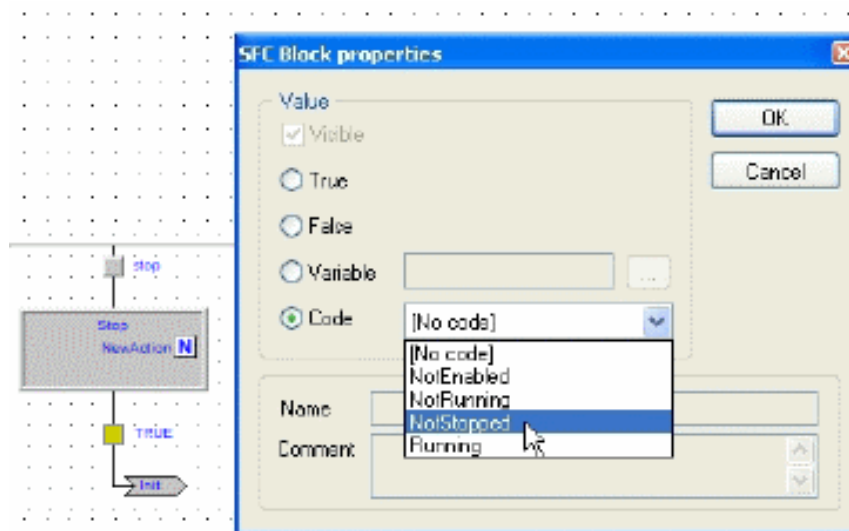
Note that you can use any language except SFC to code a condition. Select one of the languages and type the name of the new condition in the text box at the bottom of the dialog box. Then either confirm by pressing *OK*, or quit by clicking *Cancel*.

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.

## 5.5.6.2 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*.

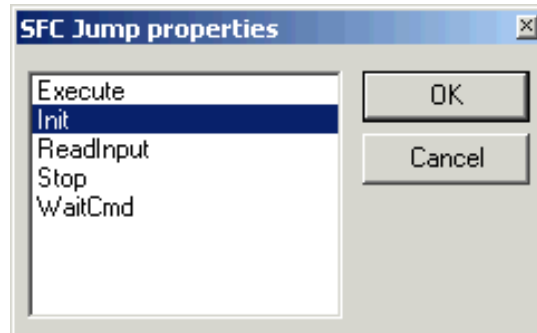
If later you need to edit the source code of the condition, you can double-click the name of the transition in the *Transitions* folder of the *Workspace* window.





## 5.5.7 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.



## 5.5.8 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.
- *Cut*, *Copy*, and *Paste* operations of a single block as well as of a set of blocks.
- Drag-and-drop.

Some of these functions are accessible through the *Edit* menu or through the *Main* toolbar.

## 5.6 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	
9	gA	BOOL	Auto		No	TRUE	..	
10	gB	REAL	Auto		[0..4]	5(0)	..	
11	gC	REAL	%MD60.20		No	1.0	..	
12	gD	INT	Auto		No	-74	CONSTANT	

The corresponding source code will look like this:

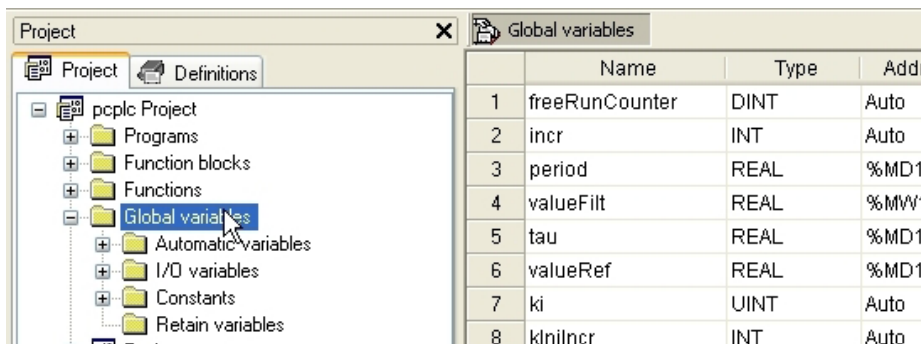
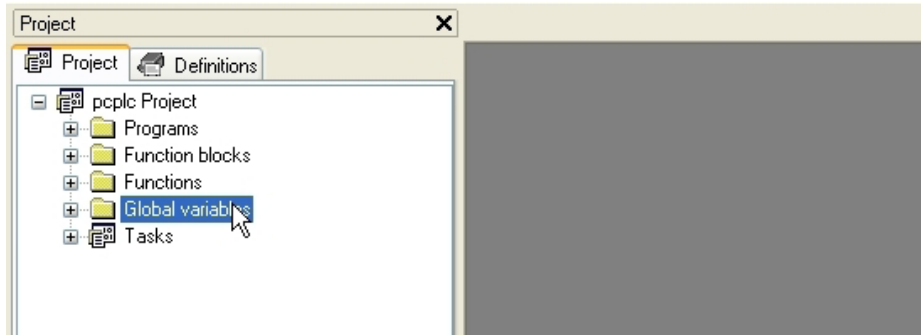
```
VAR_GLOBAL
  gA : BOOL := TRUE;
  gB : ARRAY[ 0..4 ] OF REAL;
  gC AT %MD60.20 : REAL := 1.0;
END_VAR
VAR_GLOBAL CONSTANT
  gD : INT := -74;
END_VAR
```



## 5.6.1 OPENING A VARIABLES EDITOR

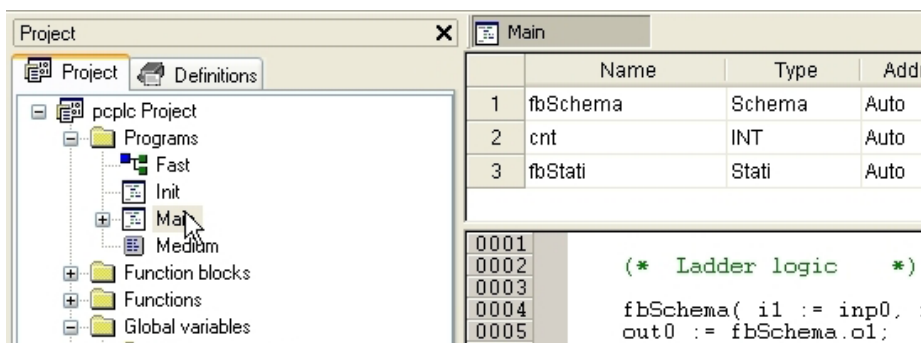
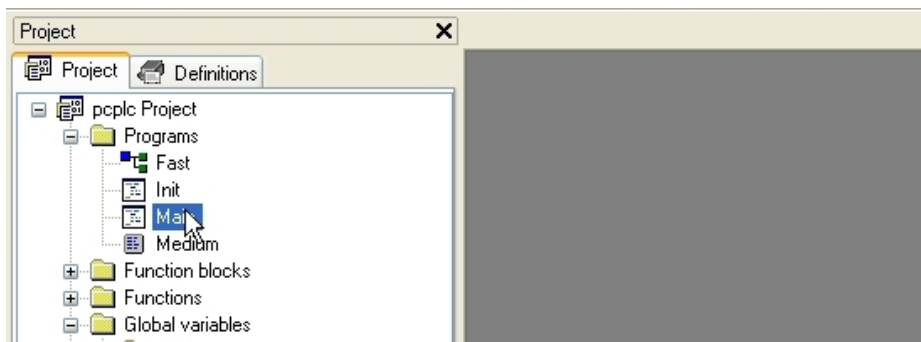
### 5.6.1.1 OPENING THE GLOBAL VARIABLES EDITOR

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



### 5.6.1.2 OPENING A LOCAL VARIABLES EDITOR

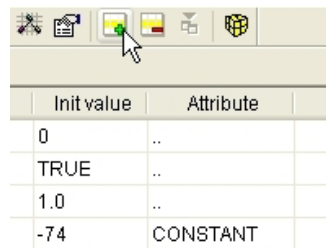
To open a local variables editor, just open the Program Organization Unit the variables you want to edit are local to.



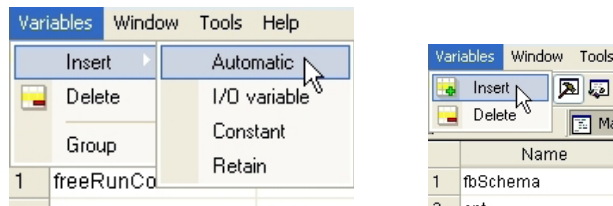


## 5.6.2 CREATING A NEW VARIABLE

In order to create a new variable, you may click on the *Insert record* item in the *Project* toolbar.



Alternatively, you may access the *Variables* menu and choose *Insert*.



## 5.6.3 EDITING VARIABLES

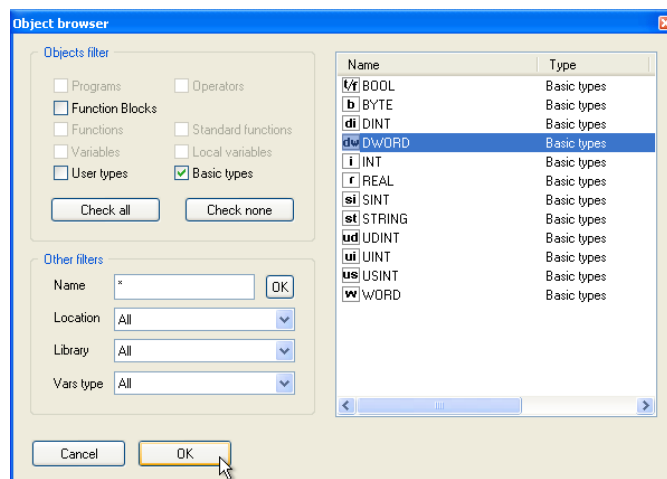
Follow this procedure to edit the declaration of a variable in a variables editor (all the following steps are optional and you will typically skip most of them when editing a variable):

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

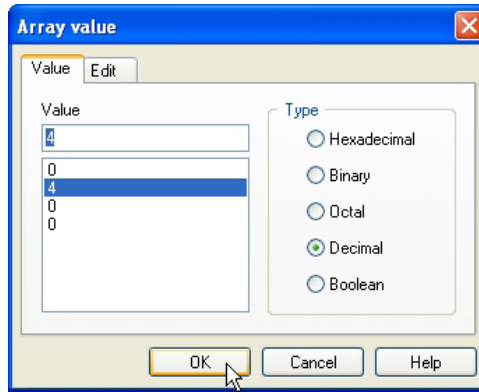
...	...	...	...	...
9	gA	BOOL	Auto	
10	globB	REAL	Auto	
11	gC	REAL	%MD60.20	

- 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.

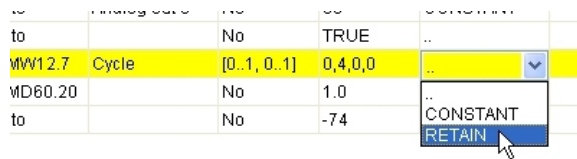
...	...	...	...	...
9	gA	BOOL	Auto	
10	globB	REAL	Auto	
11	gC	REAL	%MD60.20	



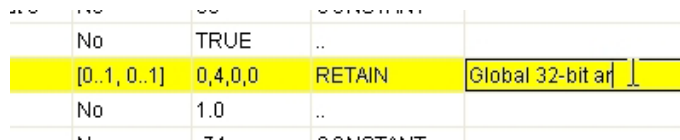




- 7) Assign an attribute to the variable (for example, `CONSTANT` or `RETAIN`), by selecting it from the list which opens when you click on the corresponding cell.



- 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.



- 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.**

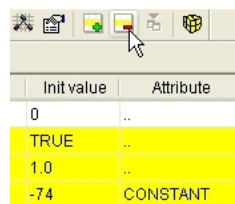


## 5.6.4 DELETING VARIABLES

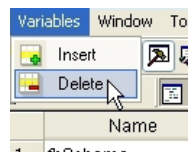
In order to delete one or more variables, select them in the editor: you may use the *CTRL* or the *SHIFT* keys to select multiple elements.

	Name	Type	Address
1	freeRunCounter	DINT	Auto
2	gA	BOOL	Auto
3	gC	REAL	%MD60.20
4	gD	INT	Auto
5	ki	UINT	Auto
6	incr	INT	Auto
7	klhlIncr	INT	Auto
8	globB	DWORD	%MW12.7
9	period	REAL	%MD1.11
10	tau	REAL	%MD1.9
11	valueFilt	REAL	%MW1.8
12	valueDef	REAL	%MD1.10

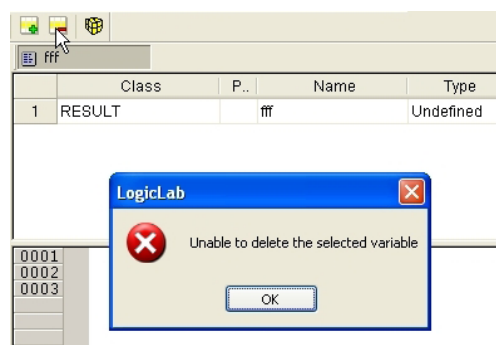
Then, click on the *Delete* record in the *Project* toolbar.



Alternatively, you may access the *Variables* menu and choose *Delete*.



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





## 5.6.5 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	%M
3	valueRef	REAL	%M
4	period	REAL	%M
5	knIncr	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	knIncr	INT	Auto
5	period	REAL	%M
6	tau	REAL	%M
7	valueFilt	REAL	%M
8	valueRef	REAL	%M

## 5.6.6 COPYING VARIABLES

The variables editor allows you to quickly copy and paste elements. You can either use keyboard shortcuts or the *Edit* menu to access these features.

The screenshot shows the 'Edit' menu with the following options: Undo (Ctrl+Z), Redo (Ctrl+Y), Cut (Ctrl+X), Copy (Ctrl+C), Paste (Ctrl+V), Delete (Del), Find in project (Ctrl+Shift+F), Medium, Function blocks, and Functions. The 'Global variables' table is open, showing a list of variables with columns for Name, Type, and Address. The variable 'tau' is highlighted in yellow.

	Name	Type	Address
1	freeRunCounter	DINT	Auto
2	incr	INT	Auto
3	ki	UINT	Auto
4	knIncr	INT	Auto
5	period	REAL	%MD1.11 Or
6	tau	REAL	%MD1.9 Or
7	valueFilt	REAL	%MW1.8 Or
8	valueRef	REAL	%MD1.10 Or
9	tau1	REAL	%MD1.9 Or





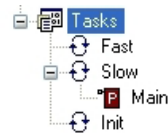


## 6. COMPILING

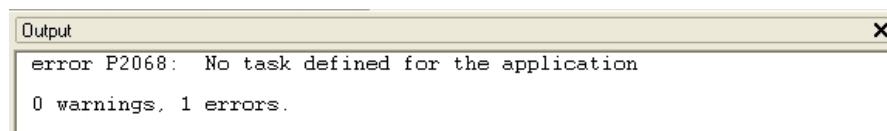
Compilation consists of taking the PLC source code and automatically translating it into binary code, which can be executed by the processor on the target device.

### 6.1 COMPILING THE PROJECT

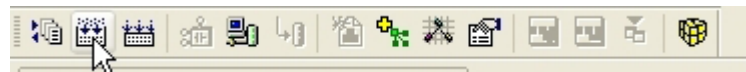
Before starting actual compilation, make sure that at least one program has been assigned to a task.



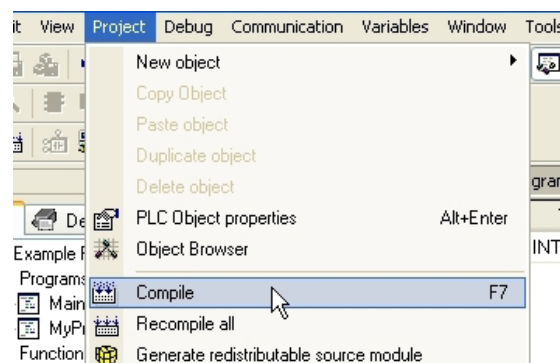
When this pre-condition does not hold, compilation aborts with a meaningful error message.



In order to start compilation, click the *Compile* button in the *Project* toolbar.



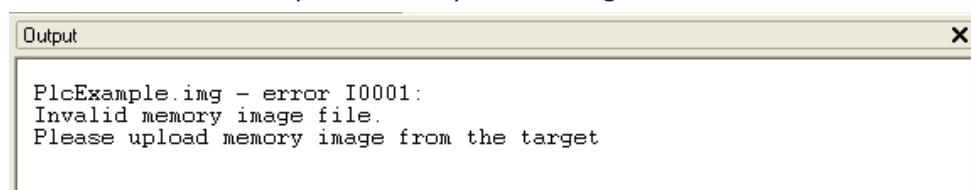
Alternatively, you can choose *Compile* from the *Project* menu or press *F7* on your keyboard.



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

#### 6.1.1 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.

## 6.2 COMPILER OUTPUT

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

```
m warnings, n errors
```

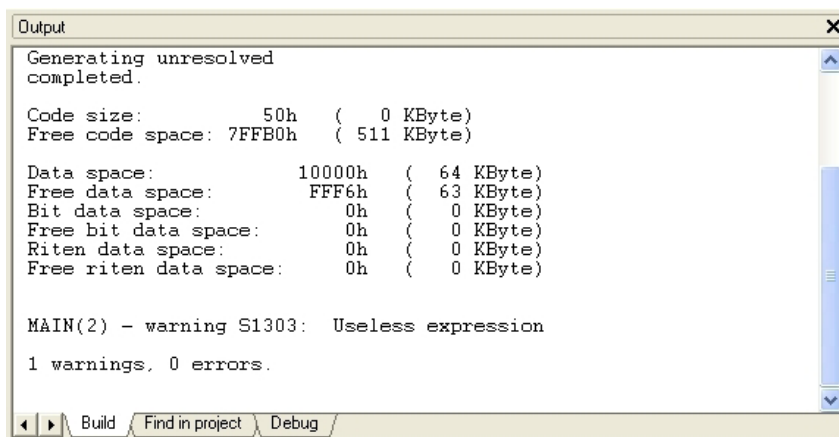
It tells the user the outcome of compilation.

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

NOTE: Be sure to eliminate all compiler errors and warnings before downloading your application to your controller or other device.

### 6.2.1 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 an unrecoverable error (*error*) or one that the compiler could work around (*warning*);
- the error code;
- the error description.



## SoMachine HVAC - Application

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.

The screenshot shows the SoMachine IDE interface. The 'Project' window on the left displays a tree view of the project structure, including 'PlcExample Project', 'Programs', 'Main', 'MyProgram', 'Function blocks', 'Functions', 'Global variables', and 'Tasks'. The 'Main' window shows a table with columns 'Name', 'Type', 'Address', and 'An'. The table contains one entry: '1 n UINT Auto No'. Below the table, the source code is displayed with line numbers 0001, 0002, and 0003. Line 0003 contains the statement 'n = n + 1;'. The 'Output' window at the bottom shows the following text:

```
Generating unresolved
completed.

Code size:          50h ( 0 KByte)
Free code space: 7FFB0h ( 511 KByte)

Data space:         10000h ( 64 KByte)
Free data space:    FFF6h ( 63 KByte)
Bit data space:     0h ( 0 KByte)
Free bit data space: 0h ( 0 KByte)
Riten data space:   0h ( 0 KByte)
Free riten data space: 0h ( 0 KByte)

MAIN(2) - warning S1303 Useless expression
1 warnings, 0 errors.
```

You can then solve the problem and re-compile.

The screenshot shows the SoMachine IDE interface after successful compilation. The 'Project' window on the left displays a tree view of the project structure, including 'PlcExample Project', 'Programs', 'Main', 'MyProgram', 'Function blocks', 'Functions', 'Global variables', and 'Tasks'. The 'Main' window shows a table with columns 'Name', 'Type', 'Address', and 'An'. The table contains one entry: '1 n UINT Auto No'. Below the table, the source code is displayed with line numbers 0001, 0002, and 0003. Line 0003 contains the statement 'n := n + 1;'. The 'Output' window at the bottom shows the following text:

```
Generating program MAIN
Generating unresolved
completed.

Code size:          40h ( 0 KByte)
Free code space: 7FFC0h ( 511 KByte)

Data space:         10000h ( 64 KByte)
Free data space:    FFF6h ( 63 KByte)
Bit data space:     0h ( 0 KByte)
Free bit data space: 0h ( 0 KByte)
Riten data space:   0h ( 0 KByte)
Free riten data space: 0h ( 0 KByte)

0 warnings, 0 errors.
```



## 6.3 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.

```
CA C:\WINDOWS\system32\cmd.exe
/D          - Download compiled project
             <if not already compiled will compile it>
/C          - Compile the project <trying to connect>
/c         - Compile the project without connecting
/A         - Rebuild and download the project
/GCI[:<file>] - Generate C headers in the destination file.
             <default 'Default.h'>
/GT[:<file>] - Generate the target variable track file.
             <default 'Default.osc'>
/GGI[:<file>] - Generate the global variable track file.
             <default 'Default.osc'>
/F:<comm>   - Force the communication properties
/T:<target> - Force the target board type
             <used to download code without opening the project>
/DR        - Perform download acknowledge request
/R         - Rebuild the project <trying to connect>
/r         - Rebuild the project without connecting
/FU       - Perform download without checking for updated source status
/Q        - Append the output to the destination file.
             <valid only with /GT e /GG flags>
/MI[:file[,pwd]] - Generate redistributable source file.

Note: the flags are case-sensitive
```



## 7. LAUNCHING THE APPLICATION

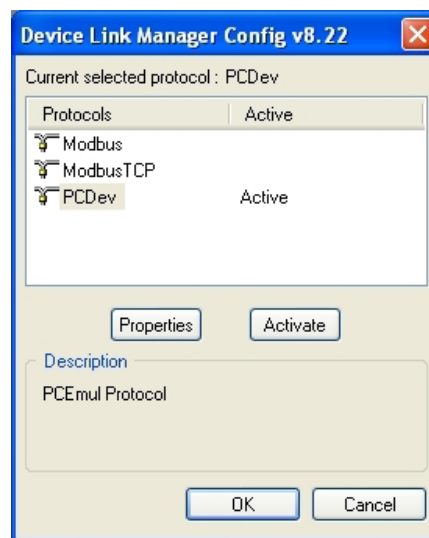
In order to download and debug the application, you have to establish a connection with the target device. This chapter focuses on the operations required to connect to the target and to download the application, while the wide range of Application's debugging tools deserves a separate chapter (see Chapter 8).

### 7.1 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).

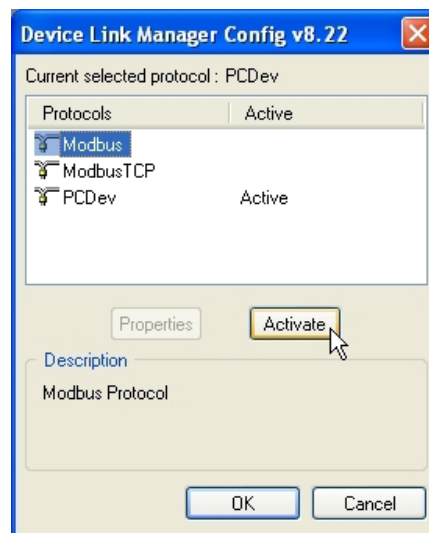
Follow this procedure to set up and establish the connection to the target device:

- 1) Click *Settings* in the *Communication* menu of the Application main window. This causes the following dialog box to appear.



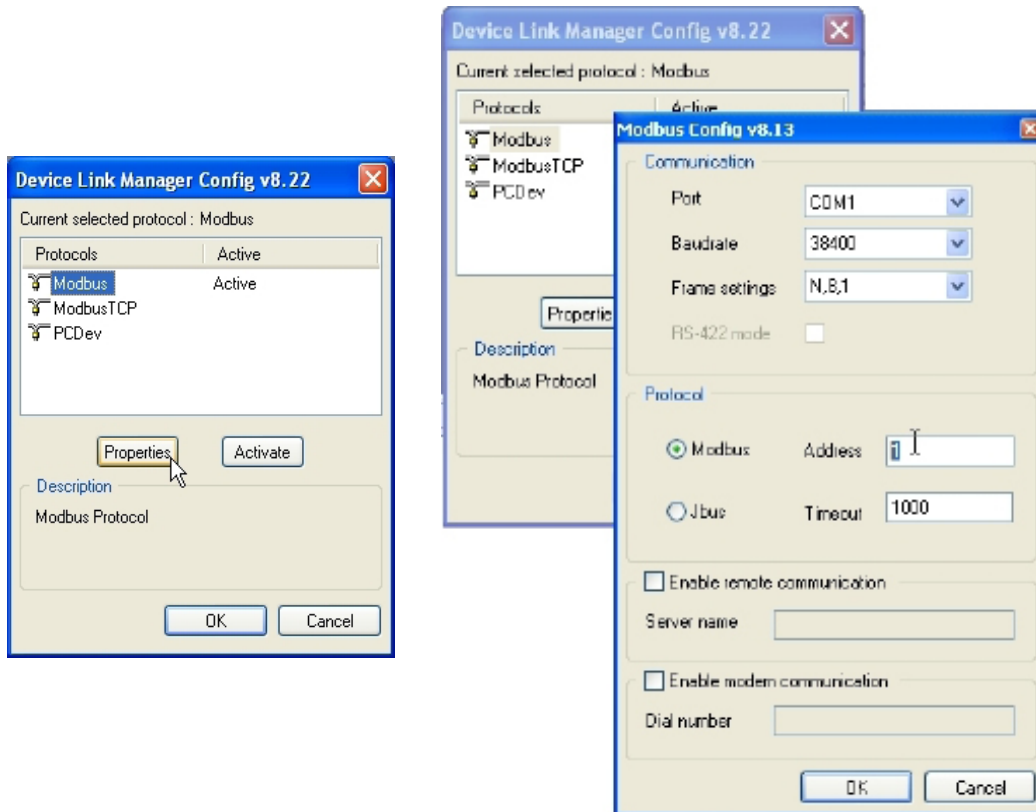
The elements in the list of communication protocols you can select from depend on the setup executable(s) you have run on your PC (refer to your hardware provider if a protocol you expect to appear in the list is missing).

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

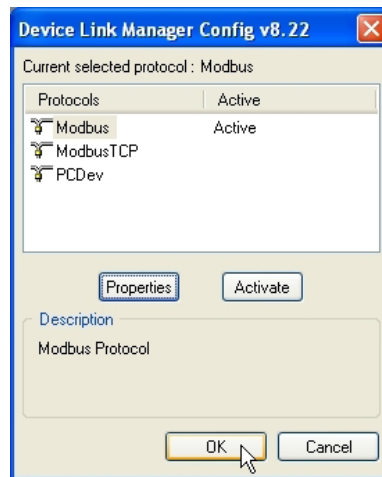




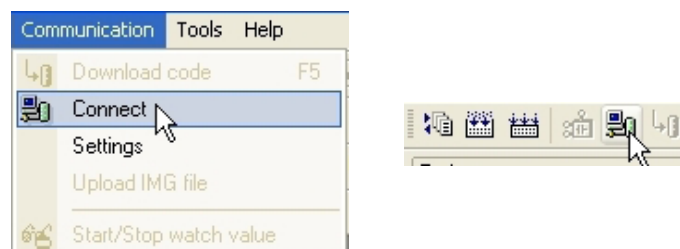
- Fill in all the protocol-specific settings (e.g., the address or the communication timeout - that is how long Application must wait for an answer from the target before displaying a communication error message).



- Apply the changes you made to the communication settings.



Now you can establish communication by clicking *Connect* in the *Communication* menu, or by pressing the *Connect* button in the *Project* toolbar.





## 7.1.1 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.



## 7.2 ON-LINE STATUS

### 7.2.1 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*.

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.

ERROR

- **Connected**: the communication has been established.

CONNECTED

### 7.2.2 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.

NO CODE



- **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 valid and the debug mode cannot be activated.

### DIFF. CODE

- **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 valid, but the debug mode still cannot be activated.

### DIFF. CODE (SYM)

- **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.

### SOURCE OK

## 7.3 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 the related button in the *Project* toolbar.



Alternatively, you can choose *Download code* from the *Project* menu or press the *F5* key.



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.

### 7.3.1 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.

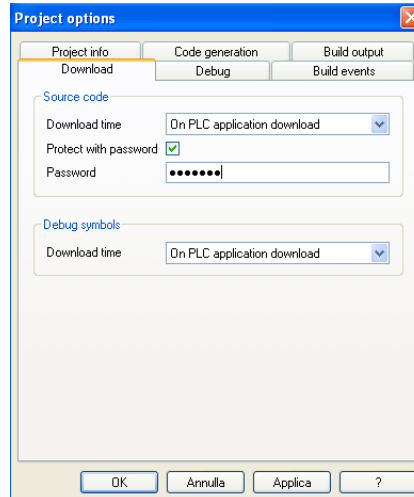




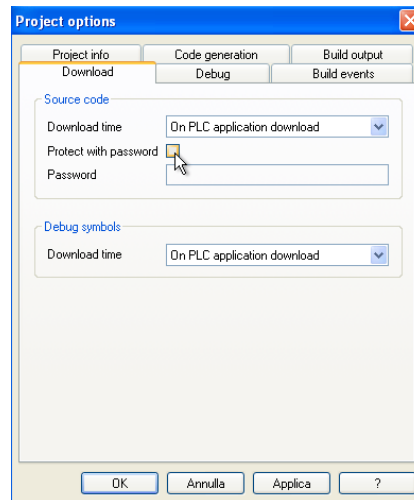
## 7.3.1.1 PROTECTING THE SOURCE CODE WITH A PASSWORD

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.

Open the *Project options* window (*Project > Options ...* menu) and set the password.



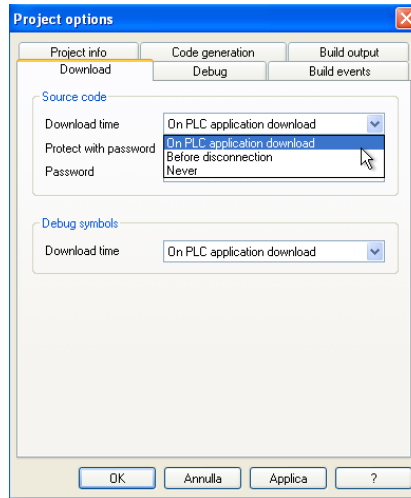
You may opt to disable the password, instead.





## 7.3.1.2 SOURCE CODE AND DEBUG SYMBOLS DOWNLOAD TIME

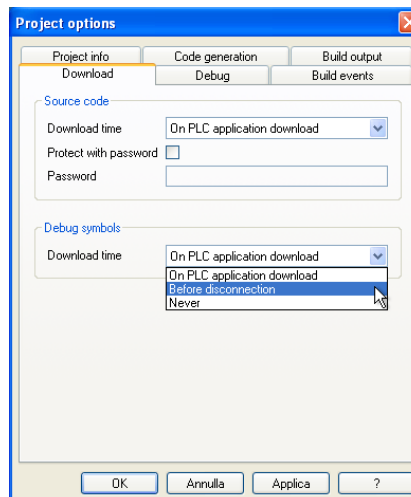
From the following select menu you can set the *Source code download time*.



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.



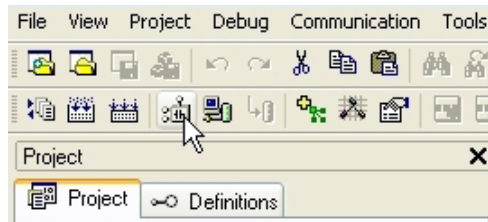
## 7.4 SIMULATION

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.



# SoMachine HVAC - Application

In order to start the simulation, just click on the appropriate item on the *Project* toolbar.



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

## 7.5 CONTROL THE PLC EXECUTION

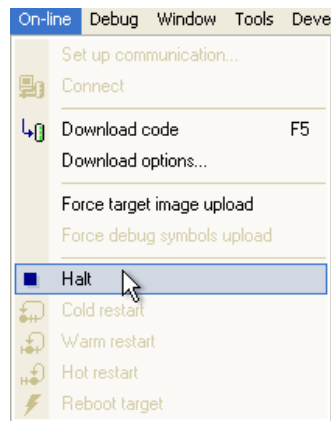
The PLC application execution can be controlled using the related functions in the project bar or by the command presents in the On-line menu.

### 7.5.1 HALT

You can stop the PLC execution selecting the following item in the project bar:



or by the following item in the On-line menu:



### 7.5.2 COLD RESTART

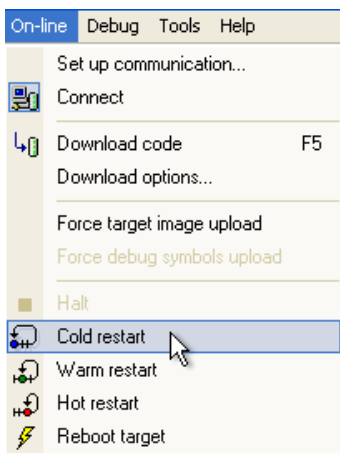
The PLC application execution will be restarted and both retain and non-retain variables will be resetted.

You can cold restart the PLC execution selecting the following item in the project bar:





or by the following item in the On-line menu:



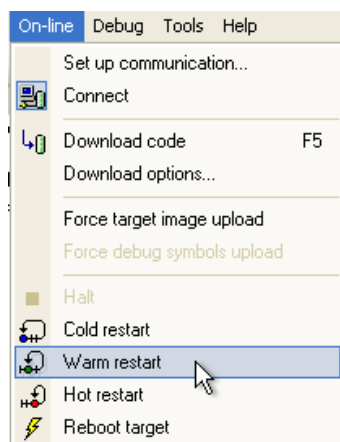
## 7.5.3 WARM RESTART

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

You can warm restart the PLC execution selecting the following item in the project bar:



or by the following item in the On-line menu:



## 7.5.4 HOT RESTART

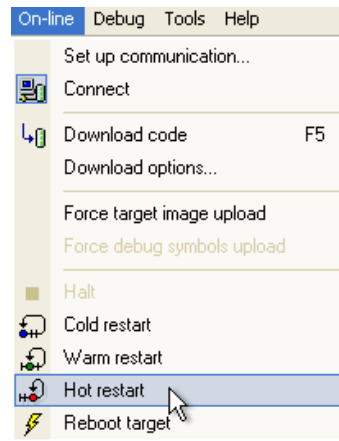
The PLC application execution will be restarted and no variables will be resetted.

You can hot restart the PLC execution selecting the following item in the project bar:





or by the following item in the On-line menu:

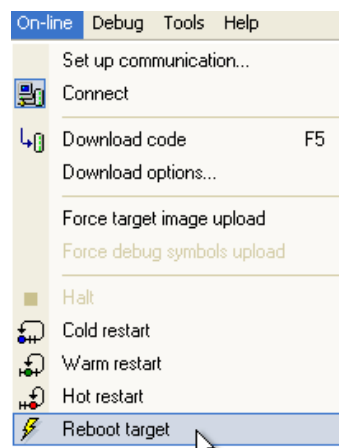


## 7.5.5 REBOOT TARGET

You can reboot the target selecting the following item in the project bar:



or by the following item in the On-line menu:







## 8. 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:

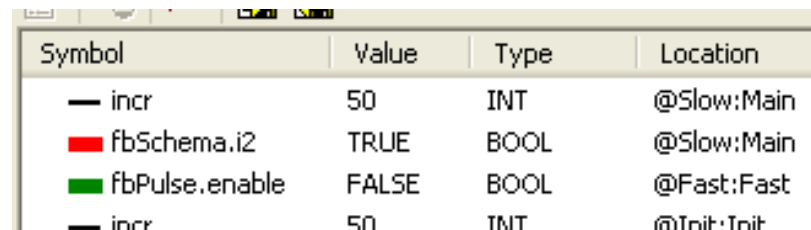
- Asynchronous debuggers. They read the values of the variables selected by the developer with successive queries issued to the target device. Both the manager of the debugging tool (that runs on the PC) and, potentially, the task which is responsible to answer those queries (on the target device) run 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.

### 8.1 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
— incr	50	INT	@Slow:Main
■ fbSchema.i2	TRUE	BOOL	@Slow:Main
■ fbPulse.enable	FALSE	BOOL	@Fast:Fast
— inrr	50	INT	@Trnit:Trnit

#### 8.1.1 OPENING AND CLOSING THE WATCH WINDOW

To open the *Watch* window, click on the *Watch* button of the *Main* toolbar.

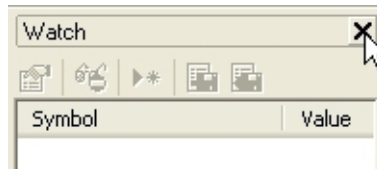


To close the *Watch* window, click on the *Watch* button again.





Alternatively, you can click on the *Close* button in the top right corner of the *Watch* window.



In both cases, 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.

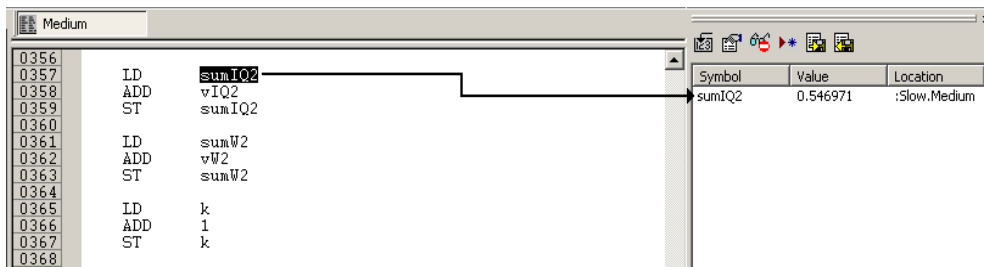
## 8.1.2 ADDING ITEMS TO THE WATCH WINDOW

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.

### 8.1.2.1 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.

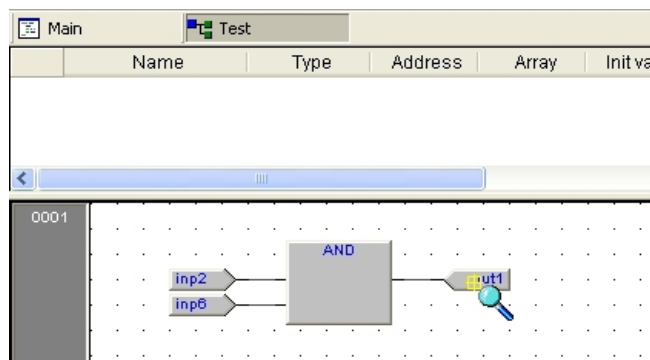
### 8.1.2.2 ADDING A VARIABLE FROM A GRAPHICAL SOURCE CODE EDITOR

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

- 1) Press the *Watch* button in the FBD bar.



- 2) Click on the block representing the variable you wish to be shown in the *Watch* window.

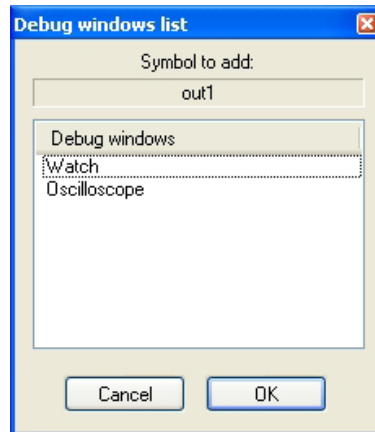




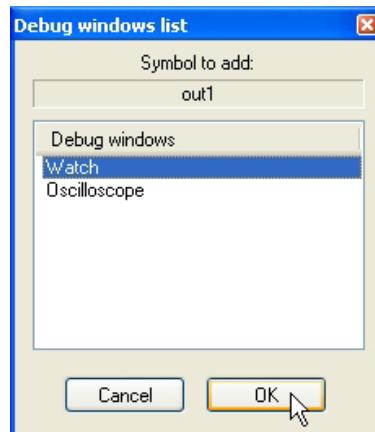


## SoMachine HVAC - Application

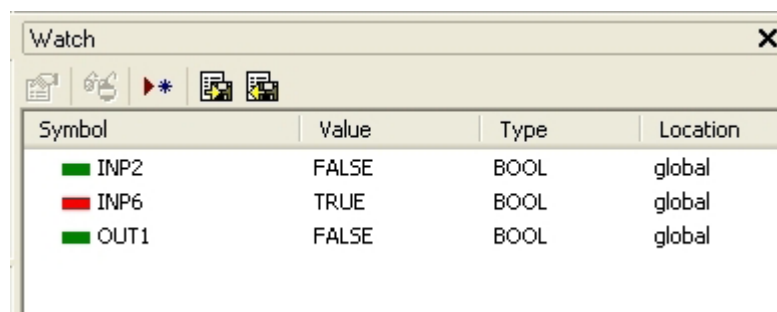
- 3) A dialog box appears listing all the currently existing instances of debug windows, and asking you which one is to receive the object you have just clicked on.



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.



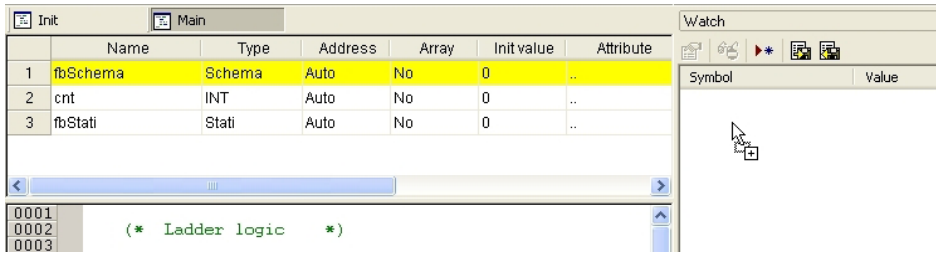
The same procedure applies to all the variables you wish to inspect.

Once you have added to the *Watch* window all the variables you want to observe, you should click on the *Select/Move* button in the FBD bar: the mouse cursor turns to its original shape.

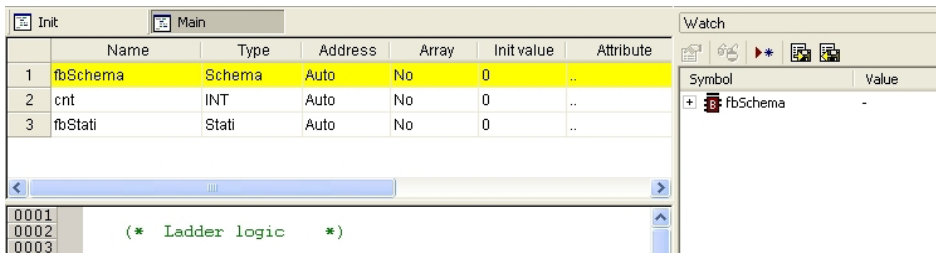


## 8.1.2.3 ADDING A VARIABLE FROM A VARIABLES EDITOR

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

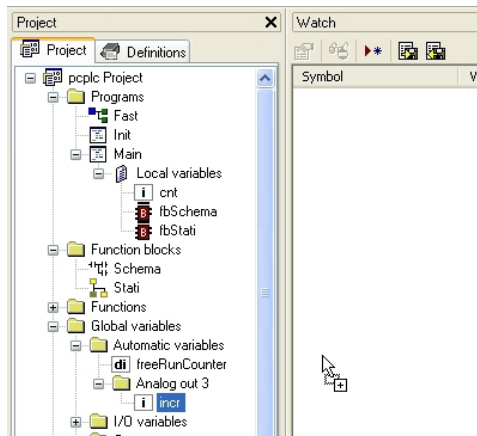


or press the *F8* key.

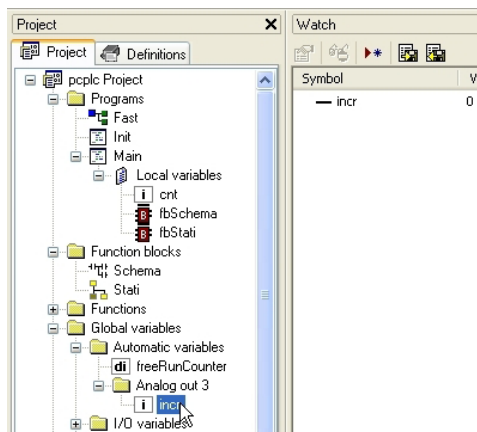


## 8.1.2.4 ADDING A VARIABLE FROM THE PROJECT TREE

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



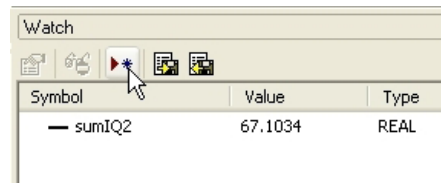
or press the *F8* key.



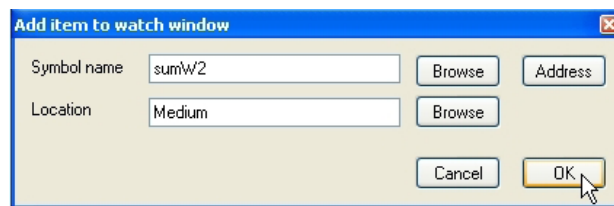


## 8.1.2.5 ADDING A VARIABLE FROM THE WATCH WINDOW TOOLBAR

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

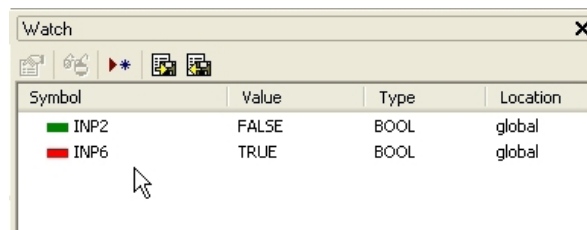
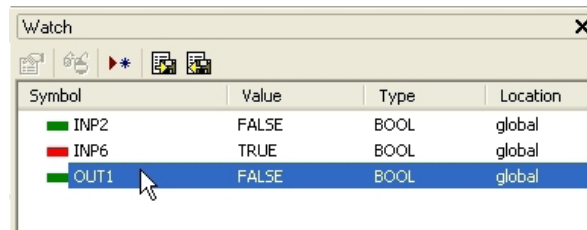


You shall type (or select by browsing the project symbols) the name of the variable and its location (where it has been declared).



## 8.1.3 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.



## 8.1.4 REFRESHMENT OF VALUES

### 8.1.4.1 NORMAL OPERATION

Let us consider the following example.

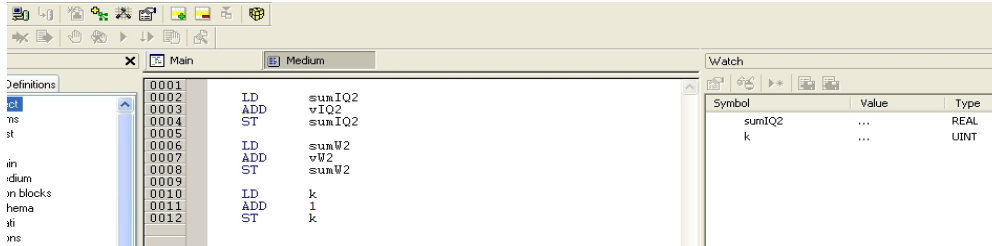




The watch window manager reads periodically from memory the value of the variables. However, this action is carried out asynchronously, that is it may happen that a higher-priority task modifies the value of some of the variables while they are being read. Thus, at the end of a refreshment process, the values displayed in the window may refer to different execution states of the PLC code.

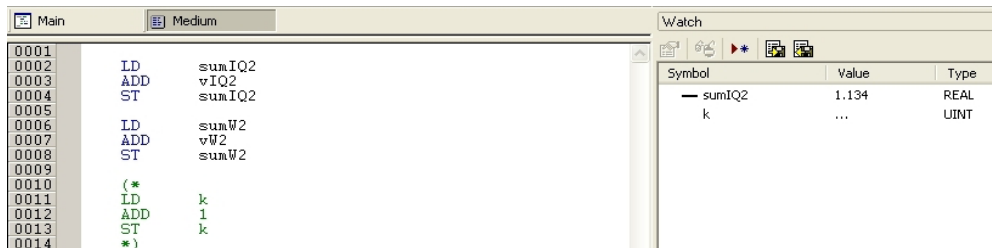
### 8.1.4.2 TARGET DISCONNECTED

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

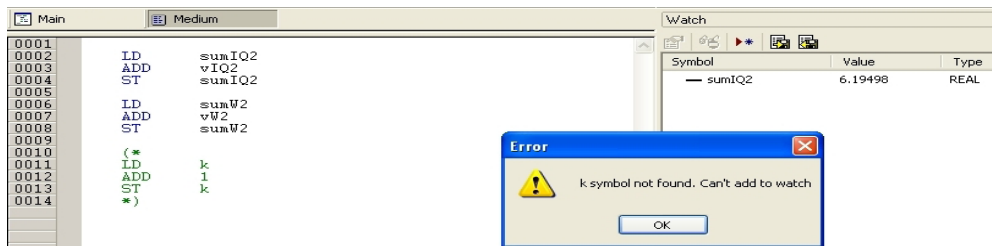


### 8.1.4.3 OBJECT NOT FOUND

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.



### 8.1.5 CHANGING THE FORMAT OF DATA

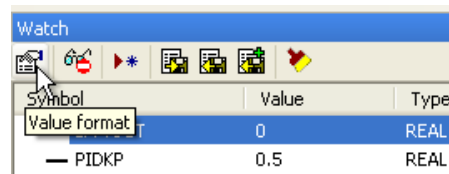
When you add a variable to the *Watch* window, Application automatically recognizes its type (unsigned integer, signed integer, floating point, hexadecimal), and displays its value consistently. Also, if the variable is floating point, Application assigns it a default number of decimal figures.

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

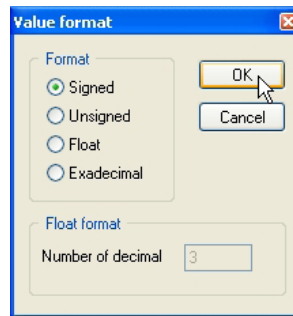


# SoMachine HVAC - Application

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.

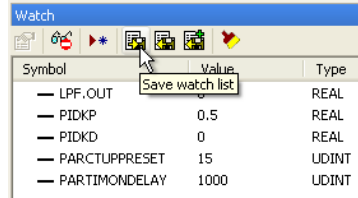


## 8.1.6 WORKING WITH WATCH LISTS

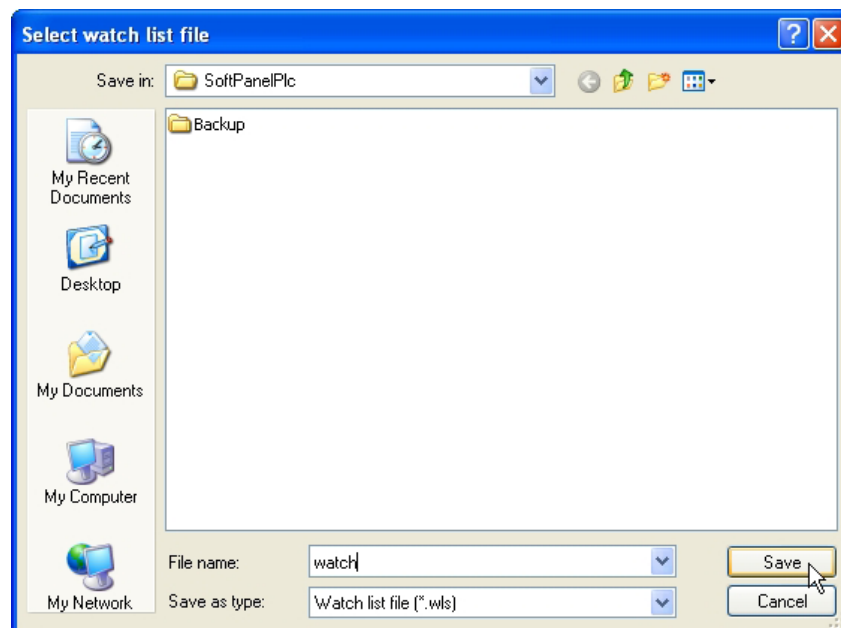
You can store to file the set of all the items in the *Watch* window, in order to easily restore the status of this debugging tools in a successive working session.

Follow this procedure to save a watch list:

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



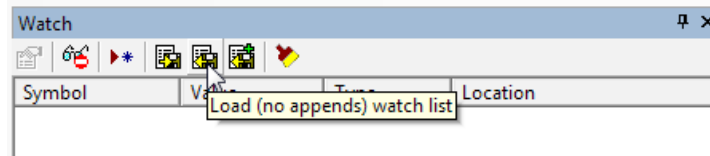
- 2) Enter the file name and choose its destination in the file system.



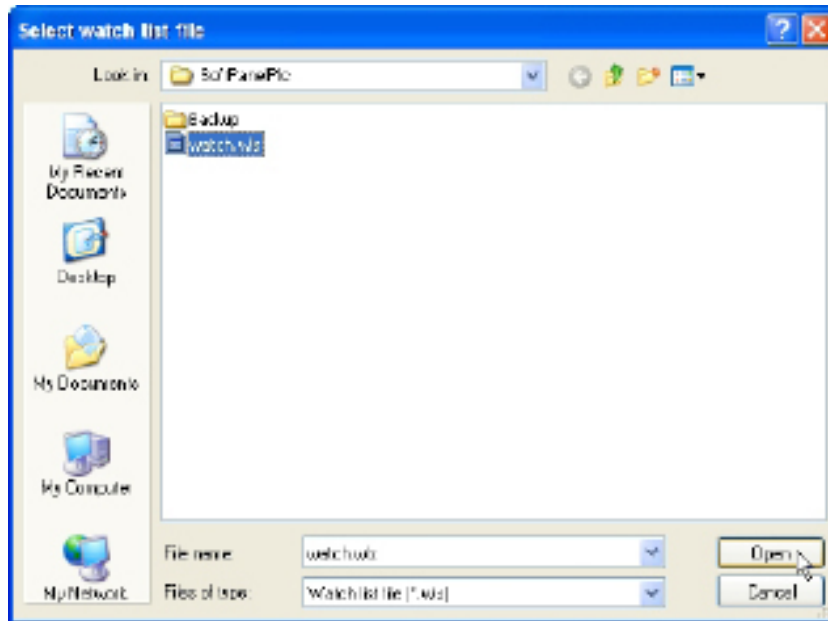


You can load a watch list from file, removing the opened one, following this procedure:

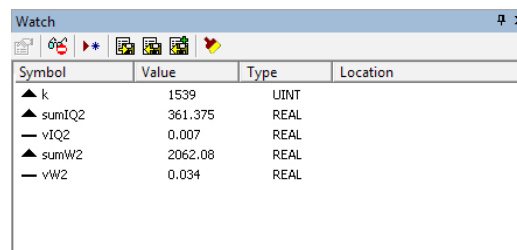
- 1) Click on the corresponding icon in the *Watch* window toolbar.



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

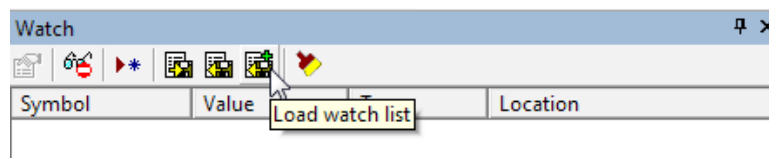


The set of symbols in the watch list is added to the *Watch* window.



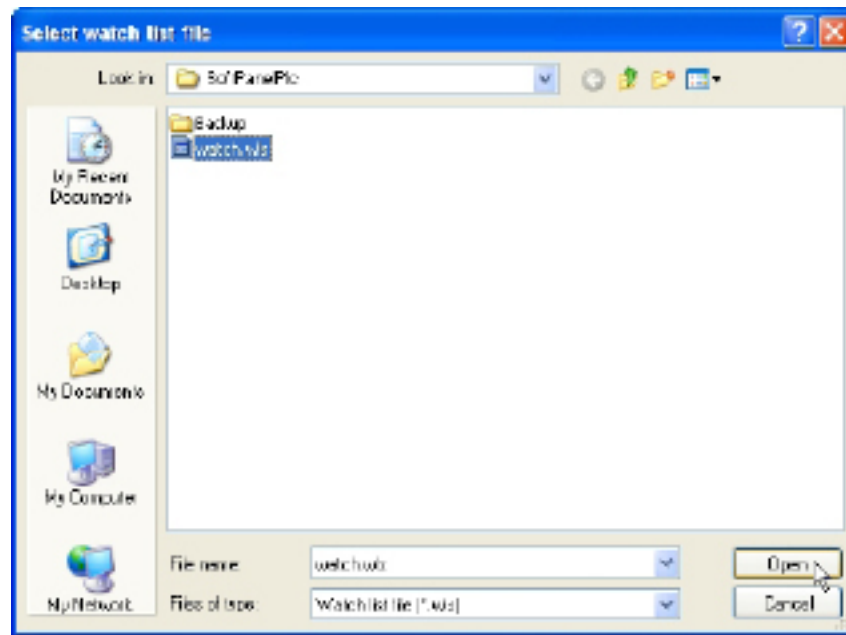
You can load a watch list from file, appending to the opened one, following this procedure:

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





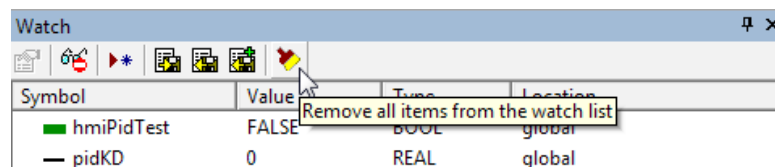
- 2) Browse the file system and select the watch list file.



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

Symbol	Value	Type	Location
▲ k	1539	UINT	
▲ sumIQ2	361.375	REAL	
— viQ2	0.007	REAL	
▲ sumW2	2062.08	REAL	
— vW2	0.034	REAL	

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



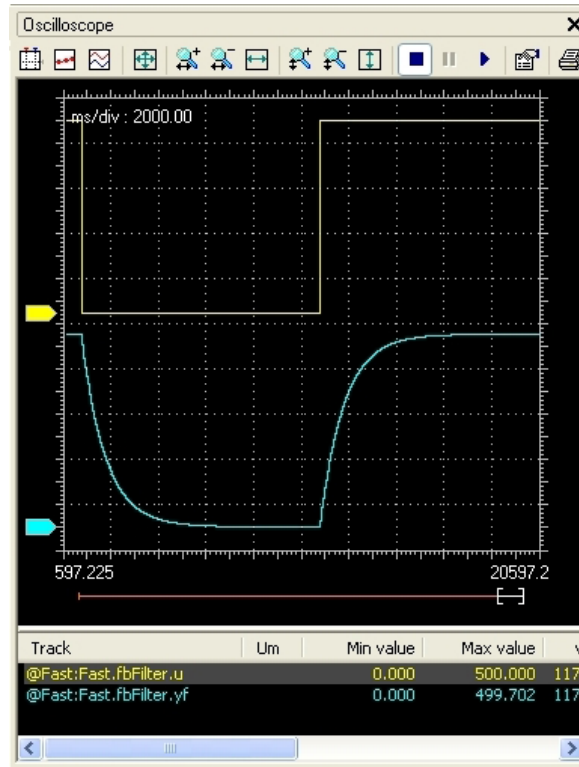
## 8.1.7 AUTOSAVE WATCH LIST

By selecting the associated option in the project options dialog (See paragraph 3.6.5 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 the project will be re-opened.

## 8.2 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.

### 8.2.1 OPENING AND CLOSING THE OSCILLOSCOPE

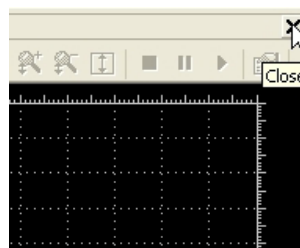
To open the Oscilloscope, click on the *Async* button of the *Main* toolbar.



To close the Oscilloscope, click on the *Async* button again.



Alternatively, you can click on the *Close* button in the top right corner of the *Oscilloscope* window.







In both cases, 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.

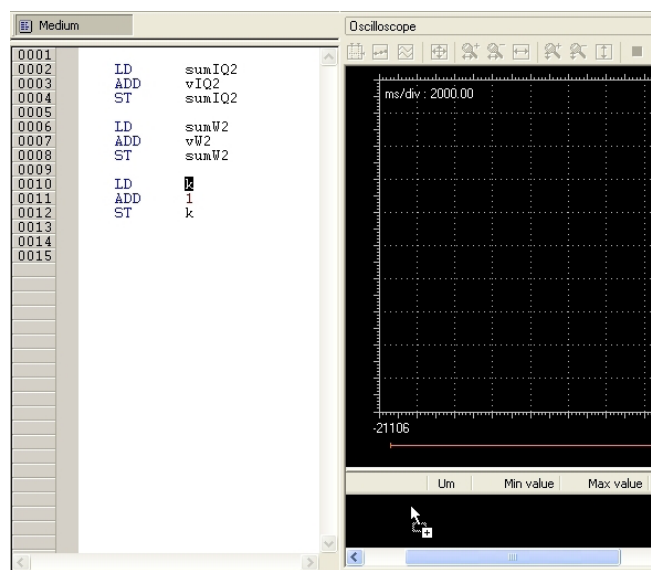
## 8.2.2 ADDING ITEMS TO THE OSCILLOSCOPE

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.

### 8.2.2.1 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.

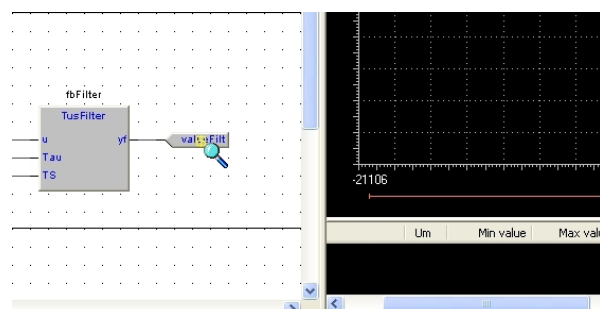
### 8.2.2.2 ADDING A VARIABLE FROM A GRAPHICAL SOURCE CODE EDITOR

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

- 1) Press the *Watch* button in the *FBD* bar.

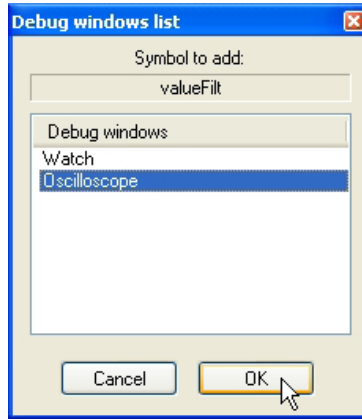


- 2) Click on the block representing the variable you wish to be shown in the Oscilloscope.





- 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.



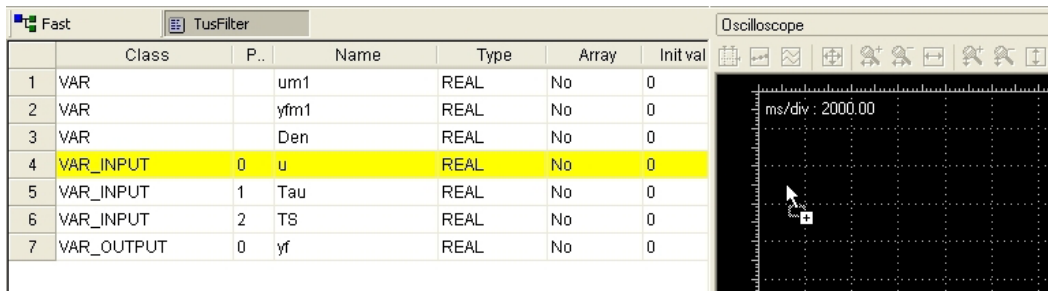
Select *Oscilloscope*, the press *OK*. The name of the variable is now displayed in the *Track* column.

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

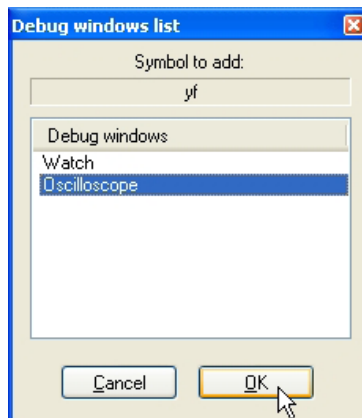
Once you have added to the Oscilloscope all the variables you want to observe, you should click on the *Select/Move* button in the *FBD* bar: the mouse cursor turns to its original shape.

### 8.2.2.3 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



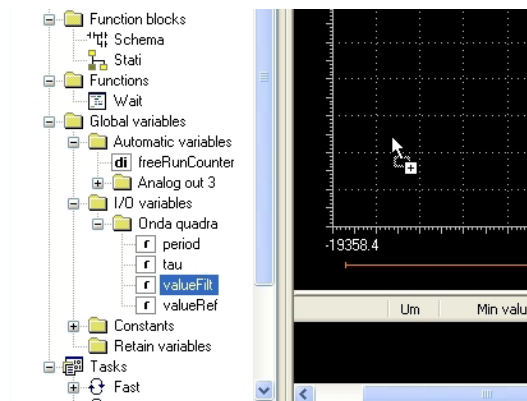
or press the *F10* key and choose *Oscilloscope* from the list of debug windows which pops up.



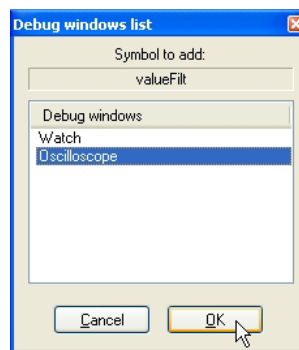


## 8.2.2.4 ADDING A VARIABLE FROM THE PROJECT TREE

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.



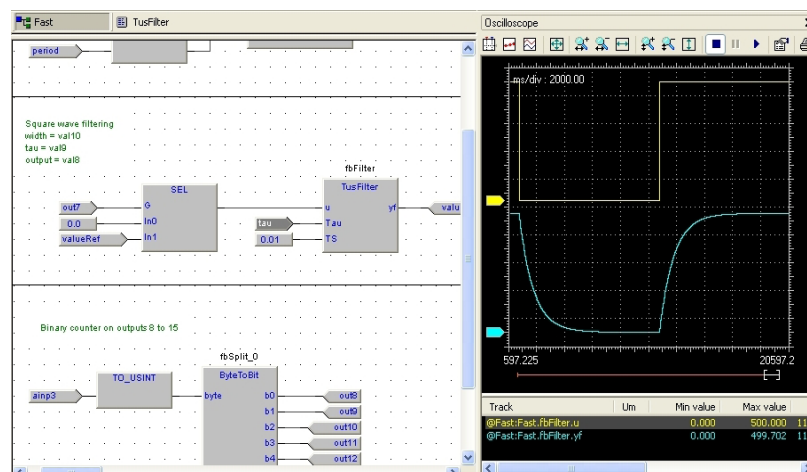
## 8.2.3 REMOVING A VARIABLE

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

## 8.2.4 VARIABLES SAMPLING

### 8.2.4.1 NORMAL OPERATION

Let us consider the following example.





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.

## 8.2.4.2 TARGET DISCONNECTED

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

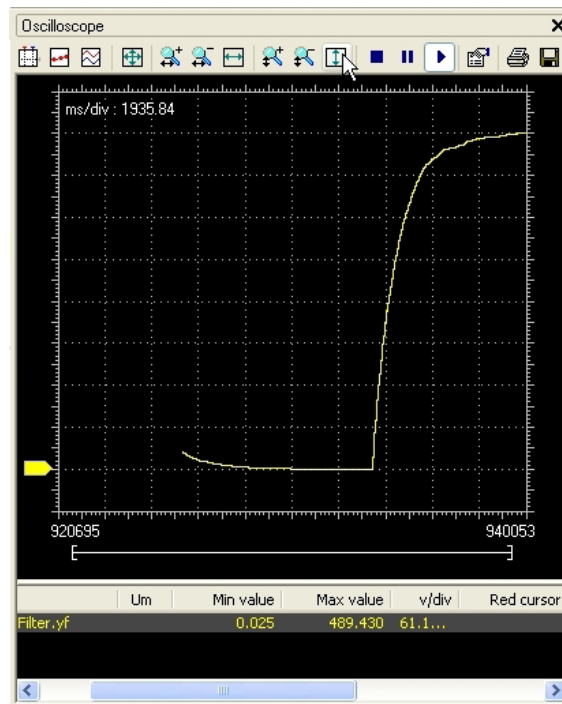
## 8.2.5 CONTROLLING DATA ACQUISITION AND DISPLAY

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.

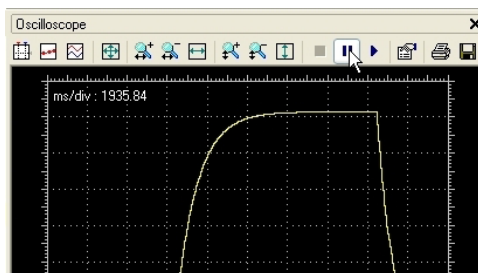
Note that all the commands in the toolbar are disabled if no variable has been added to the Oscilloscope.

### 8.2.5.1 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*.



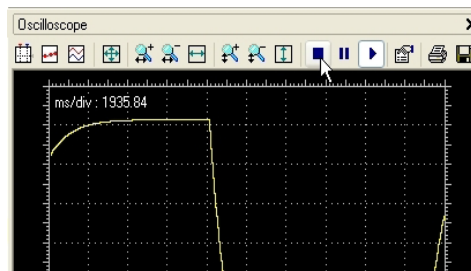


## SoMachine HVAC - Application

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*.

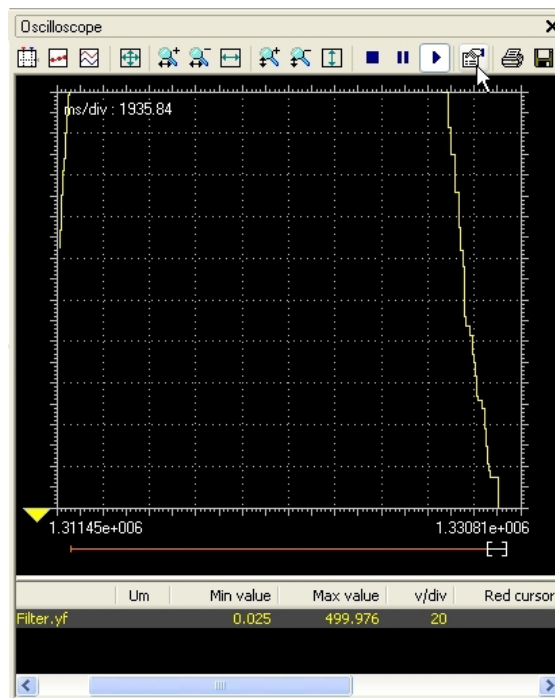


In this case, when you click on *Restart acquisition*, the evolution of the value of the variable is plotted from scratch.

### 8.2.5.2 SETTING THE SCALE OF THE AXES

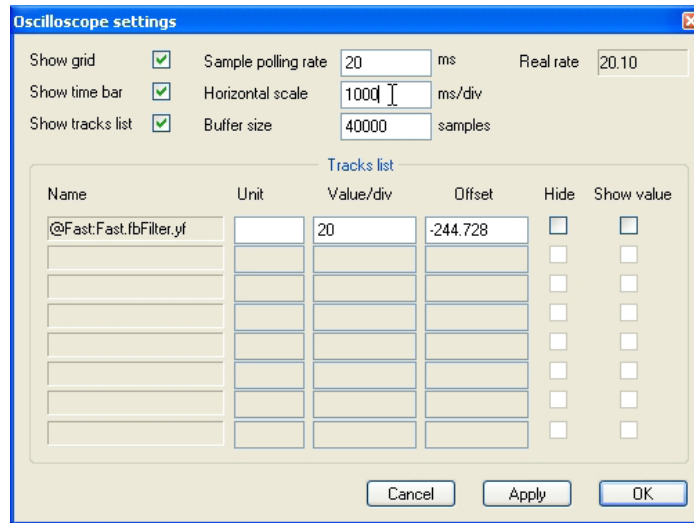
When you open the Oscilloscope, Application applies a default scale to the axes. However, if you want to set a different scale, you may follow this procedure:

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

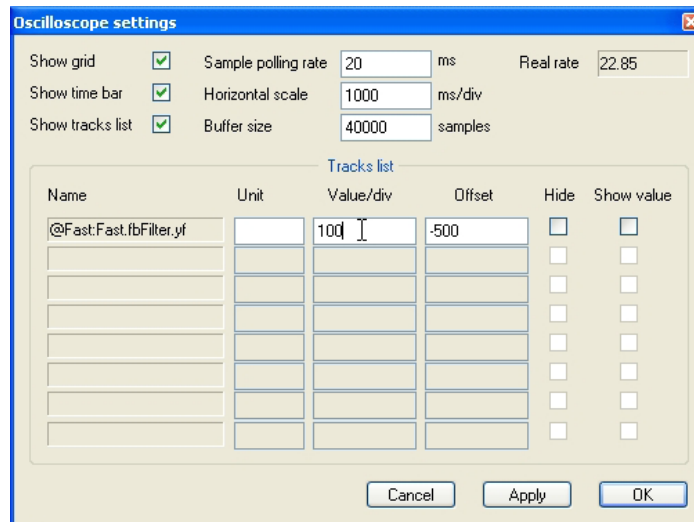




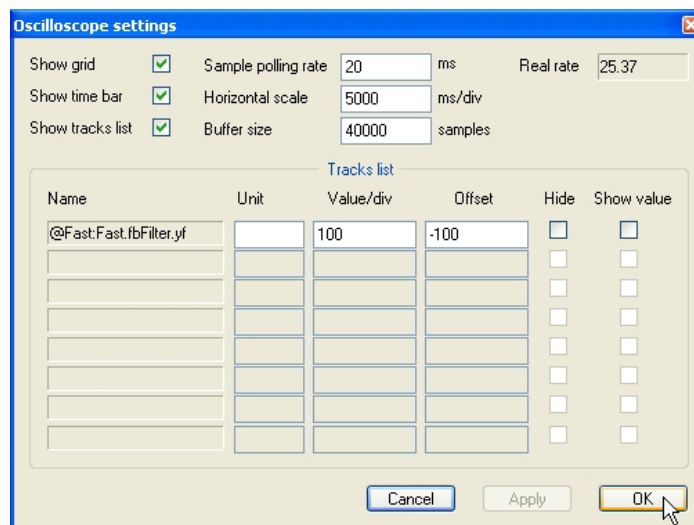
2) Set the scale of the horizontal axis, which is common to all the tracks.

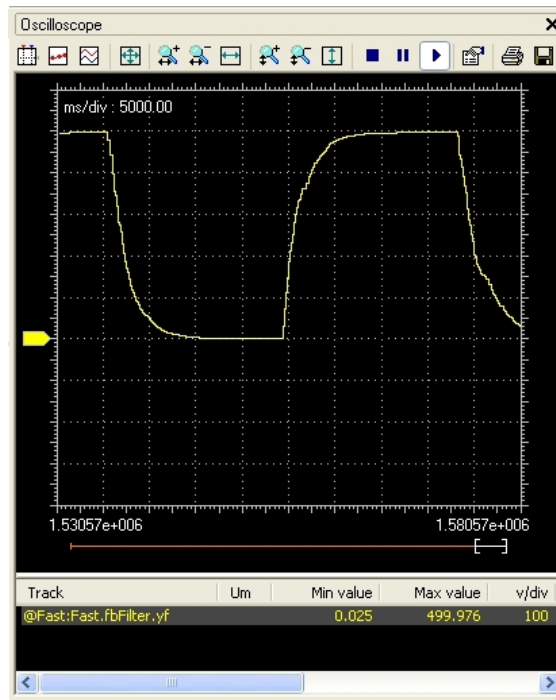


3) For each variable, you may specify a distinct scale for the vertical axis.

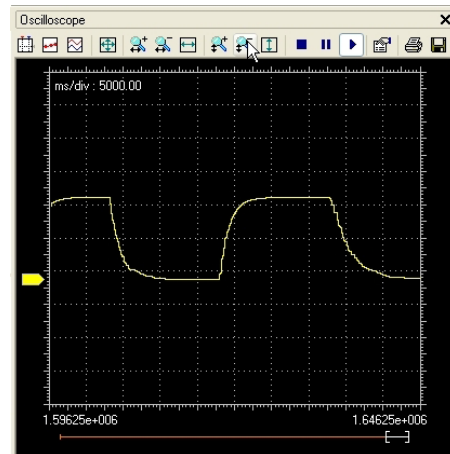
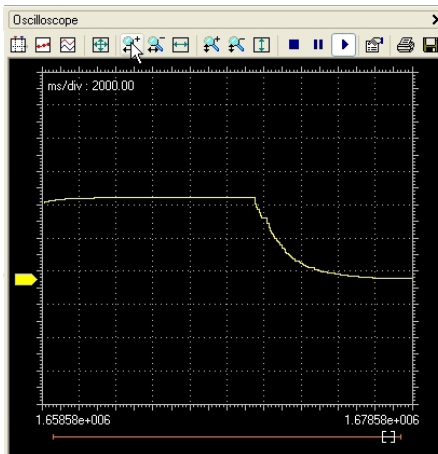


4) Confirm your settings. The graph adapts to reflect the new scale.

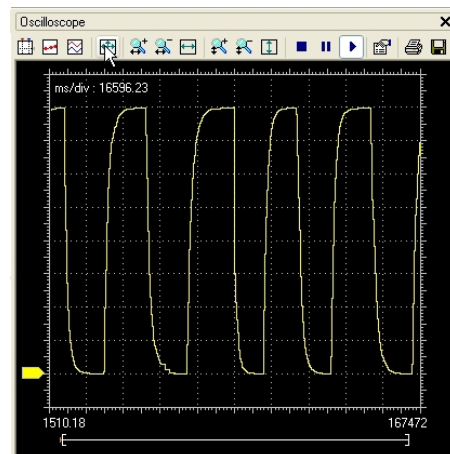
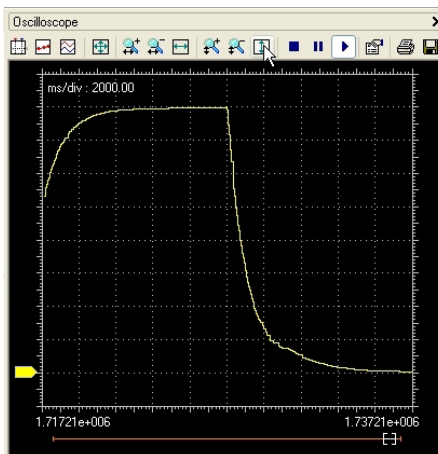




You can also zoom in and out with respect to both the horizontal and the vertical axes.



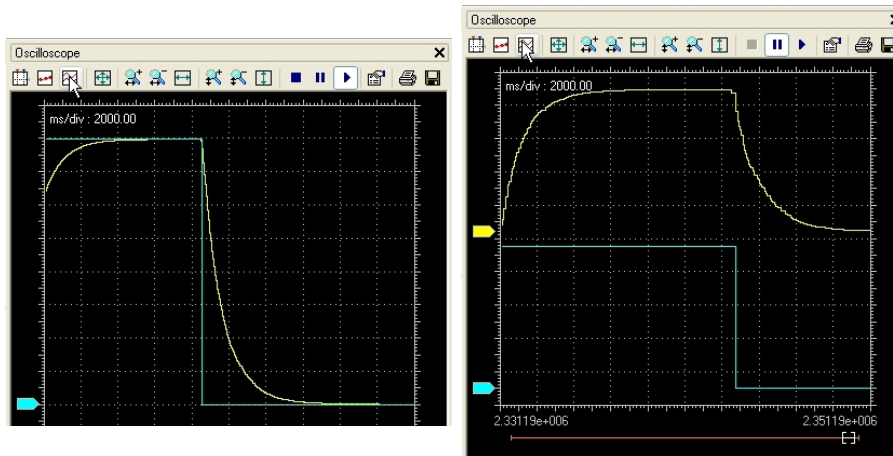
Finally, you may also quickly adapt the scale of the horizontal axis, the vertical axis, or both to include all the samples, by clicking on the corresponding item of the toolbar.





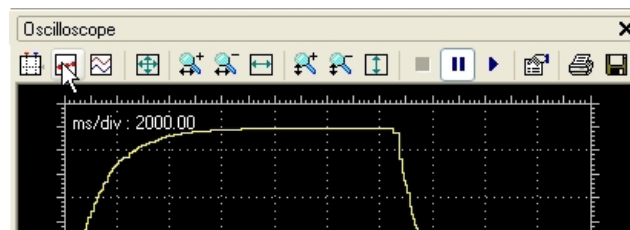
## 8.2.5.3 VERTICAL SPLIT

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.

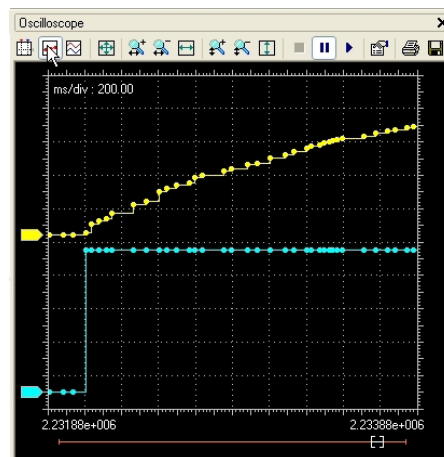


## 8.2.5.4 VIEWING SAMPLES

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.

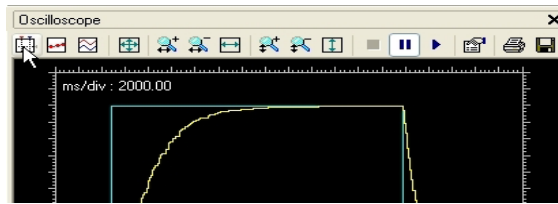




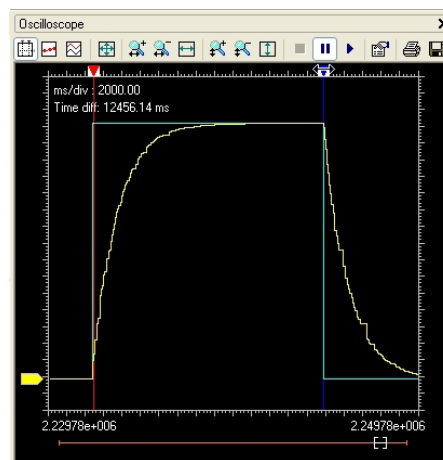


## 8.2.5.5 TAKING MEASURES

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.



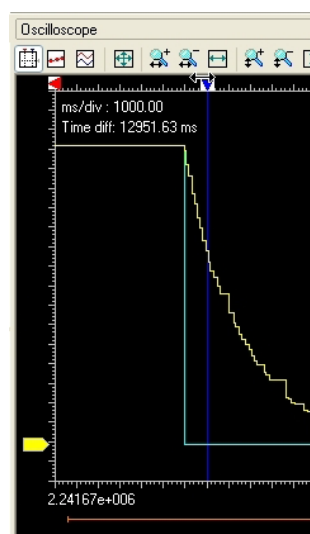
If you want to measure a time interval between two events, you just have to move one bar to the point in the graph that corresponds to the first event and the other to the point that corresponds to the second one.



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

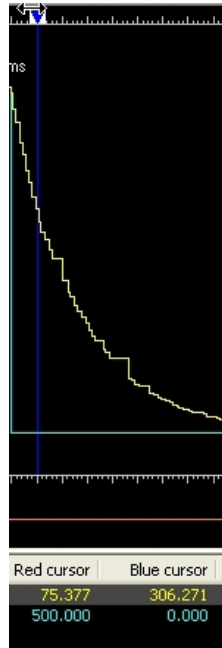


You can use a measure bar also to read the value of all the variables in the Oscilloscope at a particular moment: move the bar to the point in the graph which corresponds to the instant you want to observe.





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

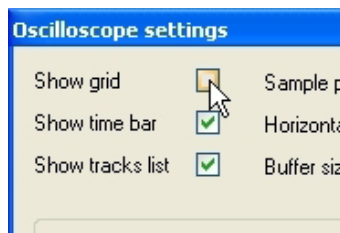


### 8.2.5.6 OSCILLOSCOPE SETTINGS

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*.



### 8.2.6 CHANGING THE POLLING RATE

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

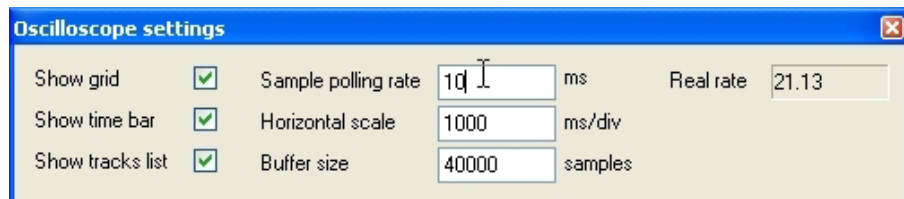
The polling rate can be configured by following this procedure:

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



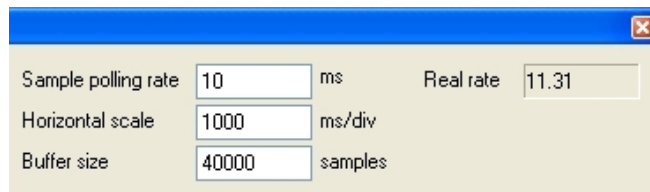


- 2) In the window that pops up edit the *Sampling polling rate*.



- 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.



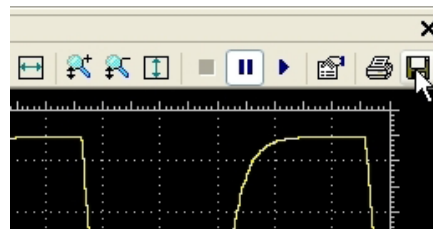
## 8.2.7 SAVING AND PRINTING THE GRAPH

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.

### 8.2.7.1 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.



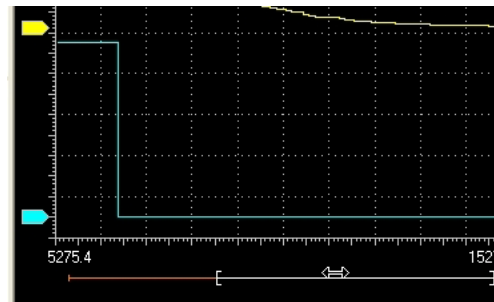
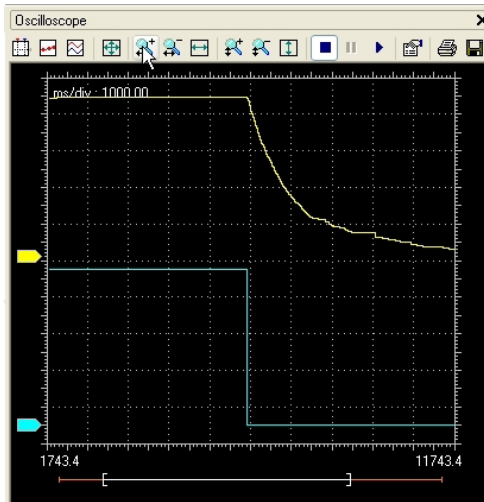
## 8.2.7.2 PRINTING THE GRAPH

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

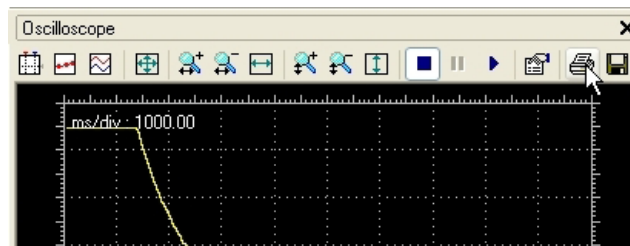
- 1) Either suspend or stop the acquisition.



- 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.



## 8.3 EDIT AND DEBUG MODE

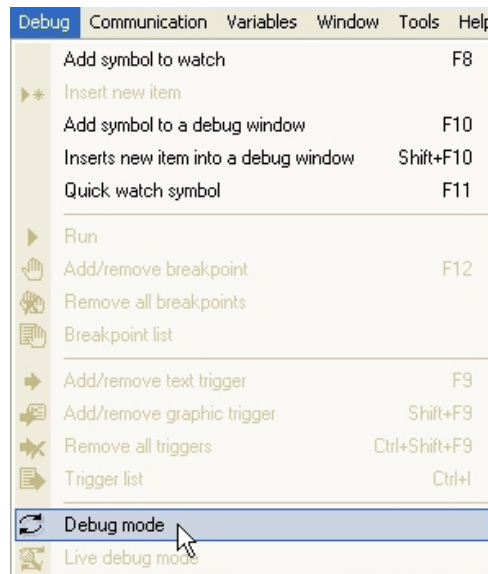
While both the *Watch* window and the Oscilloscope do not make use of the source code, all the other debuggers do: thus, Application requires the developer to switch on the debug mode, where changes to the source code are inhibited, before (s)he can access those debugging tools.

To switch on and off the debug mode, you can click on the corresponding item in the *De-*  
*bug* toolbar.





Alternatively, you can choose *Debug mode* from the *Project* menu.



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

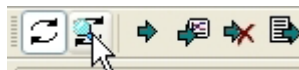


Note that you cannot enter the debug mode if the connection status differs from *Connected*.

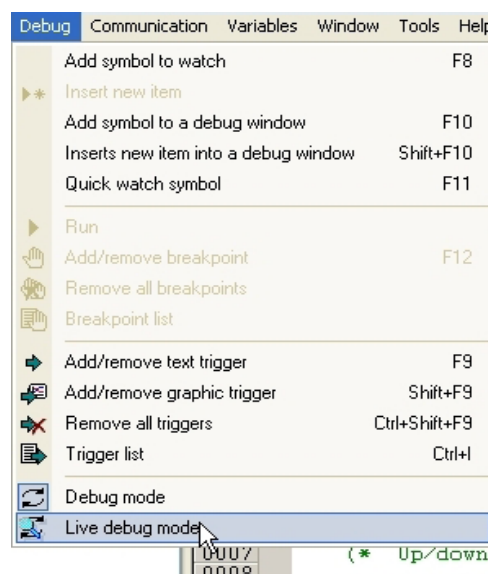
## 8.4 LIVE DEBUG

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 on the corresponding item in the *Debug* toolbar



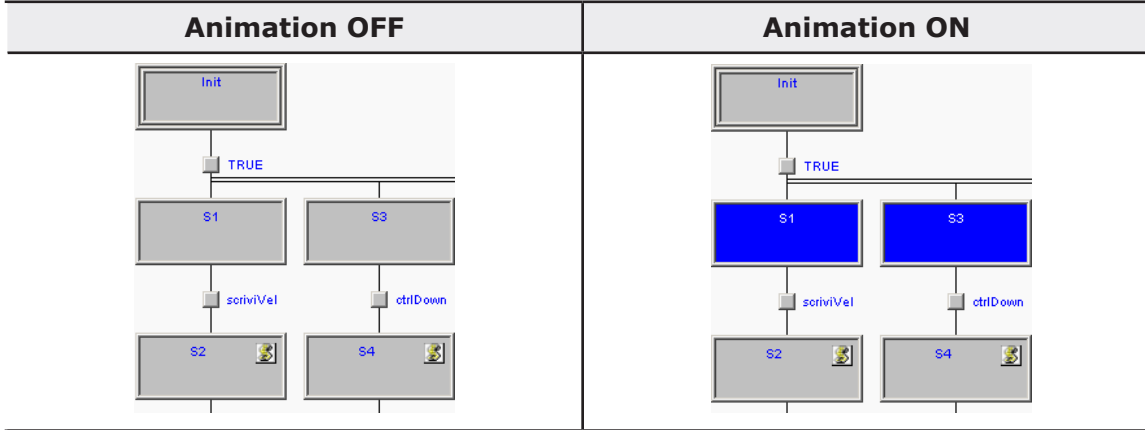
or choose *Live debug mode* from the *Project* menu.





**8.4.1 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.



In the left column, a portion of an SFC network is shown, diagram animation being off. In the right column the same portion of network is displayed when the live debug mode is active. The picture in the right column shows that steps *S1* and *S3* are currently active, whereas *Init*, *S2*, and *S4* are inactive.

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 on the video.

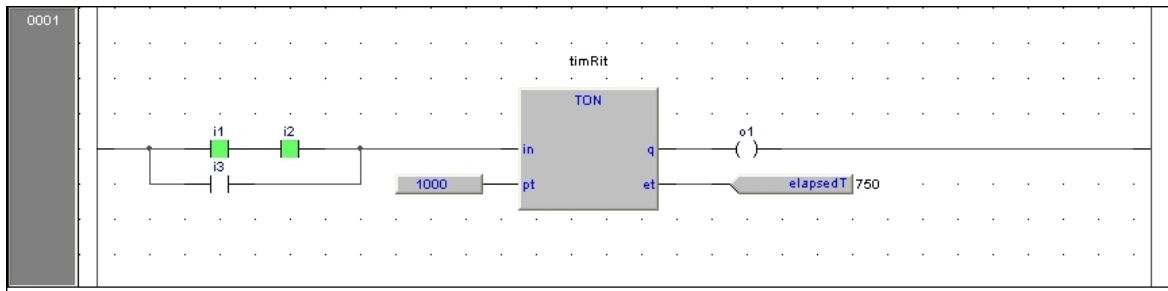
The fact that a step is never highlighted does not imply that its action is not executed, it may simply mean that the sampling rate is too slow to detect the execution.

**8.4.1.1 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.

**8.4.2 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*).

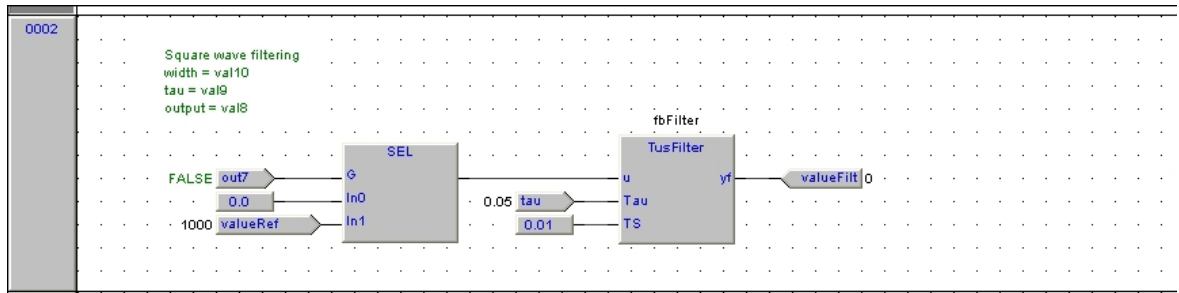




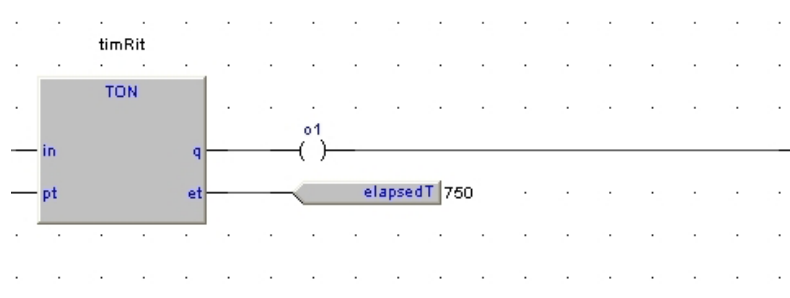
Note that the LD animation manager tests periodically the state of all the elements. It may happen that an element remains true for a slot of time too short to be displayed on the video. The fact that an element is never highlighted does not imply that its value never becomes true (the sampling rate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

## 8.4.3 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.

## 8.4.4 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.

```
0016 (* Analog output 0 = analog inp 0 + analog inp 1 *)
0017 aout0 := ainp0 + ainp1;
0018
0019 (* SFC state logic *)
0020 fbStati( enab := inp10, run := inp11, stop := inp12 );
0021
0022 cnt := cnt + 1;
0023
0024
0025
0026
0027
0028
```

## 8.5 TRIGGERS

### 8.5.1 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.



## 8.5.1.1 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.

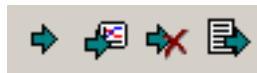
### Incompatibility with graphic trigger windows

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.

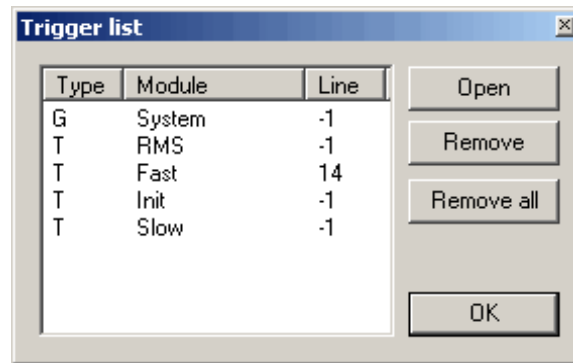
## 8.5.1.2 TRIGGER WINDOW TOOLBAR

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



Button	Command	Description
	<i>Set/Remove trigger</i>	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.
	<i>Graphic trace</i>	This button operates exactly as the above <i>Set/Remove trigger</i> , except for that it opens a graphic trigger window. It can be used likewise also to remove a graphic trigger window. Shortcut key: pressing <i>Shift + F9</i> is equivalent to clicking on <i>Set/Remove trigger</i> button.
	<i>Remove all triggers</i>	Pressing this key causes all the existing trigger windows and the graphic trigger window to be removed simultaneously. Shortcut key: pressing <i>Ctrl+Shift+F9</i> is equivalent to clicking on this button.
	<i>Trigger list</i>	This key opens a dialog listing all the existing trigger windows. Shortcut key: pressing <i>Ctrl+I</i> is equivalent to clicking on this button.



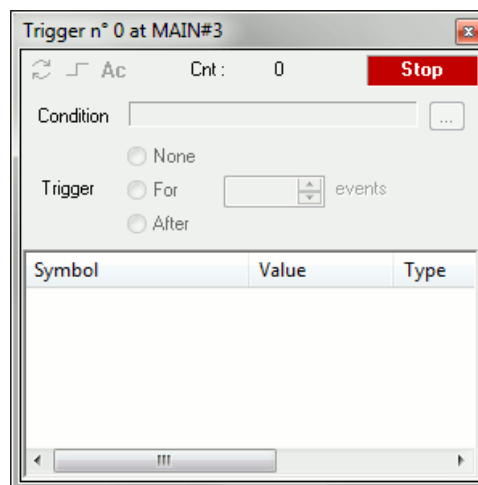


Each record refers to a trigger window, either graphic or textual. The following table explains the meaning of each field.

Field	Description
<i>Type</i>	<i>T</i> : trigger window. <i>G</i> : graphic trigger window.
<i>Module</i>	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.
<i>Line</i>	For the textual languages (IL, ST) indicates the line in which the trigger is placed. For the other languages the value is always <i>-1</i> .

### 8.5.1.3 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

<i>X</i>	Trigger identifier.
<i>ModuleName</i>	Name of the program, function, or function block where the trigger was placed.
<i>Location</i>	<p>Exact location of the trigger, within module <i>ModuleName</i>.                      If <i>ModuleName</i> is in IL, <i>Location</i> has the following format:                      N1                      Otherwise, if <i>ModuleName</i> is in FBD, it becomes:                      N2\$BT:PID                      where:                      N1 = instruction line number                      N2 = network number                      BT = block type (operand, function, function block, etc.)                      PID = block identifier</p>

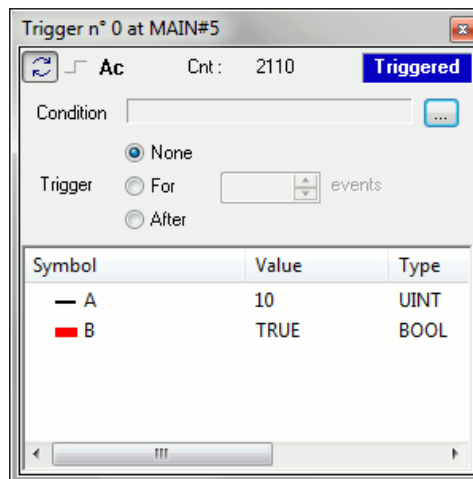
### Controls section

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 8.5.2.11).

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.



#### 8.5.1.4 TRIGGER WINDOW: DRAG AND DROP INFORMATION

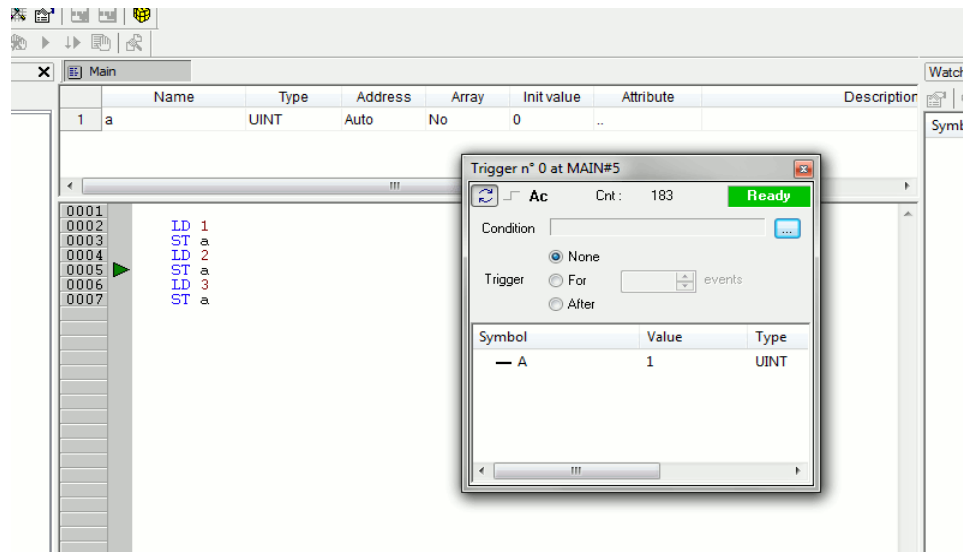
To watch a variable, you need to copy it to the lower section of the *Debug* window.

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.



## 8.5.1.5 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.

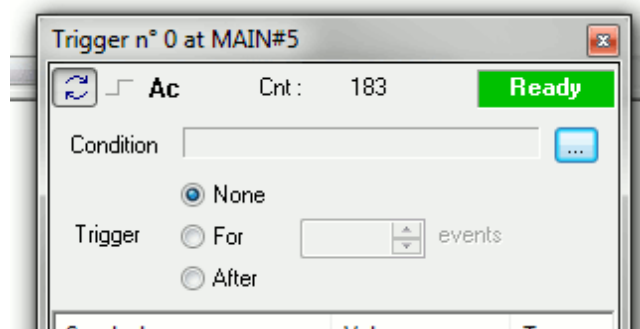
## 8.5.1.6 TRIGGER WINDOW CONTROLS

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.

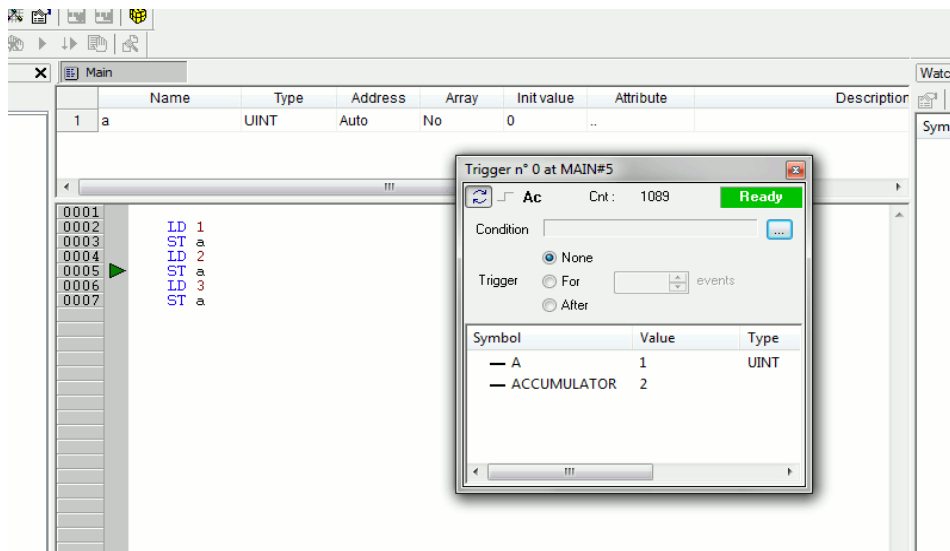
All controls except the *Accumulator display* are not accessible until at least one variable is dragged into the *Variables* window.

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





Button	Command	Description
	<i>Start/Stop</i>	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.
	<i>Single step execution</i>	This control is used to execute a single step trigger. It is enabled only when there is no active triggering session and <i>None</i> is selected. Specified condition is considered. After the single step trigger is done, triggering session automatically stops.
	<i>Accumulator display</i>	This control adds the <i>Accumulator</i> 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 <i>Accumulator</i> in column <i>Symbol</i> , the accumulator's value in column <i>Value</i> , <i>Type</i> is not specified and <i>Location</i> 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

Cnt : 26





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.

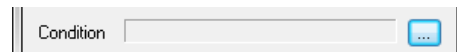


## Trigger state

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

	The trigger has not occurred during the current task execution.
	The trigger has occurred during the current task execution.
	System is not triggering. Triggering has not been started yet or it has been stopped by user or an halt condition has been reached.
	Communication with target interrupted, the state of the trigger window cannot be determined.

## User-defined condition

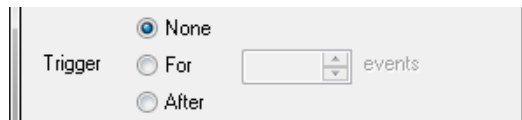


If you define a condition by using this control, the values in the *Debug* window are refreshed every time the window manager is triggered and the user-defined condition is true.

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



## 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  $N$ .

## 8.5.2 DEBUGGING WITH TRIGGER WINDOWS

### 8.5.2.1 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.



## 8.5.2.2 OPENING A TRIGGER WINDOW FROM AN IL MODULE

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

```

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 you can click the *Set/Remove trigger* button in the *Debug* toolbar



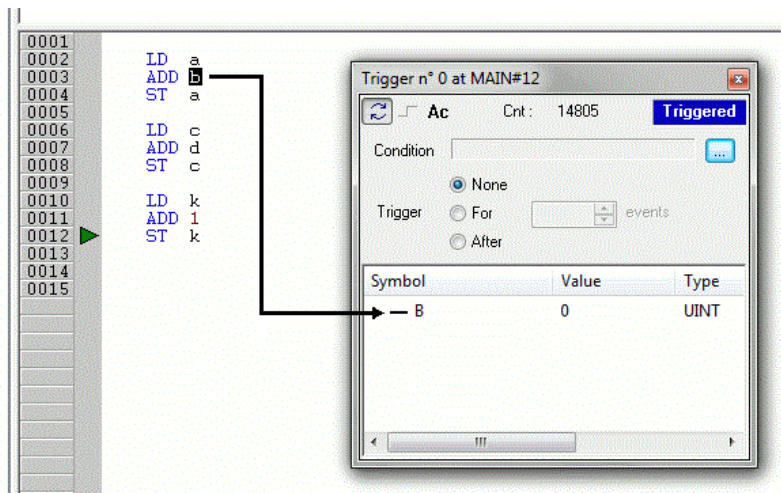
or you can press the *F9* key.

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.

## 8.5.2.3 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.

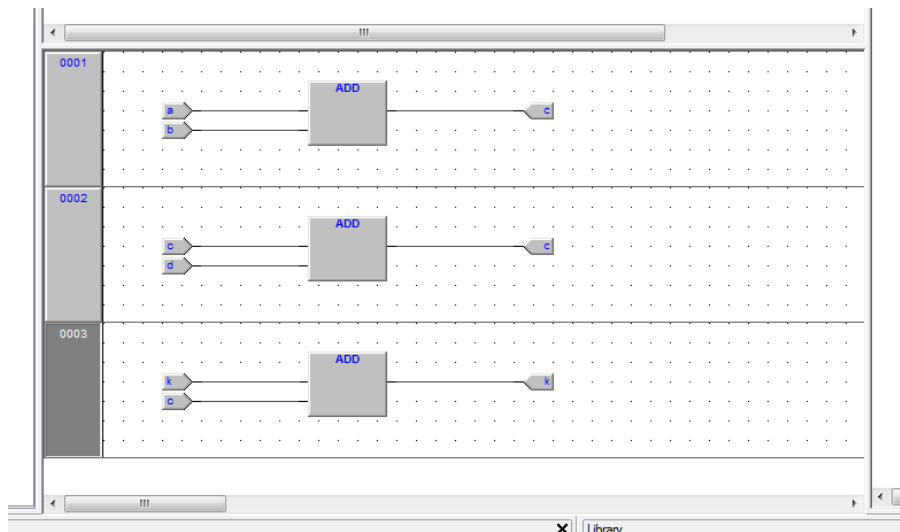


The same procedure applies to all the variables you wish to inspect.



## 8.5.2.4 OPENING A TRIGGER WINDOW FROM 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



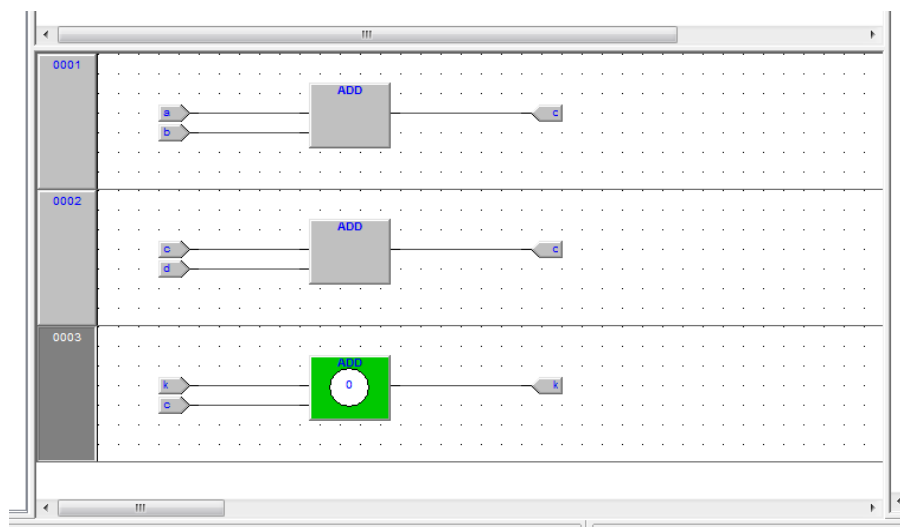
you must select the first available block preceding the selected variable. In the example of the above figure, you must move the cursor to network 3, and click the  $ADD$  block.

You can click the *Set/Remove trigger* button in the *Debug* bar



or you can press the  $F9$  key.

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.

### 8.5.2.5 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, press the *Watch* button in the FBD bar.



The cursor will become as follows.

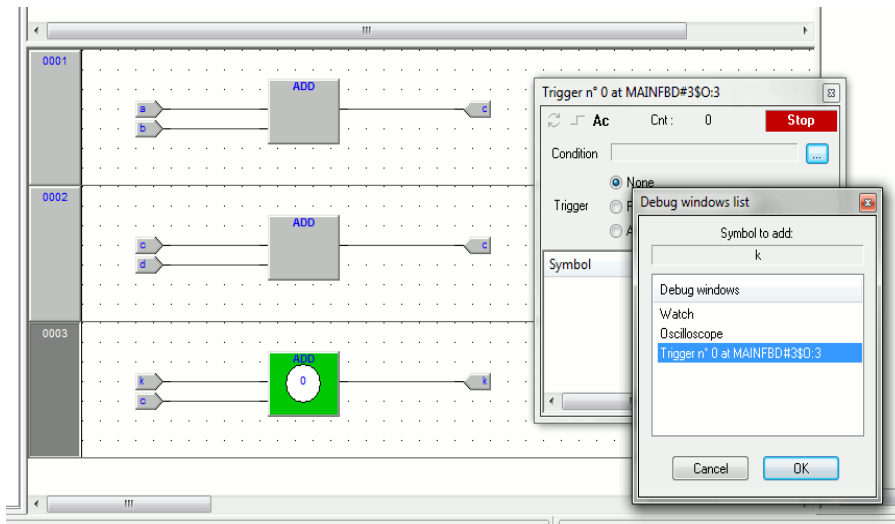


Now you can click the block representing the variable you wish to be shown in the trigger window.

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.

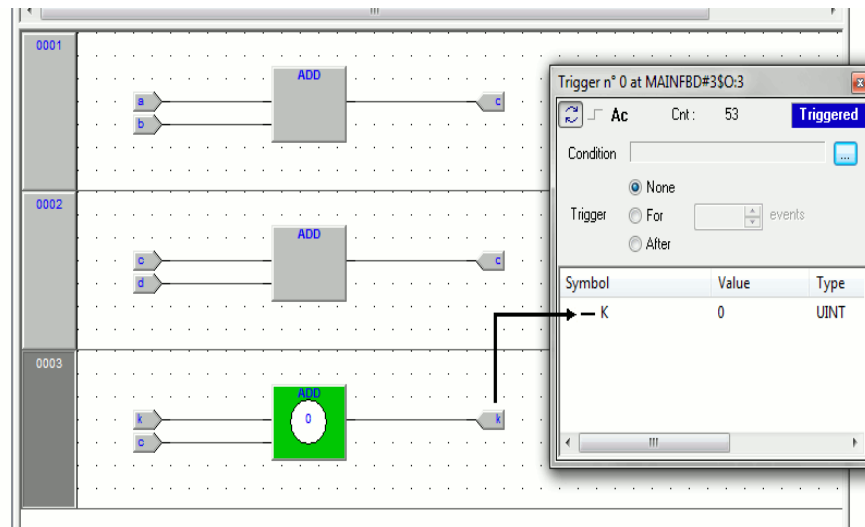






## SoMachine HVAC - Application

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.



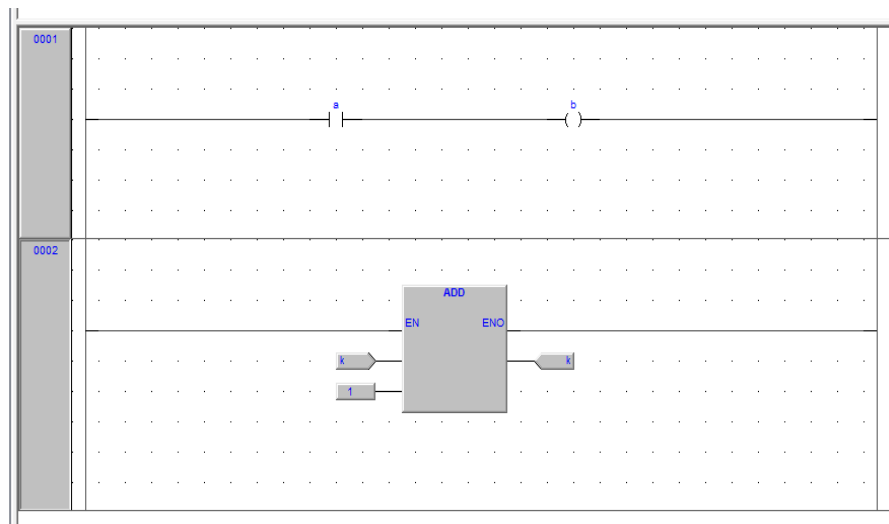
The same procedure applies to all the variables you wish to inspect.

Once you have added to the *Graphic watch* window all the variables you want to observe, you can press the normal cursor button, so as to let the cursor take back its original shape.

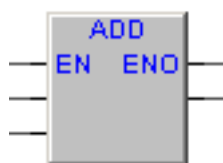


### 8.5.2.6 OPENING A TRIGGER WINDOW FROM AN LD MODULE

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



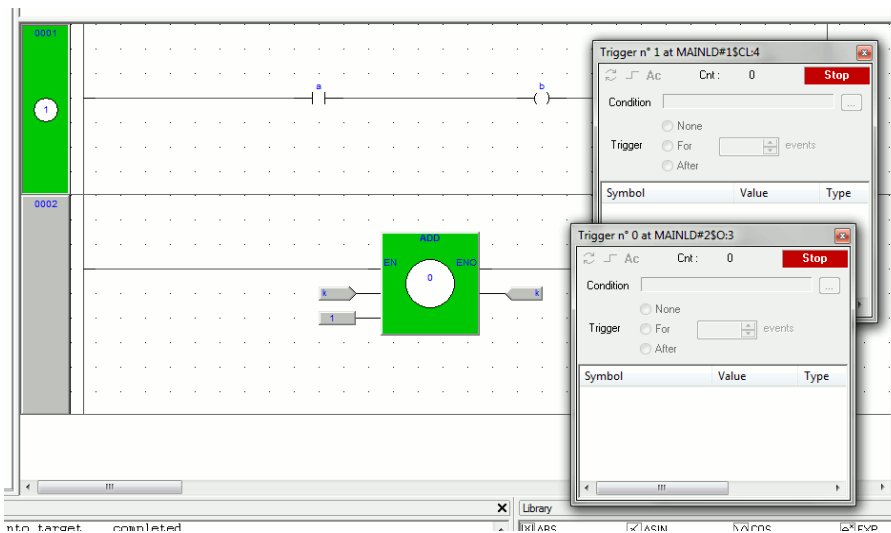
In this case, follow the SE instructions. Let us also assume that you want to know the value of some variables every time the processor reaches network number 1.

First you must click one of the items making up network number 1. Now you can click the *Set/Remove trigger* button in the *Debug* bar.



Alternatively you can press the *F9* key.

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.

### 8.5.2.7 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, press the *Watch* button in the *FBD* bar.



The cursor will become as follows.

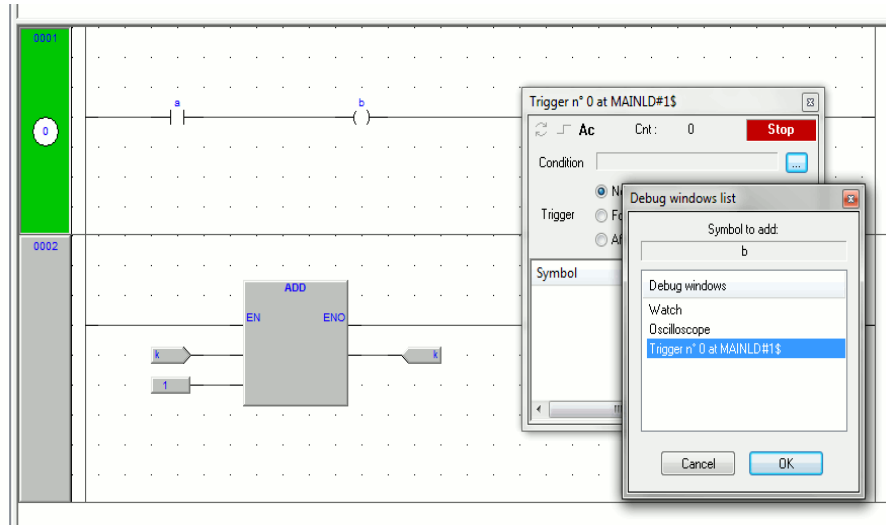




## SoMachine HVAC - Application

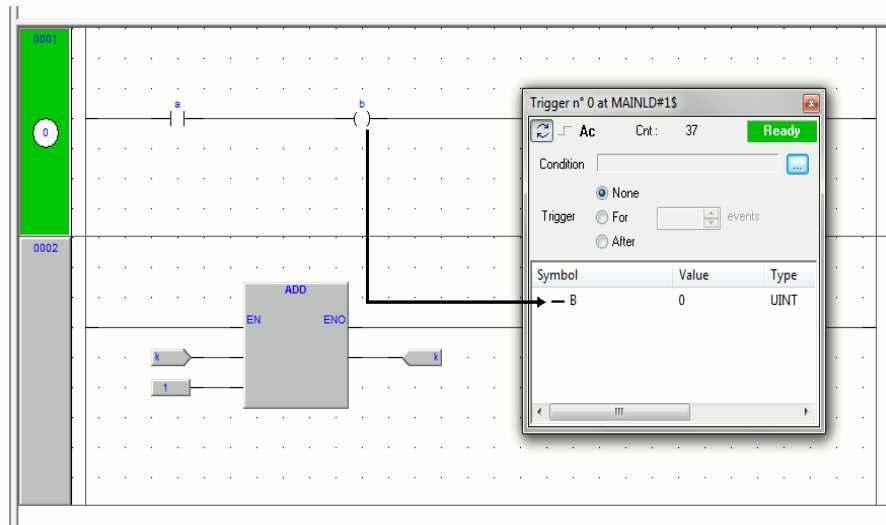
Now you can click the item representing the variable you wish to be shown in the trigger window.

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 press the *Normal cursor* button, so as to restore the original shape of the cursor.





8.5.2.8 OPENING A TRIGGER WINDOW FROM AN ST MODULE

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

```

0001
0002   a := b * b;
0003   c := c + SHR( a, 16#04 );
0004
0005   d := e * e;
0006   f := f + SHR( d, 16#04 );
0007
    
```

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

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

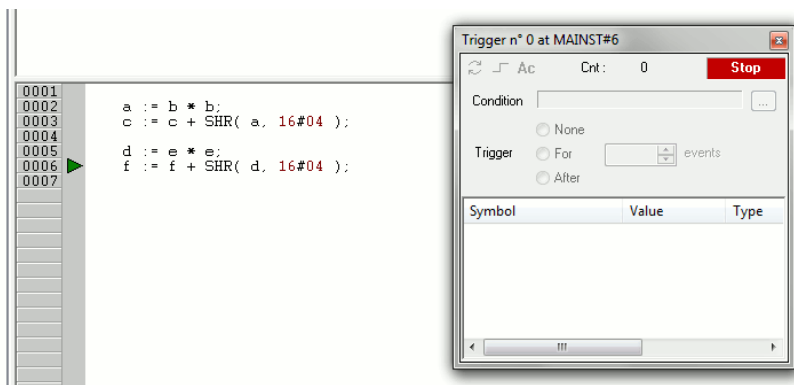
is executed. To do so, move the cursor to line 6.

Then you can click the *Set/Remove trigger* button in the *Debug* toolbar



or you can press the *F9* key.

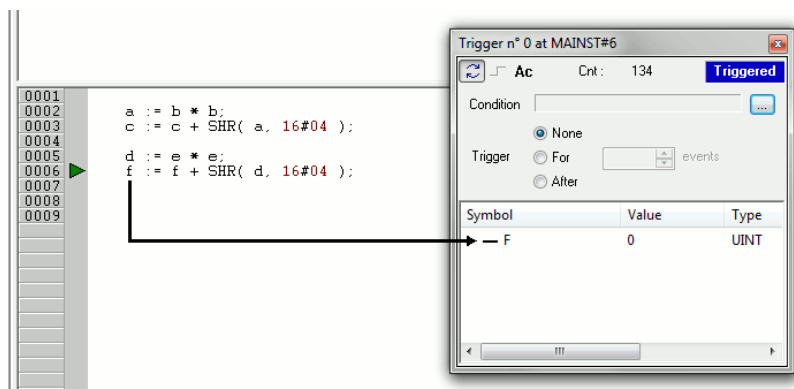
In both cases, a green arrowhead appears next to the line number, and the related trigger window pops up.



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..

8.5.2.9 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.

## 8.5.2.10 REMOVING A VARIABLE FROM THE TRIGGER WINDOW

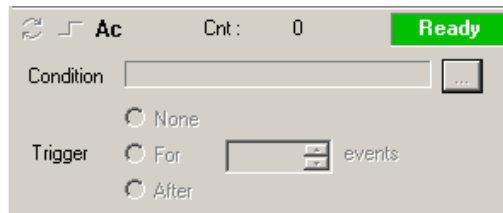
If you want a variable not to be displayed any more in the trigger window, select it by clicking its name once, then press the *Del* key.

## 8.5.2.11 USING CONTROLS

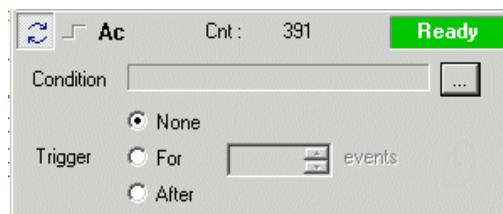
This paragraph deals with trigger windows controls, which allow you to better supervise the working of this debugging tool to get more information on the code under scope. The main purpose of trigger window controls is to let you define more limiting conditions, so that variables in *Variables* window 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 8.5.1.5), 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.

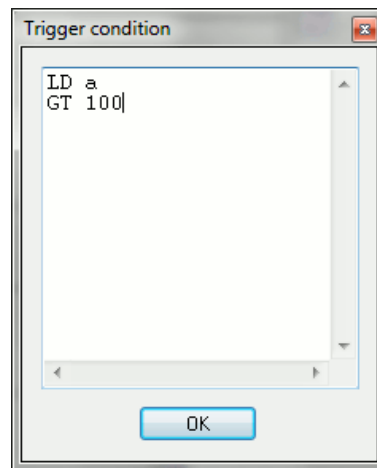
## Defining a condition

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.

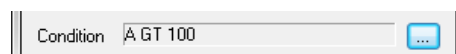


When you do so, a text window pops up, where you can write the IL code that sets the condition.



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.



## 8.5.2.12 CLOSING A TRIGGER WINDOW AND REMOVING A TRIGGER

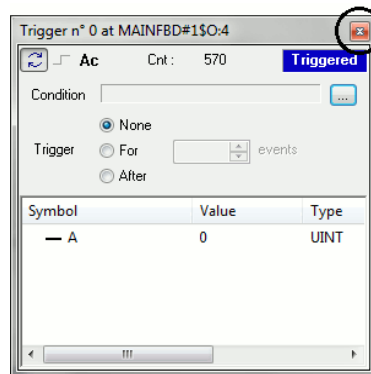
This web page deals with what you can do when you finish a debug session with a trigger window. You can choose between the following options.

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

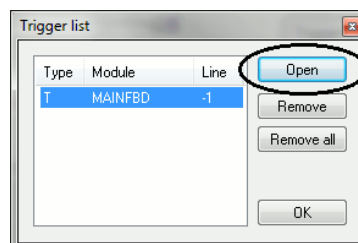
Notice that the actions listed above produce very different results.

### 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.



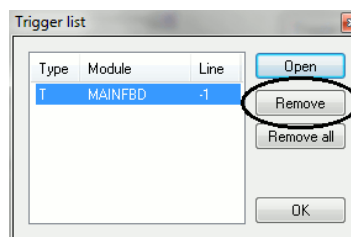
As a matter of fact, if later you want to resume debugging with a trigger window that you previously hid, you just need to open the *Trigger list* window, to select the record referred to that trigger window, and to click the *Open* button.



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.

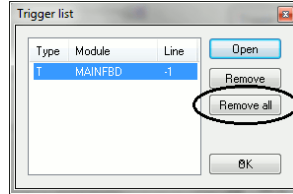




Alternatively, you can move the cursor to the line (if the module is in IL or ST), or click the block (if the module is in FBD or LD) where you placed the trigger. Now press the *Set/Remove trigger* button in the *Debug* toolbar.

### 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.



## 8.6 GRAPHIC TRIGGERS

### 8.6.1 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.

#### 8.6.1.1 PRE-CONDITIONS TO OPEN A GRAPHIC TRIGGER WINDOW

##### No need for special compilation

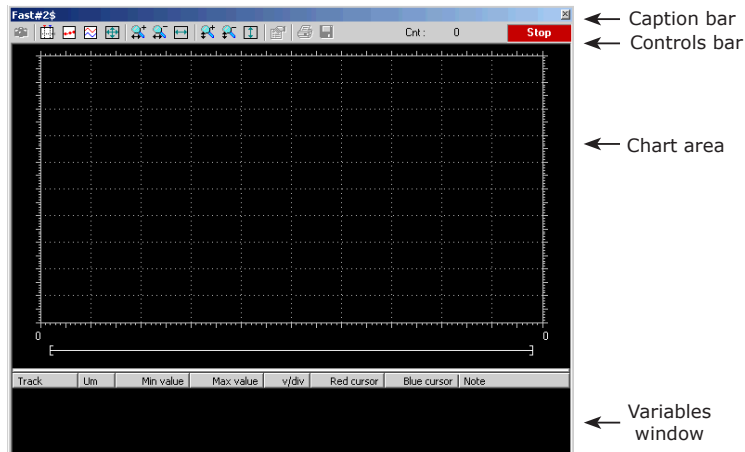
All the Application debugging tools operate at run-time. Thus, unlike other programming languages such as C++, the compiler does not need to be told whether or not to support trigger windows: given a PLC code, the compiler's output is unique, and there is no distinction between debug and release version.

##### 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.

#### 8.6.1.2 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.







## The caption bar

The *Caption* bar at the top of the pop-up window shows information on the location of the trigger which causes the variables listed in the *Variables* window to be sampled.

The text in the caption has the following format:

`ModuleName#Location`

Where

ModuleName	Name of program, function, or function block where the trigger was placed.
Location	Exact location of the trigger, within module <code>ModuleName</code> . If <code>ModuleName</code> is in IL, ST, <code>Location</code> has the format: <code>N1</code> Otherwise, if <code>ModuleName</code> is in FBD, LD, it becomes: <code>N2\$BT:PID</code> <code>N1</code> = instruction line number <code>N2</code> = network number <code>BT</code> = block type (operand, function, function block, etc.) <code>PID</code> = 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 8.6.1.5).

## 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.

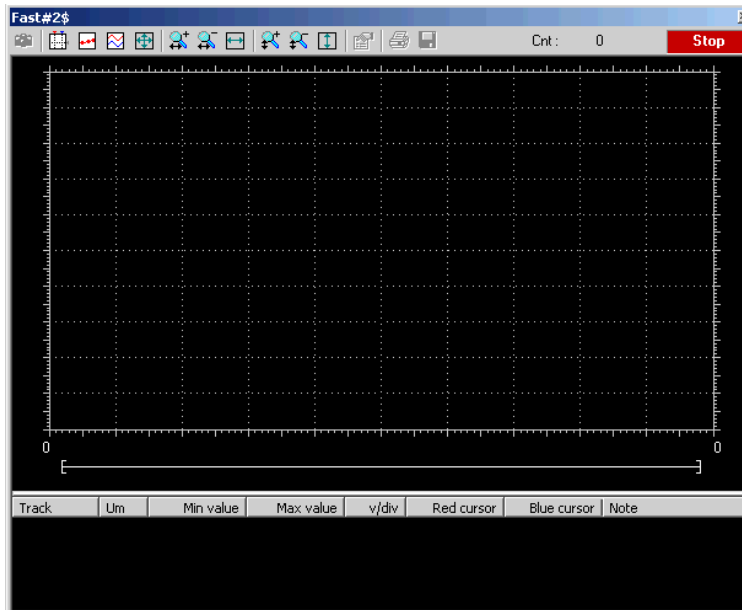
## The Variables window

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.



## 8.6.1.3 GRAPHIC TRIGGER WINDOW:DRAG AND DROP INFORMATION

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



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.

Track	Um	Min value	Max value	v/div	Red cursor	Blue cursor	Note
:Fast.Fast.fbFilter.u		0.000	1000.000	533.333	0.000	0.000	:Fast.Fast
:Fast.Fast.fbFilter.yf		0.000	1000.000	533.333	44.808	0.000	:Fast.Fast
out3		-2050.000	2050.000	2186.67	-2000.000	1100.000	global
out0		0.000	0.000	1.06667	0.000	0.000	global

Field	Description
<i>Track</i>	Name of the variable.
<i>Um</i>	Unit of measurement.
<i>Min value</i>	Minimum value in the record set.
<i>Max value</i>	Maximum value in the record set.
<i>Cur value</i>	Current value of the variable.
<i>v/div</i>	How many engineering units are represented by a unit of the y-axis (i.e. the space between two ticks on the vertical axis).
<i>Blue cursor</i>	Value of the variable at the intersection with the line identified by the blue cursor.
<i>Red cursor</i>	Value of the variable at the intersection with the line identified by the red cursor.
<i>Horz cursor</i>	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.



## 8.6.1.4 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.

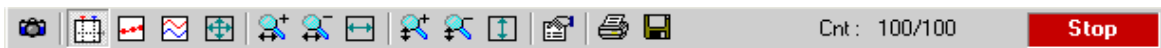
The value of the variables in the column *Track* is read from memory just before the marked instruction and immediately after the previous instruction.







## 8.6.1.5 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
	<i>Start graphic trace</i>	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.
	<i>Enable/Disable cursors</i>	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.
	<i>Show samples</i>	This control is used to put in evidence the exact point in which the variables are triggered at each sample.
	<i>Split variables</i>	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.
	<i>Show all values</i>	It is used to fill in the graph window all the values sampled for the selected variables in the current recordset.
	<i>Horizontal Zoom In and Zoom Out</i>	Zooming in is an operation that makes the curves in the <i>Chart</i> 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.



Button	Command	Description
	<i>Horizontal show all</i>	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.
	<i>Vertical Zoom In and Zoom Out</i>	<i>Vertical Zoom</i> acts only on the vertical axis.
	<i>Vertical show all</i>	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.
	<i>Graphic trigger window properties</i>	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. <a href="#">Click here to access this section.</a>
	<i>Print chart</i>	Push this button to print both the <i>Chart area</i> and the <i>Variables</i> window.
	<i>Save chart</i>	Press this button to save the chart.

## Trigger counter

Cnt: 25/100

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.

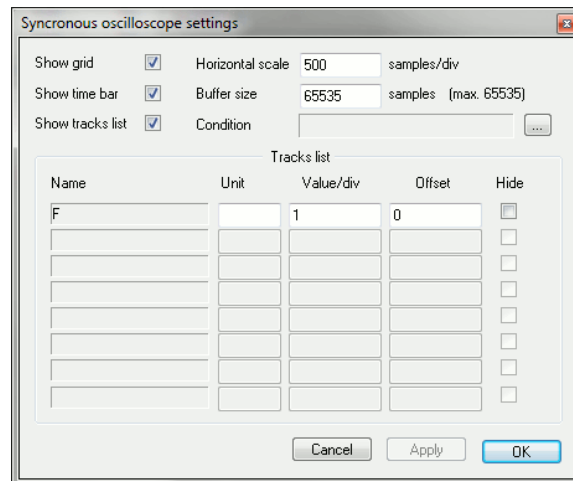
<b>Ready</b>	No sample(s) taken, as the trigger has not occurred during the current task execution.
<b>Triggered</b>	Sample(s) collected, as the trigger has occurred during the current task execution.
<b>Stop</b>	The trigger counter indicates that a number of samples has been collected satisfying the user request or memory constraints, thus the acquisition process is stopped.
<b>Error</b>	Communication with target interrupted, the state of the trigger window cannot be determined.



## 8.6.1.6 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.

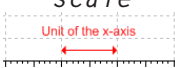
### General



### Control

Control	Description
<i>Show grid</i>	Tick this control to display a grid in the <i>Chart area</i> background.
<i>Show time bar</i>	The scroll bar at the bottom of the <i>Chart area</i> is available as long as this box is checked.
<i>Show tracks list</i>	The <i>Variables</i> window is shown as long as this box is checked, otherwise the <i>Chart area</i> extends to the bottom of the graphic trigger window.

### Values

Control	Description
<i>Horizontal scale</i> 	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.
<i>Buffer size</i>	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.

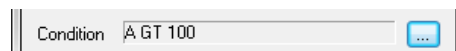
Control	Description
<i>Unit</i>	Unit of measurement, printed in the table of the <i>Variables</i> window.
<i>Value/div</i>	$\Delta$ value per unit of the y-axis. By unit of the y-axis is meant the space between two horizontal lines of the background grid.
<i>Hide</i>	Check this flag to hide selected 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.



## 8.6.2 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.

### 8.6.2.1 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.

```

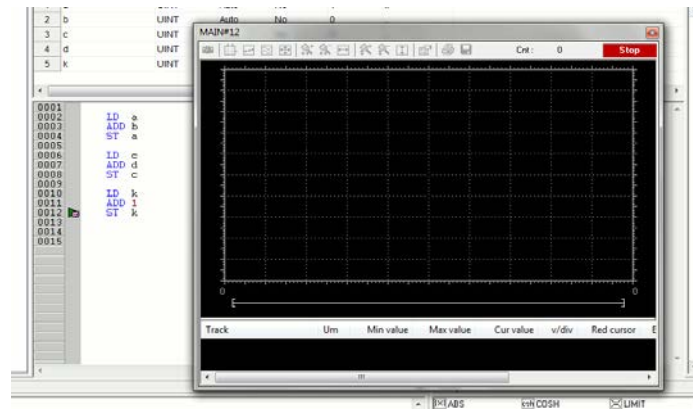
0009
0010      LD  k
0011      ADD 1
0012      ST  k
0013
    
```

Then click the *Graphic trace* button in the *Debug* toolbar.





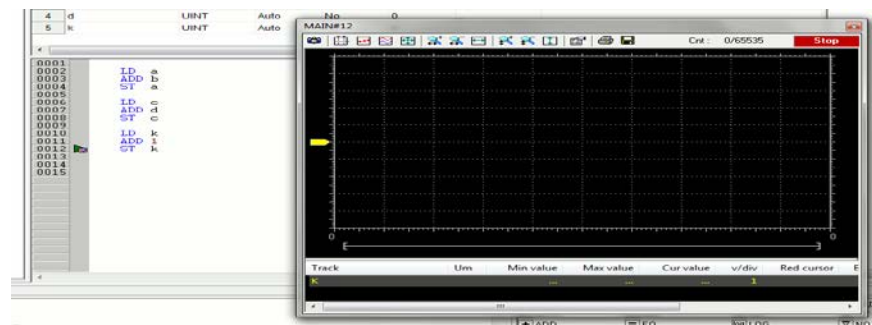
A green arrowhead appears next to the line number, and the graphic trigger window pops up.



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.

### 8.6.2.2 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.

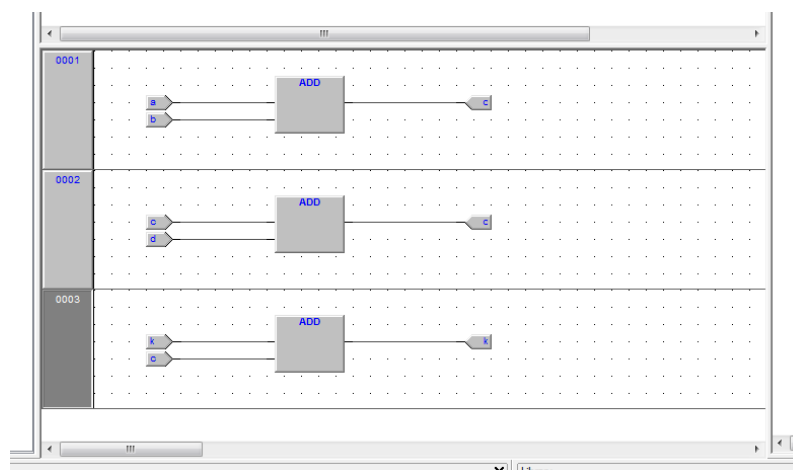


The same procedure applies to all the variables you wish to inspect.

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.

### 8.6.2.3 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

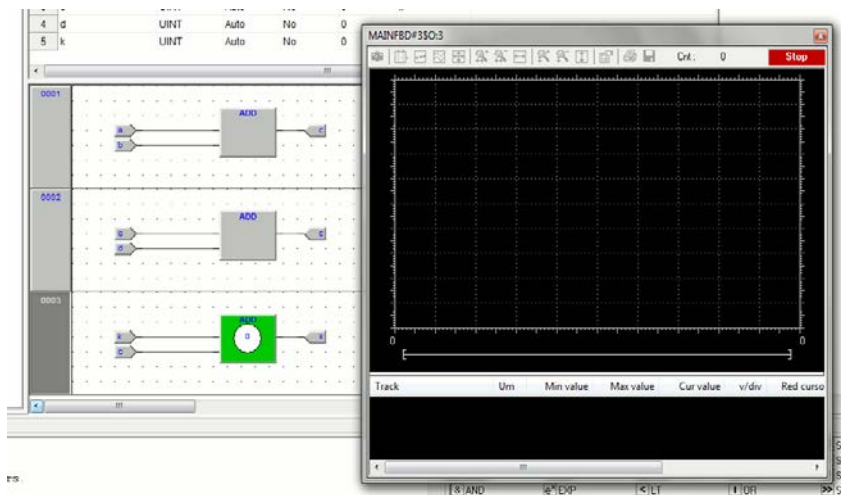


you must select the first available block preceding the selected variable. In the example of the above figure, you must move the cursor to network 3, and click the `ADD` block.

Now click the *Graphic trace* button in the *Debug* toolbar.



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.

### 8.6.2.4 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, press the *Watch* button in the FBD bar.



The cursor will become as follows.







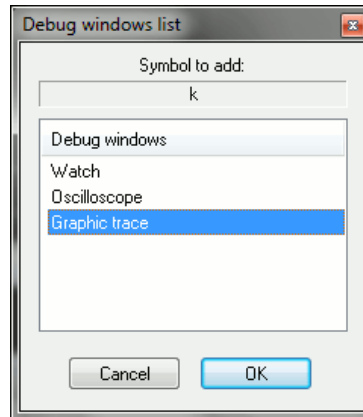
## SoMachine HVAC - Application

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

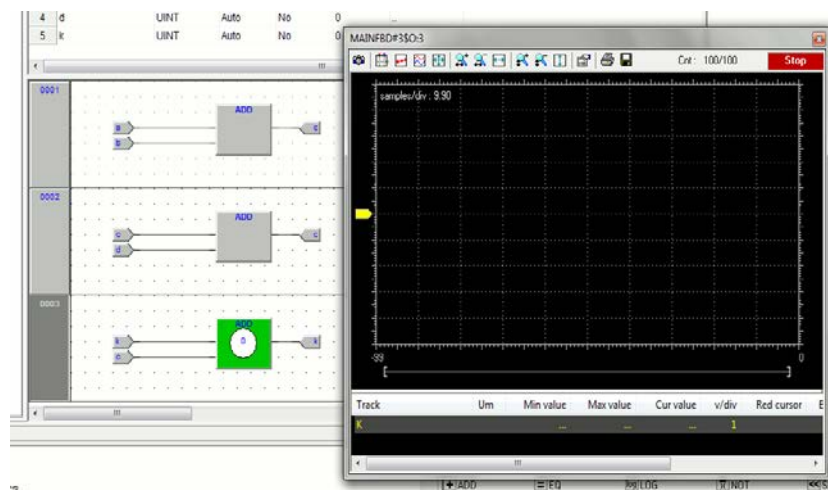
In the example we are considering, click the button block.



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 press the *Normal cursor* button, in order to restore the original cursor.

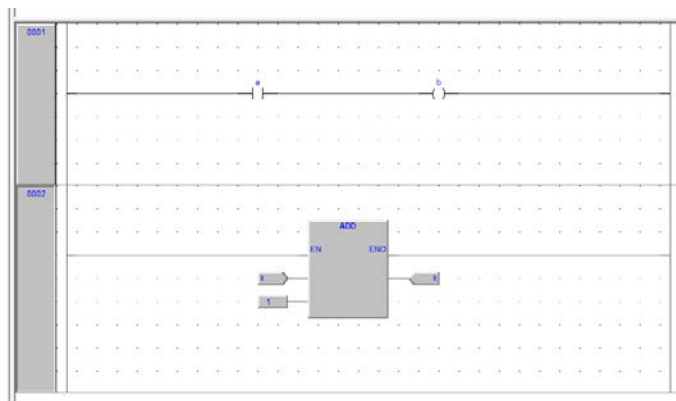


Once the first variable is dropped into a graphic trace, the *Graphic properties* window is automatically shown and allows the user to setup sampling and visualization properties.

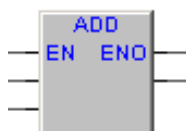


## 8.6.2.5 OPENING THE GRAPHIC TRIGGER WINDOW FROM AN LD MODULE

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

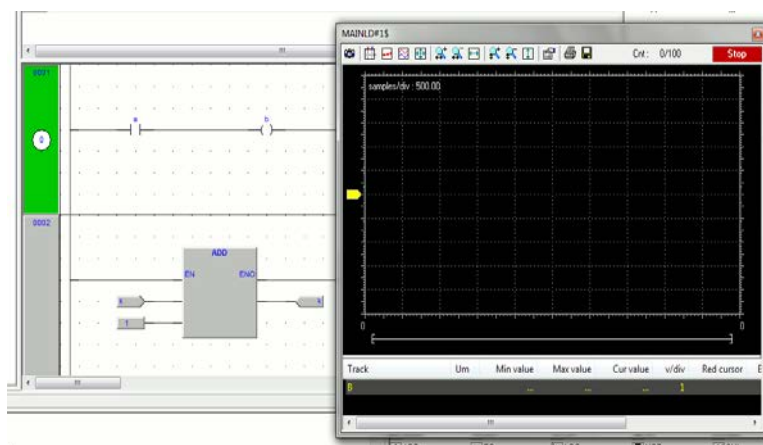


In this case, follow the instructions. Let us also assume that you want to know the value of some variables every time the processor reaches network number 1.

Click one of the items making up network nr. 1, then press the *Graphic trace* button in the *Debug* toolbar.



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.





## SoMachine HVAC - Application

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.

### 8.6.2.6 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, press the *Watch* button in the FBD bar.



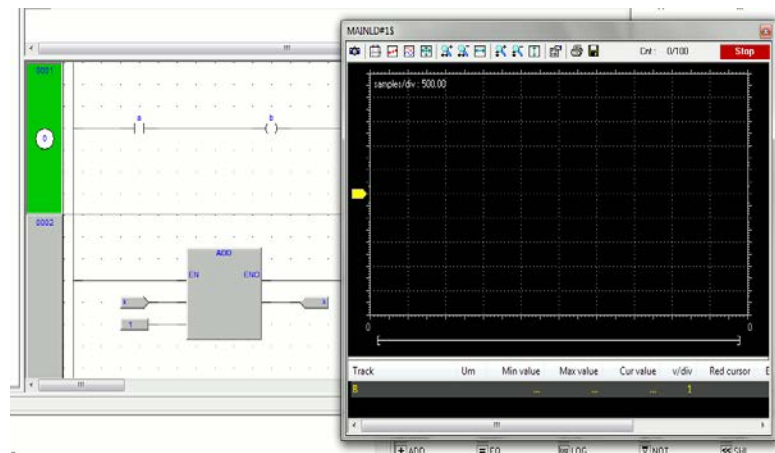
The cursor will become as follows.



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 press again the *Normal cursor* button, 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.



## 8.6.2.7 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

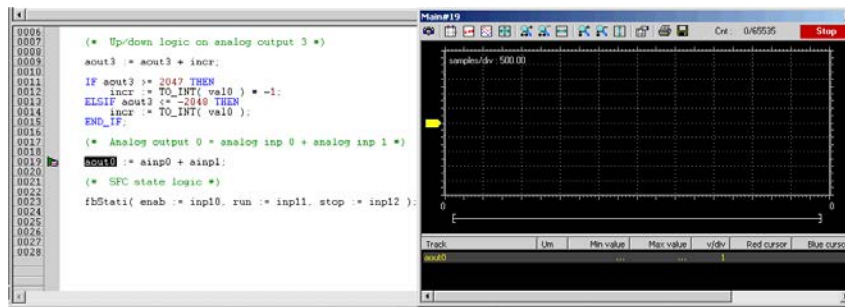
```
f := f+ SHR( d, 16#04 )
```

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

Then click the *Graphic trace* button in the *Debug* toolbar.



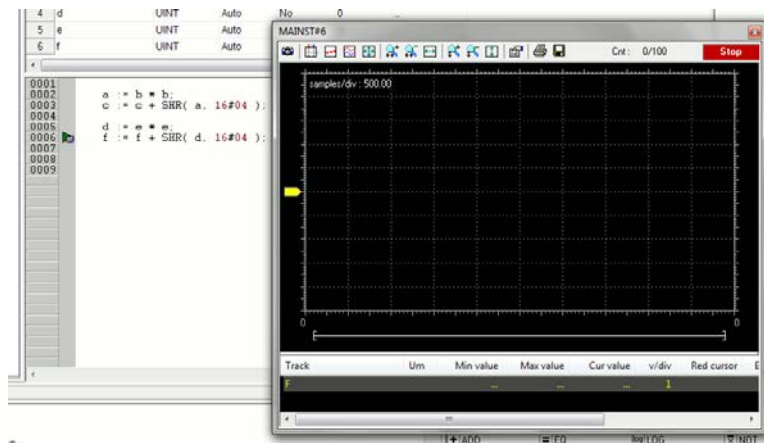
A green arrowhead appears next to the line number, and the *Graphic trigger* window pops up.



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.

## 8.6.2.8 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.

## 8.6.2.9 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.

## 8.6.2.10 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.

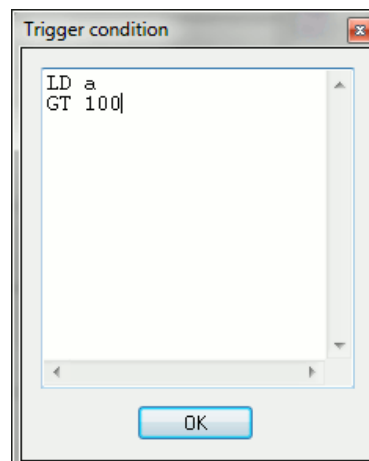
### Defining a condition

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 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.

## 8.6.2.11 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  $\text{G}$ );
- 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.







## 9. APPLICATION REFERENCE

### 9.1 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.

#### 9.1.1 FILE MENU

Command	Description
<i>New project</i>	Lets you create a new Application project.
<i>Import project from target</i>	Lets you upload the project from the target device.
<i>Open project</i>	Lets you open an existing Application project.
<i>View project</i>	Opens an existing Application project in read-only mode.
<i>Save project</i>	Same as <i>Save all</i> , but it saves also the <i>ppj</i> file. Note that, since all modifications to a Application project are first applied in memory only, you need to release the <i>Save project</i> command to make them permanent.
<i>Save project As</i>	Asks you to specify a new project name and a new location, and saves there a copy of all the files of the project.
<i>Close project</i>	Asks you whether you want to keep unsaved changes, then closes the active project.
<i>New text file</i>	Opens a blank new generic text file.
<i>Open file</i>	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.
<i>Save</i>	Lets you save the document in the currently active window.
<i>Close</i>	Closes the document in the currently active window.
<i>Options</i>	Opens the <i>Programming environment options</i> dialog box.
<i>Print</i>	Displays a dialog box, which lets you set printing options and print the document in the currently active window.
<i>Print preview</i>	Shows a picture on your video, that reproduces faithfully what you get if you print the document in the currently active window.
<i>Print project</i>	Prints all the documents making up the project.
<i>Printer setup</i>	Opens the <i>Printer setup</i> dialog box.
<i>..recent..</i>	Lists a set of <i>ppj</i> file of recently opened Application projects. Click one of them, if you want to open the relevant project.
<i>Exit</i>	Closes Application.



## 9.1.2 EDIT MENU

Command	Description
<i>Undo</i>	Cancels last change made in the document.
<i>Redo</i>	Restores the last change canceled by <i>Undo</i> .
<i>Cut</i>	Removes the selected items from the active document and stores them in a system buffer.
<i>Copy</i>	Copies the selected items to a system buffer.
<i>Paste</i>	Pastes in the active document the contents of the system buffer.
<i>Delete</i>	Deletes the selected item.
<i>Delete line</i>	Deletes the whole source code line.
<i>Find in project</i>	Opens the <i>Find in project</i> dialog box.
<i>Bookmarks</i>	Lets you set, remove, and move between bookmarks.
<i>Go to line</i>	Allows you to quickly move to a specific line in the source code editor.
<i>Find</i>	Asks you to type a string and searches for its first instance within the active document from the current location of the cursor.
<i>Find next</i>	Iterates the search previously performed by the <i>Find</i> command.
<i>Replace</i>	Allows you to automatically replace one or all the instances of a string with another string.
<i>Insert/Move mode</i>	Editing mode which allows you to insert and move blocks.
<i>Connection mode</i>	Editing mode which allows you to draw logical wires to connect pins.
<i>Watch mode</i>	Editing mode which allows you to add variables to any debugging tool.

## 9.1.3 VIEW MENU

Command	Description
<i>Main Toolbar</i>	If checked, displays the <i>Main</i> toolbar, otherwise hides it.
<i>Status bar</i>	If checked, displays the <i>Status</i> bar, otherwise hides it.
<i>Debug bar</i>	If checked, displays the <i>Debug</i> bar, otherwise hides it.
<i>FBD bar</i>	If checked, displays the <i>FBD</i> toolbar, otherwise hides it.
<i>LD bar</i>	If checked, displays the <i>LD</i> toolbar, otherwise hides it.
<i>SFC bar</i>	If checked, displays the <i>SFC</i> bar, otherwise hides it.
<i>Project bar</i>	If checked, displays the <i>Project</i> bar, otherwise hides it.
<i>Network</i>	If checked, displays the <i>Network</i> toolbar, otherwise hides it.
<i>Document bar</i>	If checked, displays the <i>Document</i> bar, otherwise hides it.
<i>Workspace</i>	If checked, displays the <i>Workspace</i> (also called <i>Project</i> window), otherwise hides it.
<i>Library</i>	If checked, displays the <i>Libraries</i> window, otherwise hides it.
<i>Output</i>	If checked, displays the <i>Output</i> window, otherwise hides it.



Command	Description
<i>Async Graphic window</i>	If checked, displays the <i>Oscilloscope</i> window, otherwise hides it.
<i>Watch window</i>	If checked, displays the <i>Watch</i> window, otherwise hides it.
<i>Force I/O bar</i>	If checked, displays the <i>Force I/O</i> bar, otherwise hides it.
<i>PLC run-time status</i>	If checked, displays the PLC run-time window, otherwise hides it.
<i>Full screen</i>	Expands the currently active document window to full screen. Press <i>Esc</i> to restore the normal appearance of the Application interface.
<i>Grid</i>	If checked, displays a dotted grid in a graphical source code editor background.

## 9.1.4 PROJECT MENU

Command	Description
<i>New object</i>	Opens another menu which lets you create a new POU or declare a new global variable.
<i>Copy object</i>	Copies the object currently selected in the Workspace.
<i>Paste object</i>	Pastes the previously copied object.
<i>Duplicate object</i>	Duplicates the object currently selected in the Workspace, and asks you to type the name of the copy.
<i>Delete object</i>	Deletes the currently selected object. As explained above, you need to release the <i>Save project</i> command to definitively erase a document from your project.
<i>PLC object properties</i>	Shows properties and description of the object currently selected in the Workspace.
<i>Object browser</i>	Opens the <i>Object browser</i> , which lets you navigate between objects.
<i>Compile</i>	Asks you whether to save unsaved changes, then launches the Application compiler.
<i>Recompile all</i>	Recompiles the project.
<i>Generate redistributable source module</i>	Generates an RSM file.
<i>Import object from library</i>	Lets you import a Application object from a library.
<i>Export object to library</i>	Lets you export a Application object to a library.
<i>Library manager</i>	Opens the <i>Library manager</i> .
<i>Macros</i>	Opens another menu which lets you create/delete macros.
<i>Select target</i>	Lets you change the target.
<i>Refresh current target</i>	Lets you update the target file for the same version.
<i>Options...</i>	Lets you specify the project options.



### 9.1.5 DEBUG MENU

Command	Description
<i>Simulation mode</i>	Open/close the integrated simulation environment.
<i>Debug mode</i>	Switches the debug mode on.
<i>Live debug mode</i>	Switches the live debug mode on.
<i>Add symbol to watch</i>	Adds a symbol to the <i>Watch</i> window.
<i>Insert new item into watch</i>	Inserts a new item into the <i>Watch</i> window.
<i>Add symbol to a debug window</i>	Adds a symbol to a debug window.
<i>Insert new item into a debug window</i>	Inserts a new item into a debug window.
<i>Run</i>	Restarts program after a breakpoint is hit.
<i>Add/Remove breakpoint</i>	Adds/removes a breakpoint.
<i>Remove all breakpoints</i>	Removes all the active breakpoints.
<i>Breakpoint list</i>	Lists all the active breakpoints.
<i>Add/remove text trigger</i>	Adds/removes a text trigger.
<i>Add/remove graphic trigger</i>	Adds/removes a graphic trigger.
<i>Remove all triggers</i>	Removes all the active triggers.
<i>Trigger list</i>	Lists all the active triggers.

### 9.1.6 ON-LINE MENU

Command	Description
<i>Set up communication...</i>	Lets you set the properties of the connection to the target.
<i>Connect</i>	Application tries to establish a connection to the target.
<i>Download code</i>	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.
<i>Download options</i>	Lets you set the properties of the source code downloaded to the target.
<i>Force image upload</i>	If the target device is connected, lets you upload the img file.
<i>Force debug symbols upload</i>	If the target device is connected, lets you upload the debug symbols file.
<i>Start/Stop watch value</i>	Freezes/resumes refreshment of the <i>Watch</i> window.



## 9.1.7 SCHEME MENU

Command	Description	Network type
<i>Network&gt; New&gt; Top</i>	Adds a blank network at the top of the active document.	LD/FBD
<i>Network&gt; New&gt; Bottom</i>	Adds a blank network at the bottom of the active document.	LD/FBD
<i>Network&gt; New&gt; Before</i>	Adds a blank network before the selected network in the active document.	LD/FBD
<i>Network &gt;New &gt; After</i>	Adds a blank network after the selected network in the active document.	LD/FBD
<i>Network &gt;Label</i>	Assigns a label to the selected network, so that it can be indicated as the target of a jump instruction.	LD/FBD
<i>Object &gt;New</i>	Lets you insert a new object into the selected network.	All
<i>Object &gt; Modify</i>	Lets you to add/remove/change pins to transitions.	SFC
<i>Object &gt; Instance name</i>	Lets you assign a name to an instance of a function block, that you have previously selected by clicking it once.	LD/FBD
<i>Object &gt; Open source</i>	Opens the editor by which the selected object was created, and displays the relevant source code: <ul style="list-style-type: none"><li>- if the object is a program, or a function, or a function block, this command opens its source code;</li><li>- if the object is a variable or a parameter, this command opens the corresponding variable editor;</li><li>- if the object is a standard function or an operator, this command opens nothing.</li></ul>	LD/FBD
<i>Code Object &gt; New Action</i>	Lets you to add an action in the active document.	SFC
<i>Code Object &gt; New Transition code</i>	Lets you to add a transition in the active document.	SFC
<i>Auto connect</i>	If checked, enables autoconnection, that is automatic creation of a logical wire linking the pins of two blocks, when they are brought close.	All
<i>Delete invalid connection</i>	Removes all invalid connections, represented by a red line in the active scheme.	All
<i>Connect Paral</i>	Activates the parallel insertion mode.	LD
<i>Connect series</i>	Activates the series insertion mode.	LD
<i>Increment pins</i>	By default some operators like <i>ADD</i> , <i>MUL</i> , etc. have two input pins, however you may occasionally need to perform such operations on more than two operands. This command allows you to add as many input pins as to reach the required number of operands.	LD/FBD



Command	Description	Network type
<i>Decrement pins</i>	Undoes the <i>Increment pins</i> command.	LD/FBD
<i>Enable EN/ENO pins</i>	Adds the <i>enable in/enable out</i> pins to the selected block. The code implementing the selected block will be executed only when the <i>enable in</i> signal is true. The <i>enable out</i> signal simply repeats the value of <i>enable in</i> , allowing you either to enable or to disable a set of blocks in cascade.	LD/FBD
<i>Object properties</i>	Shows some properties of the selected block: <ul style="list-style-type: none"> <li>- if the object is a function or a function block, displays a table with the input and output variables;</li> <li>- if the object is a variable or a parameter, opens a dialog box which lets you change the name and the logical direction (input/output).</li> </ul>	All

### 9.1.8 VARIABLES MENU

Command	Description
<i>Insert</i>	Adds a new row to the table in the currently active editor (if PLC editor, to the table of local variables; if parameters editor, to the table of parameters, etc.).
<i>Delete</i>	Deletes the variable in the selected row of the currently active table.
<i>Create multiple</i>	Lets you to create a set of multiple variables.
<i>Group</i>	Opens a dialog box which lets you create and delete groups of variables.

### 9.1.9 WINDOW MENU

Command	Description
<i>Cascade</i>	Displaces all open documents in cascade, so that they completely overlap except for the caption.
<i>Tile</i>	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.
<i>Arrange Icons</i>	Displaces the icons of the minimized documents in the bottom left-hand corner of the PLC editors area.
<i>Close all</i>	Closes all open documents.



## 9.1.10 HELP MENU












Command	Description
<i>Index</i>	Lists all the <i>Help keywords</i> and opens the related topic.
<i>Context</i>	Context-sensitive help. Opens the topic related to the currently active window.
<i>About...</i>	Information on producers and version.

## 9.2 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.

### 9.2.1 MAIN TOOLBAR



Button	Command	Description
	<i>New project</i>	Creates a new project.
	<i>Open project</i>	Opens an existing project.
	<i>Save project</i>	Saves all documents in the currently open windows, including the project file. Note that, since all modifications to a Application project are first applied in memory only, you need to release the <i>Save project</i> command to make them permanent.
	<i>Undo</i>	Cancels last change made in the document.
	<i>Redo</i>	Restores the last change canceled by <i>Undo</i> .
	<i>Cut</i>	Removes the selected items from the active document and stores them in a system buffer.
	<i>Copy</i>	Copies the selected items to a system buffer.
	<i>Paste</i>	Pastes in the active document the contents of the system buffer.
	<i>Find</i>	Asks you to type a string and searches for its first instance within the active document from the current location of the cursor.
	<i>Find next</i>	Iterates the search previously performed by the <i>Find</i> command.
	<i>Find in project</i>	Opens the <i>Find in project</i> dialog box.



Button	Command	Description
	<i>Print</i>	Displays a dialog box, which lets you set printing options and print the document in the currently active window.
	<i>Print preview</i>	Shows a picture on your video, that reproduces faithfully what you get if you print the document in the currently active window.
	<i>Workspace</i>	If pressed, displays the <i>Workspace</i> (also called <i>Project</i> window), otherwise hides it.
	<i>Output</i>	If pressed, displays the <i>Output</i> window, otherwise hides it.
	<i>Library</i>	If pressed, displays the <i>Libraries</i> window, otherwise hides it.
	<i>Watch</i>	If checked, displays the <i>Watch</i> window, otherwise hides it.
	<i>Async</i>	If checked, displays the <i>Oscilloscope</i> window, otherwise hides it.
	<i>Force I/O</i>	If pressed, displays the <i>Force I/O</i> window, otherwise hides it.
	<i>PLC run-time monitor</i>	If checked, displays the PLC run-time window, otherwise hides it.
	<i>Full screen</i>	Expands the currently active document window to full screen. Press <i>Esc</i> or release the <i>Full screen</i> button to restore the normal appearance of the Application interface.

## 9.2.2 FBD TOOLBAR



Button	Command	Description
	<i>Move/Insert</i>	Editing mode which allows you to insert and move blocks.
	<i>Connection</i>	Editing mode which allows you to draw logical wires to connect pins.
	<i>Watch</i>	Editing mode which allows you to add variables to any debugging tool.
	<i>New block</i>	Lets you insert a new block into the selected network.
	<i>Constant</i>	Adds a constant to the selected network.
	<i>Return</i>	Adds a conditional return block to the selected network.
	<i>Jump</i>	Adds a conditional jump block to the selected network.
	<i>Comment</i>	Adds a comment to the selected network.





Button	Command	Description
	<i>Inc pins</i>	By default some operators like <i>ADD</i> , <i>MUL</i> , etc. have two input pins, however you may occasionally need to perform such operations on more than two operands. This command allows you to add as many input pins as to reach the required number of operands.
	<i>Dec pins</i>	Undoes the <i>Inc pins</i> command.
	<i>EN/ENO</i>	Adds the <i>enable in/enable out</i> pins to the selected block. The code implementing the selected block will be executed only when the <i>enable in</i> signal is true. The <i>enable out</i> signal simply repeats the value of <i>enable in</i> , allowing you either to enable or to disable a cascade of blocks.
	<i>FBD properties</i>	Shows some properties of the selected block: <ul style="list-style-type: none"> <li>- if the object is a function or a function block, displays a table with the input and output variables;</li> <li>- if the object is a variable or a parameter, opens a dialog box which lets you change the name and the logical direction (input/output).</li> </ul>
	<i>View source</i>	Opens the editor by which the selected object was created, and displays the relevant source code: <ul style="list-style-type: none"> <li>- if the object is a program, or a function, or a function block, this command opens the relevant source code editor;</li> <li>- if the object is a variable or a parameter, then this command opens the corresponding variable editor;</li> <li>- if the object is a standard function or an operator, this command opens nothing.</li> </ul>

## 9.2.3 LD TOOLBAR

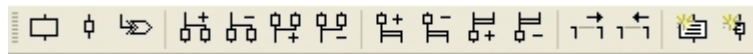


Button	Command	Description
	<i>Insert parallel</i>	Activates the parallel insertion mode. All contacts inserted in this mode will be inserted in parallel with the actually selected contacts.
	<i>Insert series</i>	Activates the series insertion mode. All contacts inserted in this mode will be inserted on the right of the currently selected contact/block. If a connection is selected, the new contact will be placed in the middle of the connection segment.
	<i>Insert contact</i>	Insertion of a new contact according to the selected mode (series or parallel).



Button	Command	Description
	<i>Insert negated contact</i>	Insertion of a new negative contact according to the selected mode (series or parallel).
	<i>Insert rising edge contact</i>	Insertion of a new rising edge contact according to the selected mode (serial or parallel).
	<i>Insert falling edge contact</i>	Insertion of a new falling edge contact according to the selected mode (serial or parallel).
	<i>Insert coil</i>	Insertion of a new coil attached to the right power rail.
	<i>Insert negated coil</i>	Insertion of a new negative coil attached to the right power rail.
	<i>Insert set contact</i>	Insertion of a new set coil attached to the right power rail.
	<i>Insert reset coil</i>	Insertion of a new reset coil attached to the right power rail.
	<i>Insert rising edge contact</i>	Insert positive transition-sensing coil to the right power rail.
	<i>Insert falling edge contact</i>	Insert negative transition-sensing coil to the right power rail.







9.2.4 SFC TOOLBAR



Button	Command	Description
	<i>New step</i>	Inserts a new step into the currently open SFC document.
	<i>Add transition</i>	Adds a new transition to the currently open SFC document.
	<i>Add jump</i>	Adds a new jump block to the currently open SFC document.
	<i>Add divergent pin</i>	Adds a new pin to the selected divergent transition.
	<i>Remove divergent pin</i>	Removes the rightmost pin from the selected divergent transition.
	<i>Add convergent pin</i>	Adds a new pin to the selected convergent transition.
	<i>Remove convergent pin</i>	Removes the rightmost pin from the selected convergent transition.
	<i>Add simultaneous divergent pin</i>	Adds a new pin to the selected simultaneous divergent transition.
	<i>Remove simultaneous divergent pin</i>	Removes the rightmost pin from the selected simultaneous divergent transition.











## SoMachine HVAC - Application




Button	Command	Description
	<i>Add simultaneous convergent pin</i>	Adds a new pin to the selected simultaneous convergent transition.
	<i>Remove simultaneous convergent pin</i>	Removes the rightmost pin from the selected simultaneous divergent transition.
	<i>Shift pin right</i>	Increases the distance between the two rightmost pins of the currently selected transition, in order to let the SFC subnet linked to the pin on the left contain divergent branches.
	<i>Shift pin left</i>	Decreases the distance between the two rightmost pins of the currently selected transition.
	<i>New action code</i>	Allows the user to create a new action to be associated with one of the steps. When you press this button, Application asks you which language you want to use to implement the new action, then opens the corresponding editor.
	<i>New transition code</i>	Allows the user to write the code to be associated with one of the transitions. When you press this button, Application asks you which language you want to use to implement the new transition, then opens the corresponding editor.

### 9.2.5 PROJECT TOOLBAR

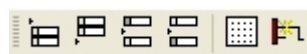








Button	Command	Description
	<i>Library manager</i>	Opens the library manager.
	<i>Compile</i>	Asks you whether to save unsaved changes, then launches the Application compiler.
	<i>Recompile all</i>	Asks you whether to save unsaved changes, then launches the Application compiler to recompile the whole project.
	<i>Connect to the target</i>	Application tries to establish a connection to the target.
	<i>Code download</i>	Application checks if any changes have been applied since last compilation, and compiles the project if this is the case. Then, it sends the target the compiled code.
	<i>New macro</i>	Defines a new macro.
	<i>Object browser</i>	Opens the object browser, which lets you navigate between objects.
	<i>PLC Obj properties</i>	Shows properties and description of the object currently selected in the Workspace.



Button	Command	Description
	<i>Insert record</i>	Adds a new row to the table in the currently active editor (if PLC editor, to the table of local variables; if parameters editor, to the table of parameters, etc.).
	<i>Delete record</i>	Deletes the variable in the selected row of the currently active table.
	<i>Generate redistributable source module</i>	Creates an RSM file of the project.






## 9.2.6 NETWORK TOOLBAR



Button	Command	Description
	<i>Insert Top</i>	Adds a blank network at the top of the active LD/FBD document.
	<i>Insert Bottom</i>	Adds a blank network at the bottom of the active LD/FBD document.
	<i>Insert After</i>	Adds a blank network after the selected network in the active LD/FBD document.
	<i>Insert Before</i>	Adds a blank network before the selected network in the active LD/FBD document.
	<i>View grid</i>	If checked, displays a dotted grid in the LD/FBD editor background.
	<i>Auto connect</i>	If checked, enables auto connection, that is automatic creation of a logical wire linking the pins of two blocks, when they are brought close.







## 9.2.7 DEBUG TOOLBAR



Button	Command	Description
	<i>Debug mode</i>	Switch on/off the <i>Debug</i> mode.
	<i>Live debug mode</i>	Switch on/off the <i>Live debug</i> mode.
	<i>Set/Remove trigger</i>	Sets/removes a trigger at the current source code line.
	<i>Graphic trigger</i>	Sets/removes a graphic trigger at the current source code line.
	<i>Remove all triggers</i>	Removes all triggers.



## SoMachine HVAC - Application

Button	Command	Description
	<i>Trigger list</i>	Lists all triggers.
	<i>Set breakpoints</i>	Sets a breakpoint at the current source code line.
	<i>Remove all breakpoints</i>	Removes all breakpoints.
	<i>Run</i>	Restarts program execution after a breakpoint is hit.
	<i>Breakpoint list</i>	Lists all breakpoints.
	<i>Change current instance</i>	Changes the current function block instance (live debug mode).





## 10. 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.

### 10.1 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 the most part 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 as a language specification. To correctly manage common elements refer to the Application user guide.

#### 10.1.1 BASIC ELEMENTS

##### 10.1.1.1 CHARACTER SET

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

##### 10.1.1.2 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.

#### 10.1.2 ELEMENTARY DATA TYPES

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

The 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.

### 10.1.3 DERIVED DATA TYPES

Derived data types can be declared using the `TYPE...END_TYPE` construct. These derived data types can then 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 that a variable of its parent type can be used.

#### 10.1.3.1 TYPEDEFS

The purpose of typedefs is to assign alternative names to existing types. No difference between a typedef and its parent type exists, apart from 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
```

#### 10.1.3.2 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.

### 10.1.3.3 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
```

### 10.1.3.4 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 structure 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
    
```

### 10.1.4 LITERALS

#### 10.1.4.1 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 <code>FALSE</code> and <code>TRUE</code>	FALSE TRUE

#### 10.1.4.2 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 <i>space</i> character
'\$''	String of length one containing the <i>single quote</i> character



Example	Explanation
'"'	String of length one containing the <i>double quote</i> character
'\$R\$L'	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

## 10.1.5 VARIABLES

### 10.1.5.1 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.

### 10.1.5.2 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
```



## 10.1.5.3 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 only accessible to a program organization unit via a VAR\_EXTERNAL declaration. The type of a variable declared in a VAR\_EXTERNAL must agree with the type declared in the VAR\_GLOBAL block.

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.

## 10.1.5.4 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.

## 10.1.5.5 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.



## 10.1.5.6 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.

## 10.1.5.7 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
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 7 <sup>th</sup> element of memory datablock 4.
%IX0.4	First bit of the first byte of the 5 <sup>th</sup> 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 datablocks, 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 7 <sup>th</sup> 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 33<sup>rd</sup> element of datablock 4.
- Variable Counters is an array of 10 independent variables of type unsigned integer.

### 10.1.5.8 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.

### 10.1.6 PROGRAM ORGANIZATION UNITS

Program organization units are functions, function blocks, and programs. These 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.



## 10.1.6.1 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.

## 10.1.6.2 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.





## 10.1.6.3 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.

## 10.1.7 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	
<b>Description</b>	Conversion to BOOL (boolean)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>out := TO_BOOL( 0 ); (* out = FALSE *) out := TO_BOOL( 1 ); (* out = TRUE *) out := TO_BOOL( 1000 ); (* out = TRUE *)</pre>



TO_SINT	
<b>Description</b>	Conversion to SINT (8-bit signed integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	SINT
<b>Examples</b>	<pre>out := TO_SINT( -1 ); (* out = -1 *) out := TO_SINT( 16#100 ); (* out = 0 *)</pre>

TO_USINT	
<b>Description</b>	Conversion to USINT (8-bit unsigned integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	USINT
<b>Examples</b>	<pre>out := TO_USINT( -1 ); (* out = 255 *) out := TO_USINT( 16#100 ); (* out = 0 *)</pre>

TO_INT	
<b>Description</b>	Conversion to INT (16-bit signed integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	INT
<b>Examples</b>	<pre>out := TO_INT( -1000.0 ); (* out = -1000 *) out := TO_INT( 16#8000 ); (* out = -32768 *)</pre>

TO_UINT	
<b>Description</b>	Conversion to UINT (16-bit unsigned integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	UINT
<b>Examples</b>	<pre>out := TO_UINT( 1000.0 ); (* out = 1000 *) out := TO_UINT( 16#8000 ); (* out = 32768 *)</pre>

TO_DINT	
<b>Description</b>	Conversion to DINT (32-bit signed integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	DINT
<b>Examples</b>	<pre>out := TO_DINT( 10.0 ); (* out = 10 *) out := TO_DINT( 16#FFFFFFFF ); (* out = -1 *)</pre>



<b>TO_UDINT</b>	
<b>Description</b>	Conversion to UDINT (32-bit unsigned integer)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	UDINT
<b>Examples</b>	<code>out := TO_UDINT( 10.0 ); (* out = 10 *)</code> <code>out := TO_UDINT( 16#FFFFFFFF ); (* out = 4294967295 *)</code>

<b>TO_BYTE</b>	
<b>Description</b>	Conversion to BYTE (8-bit string)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	BYTE
<b>Examples</b>	<code>out := TO_BYTE( -1 ); (* out = 16#FF *)</code> <code>out := TO_BYTE( 16#100 ); (* out = 16#00 *)</code>

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

<b>TO_DWORD</b>	
<b>Description</b>	Conversion to DWORD (32-bit string)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	DWORD
<b>Examples</b>	<code>out := TO_DWORD( 10.0 ); (* out = 16#0000000A *)</code> <code>out := TO_DWORD( -1 ); (* out = 16#FFFFFFFF *)</code>

<b>TO_REAL</b>	
<b>Description</b>	Conversion to REAL (32-bit floating point)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	REAL
<b>Examples</b>	<code>out := TO_REAL( -1000 ); (* out = -1000.0 *)</code> <code>out := TO_REAL( 16#8000 ); (* out = -32768.0 *)</code>



<b>TO_LREAL</b>	
<b>Description</b>	Conversion to LREAL (64-bit floating point)
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	LREAL
<b>Examples</b>	<pre>out := TO_LREAL( -1000 ); (* out = -1000.0 *) out := TO_LREAL( 16#8000 ); (* out = -32768.0 *)</pre>

## Numerical functions

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

<b>ABS</b>	
<b>Description</b>	Absolute value. Computes the absolute value of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	Any numerical type
<b>Output data type</b>	Same as input
<b>Examples</b>	<pre>OUT := ABS( -5 ); (* OUT = 5 *) OUT := ABS( -1.618 ); (* OUT = 1.618 *) OUT := ABS( 3.141592 ); (* OUT = 3.141592 *)</pre>

<b>SQRT</b>	
<b>Description</b>	Square root. Computes the square root of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	<pre>OUT := SQRT( 4.0 ); (* OUT = 2.0 *)</pre>

<b>LN</b>	
<b>Description</b>	Natural logarithm. Computes the logarithm with base e of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	<pre>OUT := LN( 2.718281 ); (* OUT = 1.0 *)</pre>

<b>LOG</b>	
<b>Description</b>	Common logarithm. Computes the logarithm with base 10 of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	<pre>OUT := LOG( 100.0 ); (* OUT = 2.0 *)</pre>



<b>EXP</b>	
<b>Description</b>	Natural exponential. Computes the exponential function of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := EXP( 1.0 ); (* OUT ~ 2.718281 *)

<b>SIN</b>	
<b>Description</b>	Sine. Computes the sine function of input #0 expressed in radians
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := SIN( 0.0 ); (* OUT = 0.0 *) OUT := SIN( 2.5 * 3.141592 ); (* OUT ~ 1.0 *)

<b>COS</b>	
<b>Description</b>	Cosine. Computes the cosine function of input #0 expressed in radians
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := COS( 0.0 ); (* OUT = 1.0 *) OUT := COS( -3.141592 ); (* OUT ~ -1.0 *)

<b>TAN</b>	
<b>Description</b>	Tangent. Computes the tangent function of input #0 expressed in radians
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := TAN( 0.0 ); (* OUT = 0.0 *) OUT := TAN( 3.141592 / 4.0 ); (* OUT ~ 1.0 *)

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



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

<b>ATAN</b>	
<b>Description</b>	Arc tangent. Computes the principal arc tangent of input #0; result is expressed in radians
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := ATAN( 0.0 ); (* OUT = 0.0 *) OUT := ATAN( 1.0 ); (* OUT = PI / 4 *)

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

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

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



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

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

<b>POW</b>	
<b>Description</b>	Exponentiation. Raises Base to the power Expo
<b>Number of operands</b>	2
<b>Input data type</b>	LREAL where available, REAL otherwise; LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := POW( 2.0, 3.0 ); (* OUT = 8.0 *) OUT := POW( -1.0, 5.0 ); (* OUT = -1.0 *)

<b>ATAN2*</b>	
<b>Description</b>	Arc tangent (with 2 parameters). Computes the principal arc tangent of Y/X; result is expressed in radians
<b>Number of operands</b>	2
<b>Input data type</b>	LREAL where available, REAL otherwise; LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	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 *)

<b>SINH*</b>	
<b>Description</b>	Hyperbolic sine. Computes the hyperbolic sine function of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := SINH( 0.0 ); (* OUT = 0.0 *)



<b>COSH*</b>	
<b>Description</b>	Hyperbolic cosine. Computes the hyperbolic cosine function of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := COSH( 0.0 ); (* OUT = 1.0 *)

<b>TANH*</b>	
<b>Description</b>	Hyperbolic tangent. Computes the hyperbolic tangent function of input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := TANH( 0.0 ); (* OUT = 0.0 *)

<b>CEIL*</b>	
<b>Description</b>	Rounding up to integer. Returns the smallest integer that is greater than or equal to input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	OUT := CEIL( 1.95 ); (* OUT = 2.0 *) OUT := CEIL( -1.27 ); (* OUT = -1.0 *)

<b>FLOOR*</b>	
<b>Description</b>	Rounding down to integer. Returns the largest integer that is less than or equal to input #0
<b>Number of operands</b>	1
<b>Input data type</b>	LREAL where available, REAL otherwise
<b>Output data type</b>	LREAL where available, REAL otherwise
<b>Examples</b>	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

<b>SHL</b>	
<b>Description</b>	Input#0 left-shifted of Input #1 bits, zero filled on the right.
<b>Number of operands</b>	2
<b>Input data type</b>	Any numerical type, Any numerical type
<b>Output data type</b>	Same as Input #0
<b>Examples</b>	OUT := SHL( IN := 16#1000CAFE, 16 ); (* OUT = 16#CAFE0000 *)





<b>SHR</b>	
<b>Description</b>	Input #0 right-shifted of Input #1 bits, zero filled on the left.
<b>Number of operands</b>	2
<b>Input data type</b>	Any numerical type, Any numerical type
<b>Output data type</b>	Same as Input #0
<b>Examples</b>	<pre>OUT := SHR( IN := 16#1000CAFE, 24 ); (* OUT = 16#00000010 *)</pre>

<b>ROL</b>	
<b>Description</b>	Input #0 left-shifted of Input #1 bits, circular.
<b>Number of operands</b>	2
<b>Input data type</b>	Any numerical type, Any numerical type
<b>Output data type</b>	Same as Input #0
<b>Examples</b>	<pre>OUT := ROL( IN := 16#1000CAFE, 4 ); (* OUT = 16#000CAFE1 *)</pre>

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

<b>AND</b>	
<b>Description</b>	Logical AND if both Input #0 and Input #1 are BOOL, otherwise bitwise AND.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but STRING, Any but STRING
<b>Output data type</b>	Same as Inputs
<b>Examples</b>	<pre>OUT := TRUE AND FALSE; (* OUT = FALSE *) OUT := 16#1234 AND 16#5678; (* OUT = 16#1230 *)</pre>

<b>OR</b>	
<b>Description</b>	Logical OR if both Input #0 and Input #1 are BOOL, otherwise bitwise OR.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but STRING, Any but STRING
<b>Output data type</b>	Same as Inputs
<b>Examples</b>	<pre>OUT := TRUE OR FALSE; (* OUT = FALSE *) OUT := 16#1234 OR 16#5678; (* OUT = 16#567C *)</pre>



<b>XOR</b>	
<b>Description</b>	Logical XOR if both Input #0 and Input #1 are BOOL, otherwise bitwise XOR.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but STRING, Any but STRING
<b>Output data type</b>	Same as Inputs
<b>Examples</b>	OUT := TRUE OR FALSE; (* OUT = TRUE *) OUT := 16#1234 OR 16#5678; (* OUT = 16#444C *)

<b>NOT</b>	
<b>Description</b>	Logical NOT if Input is BOOL, otherwise bitwise NOT.
<b>Number of operands</b>	1
<b>Input data type</b>	Any but STRING
<b>Output data type</b>	Same as Inputs
<b>Examples</b>	OUT := NOT FALSE; (* OUT = TRUE *) OUT := NOT 16#1234; (* OUT = 16#EDCB *)

### Selection functions

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

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

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



<b>LIMIT</b>	
<b>Description</b>	Limits Input #0 to be equal or more than Input#1, and equal or less than Input #2.
<b>Number of operands</b>	3
<b>Input data type</b>	Any numerical type, Any numerical type, Any numerical type
<b>Output data type</b>	Same as Inputs
<b>Examples</b>	<pre>OUT := LIMIT( IN := 4, MN := 0, MX := 5 ); (* OUT = 4 *) OUT := LIMIT( IN := 88, MN := 0, MX := 5 ); (* OUT = 5 *) OUT := LIMIT( IN := -1, MN := 0, MX := 5 ); (* OUT = 0 *)</pre>

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

## Comparison functions

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

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

<b>GE</b>	
<b>Description</b>	Greater than or equal to. Returns TRUE if Input #0 >= Input #1, otherwise FALSE.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but BOOL, Any but BOOL
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>OUT := GE( 20, 20 ); (* OUT = TRUE *) OUT := GE( 'pippo', 'pluto' ); (* OUT = FALSE *)</pre>



<b>EQ</b>	
<b>Description</b>	Equal to. Returns TRUE if Input #0 = Input #1, otherwise FALSE.
<b>Number of operands</b>	2
<b>Input data type</b>	Any, Any
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>OUT := EQ( TRUE, FALSE );      (* OUT = FALSE *) OUT := EQ( 'pippo', 'pluto' ); (* OUT = FALSE *)</pre>

<b>LT</b>	
<b>Description</b>	Less than. Returns TRUE if Input #0 < Input #1, otherwise FALSE.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but BOOL, Any but BOOL
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>OUT := LT( 0, 20 ); (* OUT = TRUE *) OUT := LT( 'pipp', 'pluto' ); (* OUT = TRUE *)</pre>

<b>LE</b>	
<b>Description</b>	Less than or equal to. Returns TRUE if Input #0 <= Input #1, otherwise FALSE.
<b>Number of operands</b>	2
<b>Input data type</b>	Any but BOOL, Any but BOOL
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>OUT := LE( 20, 20 ); (* OUT = TRUE *) OUT := LE( 'pipp', 'pluto' ); (* OUT = TRUE *)</pre>

<b>NE</b>	
<b>Description</b>	Not equal to. Returns TRUE if Input #0 != Input #1, otherwise FALSE.
<b>Number of operands</b>	2
<b>Input data type</b>	Any, Any
<b>Output data type</b>	BOOL
<b>Examples</b>	<pre>OUT := NE( TRUE, FALSE ); (* OUT = TRUE *) OUT := NE( 'pipp', 'pluto' ); (* OUT = TRUE *)</pre>



## String functions

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

CONCAT	
<b>Description</b>	Character string concatenation
<b>Number of operands</b>	2
<b>Input data type</b>	STRING, STRING
<b>Output data type</b>	STRING
<b>Examples</b>	<code>OUT := CONCAT( 'AB', 'CD' ); (* OUT = 'ABCD' *)</code>

DELETE	
<b>Description</b>	Delete L characters of IN, beginning at the P-th character position
<b>Number of operands</b>	3
<b>Input data type</b>	STRING, UINT, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	<code>OUT := DELETE( IN := 'ABXYC', L := 2, P := 3 );</code> <code>(* OUT = 'ABC' *)</code>

FIND	
<b>Description</b>	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.
<b>Number of operands</b>	2
<b>Input data type</b>	STRING, STRING
<b>Output data type</b>	UINT
<b>Examples</b>	<code>OUT := FIND( IN1 := 'ABCBC', IN2 := 'BC' ); (* OUT = 2 *)</code>

INSERT	
<b>Description</b>	Insert IN2 into IN1 after the P-th character position
<b>Number of operands</b>	3
<b>Input data type</b>	STRING, STRING, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	<code>OUT := INSERT( IN1 := 'ABC', IN2 := 'XY', P := 2 );</code> <code>(* OUT = 'ABXYC' *)</code>

LEFT	
<b>Description</b>	Leftmost L characters of IN
<b>Number of operands</b>	2
<b>Input data type</b>	STRING, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	<code>OUT := LEFT( IN := 'ASTR', L := 3 ); (* OUT = 'AST' *)</code>



<b>MID</b>	
<b>Description</b>	L characters of IN, beginning at the P-th
<b>Number of operands</b>	3
<b>Input data type</b>	STRING, UINT, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	OUT := MID( IN := 'ASTR', L := 2, P := 2 ); (* OUT = 'ST' *)

<b>REPLACE</b>	
<b>Description</b>	Replace L characters of IN1 by IN2, starting at the P-th character position
<b>Number of operands</b>	4
<b>Input data type</b>	STRING, STRING, UINT, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	OUT := REPLACE( IN1 := 'ABCDE', IN2 := 'X', L := 2, P := 3 ); (* OUT = 'ABXE' *)

<b>RIGHT</b>	
<b>Description</b>	Rightmost L characters of IN
<b>Number of operands</b>	2
<b>Input data type</b>	STRING, UINT
<b>Output data type</b>	STRING
<b>Examples</b>	OUT := RIGHT( IN := 'ASTR', L := 3 ); (* OUT = 'STR' *)

## 10.2 INSTRUCTION LIST (IL)

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

### 10.2.1 SYNTAX AND SEMANTICS

#### 10.2.1.1 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).



## 10.2.2 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.

## 10.2.3 CALLING FUNCTIONS AND FUNCTION BLOCKS

### 10.2.3.1 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 2<sup>nd</sup>), 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
<code>FBInstanceName</code>	Name of the instance to be invoked.
<code>IO_var</code>	Input or output variable to be written / read.

## 10.3 FUNCTION BLOCK DIAGRAM (FBD)




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

### 10.3.1 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	

## 10.3.2 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.

## 10.3.3 EVALUATION OF NETWORKS

### 10.3.3.1 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 said 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 assigned to every network. Within a program organization unit (POU), networks are evaluated according to the sequence of their number: network  $N$  is evaluated before network  $N+1$ , unless otherwise specified by means of the execution control elements.

### 10.3.3.2 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

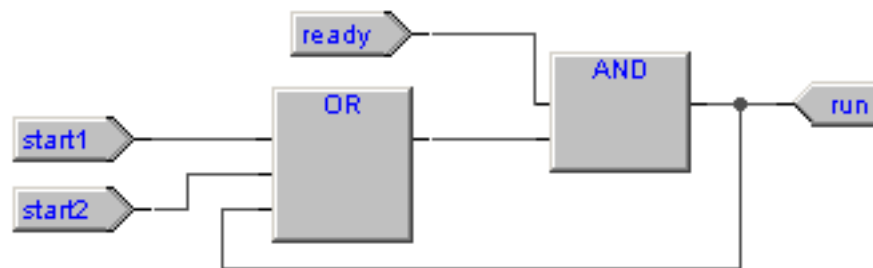
A feedback path is said to exist in a network when the output of a function or function block is used as the input to a function or function block which precedes it in the network; the associated variable is called a feedback variable.

Feedback paths can be utilized subject to the following rules:

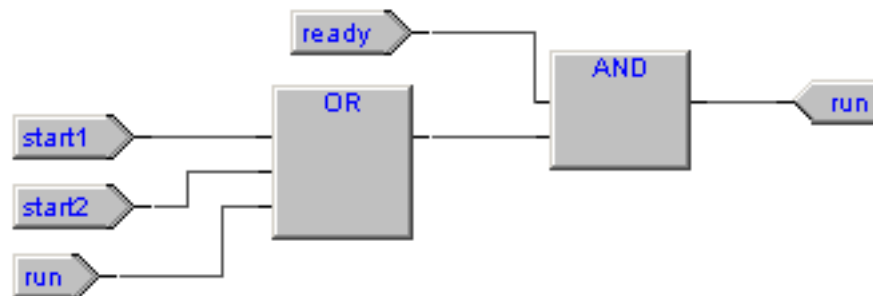
- 1) Feedback variables must be initialized, and the initial value is used during the first evaluation of the network. Look the Global variables editor, the Local variables editor, or the Parameters editor to know how to initialize the respective item.
- 2) Once the element with a feedback variable as 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



## 10.3.4 EXECUTION CONTROL ELEMENTS

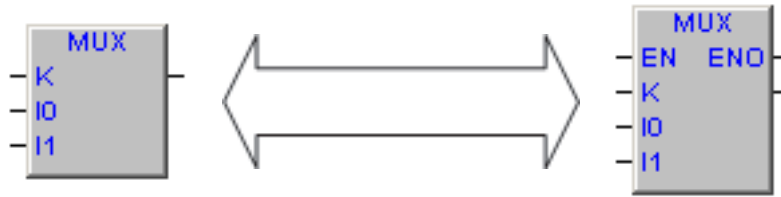
### 10.3.4.1 EN/ENO SIGNALS

Additional Boolean `EN` (Enable) input and `ENO` (Enable Out) characterize Application blocks, according to the declarations

EN	ENO
VAR_INPUT	VAR_OUTPUT
EN: BOOL := 1;	ENO: BOOL;
END_VAR	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.

### 10.3.4.2 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
	Example: Jump Condition Network

### 10.3.4.3 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

## 10.4 LADDER DIAGRAM (LD)

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

### 10.4.1 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)	
Power rails connected by the signal link	

### 10.4.2 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	

### 10.4.3 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.	



Name	Description	Symbol
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.	

## 10.4.4 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.

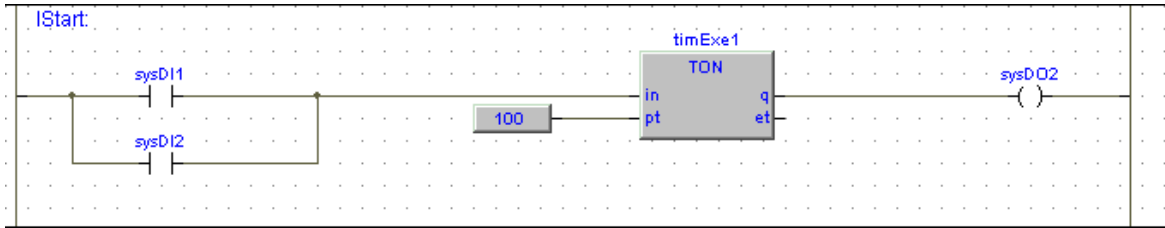
Standard coil symbols are shown in the following table.

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.	
RESET (unlatch) coil	The associated Boolean variable is reset to the OFF state when the left link is in the ON state, and remains reset until set by a SET coil.	
Positive transition-sensing coil	The state of the associated Boolean variable is ON from one evaluation of this element to the next when a transition of the left link from OFF to ON is sensed.	
Negative transition-sensing coil	The state of the associated Boolean variable is ON from one evaluation of this element to the next when a transition of the left link from ON to OFF is sensed.	



## 10.4.5 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.



## 10.5 STRUCTURED TEXT (ST)

This section defines the semantics of the ST (Structured Text) language.

### 10.5.1 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.

#### 10.5.1.1 OPERANDS

An operand can be a literal, a variable, a function invocation, or another expression.

#### 10.5.1.2 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.

#### 10.5.1.3 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.





## 10.5.1.4 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

## 10.5.2 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.

### 10.5.2.1 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

## 10.5.2.2 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.

## 10.5.2.3 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 ;
```

## 10.5.2.4 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 ;
    
```

## 10.6 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 changes necessary to convert the representations from a documentation standard 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

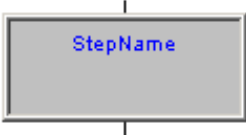
The SFC elements provide a means of partitioning a PLC program organization unit into a set of steps and transitions interconnected by directed links. Associated with each step is a set of actions, and with each transition is associated a transition condition.

### 10.6.1 STEPS

#### 10.6.1.1 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
	<p style="text-align: center;">Step (graphical representation with direct links)</p>



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.

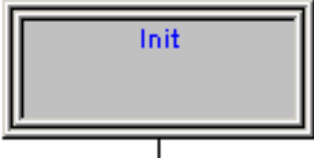
Representation	Description
<code>Step Name_x</code>	Step flag = <code>TRUE</code> when <code>Step Name_x</code> is active = <code>FALSE</code> otherwise

Users cannot assign a value directly to a step state.

### 10.6.1.2 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)

### 10.6.1.3 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
<pre>STEP StepName : (* Step body *) END_STEP</pre>	Step (textual form)
<pre>INITIAL_STEP StepName : (* Step body *) END_STEP</pre>	Initial step (textual form)



Such a structuring element exists in the lsc file for every step having at least one associated action.

### 10.6.1.4 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
<i>N</i>	Non-stored (null qualifier).	The action is executed as long as the step remains active.
<i>P</i>	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.

### 10.6.1.5 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)

## 10.6.2 TRANSITIONS

### 10.6.2.1 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).

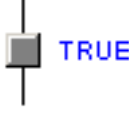


### 10.6.2.2 TRANSITION CONDITION

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) in which the transition is located.

## 10.6.3 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.

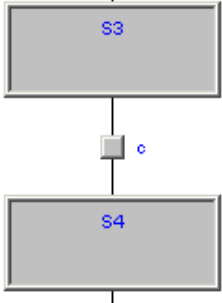
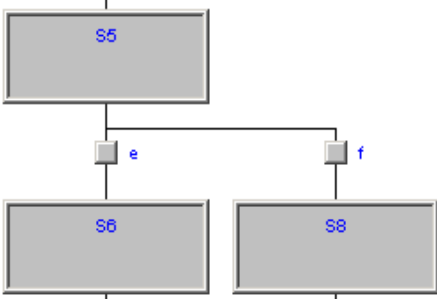
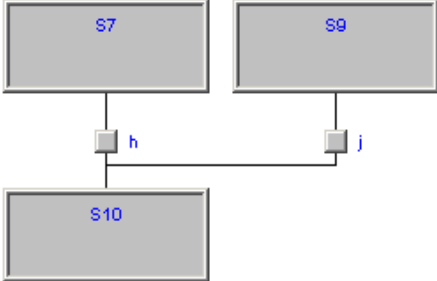
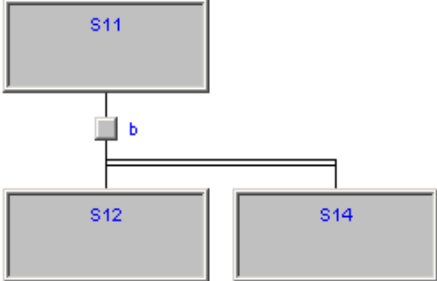
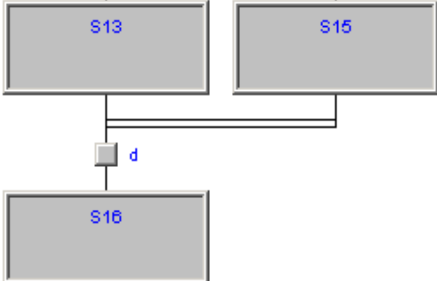
When the clearing of a transition leads to the activation of several steps at the same time, the sequences to which these steps belong are called simultaneous sequences. After their simultaneous activation, the evolution of each of these sequences becomes independent. In order to emphasize the special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



**Sequence evolution table**

This table defines the syntax and semantics of the allowed combinations of steps and transitions.

Example	Rule
	<p>Normal transition</p> <p>An evolution from step <math>S_3</math> to step <math>S_4</math> takes place if and only if step <math>S_3</math> is in the active state and the transition condition <math>c</math> is TRUE.</p>
	<p>Divergent transition</p> <p>An evolution takes place from <math>S_5</math> to <math>S_6</math> if and only if <math>S_5</math> is active and the transition condition <math>e</math> is TRUE, or from <math>S_5</math> to <math>S_8</math> only if <math>S_5</math> is active and <math>f</math> is TRUE and <math>e</math> is FALSE.</p>
	<p>Convergent transition</p> <p>An evolution takes place from <math>S_7</math> to <math>S_{10}</math> only if <math>S_7</math> is active and the transition condition <math>h</math> is TRUE, or from <math>S_9</math> to <math>S_{10}</math> only if <math>S_9</math> is active and <math>j</math> is TRUE.</p>
	<p>Simultaneous divergent transition</p> <p>An evolution takes place from <math>S_{11}</math> to <math>S_{12}</math>, <math>S_{14}</math>,... only if <math>S_{11}</math> is active and the transition condition <math>b</math> associated to the common transition is TRUE. After the simultaneous activation of <math>S_{12}</math>, <math>S_{14}</math>, etc., the evolution of each sequence proceeds independently.</p>
	<p>Simultaneous convergent transition</p> <p>An evolution takes place from <math>S_{13}</math>, <math>S_{15}</math>,... to <math>S_{16}</math> only if all steps above and connected to the double horizontal line are active and the transition condition <math>d</math> associated to the common transition is TRUE.</p>



## Examples

Invalid scheme	Equivalent allowed scheme	Note
		<p>Expected behavior: an evolution takes place from S30 to S33 if a is FALSE and d is TRUE.</p> <p>The scheme in the leftmost column is invalid because conditions d and TRUE are directly linked.</p>
		<p>Expected behavior: an evolution takes place from S32 to S31 if c is FALSE and d is TRUE.</p> <p>The scheme in the leftmost column is invalid because direct links flow only downwards. Upward transitions can be performed via jump blocks.</p>

## 10.7 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.

### 10.7.1 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 );
```

## 10.7.2 POINTERS

Pointers are a special kind of variables which act as a reference to another variable (the 1pointed 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 hazardous: 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.**

## **10.7.3 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 3.6.2 for more details).





## 11. ERRORS REFERENCE

### 11.1 COMPILE TIME ERROR MESSAGES

ERROR CODE	SHORT DESCRIPTION	EXPLANATION
A04097	Object not found	The object indicated (variable or function block) has not been defined in the application.
A04098	Unsupported data type	The size (in bits) requested by the indicated data type isn't supported by the target system.
A04099	Auto vars space exhausted	The total allocation space requested by all local variables exceeds the space available on the target system.
A04100	Retentive vars space exhausted	The total allocation space requested by all local retentive variables exceeds the space available on the target system.
A04101	Bit vars space exhausted	The total allocation space requested by all local bit (boolean) variables exceeds the space available on the target system.
A04102	Invalid ++ in data block	The variable indicated is associated with an index that is not available in the relative data block.
A04103	Data block not found	The variable indicated is associated with a data block that doesn't exist (isn't defined) in the target system.
A04104	Code space exhausted	The total size of code used for POU (programs, functions and function blocks) exceed the space available on the target system.
A04105	Invalid bit offset	The variable indicated is associated with a bit index that is not available in the relative data block.
A04106	Image variable requested	Error code superseded.
A04107	Target function not found	The function indicated isn't available on the target system.
A04108	Base object not found	The indicated instance refers to a function block definition non defined.
A04109	Invalid base object type	The indicated variable is associated with a data type (including function block definition) that isn't defined.
A04110	Invalid data type	The data type used in the variable definition doesn't exist.
A04111	Invalid operand type	The operand type is not allowed for the current operator.
A04112	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.
A04113	Temporary variables allocation error	Internal compiler error.
A04114	Embedded functions do not support arrays as input variables	
A04115	Too many parameters input to embedded function	
A04116	Incremental build failed, perform a full build command	
A04117	Less then 10% of free data	
A04118	Less then 10% of free retain data	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
A04119	Less then 10% of free bit data	
A04120	Variable exceeds data block space	
A04121	Element not found	
A04123	Invalid access to private member	
A04129	Not a structured type	
A04130	Not a function block instance	
A04131	Incompatible external declaration	
A04133	Not a variable	
A04134	Index exceeds array size	
A04135	Invalid index data type	
A04136	Missing index(es)	
A04137	Function block instance required	
A04138	Simple variable required	
A04139	Too many indexes	
A04140	Not a structure instance	
A04141	Not an array	
A04143	Not a pointer	
A04144	Double pointer indirection not allowed	
A04145	To be implemented	
A04146	Bit datatype not allowed	
A04147	Unable to calculate variable offset	
A04148	Complex variables cannot have process image	
A04149	Cannot use directly represented variables with process image in function blocks (not implemented)	
A04150	Function block instance not allowed	
A04151	Structure not allowed	
A04152	16-bit variables must be aligned to a 16-bit boundary	
A04153	32-bit variables must be aligned to a 32-bit boundary	
A04154	Temporary string variable allocation error. Instruction shall be split.	
A04155	Ext/aux auto vars space exhausted	
A04156	Ambiguous enum value, <enum># prefix required	
B00001	Data block not found	The variable indicated is associated with a data block that doesn't exist (isn't defined) in the target system.
B00002	Error on create file	The indicated file can't be created due to a file system error or to a missing source file.
C00001	Parser not inzialized	Internal compiler error.
C00002	Invalid token	Invalid word for the current language syntax





ERROR CODE	SHORT DESCRIPTION	EXPLANATION
C00003	Invalid file specification	Internal compiler error.
C00004	Can't open file	The indicated file can't be opened due to a file system error or to a missing source file.
C00005	Parser tabel error	Internal compiler error.
C00006	Parser non specified	Internal compiler error.
C00007	Unexpected end of file	The indicated file is truncated or the syntax is incomplete.
C00009	Reserved keyword	The indicated word can't be used for declaration purposes because is a keyword of the language.
C00010	Invalid element	The indicated word isn't a valid one for the language syntax.
C00011	Aborted by user	
C00032	Too many parameters in macro call	
C00033	Invalid number of parameters in macro call	
C00034	Too many macro calls nested	
C04097	Invalid variable type	The data type indicated isn't allowed.
C04098	Invalid location prefix	The address string of the indicated variable isn't correct, '%' missing.
C04099	Invalid location specification	The address string of the indicated variable isn't correct, the data access type indication isn't 'I', 'Q' or 'M'.
C04100	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'.
C04101	Invalid location index specification	The address string of the indicated variable isn't correct, the index isn't correct.
C04102	Duplicate variable name	The name of the indicated variable has already been used for some other project object.
C04103	Only 0 admitted here	The compiler uses only arrays zero-index based
C04104	Invalid array dimension	The dimension of the array isn't indicated in the correct way (e.g.: contains invalid characters, negative numbers etc.).
C04105	Constant not initialized	Every constant need to have an initial value.
C04106	Invalid string size	
C04107	Initialization exceeding string size	
C04108	Invalid repetition in initialization	
C04109	Invalid data type for initialization	
C04353	Duplicate label	The indicated label has already been defined in the current POU (program, function or function block).
C04354	Constant not admitted	The operation indicated doesn't allow to use constants (typically store or assign operations).
C04355	Address of explicit constant not defined	
C04356	Maximum number of subscripts exceeded	
C04358	Invalid array base	
C04359	Invalid operand	
C04609	Invalid binary constant	A constant value with 2# prefix must contain only binary digits (0 or 1).



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
C04610	Invalid octal constant	A constant value with 8# prefix must contain only octal digits (between 0 and 7).
C04611	Invalid hexadecimal constant	A constant value with 16# prefix must contain only hexadecimal digits (between 0 and 9 and between A and F).
C04612	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'.
C04613	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'.
C04614	Invalid constant string	
C04864	Duplicate function name	The indicated function name has already been used for another application object.
C04865	Invalid function type	The data type returned by the indicated function is not correct.
C05120	Duplicate program name	The indicated program name has already been used for another application object.
C05376	Duplicate function block name	The indicated function block name has already been used for another application object.
C05632	Invalid pragma	
C05633	Invalid pragma value	
C05889	Duplicate macro name	
C05890	Duplicate macro parameter name	
C06144	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.
F01025	Invalid network	The indicated FBD or LD network contains a connection error (the errors are normally indicated by red connections).
F01026	Unconnected pin	The indicated block (operator, function, contact or coil) has an unconnected pin.
F01027	Invalid connection (incomplete, more than a source etc.)	Internal compiler error.
F01028	More than one network per block	The network indicated contains more networks of blocks and variables not connected between them.
F01029	Ambiguous network evaluation	The compiler is not able to find an univocal way to establish the order of blocks execution.
F01030	Temporary variables allocation error	Internal compiler error.
F01031	Inconsistent network	The network indicated doesn't have input or output variables.
F01032	Invalid object connected to power rail	
F01033	Invalid use of pin negation (ADR operator does not allow negated input)	
F01034	Invalid use of pin negation (SIZEOF operator does not allow negated input)	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G00001	Invalid operand number	The number of operands is not correct for the operand or the function indicated.
G00002	Variable not defined	The variable has not been defined in the local or global context.
G00003	Label not defined	The label indicated for the JMP operand isn't defined in the current POU (program, function or function block).
G00004	Function block not defined	The indicated instance refers to a function block not defined in the whole project.
G00005	Reference to object not defined	The indicated instance refers to an object not defined in the whole project.
G00006	Constant not admitted	The operation indicated doesn't allow to use constants (typically store or assign operations).
G00007	Code buffer overflow	The total size of code used for POU (programs, functions and function blocks) exceed the space available on the target system.
G00008	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.
G00009	Program not found	The indicated program doesn't exist in the current project.
G00010	Program already assigned to a task	The indicated program has been assigned to more than one task of the target system.
G00011	Can't allocate code buffer	There isn't enough memory on the PC to create the image of the code of the target system.
G00012	Function not defined	The indicated function doesn't exist in the current project.
G00013	Cyclic declaration of function blocks	The indicated function block call itself directly or by means of other functions.
G00014	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.
G00015	Accumulator extension	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.
G00016	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).
G00017	Program is not assigned to a task	The indicated program hasn't been assigned to a task in the target system.
G00018	Task not found in resources	The indicated task isn't defined in the target system.
G00019	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.
G00020	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).
G00021	Invalid processor type	The processor indicated into the target definition file (*.TAR) isn't correct or isn't supported by the compiler.
G00022	Function block with process image variables can't be used in event tasks	



<b>ERROR CODE</b>	<b>SHORT DESCRIPTION</b>	<b>EXPLANATION</b>
G00023	Process image variables can't be used in event tasks	
G00024	Accumulator undefined	
G00025	Invalid index	
G00026	Only constant index allowed	
G00027	Illegal reference to the address of a register	
G00028	Less then 10% of free code	
G00029	Index exceeds array size	
G00030	Access to array as scalar - assuming index 0	
G00031	Number of indexes not matching the var size	
G00032	Multidimensional variables not supported	
G00033	Invalid data type	
G00034	Invalid operand type	
G00035	Assembler error	
G00036	Aborted by user	
G00037	Element not defined	
G00038	Cyclic declaration of structures	
G00039	Cyclic declaration of typedefs	
G00040	Unresolved definition of typedef	
G00041	Exceeding dimensions in typedef	
G00042	Unable to allocate compiler internal data	
G00043	CODE GENERATOR INTERNAL ERROR	
G00044	Real data not supported	
G00045	Long real data not supported	
G00046	Long data not supported	
G00047	Operation not implemented	
G00048	Invalid operator	
G00049	Invalid operator value	
G00050	Unbalanced parentheses	
G00051	Data conversion	
G00052	To be implemented	
G00053	Invalid index data type	
G00054	Negation without condition	
G00055	Operation not allowed on boolean	
G00056	Negation of a non-boolean operand	
G00057	Boolean operand required	
G00058	Floating point parameter not allowed	
G00059	Operand extension	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G00060	Division by zero	
G00061	Illegal comparison	
G00062	Function block must be instanciated	
G00063	String operand not allowed	
G00064	Operation not allowed on pointers	
G00065	Destination may be too small to store current result	
G00066	Cannot use a function block containing external variables with process image in more than one task	
G00067	Cannot load the address of an explicit constant	
G00068	Writing a real value into an integer variable	
G00069	Cannot use complex variables in functions. Not implemented	
G00070	Signed/unsigned mismatch	
G00071	Conversion data types mismatch, possible loss of data	
G00072	Implicit type conversion of boolean to integer	
G00073	Implicit type conversion of boolean to real	
G00074	Implicit type conversion of integer to boolean	
G00075	Implicit type conversion of integer to boolean	
G00076	Implicit type conversion of real to boolean	
G00077	Implicit type conversion of real to integer	
G00078	Arithmetic operations require numerical operands	
G00079	Bitwise logical operations require bitstring/integer operands	
G00080	Comparison operations require elementary (i.e., not user-defined) operands	
G00081	Cannot take the address of a bit variable	
G00082	Writing a signed value into an unsigned variable	
G00083	Writing an unsigned value into a signed variable	
G00084	Implicit conversion from single to double precision	
G00085	Implicit conversion from double to single precision	
G00086	Function parameter extension	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G00087	Casting to the same type has no effects	
G00088	Function parameters wrong number	
G00089	Embedded target function not found	
G00090	Recursive type declaration	
G00091	Wrong initial value. Signed/unsigned mismatch	
G00092	Wrong initial value. Conversion data types mismatch, possible loss of data	
G00093	String will be truncated	
G00094	Init value type mismatch	
G00095	Improper init value	
G00096	Init value object not found	
G00097	Invalid assignment to pointer	
G00513	Invalid operator	The operator indicated is not allowed for the indicated operation.
G00514	Operation not implemented	The operator indicated isn't supported by the current target system.
G00515	Real data not supported	The target system in use doesn't support floating point operations.
G00516	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 lose 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.
G00517	Long data not supported	The target system in use doesn't support long data operations.
G00518	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.).
G00519	Assembler error	Internal compiler error.
G00520	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.
G00521	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.
G00522	Instruction has constant result	The indicated operation has a result that is constant (ex. multiply by 0, AND with FALSE).
G00523	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
G00524	Unbalanced parentheses	The number of opened parentheses doesn't match with the number of the closed parentheses in the indicated code block.
G00525	Operation not allowed on boolean	The indicated operation can't be performed on boolean operands (ex. the arithmetic operations).
G00526	Can't perform modulo with long values	The current target system doesn't allow the modulo operation with long data types.
G00527	Division by 0	The indicated division operation has the constant value 0 as denominator.
G00528	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.
G00529	Initial value not defined	Internal compiler error.
G00530	Invalid initial value	The initial value of the variable isn't indicated correctly.
G00531	Invalid accumulator type	The accumulator has a data type not allowed for the indicated operation (ex. MUX operator with REAL accumulator).
G00532	Code generator internal error	Internal compiler error.
G00533	Invalid operator value	The operator has a value not acceptable for the indicated operation (ex. SHL with constant value bigger than 32).
G00534	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G00535	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G00536	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.
G00537	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.
G00538	Return not allowed from programs	The RET operator isn't allowed in PROGRAM blocks.
G00539	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.
G00540	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.
G00541	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.
G00542	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.
G00543	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
G00544	Long real data not supported	The long real data type LREAL isn't supported by the compiler.
G00769	Invalid operator	The operator indicated is not allowed for the indicated operation.
G00770	Operation not implemented	The operator indicated isn't supported by the current target system.
G00771	Assembler error	Internal compiler error.
G00772	Long real data not supported	The long real data type LREAL isn't supported by the compiler.
G00773	Long data not supported	The long data type LINT isn't supported by the compiler.
G00774	Negation of a non-boolean parameter	The negation modifier 'N' can't be used in operations with data types different than boolean.
G00775	Operation not allowed on boolean	The indicated operation can't be performed on boolean operands (ex. the arithmetic operations).
G00776	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.).
G00777	Accumulator undefined	The operation is performed without a previously loaded value into the accumulator.
G00778	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 lose 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.
G00779	Division by zero	The indicated division operation has the constant value 0 as denominator.
G00780	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.
G00781	Illegal comparison	The indicated comparison operation is executed between non homogeneous data types.
G00782	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.
G00783	Boolean parameter required	The IL operator indicated (typically 'S' or 'R') can't be used when the accumulator has a type other than BOOL.
G00784	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.
G00785	Does not support float accumulator	The accumulator has REAL data type and it's not allowed for the indicated operation (typically MUX operation).
G00786	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
G00787	Comparison of unsigned type and signed type	The compare operation indicated is performed using operators that have signed and unsigned data type. Indeterminate results may be possible.
G00788	Illegal conversion	Internal compiler error.
G00789	Conversion may result in loss or corruption of data	Error code not used.
G00790	Illegal negation of a real parameter	Error code not used.
G00791	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.
G00792	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.
G00793	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. Indeterminate results may be possible.
G00794	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. Indeterminate results may be possible.
G00795	Unbalanced parentheses	The number of opened parentheses doesn't match with the number of the closed parentheses in the indicated code block.
G00796	Error while extending parameters	Internal compiler error.
G00797	Invalid index	The constant index value used in the indicated expression is too big for the array dimension. See the array declaration string.
G00798	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.
G00799	Return not allowed from programs	The RET operator isn't allowed in PROGRAM blocks.
G00800	Boolean accumulator required	The indicated SEL operator requires that the accumulator has the boolean data type.
G00801	Operators have mismatching type	The selection performed by MUX and SEL operators shall be done between elements that have homogeneous data types.
G00802	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.
G01537	Using a boolean index to access an element of array	
G01538	Does not support boolean accumulator	
G01539	Does not support float accumulator	
G01540	Error while extending operand(s)	
G01541	Writing a signed value into an unsigned variable	
G01542	Writing an unsigned value into a signed variable	
G01543	Writing a real value into an integer variable	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
G01544	Writing an integer value into a real variable	
G01545	Converting a string into a number	
G01546	Converting a number into a string	
G01547	FPU stack full	
G01548	FPU stack empty	
G01549	FPU stack size error	
G01550	Illegal access to variable through function	
G01551	Illegal reference to address of variable accessible through function	
G01552	Invalid access through function	
G01553	Two variables with the same handle	
G01554	Invalid index for variable accessible through function	
G01555	Invalid instruction with non-empty FPU stack	
G01556	Function result of type string requires store to variable	
G08193	Type definition of unknown data type	
G08194	Type definition has exceeded array dimensions	
G08195	Cyclic definition of data type	
G08196	Double pointers are not supported	
G08197	No enumerative elements	
G08199	Invalid or undefined initialization constant	
G10241	Too many initializer for variable	
G10242	Too less initializer for variable	
G10243	Constant without init values	
P02048	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.
P02049	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.
P02050	Can't create parameters file	The parameters file (with PAR extension) can't be written because of disk write protection or insufficient disk space.
P02051	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
P02052	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.
P02053	Save project error	The new project can't be saved due to disk write protection, non existing destination directory or file system lock.
P02054	Generic file error	A non specific error occurred during file operations.
P02055	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.
P02056	Can't save file	The indicated file can't be saved because of disk write protection or destination file existing and protected.
P02057	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.
P02058	Can't open library file	The indicated library file doesn't exists or can't be opened due to file system locking.
P02059	Listing file not created	
P02060	Cannot create PLC application binary file	
P02061	Can't open template project	
P02062	Support for processor isn't available	
P02063	Less than 10% of free code	
P02064	Less than 10% of free data	
P02065	Less than 10% of free retain data	
P02066	Less than 10% of free bit data	
P02067	Task not found in resources	
P02068	No task defined for the application	
P02069	Project is in the old PPJ format. It will be saved in the actual PPJX format	
P02070	Can't open auxiliary source file	
P02071	Can't read file	
P02072	Application name is longer than 10 characters: only the first 10 characters will be downloaded into the target	
P02073	Downloadable source code file is not password-protected	
P02074	Downloadable PLC application binary file not created	
P02075	Less than 10% of free ext/aux data	
P02076	Project private copy of this library was missing and has been replaced with a new copy of the library (from the original path)	



<b>ERROR CODE</b>	<b>SHORT DESCRIPTION</b>	<b>EXPLANATION</b>
P02077	Cannot load library! Project private copy of this library was missing and the original path to the library is invalid: library has been dropped	
P02078	PLC variables export file not created	
P02079	Debug symbols package (for following download to the target device) not created	
P02080	Source code package (for following download to the target device) not created	
P02081	Invalid task definition	
P02083	Invalid or incoherent task period	
P02084	Broken library link	
S01281	Generic ST error	
S01282	Too many expressions nested	
S01283	No iteration to exit from	
S01284	Missing END_IF	
S01285	Invalid ST statement	
S01286	Invalid assignment	
S01287	Missing;	
S01288	Invalid expression	
S01289	Invalid expression or missing DO	
S01290	Missing END_WHILE	
S01291	Missing END_FOR	
S01292	Missing END_REPEAT	
S01293	Invalid expression or missing THEN	
S01294	Invalid expression or missing TO	
S01295	Invalid expression or missing BY	
S01296	Invalid statement or missing UNTIL	
S01297	Invalid assignment, := expected	
S01298	Invalid address expression	
S01299	Invalid size expression	
S01300	Function return value ignored	
S01301	Invalid parameter passing	
S01302	Function parameter not defined	
S01303	Useless expression	
S01304	Unbalanced parentheses	
S01305	Unknown function	
S01306	Invalid function parameter(s) specification	
S01307	Function parameter doesn't exist	



ERROR CODE	SHORT DESCRIPTION	EXPLANATION
S01308	Multiple assignment not allowed (in accordance with IEC 61131-3)	
S01309	ST preprocessor buffer overflow	
S01310	Function block invocation of a non-function block instance	
S01311	Missing END_WAITING	
S01312	Syntax error	
S01537	Generic SFC error	
S01538	Initial step missing	
S01539	Output connection missing	
S01540	The output pin must be connected to a transition	
S01541	Every output pin of a transition must be connected to a step/jump block	
S01542	Transition expected	
S01543	Step or jump expected	
S01544	Could not find the associate program code	
S01545	Could not find the condition code	
S01546	Unknown-type transition	
S01547	Invalid destination	
S01548	Duplicates action. Same SFC action cannot be used in more than one step	
T08193	Communication timeout	The communication with the target system is unsuccessful because there is no answer from the system itself. More common causes of this problem are incorrect cable connection, invalid target address in communication settings, invalid settings of communication parameters (such as baud rate), target system availability.
T08194	Incompatible target version	Error code not used.
T08195	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.
T08196	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.
T08197	Invalid target information address	Internal compiler error.
T08198	Flash erase failure	The target system was not able to complete the flash erasure procedure. Contact the target system vendor for details.
T08199	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
T08200	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.
T08201	Invalid function index	Internal compiler error.
T08202	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.
T08203	Invalid target information	
T08204	Rebuild required	
T08205	Invalid task	
T08206	Application-level communication protocol error: PLC run-time was not able to understand the received command	
T08207	Not implemented	
T08209	No room for source file on the target	
T08210	Error while uploading source code from target device	
T08211	No room for debug symbols on the target	
T08212	Memory read error	
T08213	Memory write error	
T08214	Not enough space available on the target device for the PLC application binary	







## Contents

<b>1.</b>	<b>Overview</b>	309
1.1	Main elements	309
1.2	Run-time functionalities	312
1.3	Communicating with the target	312
<b>2.</b>	<b>Creating a simple UserInterface project</b>	313
2.1	Purpose of this chapter	313
2.2	Creating a new project	313
2.3	Inserting the first page in the project	314
2.3.1	Creating a new page	314
2.3.2	Editing the colors of the page	315
2.4	Inserting a secondary page	316
2.4.1	Creating a secondary page	316
2.4.2	Dimensioning and setting the secondary page	316
2.4.3	Viewing the title bar and the system button	317
2.4.4	Assigning a style to the window	318
2.4.5	Choosing the start window	319
2.5	Inserting static controls	319
2.5.1	Inserting a line	320
2.5.2	Inserting a rectangle in the page	320
2.6	Inserting static images	322
2.6.1	Importing a bitmap in the project	322
2.6.2	Associating an imported bitmap with an image control	324
2.7	Text strings	325
2.7.1	Inserting a text string	325
2.8	Data management in UserInterface	325
2.8.1	Declaring a local variable	326
2.8.2	Declaring a global variable	327
2.8.3	Importing the PLC variables in the UserInterface project	327
2.8.4	Inserting field parameters	328
2.9	Inserting edit box	329
2.9.1	Inserting an edit box in the page	329
2.9.2	Edit box and UserInterface local variable association	332
2.9.3	Edit box and UserInterface global variable association	333
2.9.4	Linking an edit box with a target (or system) variable	334
2.9.5	Linking an edit box with a PLC Application variable	335
2.9.6	Linking an edit box to a parameter	335
2.9.7	Linking an edit box to a variable by dragging and dropping	336



2.10	Inserting buttons	337
2.10.1	Inserting a led-button	337
2.10.2	Inserting a boolean variable command button	339
2.10.3	Inserting a button to open a child page	340
2.10.4	Inserting a button aimed at launching a procedure of the user	342
2.11	Visibility and updating of controls	343
2.11.1	The visibility property	343
2.11.2	The refresh property	345
2.12	Compiling and downloading the project on the target	345
2.12.1	Connecting to the target	346
2.12.2	Compiling pages for the target	346
2.12.3	Downloading and executing the compiled pages on the target	347
2.12.4	Simulation	347
<b>3.</b>	<b>UserInterface layout</b>	349
3.1	Project window	349
3.2	Embedded editors	350
3.3	Properties window	350
3.4	Toolbars	350
3.5	The output window	351
3.6	Target variables and parameters	351
3.7	Table of keys-actions associations	351
<b>4.</b>	<b>HMI project in UserInterface</b>	353
4.1	Project properties	353
4.1.1	General	353
4.1.2	System options	354
4.1.3	Language selection	354
4.1.4	Global periodic procedure	356
4.2	Frame set	356
4.3	Pages	357
4.3.1	Navigating between pages	357
4.3.2	Child Pages	357
4.3.3	Pop-up pages	358
4.3.4	Asynchronous messages	358
4.4	Controls	359
4.4.1	Static	359
4.4.2	Graphic element	359
4.4.3	Edit box	359
4.4.4	Text box	360



4.4.5	Image	360
4.4.6	Animation	360
4.4.7	Button	360
4.4.8	Chart	361
4.4.9	Trend	361
4.4.10	Progress bar	361
4.4.11	Combo box	361
4.4.12	Checkbox	361
4.4.13	Custom control	362
<b>4.5</b>	<b>Variables</b>	<b>362</b>
4.5.1	Local variables	362
4.5.2	Global variables	363
4.5.3	Variables imported from PLC	363
4.5.4	System variables	364
<b>4.6</b>	<b>Multiple pages management</b>	<b>364</b>
4.6.1	Association of elements of a set	364
4.6.2	Navigation of the elements of a set	365
4.6.3	Pages numbering	365
<b>4.7</b>	<b>Advanced operations on pages</b>	<b>366</b>
4.7.1	Export/import of pages to/from files	366
4.7.2	Export/import procedures and variables	367
4.7.3	Copy/paste of pages in the project	367
4.7.4	Rename pages	368
4.7.5	Templates of page management	368
<b>4.8</b>	<b>Events</b>	<b>371</b>
4.8.1	Page or control events	371
4.8.2	Key pressure events	372
4.8.3	Events raised by software	372
4.8.4	Procedures that can be associated to events	373
4.8.5	Actions that can be associated to key pressure	373
<b>4.9</b>	<b>Resources</b>	<b>374</b>
4.9.1	Fonts	374
4.9.2	Bitmaps	374
4.9.3	Strings table	375
4.9.4	Enumeratives	376
4.9.5	Images lists	376
4.9.6	Sets	377
<b>4.10</b>	<b>Automatic documentation</b>	<b>378</b>
<b>4.11</b>	<b>Managing projects</b>	<b>379</b>
4.11.1	Selecting the target device	379



<b>5.</b>	<b>Appendix I: page properties and object properties</b>	381
5.1	Frame set	381
5.1.1	Properties	381
5.2	Child page	382
5.2.1	Properties	382
5.2.2	Events	382
5.3	Pop-up page	383
5.3.1	Properties	383
5.3.2	Events	384
5.4	Static	384
5.4.1	Properties	384
5.4.2	Events	385
5.5	Line	385
5.5.1	Properties	385
5.6	Rectangle	386
5.6.1	Properties	386
5.7	Edit box	386
5.7.1	Properties	386
5.7.2	Format specification - printf	388
5.7.3	Events	389
5.8	Text box	389
5.8.1	Properties	389
5.8.2	Events	391
5.9	Image	391
5.9.1	Properties	391
5.10	Animation	392
5.10.1	Properties	392
5.10.2	Events	392
5.11	Button	393
5.11.1	Properties	393
5.11.2	Events	394
5.12	Progress bar	395
5.12.1	Properties	395
5.12.2	Events	396
5.13	Custom control	396
5.13.1	Properties	396
5.13.2	Events	397
5.14	Chart	397
5.14.1	Properties	397



5.14.2	Events	399
5.15	Trend	399
5.15.1	Properties	399
5.15.2	Events	402
<b>6.</b>	<b>APPENDIX II: FILE FOR TARGET DESCRIPTION</b>	<b>403</b>
6.1	Target properties	403
6.1.1	Description	403
6.2	Object version	404
6.3	System enumeratives	404
6.3.1	Descriptions	404
6.3.2	Example	406
<b>7.</b>	<b>Appendix III: Description of parameter file</b>	<b>409</b>
<b>8.</b>	<b>Appendix IV: elements of HMI runtime</b>	<b>411</b>
8.1	Functions	411
8.1.1	System functions: hardware and operating system	411
8.1.2	Function for managing project resources and common properties	413
8.1.3	Functions for operating with pages	415
8.1.4	Function for objects	417
8.1.5	Drawing functions	419
8.1.6	Functions for text	421
8.1.7	Functions for parameter access	422
8.1.8	Functions for events	424
8.2	Function Blocks	425



## 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](http://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.<sup>(1)</sup>
- 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. OVERVIEW

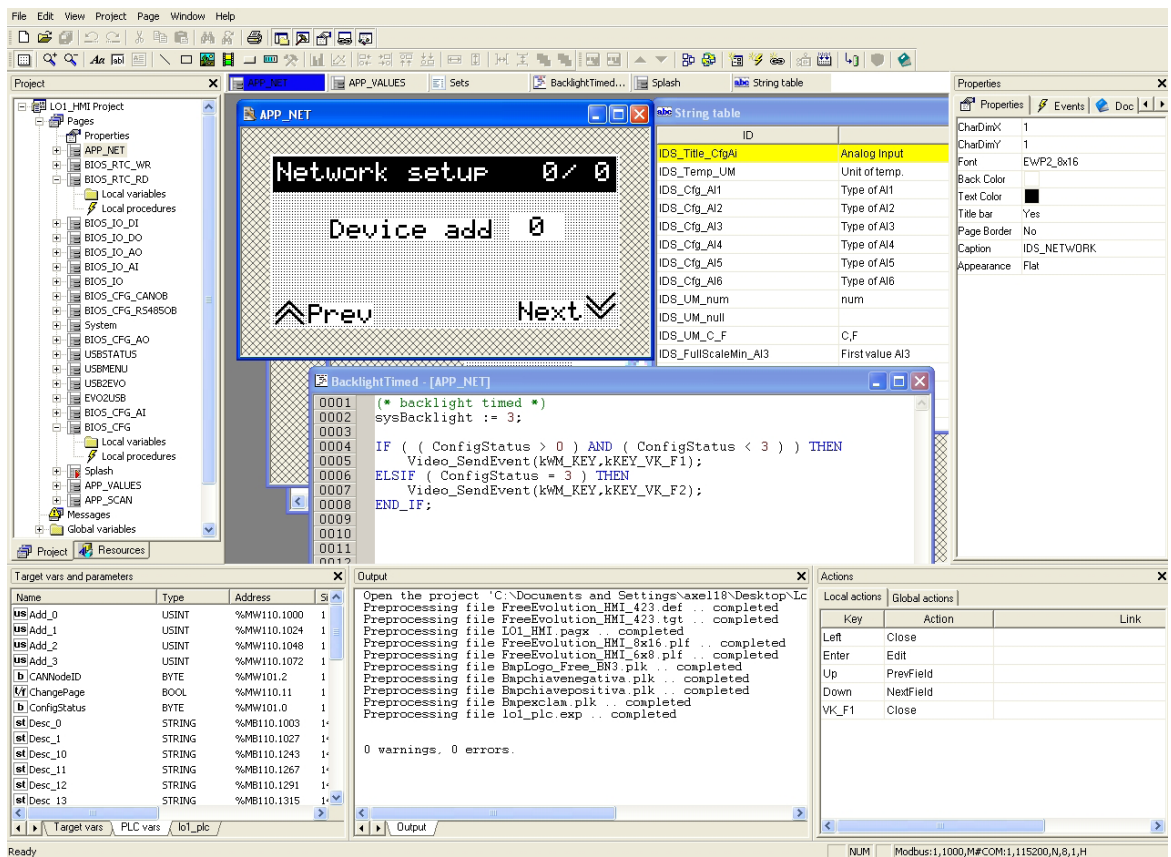
UserInterface is a software application that allows the developer to create user interfaces for embedded systems based on HMI runtime.

UserInterface is an easy to learn and use software, which allows the user to implement graphical interfaces in a visual way. The realized pages are viewed in UserInterface as they will appear on the final target.

Thanks to its multi-pages structure, UserInterface can support HMI (Human Machine Interface) applications with an arbitrary number of pages.

It is equipped with a considerable number of tools to realize even complex applications and it interfaces directly to the PLC IEC1131 Application compiler for managing the variables which are defined in the target PLC application.

The following paragraphs show you the main features of this product.



### 1.1 MAIN ELEMENTS

#### Set of controls



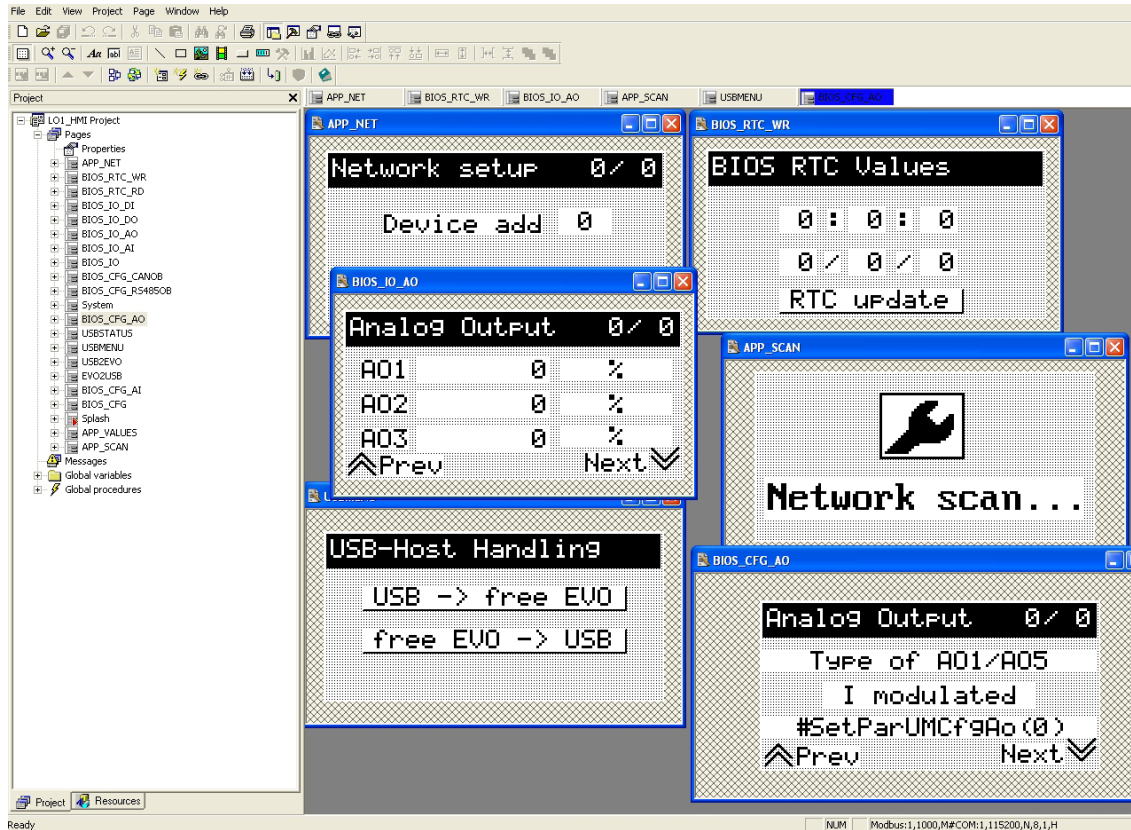
Each page may contain an arbitrary number of defined graphic controls. There are two classes of graphic controls:

- Static controls: drawing tools such as lines, rectangles, and figures.
- Dynamic controls: multilayered objects, which enable data and images display and user interaction (strings, editboxes, textboxes, buttons, progress, charts and trends, custom controls).



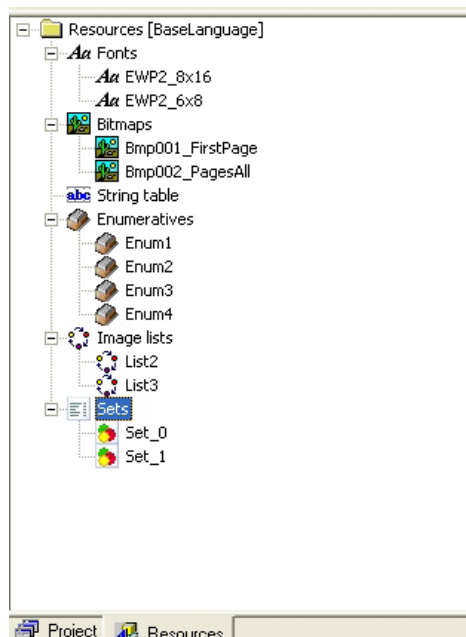
UserInterface is an open system, allowing the implementation of custom controls which may be included in the target system.

## Multi-pages structure



UserInterface supports the definition of an arbitrary number of pages (full-screen or pop-up). Each page may contain links to other pages, so that the whole project takes a tree structure.

## Resources management





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

Resources include fonts for characters display, images, string table, enumerated data types, and elements sets.

Specifically regarding the images, UserInterface allows to import bitmap files directly from the Windows-formatted file (*.bmp*, *.gif*, *.emf*, *.jpg*, *.ico* etc.).

## Languages management

ID	Caption
IDW_DIAGNOSTICA	Diagnostica v1.00
IDS_MEMO	Memoria
IDS_SERIALE	Connessione seriale
IDS_TEMPI	Tempi di esecuzione (ms)
IDS_LINGUA	Lingua
IDS_SERBAUD	Baud rate
IDS_GDBRXPKC	Pacchetti RX
IDS_GDBTXPKC	Pacchetti TX
IDS_GDBRXNAK	Pacchetti RX errati
IDS_GDBTXNAK	Pacchetti TX errati
IDW_DEBUG	Debug di sistema
IDW_TESTVIDEO	Test touch screen
IDW_VIDEO	Video e touch screen
IDS_LCDCONTRASTO	Luminosità LCD
IDS_STATOPLC	Stato PLC
IDS_TEMPIFAST	Ciclo fast (2 ms)
IDS_TEMPIINOUT	Ciclo in/out
IDS_MEMODISK	Spazio su disco (byte)

ID	Caption
IDW_DIAGNOSTICA	Diagnostics
IDS_MEMO	Memory
IDS_SERIALE	Serial connection
IDS_TEMPI	Execution time
IDS_LINGUA	Language
IDS_SERBAUD	Baud rate
IDS_GDBRXPKC	RX packets
IDS_GDBTXPKC	TX packets
IDS_GDBRXNAK	Bad RX packets
IDS_GDBTXNAK	Bad TX packets
IDW_DEBUG	System debug
IDW_TESTVIDEO	Test touch screen
IDW_VIDEO	Video e touch screen
IDS_LCDCONTRASTO	LCD contrast
IDS_STATOPLC	PLC status
IDS_TEMPIFAST	Fast cycle (2ms)
IDS_TEMPIINOUT	In/out cycle
IDS_MEMODISK	Disk space (byte)

Strings and enumerated data types are structured as to ease the multilingual device; moreover UserInterface provides a function to export/import the above mentioned elements to/from a text file, in order to simplify the translation from a language to another.

## Variables and procedures

Name	Type	Array	Init value	Description
xll	UINT	No	0	
xlr	UINT	No	0	
xhl	UINT			
yll	UINT			
ylr	UINT			
yhl	UINT			
showLL	BOOL			
showLR	BOOL			
showHL	BOOL			
go	BOOL			
iniX	UINT			
iniY	UINT			
nkToSet	BOOL			

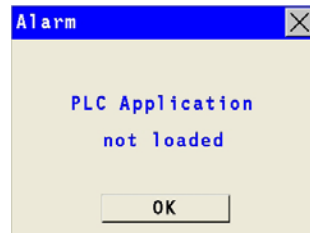
```
0001
0002 (* Non scrive sopra la title bar *)
0003 IF sysTCHPixY > 48 THEN
0004
0005 (* Croce nella posizione del touchscreen *)
0006 sysVoid := Video_Line( sysTCHPixX,
0007                       sysTCHPixY - 1,
0008                       sysTCHPixX,
0009                       sysTCHPixY + 1,
0010                       1,
0011                       0 );
0012
0013 sysVoid := Video_Line( sysTCHPixX - 1,
0014                       sysTCHPixY,
0015                       sysTCHPixX + 1,
0016                       sysTCHPixY,
```

UserInterface enables the implementation of procedures which may be as complex as you want in the ST language. Through these procedures, the user can interact with the UserInterface application, the PLC application or the target system variables to customize the interface's behaviour or the whole CNC.



## 1.2 RUN-TIME FUNCTIONALITIES

### Asynchronous messages management<sup>(1)</sup>

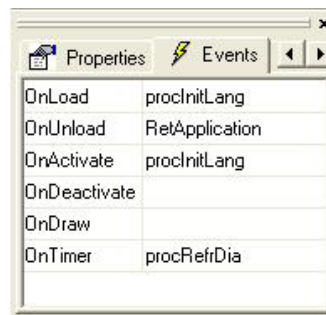


UserInterface supports the issue of asynchronous messages whatever their complexity. You can entirely customize the issue messages management by typing a simple ST procedure.

### Multilingual support

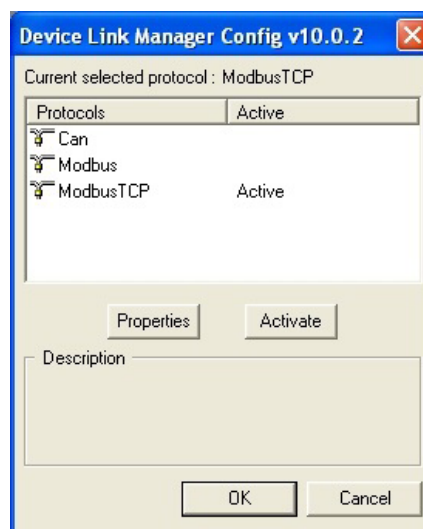
UserInterface allows you to change strings, resources, and enumerations language without recompiling nor reloading the application.

### Events management



UserInterface applications are structured in events; the user may seize the available events and manage them through ST-coded procedures.

## 1.3 COMMUNICATING WITH THE TARGET



You can establish the communication with the target device through the PC communication drivers, thus using one of the available custom protocols (which can be easily implemented thanks to the modular structure of the communication system).



## 2. CREATING A SIMPLE USERINTERFACE PROJECT

### 2.1 PURPOSE OF THIS CHAPTER

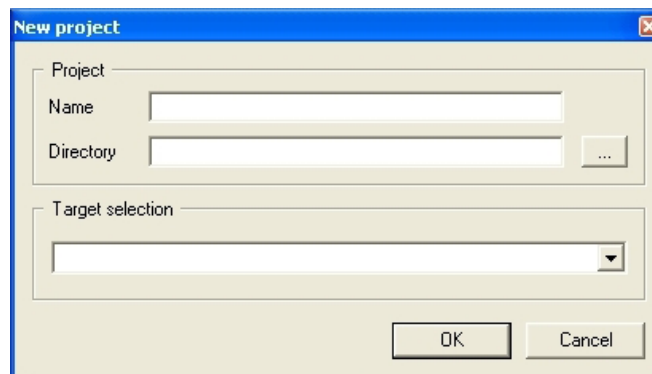
This chapter aims to lead the user to realize a simple HMI project with UserInterface, through a sequence of easy steps.

Here below you can find the list of this chapter's topics.

- Creating a new project: starting at zero the realization of a HMI project.
- Inserting the first page in the project.
- Inserting a secondary page.
- Inserting static controls: how to insert simple objects (lines, rectangles, etc.) in a page.
- Inserting static images: how to insert an image in a page, starting at a *.bmp* file.
- Inserting strings: how to insert a text label.
- Inserting edit boxes: how to access the data of the system and the control PLC, how to declare new variables, how to insert text frames to view/edit these data.
- Inserting buttons: learning to use an essential control for the interaction between the user and the system.
- Compiling and downloading the project.

### 2.2 CREATING A NEW PROJECT

Launch UserInterface, then select the *New Project* command from the *File* menu. The following dialog box appears.



Type the name you want to assign to the project in the *Name* field, and in the *Directory* field specify the directory where you want to create the project folder.

Select the target which will execute the HMI from the *Target selection* menu. The contents of this menu can be customized: if the desired target does not appear in the list, refer to your hardware provider.

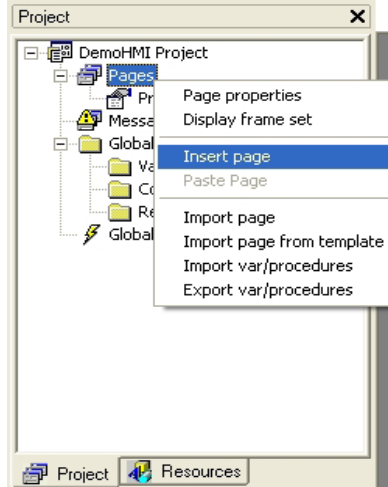
Confirm your choice by pressing *OK*. UserInterface automatically creates the folder *I:\Demo manuale\Demo HMI* as specified in *Directory*.



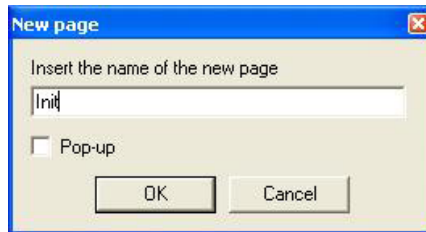
## 2.3 INSERTING THE FIRST PAGE IN THE PROJECT

### 2.3.1 CREATING A NEW PAGE

To insert a new page in the project, right-click on the *Pages* item of the project tree.

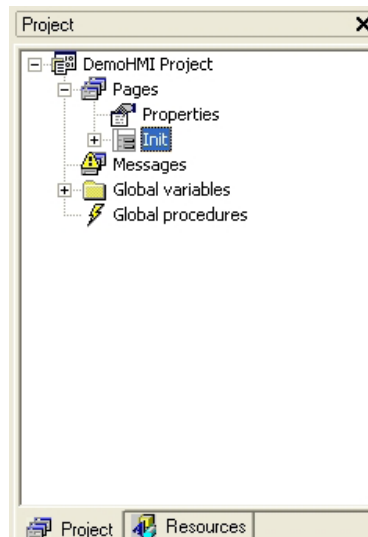


Select the *Insert page* option from the menu which has just shown up. This causes a dialog box to appear where you have to specify the page name and whether the page is a pop-up one or not.



If you do not select the *Pop-up* property when creating it, the page is called *Child Page*. Its main feature is that it fits the whole video area. Consequently the user cannot define position and size of a child page because they are automatically set depending on the video area and on an eventual frame set (see paragraph 4.2).

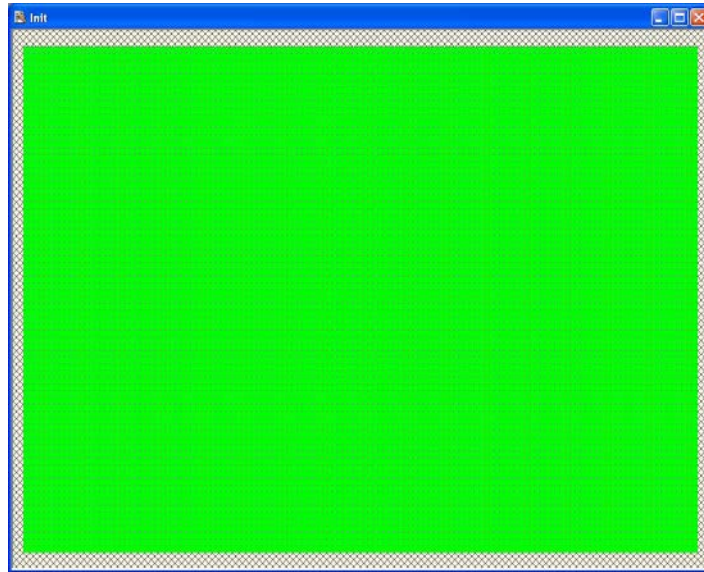
Choose to create a child page and call it *Init*: type the name *Init* in the apposite field and press *OK* to confirm your choice. A new node appears in the pages folder of the project tree.







Double-click on the *Init* item to open the document with this page preview<sup>(1)</sup>, which is blank at the moment.

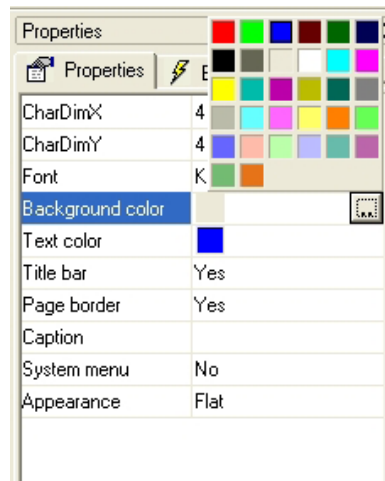


## 2.3.2 EDITING THE COLORS OF THE PAGE

You can edit the background color of the page and the foreground default text color through the page properties: double-click in the *Background Color* field. A little button appears.



Pressing it, the colors palette appears<sup>(1)</sup>. Then you can select the desired color.



Choose grey as background color and black as default text color<sup>(1)</sup>.

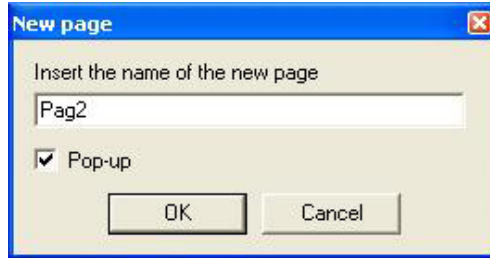




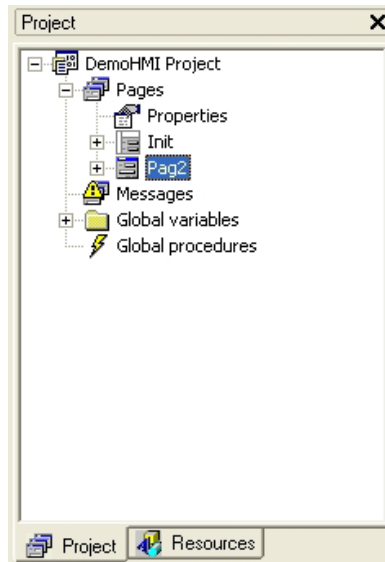
## 2.4 INSERTING A SECONDARY PAGE

### 2.4.1 CREATING A SECONDARY PAGE

Let us assume that you want to create a secondary page: right-click on the *Pages* item of the project tree and choose the *Insert page* option from the contextual menu. Type the name *Pag2* in the dialog box which appears and select the pop-up property.



Consequently a new item appears in the *Pages* folder of the project tree.

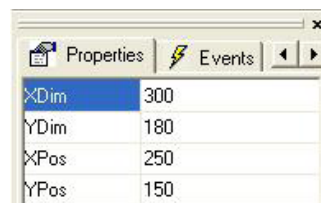


### 2.4.2 DIMENSIONING AND SETTING THE SECONDARY PAGE

Note that the icon of the *Init* page different from the new *Pag2* one. In fact, the last one has been created as pop-up page, whereas the first one has been created as child page.

Pop-up pages are not subjected to any restriction from the frame set (see paragraph 4.2): their dimensions and positions can be chosen by the user.

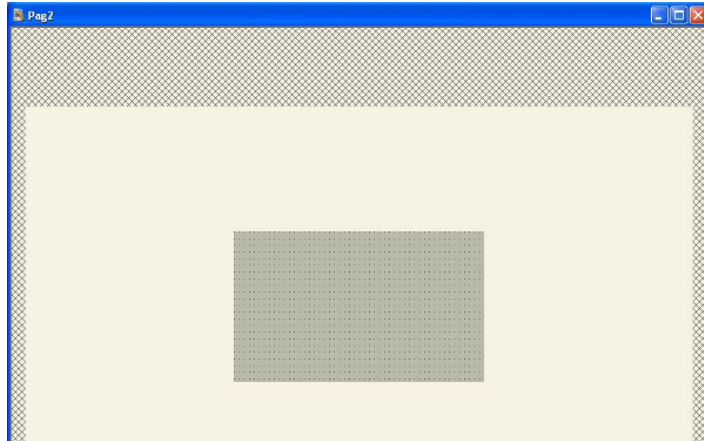
Assign to the secondary window the dimensions 300x180 pixel and set it (x, y) = (250, 150) because these are the top left-hand corner's coordinates of the window. Double-click on the *Pag2* item of the project tree. In this way you open the corresponding document. Assign dimensions and position.







After editing the colors, too, the new window will look like the picture below<sup>(1)</sup>.



The grey area in the centre is the active area of the *Pag2* page, whereas the clearer area which surrounds it represents the video area of the target system. In this way you obtain a clear vision of the new page placement.

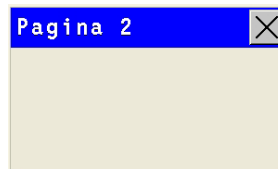
### 2.4.3 VIEWING THE TITLE BAR AND THE SYSTEM BUTTON

UserInterface enables the automatic creation of a title bar (*Title bar* properties = *Yes*) and of a button to close the page (*System menu* properties = *Yes*), besides the print of a text string as title (*Caption* properties).

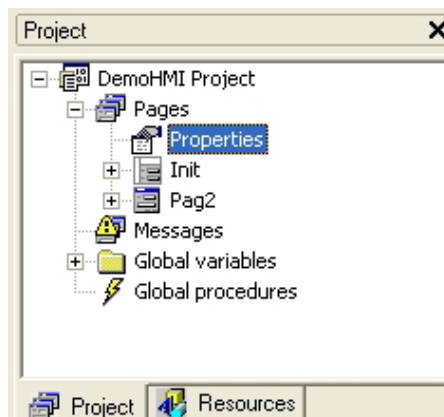
Let us assume that you want to activate the title bar and the close button, and to print the *Pagina 2* string as title.

Title bar	Yes
Page border	Yes
Caption	Pagina 2
System menu	No
Appearance	Flat

Then the secondary page looks like the following picture<sup>(1)</sup>.

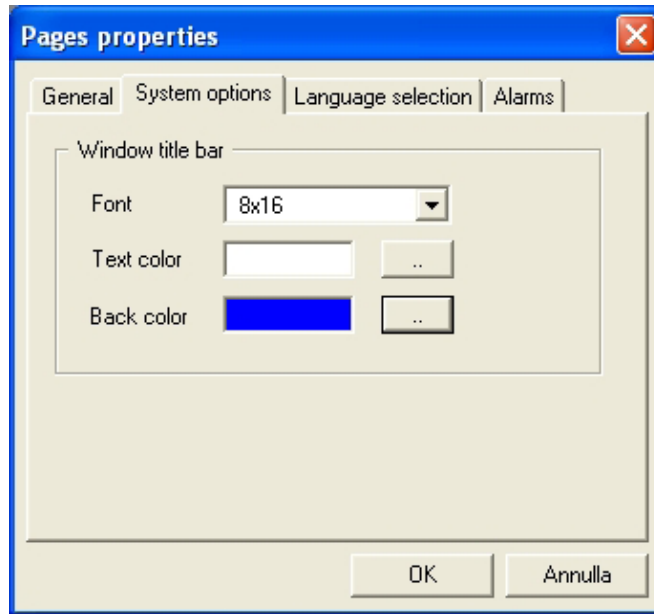


The text and the background color and the used font are the same for all the pages of the project, so you will not find them in this specific page properties. In order to customize these features, double-click on the *Properties* item of the project tree.

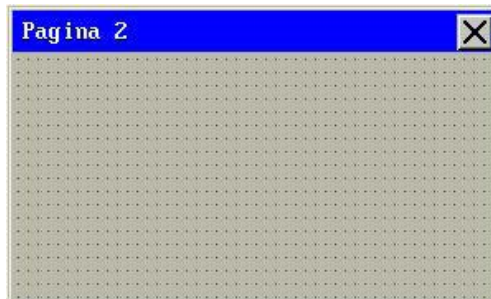




A multi-tabs window opens. In *System options* assign the font (in this case 8x16), the text color and the background color (in this case respectively white and blue).



Then the secondary page looks like the following figure<sup>(1)</sup>.



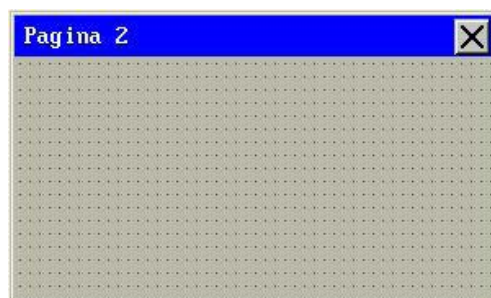
## 2.4.4 ASSIGNING A STYLE TO THE WINDOW

UserInterface supports three styles for the windows, which you can select through the *Appearance* property: *Flat* (the default style when you create a window), *Sunken* and *Raised*.

Choose the last one.



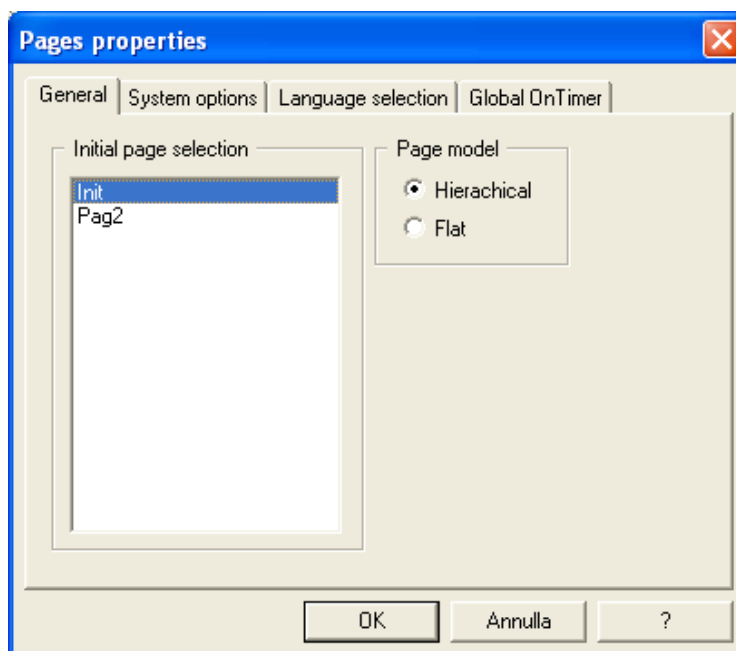
The window looks like the picture below<sup>(1)</sup>.





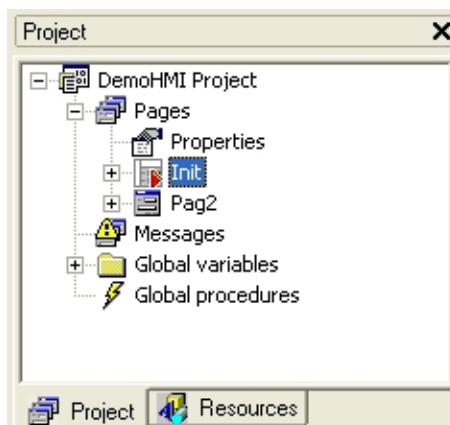
## 2.4.5 CHOOSING THE START WINDOW

The user has to indicate the start window of the whole HMI project. The start window will open at the HMI application start. If the project consists in one single page, the system will take this one as start page. You can indicate the start page in the project properties window, which you can open by double-clicking on the *Properties* item of the project tree. The *General* window is used for this purpose.



In order to indicate the start page, select the desired one from the list. Then confirm your choice by clicking *OK*.

The start page is marked in the project tree by a red triangle.



## 2.5 INSERTING STATIC CONTROLS

The two pages which you have just created are blank yet. Go back to the first page (*Init*) and start inserting some controls.

Static controls are objects which are drawn once, when opening the page, and they do not change until the page is active.



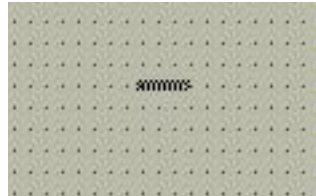
## 2.5.1 INSERTING A LINE

Insert a line by clicking the corresponding button in the *Page toolbar*.

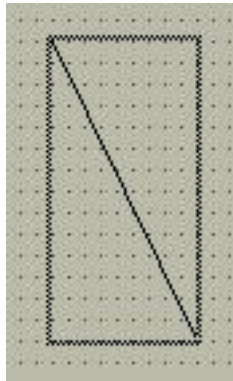


Move the mouse to the active area of the page. A cross + appears. The object will be inserted in the grid near to the mouse cursor.

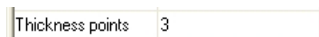
Confirm the insertion point by left-clicking. A new *Line* control appears<sup>(1)</sup>. It has a default size and horizontal alignment.



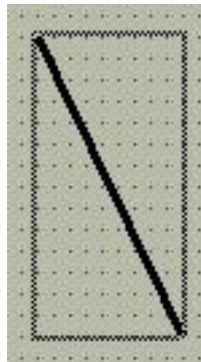
You can resize it by dragging one of the two ends of the line<sup>(1)</sup>.



You can edit the line thickness through the *Thickness points* property of the control. For example, assign a 3 pixel thickness.



In the page preview you can see how the line looks like<sup>(1)</sup>.



## 2.5.2 INSERTING A RECTANGLE IN THE PAGE

Press the corresponding button in the *Page toolbar*.

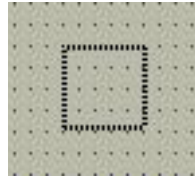




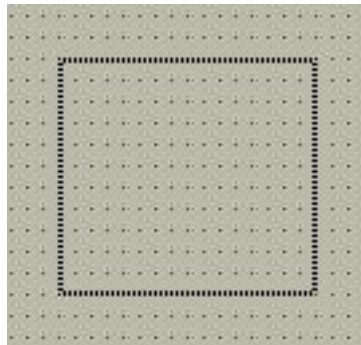
## SoMachine HVAC - UserInterface

Move the mouse to the active area of the page. A cross + appears. The object will be inserted in the grid near to the mouse cursor.

Confirm the insertion point by left-clicking. A new *Rectangle* control appears<sup>(1)</sup>. It has a default size.



You can edit both the dimensions dragging one of the rectangle vertexes, or one dimension at a time dragging one of the rectangle's sides<sup>(1)</sup>.



You can customize the border and the background color and the transparency through the control properties. For example, make the rectangle white and opaque with white border and thickness set to 1.

Border points	1
Border color	<input type="text"/>
Background color	<input type="text"/>
Transparent	TRUE

If the target has new feature of transparency the properties are now like this.

Border points	1
Border color	<input type="text"/>
Background color	<input type="text"/>

In the page preview you can see how the rectangle looks like<sup>(1)</sup>.



Now superimpose another rectangle to the first one. Let us assume that you want the new rectangle to be transparent with black borders, and thickness set to 2.

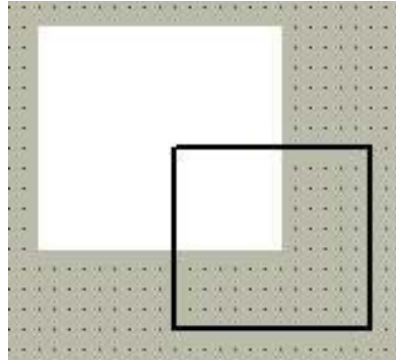
Border points	2
Border color	<input type="text"/>
Background color	<input type="text"/>
Transparent	TRUE



If the target has new feature of transparency the properties are now like this.

Border points	2
Border color	■
Background color	T

In the page preview you will see the following image.



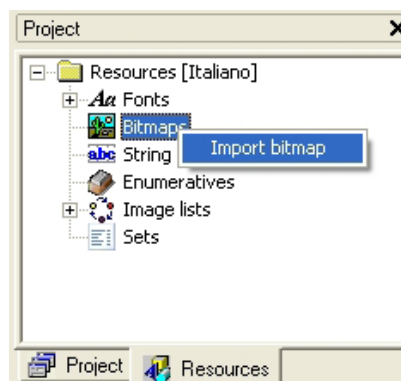
## 2.6 INSERTING STATIC IMAGES

The following paragraph shows you how to insert static images in the page. Static images are different from animations (images which may change dynamically, even though they have fixed position and dimensions) and from floating images (images which move in the page).

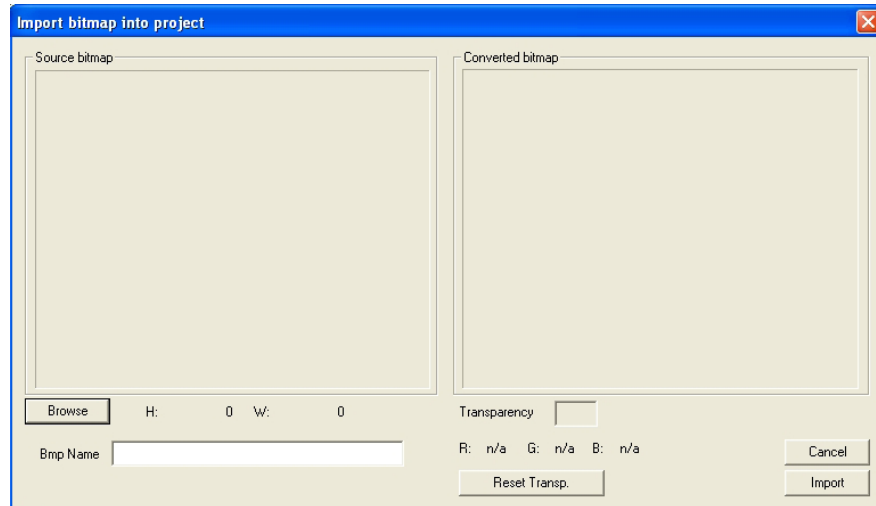
### 2.6.1 IMPORTING A BITMAP IN THE PROJECT

Image that has to be visualized must be available on PC as a basic Windows image file (.bmp, .dib, .emf, .gif, .ico, .jpg, .wmf ...). If this pre-condition holds, you can start the importing procedure.

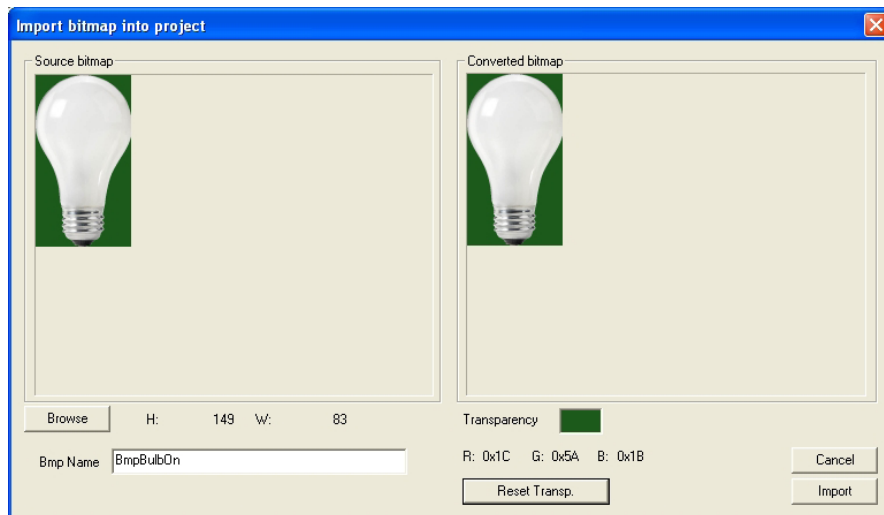
Right-click the *Bitmaps* item in the resources tree and select the *Import bitmap* command in the contextual menu which appears.



A dialog window opens.



Pressing the *Browse* button, you can navigate in the computer resources and select the source file. In this case, the source file is *Bulb0n.jpg*, which represents a lighted bulb<sup>(1)</sup>.



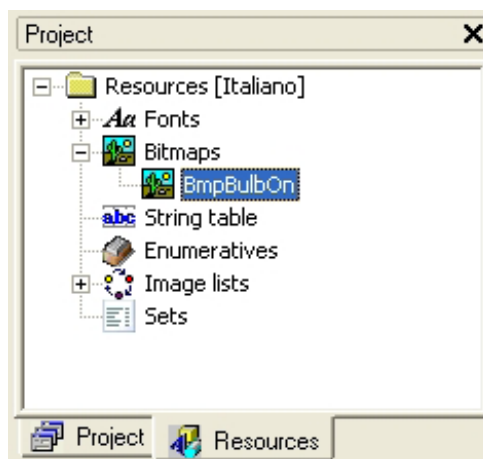
In the *Bitmap Name* field, you can assign the bitmap name which will appear in the resources tree; the default name is the file name without extension and preceded by the *Bmp* prefix.

The *Transparency color* field lets you specify the transparency color, that is a color which will not be really drawn but will let the elements appear through the bitmap background.

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

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

At last you can confirm the operation by clicking the *Import* button. The imported bitmap appears as a new item in the resources tree.



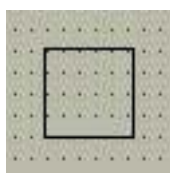
## 2.6.2 ASSOCIATING AN IMPORTED BITMAP WITH AN IMAGE CONTROL

The control which is aimed to display the static images is called *Image*: press the corresponding button in the *Page toolbar*.



Move the mouse to the active area of the page. A cross + appears. The object will be inserted in the grid near to the mouse cursor.

Confirm the insertion point by left-clicking. A new blank frame appears<sup>(1)</sup>.



Trough the *Bitmap* property specify the image which this *Image* control must display. Choose the desired bitmap from the list; in this case, you can see and select the only bitmap which you have imported: *BmpBulbOn*.



The control changes its size to be compatible with the assigned bitmap measures. The image in the page preview looks like the following picture<sup>(1)</sup>.







## 2.7 TEXT STRINGS

Text strings are not part of static controls because they have some properties which let them change in a page through time. Visibility, selection, and refresh may be assigned to variables, which may change their value at any time.

### 2.7.1 INSERTING A TEXT STRING

Click the corresponding button in the *Page toolbar*.



Move the mouse to the active area of the page. A cross + appears. The object will be inserted in the grid near to the mouse cursor.

Confirm the insertion point by left clicking. A new *Static* (that is string) control with the default text *str* appears<sup>(1)</sup>.



You can edit the contents of the string through the *Text* property of the control. For example, *Text string*.



The page preview looks like the image below.



This is the basic use of the string. Alternatively you can assign strings by taking them from the resources (see paragraph 4.9.3).

## 2.8 DATA MANAGEMENT IN USERINTERFACE

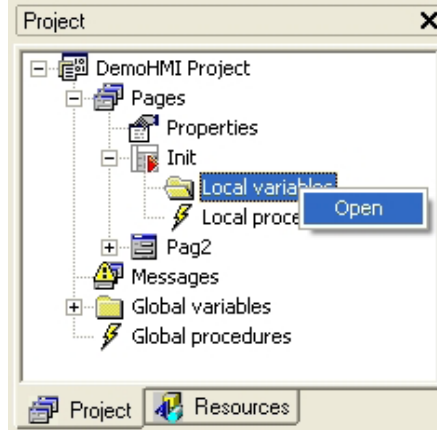
This paragraph shows you the variables management in UserInterface. It is possible to distinguish the data in local variables (visible in the page scope only) and global variables (visible from every page). For some controls it is possible to use parameters and sets.



### 2.8.1 DECLARING A LOCAL VARIABLE

First of all declare a local variable, which you can use just in the specific page where the declaration takes place.

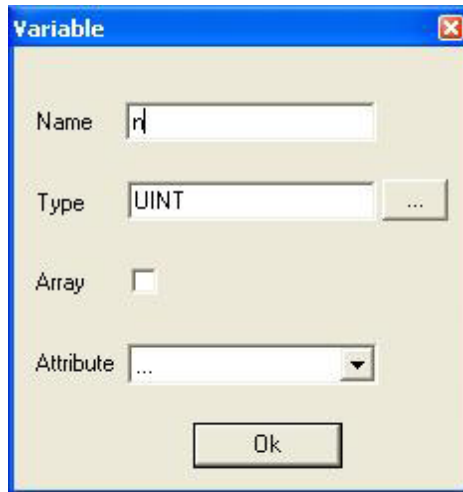
In the pages tree, under the *Init* page item, right-click on the *Local variables* item and select *Open* in the contextual menu which appears.



The local variables editor window opens. It is blank at present. Click the *New record* button in the *Project toolbar*.



A dialog window opens requesting to specify the new variable's basic features. We can declare *n* as a new 16 bit unsigned integer variable.



Confirm the operation by clicking *Ok*. The new corresponding record is added to the variables editor.

	Name	Type	Array	Init value	Description
1	n	UINT	No	0	

You can change this new variable's features editing the fields of the record which you have just created. For example, you may assign an initial value different from null and a comment.

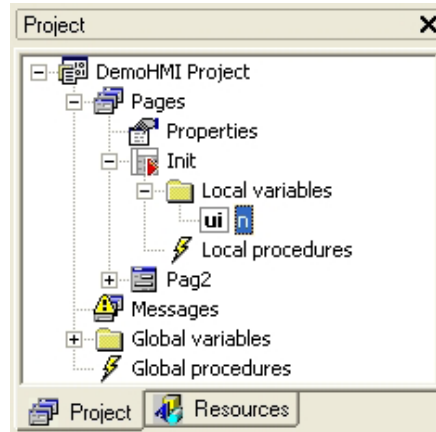
	Name	Type	Array	Init value	Description
1	n	UINT	No	100	Variabile contatore



When you save the project by clicking the apposite button

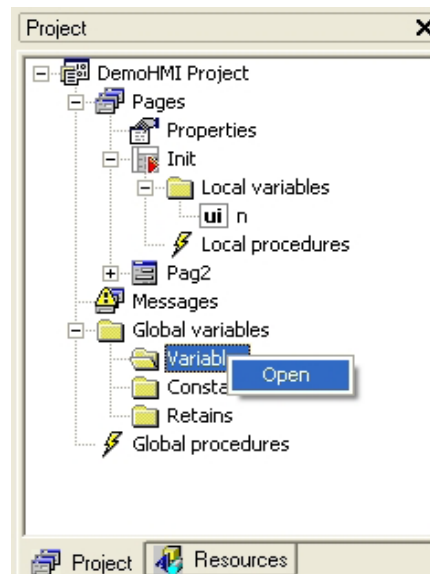


or when you close the variables editor, UserInterface adds a new item in the pages tree. It corresponds to the local variable which you have just declared.



## 2.8.2 DECLARING A GLOBAL VARIABLE

Let us assume that you want to declare a floating point global variable *t*: right-click on the *Variables* item under the *Global variables* node of the resources tree and select the *Open* command in the contextual menu which appears.



Follow the steps as shown in paragraph 2.8.1, until the new global variable appears as a new item in the pages tree.

## 2.8.3 IMPORTING THE PLC VARIABLES IN THE USERINTERFACE PROJECT

Usually an HMI project is not a stand-alone one, but is an interface for a PLC. More precisely, if the PLC project has been carried out with Application, you can easily publish some variables to UserInterface.



A variable of the Application project can be exported to UserInterface if it has been allocated on a datablock (it is not an automatic variable). If this pre-condition holds, when compiling the PLC, the program automatically creates an `.exp` file, which contains a list of the exported variables with their location in the datablocks, which the UserInterface program can work out.

In order to import in UserInterface the variables which have been exported from the PLC Application project, you have to select the *Link PLC variables file...* from the *Project* menu.

A window opens and lets you select the file which contains the exported variables.

If you confirm to include the `.exp` file in the UserInterface project, a new table called *PLC vars* appears in the libraries window. It contains the list of the exported variables.



When you need to update the list of the exported variables, if the `.exp` file has not been moved to another directory, it is not necessary to repeat the above mentioned procedure. It is enough to launch the *Refresh PLC variables* command from the *Project* menu.

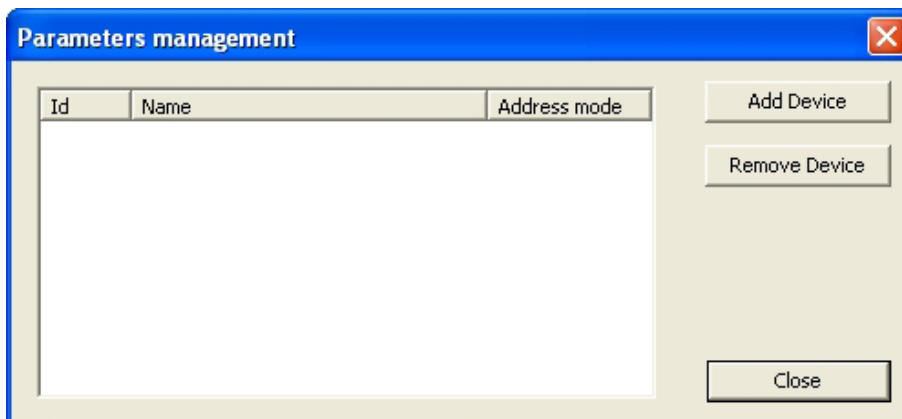
## 2.8.4 INSERTING FIELD PARAMETERS

Target system usually has internal variables and is connected on a fieldbus, so it needs to show some variables of the different devices which are connected on the net.

For this reason, UserInterface lets you link a specific file which contains the variables definition on the bus. Click the apposite button in the toolbar.



The parameters management window appears.

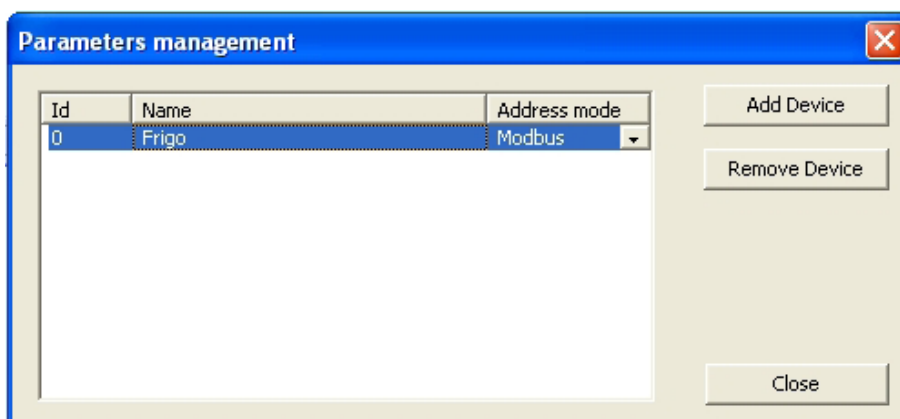


Through the *Add Device* button you can add a new object linked to the target on the fieldbus.

The selection window appears. Then you have to take from your PC a `.parx` file (see



chapter 7). After inserting this file, the parameters management window will look like the image below.



A device called *Frigo* has been inserted. In order to see the relevant parameters, click the *Close* button.

In the *Window target vars and parameters* you will see the device and its parameters.

Name	Type	Address	Min	Max	Um	Description
Par_TAB	UINT	Modbus:15716:0	0	65535	num	Tab (map code)
Par_POLI	UINT	Modbus:15717:0	0	65535	num	Polycarbonate code
Par_PARMOD	BOOL	Modbus:15719:0	0	1	flag	Parameter modified
Gain_Ntc_AI1	UINT	Modbus:15616:0	0	65535	num	NTC calibration gain AI1
Gain_Ntc_AI2	UINT	Modbus:15617:0	0	65535	num	NTC calibration gain AI2
Gain_Ntc_AI3	UINT	Modbus:15618:0	0	65535	num	NTC calibration gain AI3
Gain_PT1000_AI3	UINT	Modbus:15619:0	0	65535	num	PT1000 calibration gain AI3
Gain_5V_AI3	UINT	Modbus:15620:0	0	65535	num	0-5V calibration gain AI3
Gain_10V_AI3	UINT	Modbus:15621:0	0	65535	num	0-10V calibration gain AI3
Gain_mA_AI3	UINT	Modbus:15622:0	0	65535	num	4-20mA calibration gain AI3
Gain_Ntc_AI4	UINT	Modbus:15623:0	0	65535	num	NTC calibration gain AI4
Gain_PT1000_AI4	UINT	Modbus:15624:0	0	65535	num	PT1000 calibration gain AI4
Gain_5V_AI4	UINT	Modbus:15625:0	0	65535	num	0-5V calibration gain AI4
Gain_10V_AI4	UINT	Modbus:15626:0	0	65535	num	0-10V calibration gain AI4
Gain_mA_AI4	UINT	Modbus:15627:0	0	65535	num	4-20mA calibration gain AI4
Gain_Ntc_AI5	UINT	Modbus:15628:0	0	65535	num	NTC calibration gain AI5
Gain_PT1000_AI5	UINT	Modbus:15629:0	0	65535	num	PT1000 calibration gain AI5
Gain_5V_AI5	UINT	Modbus:15630:0	0	65535	num	0-5V calibration gain AI5

When you need to update the list of parameters, if the *.parx* file has not been moved to another directory, it is not necessary to repeat the above mentioned procedure, but it is enough to press the button



## 2.9 INSERTING EDIT BOX

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

### 2.9.1 INSERTING AN EDIT BOX IN THE PAGE

Insert an *Edit box* control in the page by pressing the corresponding button in the *Page toolbar*.

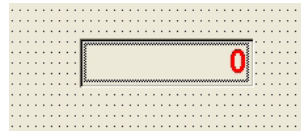


Move the mouse to the active area of the page. A cross + appears. The object will be inserted in the grid near to the mouse cursor.

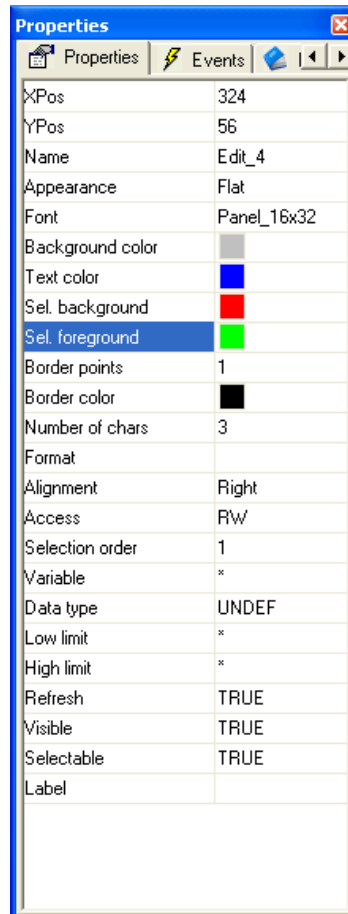
Confirm the insertion point by left-clicking. A new text frame appears<sup>(1)</sup>. It consists by



default in a certain number of characters and its font is specified in the *Font* property of the page.



Edit this control's properties as you can see below<sup>(1)</sup>.



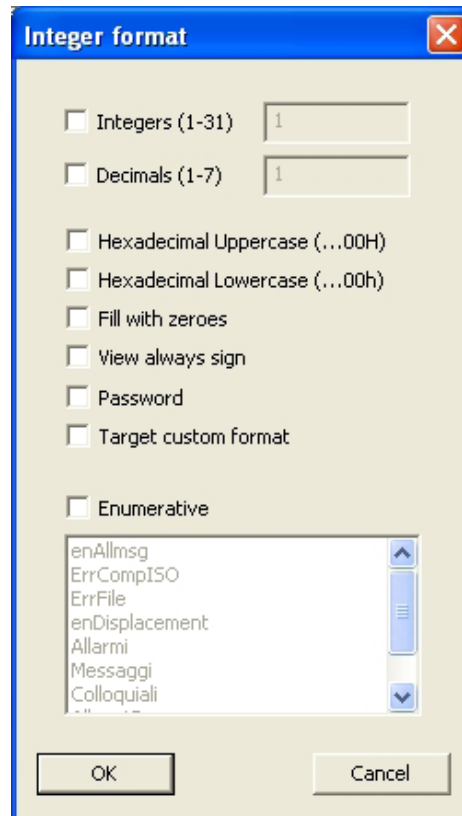
In the following list you can find all the changes which may be carried out:

- *Appearance*: you can make the edit box appearance "sunken" by assigning the *Sunken* property.
- *Font*: you can customize font by choosing, for example, a 16x32 font instead of the default 8x16 font.
- *Select background* and *Select Foreground*: respectively text and background colors when the edit box is selected.
- *Number of Chars*: maximum number of characters which can be displayed.
- *Access*: in order to set the read-only mode, replace *RW* (read-write) with *RO* (read-only).
- *Refresh*: in order to constantly update the contents of the edit box, select the *TRUE* option. Otherwise, the contents are refreshed just when drawing the page for the first time.
- *Label*: if the target has a touchscreen display, shows keyboard and has this feature enabled, it is possible to add this text/'string resource' as header of keyboard.

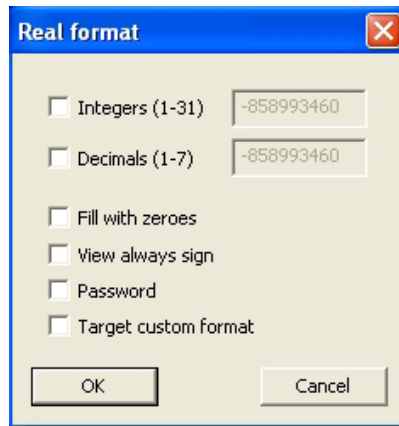


## SoMachine HVAC - UserInterface

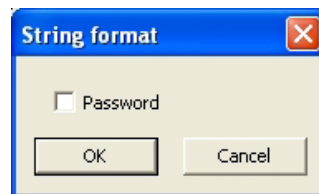
- *Format*: it represents the display format of the associated variable's value. The format value can be inserted only if a variable is just available. It opens a dialog window with these settings according to the type of variable (integer, real, string).



- *Integers*: number of digit before comma
- *Decimals*: number of digit after comma
- *Hexadecimal Uppercase*: the number is shown as 0...0H representation with uppercase H letter
- *Hexadecimal Lowercase*: the number is shown as 0...0h representation with lowercase h letter
- *Fill with zeros*: fill the entire editbox controls with 0 where there are not numbers
- *View always sign*: show the +/- symbol in editbox
- *Password*: show only \* symbols
- *Target custom format*: the target can define custom format to show the data in a particular way. In that case there is a variable on the target with the value of the corresponding user mode.
- *Enumerative*: this representation allows to select a string value corresponding to numeric value defined in *Resources*, under *Enumeratives*.



- *Integers*: number of digit before comma
- *Decimals*: number of digit after comma
- *Fill with zeros*: fill the entire editbox controls with 0 whrere there are not numbers
- *View always sign*: show the +/- symbol in editbox
- *Password*: show only \* symbols
- *Target custom format*: the target can define custom format to show the data in a particular way. In that case there is a variable on the target with the value of the corresponding user mode.



- *Password*: show only \* symbols.

The *Target custom format* is a special feature which enables a particular custom format implemented on the target.

The format is specified according with language *printf* syntax (see paragraph 5.7.2).

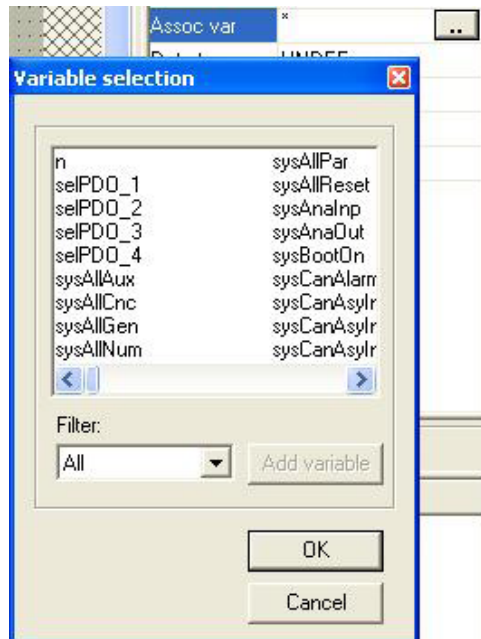
## 2.9.2 EDIT BOX AND USERINTERFACE LOCAL VARIABLE ASSOCIATION

The edit box which you have just inserted lacks an essential element: the associated variable to take the values to display from. Let us assume that you want to link the edit box to a local variable (in order to get information on how to declare a local variable, see paragraph 2.8.1).

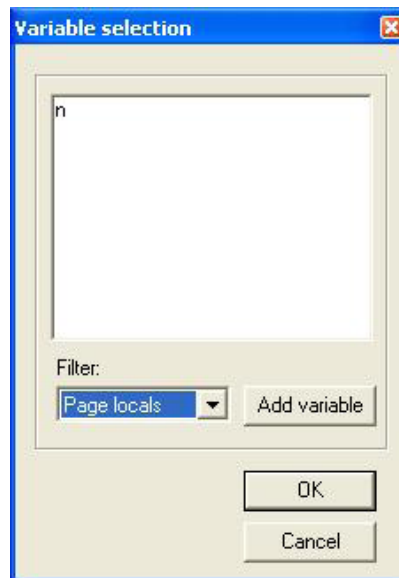
Select the edit box by clicking it once and select the *Variable* property.

You can either type the name of the variable or click on the field and open the dialog window by clicking on the apposite button.

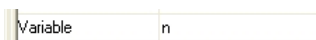




You can restrict the research just to the local variables of the *Init* page (consequently only the *n* variable) by using the *Filter* tool.



Select the local variable. The *Variable* field in the table properties refreshes accordantly.



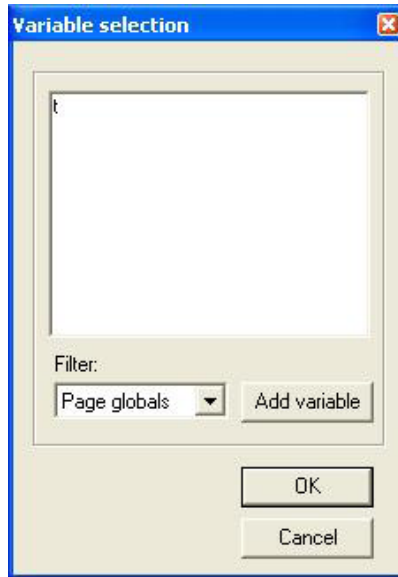
Then the *Edit box* control shows the *n* local variable's value constantly refreshed.

### 2.9.3 EDIT BOX AND USERINTERFACE GLOBAL VARIABLE ASSOCIATION

The principle to associate the *Edit box* control with a global variable is similar to the one to associate the *Edit box* control with a local variable. The difference consists in the variable declaration (in order to get information on how to declare a global variable, see paragraph 2.8.2).



You can associate the Edit box with the global variable through the dialog window which was introduced in the preceding paragraph, but in this case it is necessary to use a different filter in the *Filter* field.

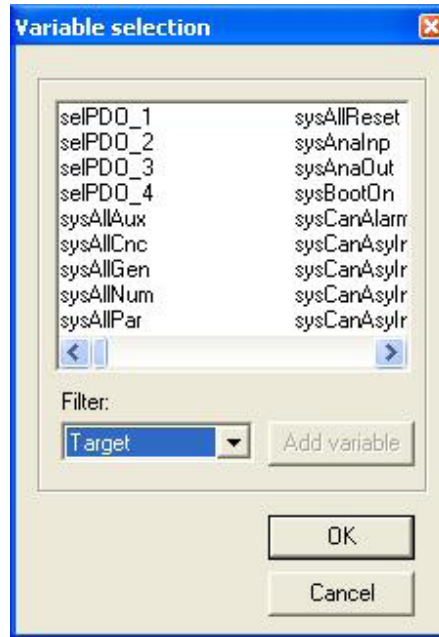


## 2.9.4 LINKING AN EDIT BOX WITH A TARGET (OR SYSTEM) VARIABLE

The target system executing PLC and HMI often publishes some variables which allow the interaction between user interface and system. In UserInterface, such variables are called target variables. You can view them in the *Target vars* table of the *Target Vars and Parameters* window.

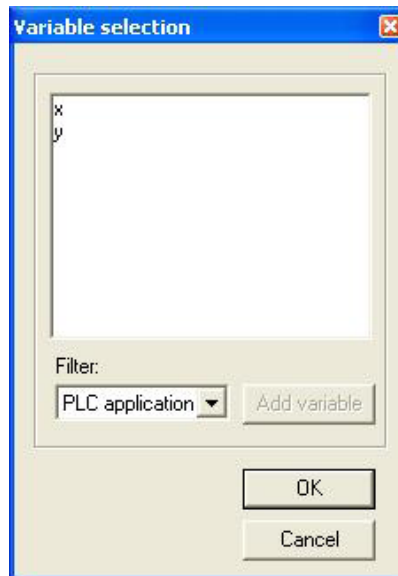


You can associate an Edit box with a target variable through the dialog window which opens from the *Variable* field, but in this case it is necessary to use a different filter in the *Filter* field.



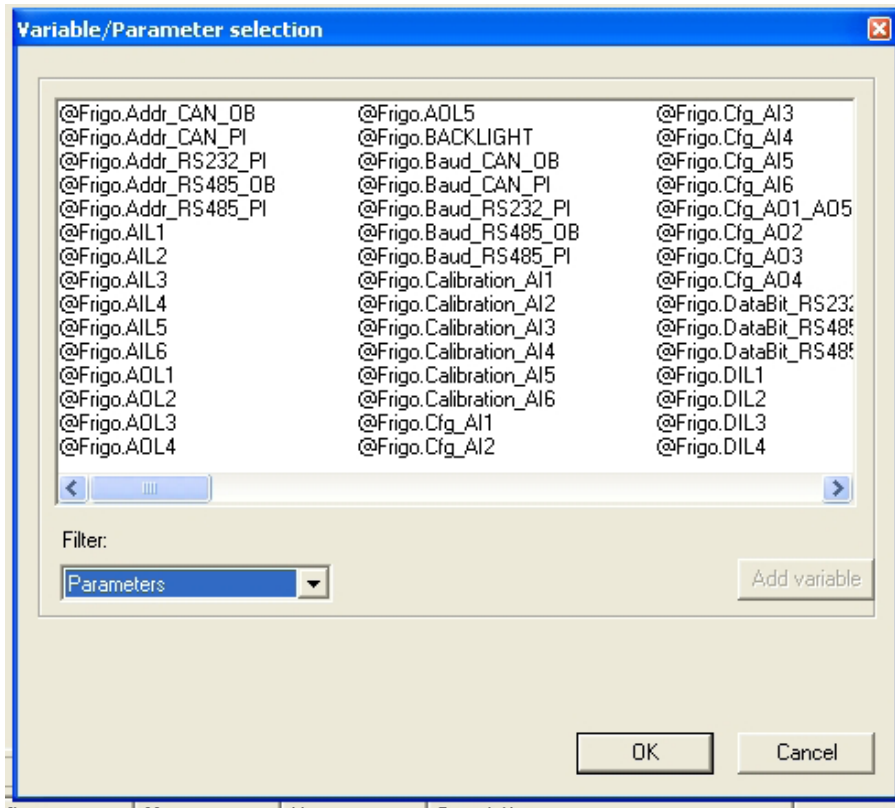
## 2.9.5 LINKING AN EDIT BOX WITH A PLC APPLICATION VARIABLE

You can associate an Edit box with a PLC Application variable through the dialog window which opens from the *Assoc var* field (see paragraph 2.9.2), but in this case it is necessary to use a different filter in the *Filter* field.



## 2.9.6 LINKING AN EDIT BOX TO A PARAMETER

You can associate an Edit box with a parameter through the dialog window which opens from the *Variable* field (see paragraph 2.9.2), but in this case it is necessary to use a different filter in the *Filter* field.



The name of parameters is composed of *@device.variable name*, differently from variables which show just their name.

The parameter may be inserted in the apposite controls property in the following forms:

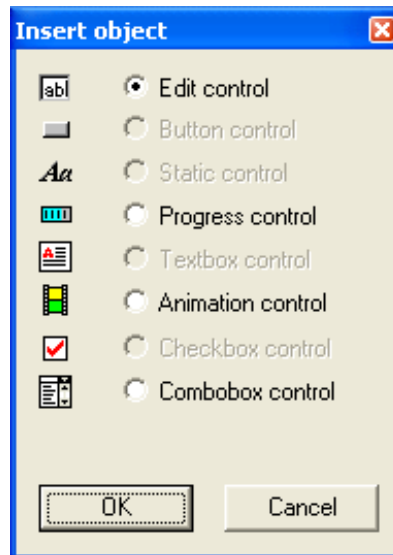
- explicit form = *@d.oi.os:type*: *d* = numerical ID of the device, *oi* = *object index*, *os* = *object subindex type= PLC type* (e. g. *@1.2010.0:UINT*);
- implicit form = *@dev.name*: *dev* = symbolic identifier of the device, *name* = symbolic name of the parameter (e. g. *Frigo.AIL1*).

The *d* (ID) field of the device is a numerical or symbolic identifier (to be defined at project creation). It refers to a specific device which may be local (the device which executes the pages itself) or on the fieldbus.

The *dev* field is a symbolic identifier of a device whose numerical ID can be retrieved by UserInterface.

### 2.9.7 LINKING AN EDIT BOX TO A VARIABLE BY DRAGGING AND DROPPING

You may add variables and parameters to the *Target vars and parameters* window by dragging and dropping them in the page. UserInterface will request to define the type of control to insert, to associate it with the variable.



## 2.10 INSERTING BUTTONS

Buttons are very versatile controls which play an essential role in the interaction between user and system, particularly in case of touchscreen systems without keyboard.

This chapter's aim is to show four kinds of use of the button control:

- LED-button to view the state of a boolean variable;
- command button of a boolean variable's state;
- opening button of a secondary page;
- activation button to start the execution of a customized procedure.

### 2.10.1 INSERTING A LED-BUTTON

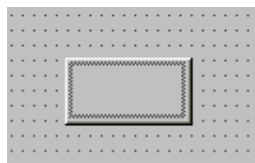
The following paragraph teaches you how to use a button which shows an associated boolean variable's state.

Insert a new button in the page by pressing the corresponding button in the *Page toolbar*.



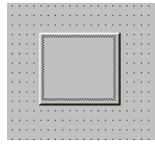
Move the mouse in the active area of the page; a cross + appears. The object will be inserted in the grid near to the mouse cursor.

Confirm the insertion point by left-clicking. A new *Button* control appears. It has a default size.





You may edit both the dimensions by dragging one of the button's vertexes or one dimension at a time by dragging one of the button's sides.



The *Border color* and the *Background Color* properties determine the border and the background color when the button is inactive, whereas the *Selection Border* and *Selection Background* properties define the border and the background color when the button is selected<sup>(1)</sup>.

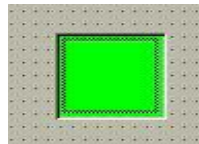
Border color	Black
Background color	Green
Selection border	Black
Sel. background	Red

The *Selection variable* property determines the state of the button and, consequently, the couple of colors related to the control. This property may be associated either with a constant value (*FALSE* = the control is always inactive, *TRUE* = the control is always selected, on *PRESS* = the control is automatically selected when the user presses the button) or with a boolean variable whose value determines dynamically the selection state.

Declare a boolean global variable *b* and associate it with the control button as selection variable.

Selection variable	b
--------------------	---

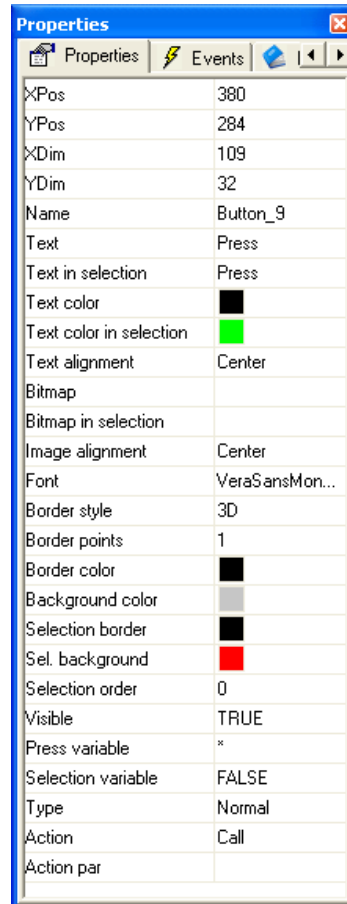
You may customize the button appearance through the *Appearance* property. For example, choose the *Sunken* option<sup>(1)</sup>.





## 2.10.2 INSERTING A BOOLEAN VARIABLE COMMAND BUTTON

Insert a new button in the page by following the aforesaid instructions (see paragraph 2.10.1). Set it beside the LED button and let a text string show on it by means of the *Text* property.



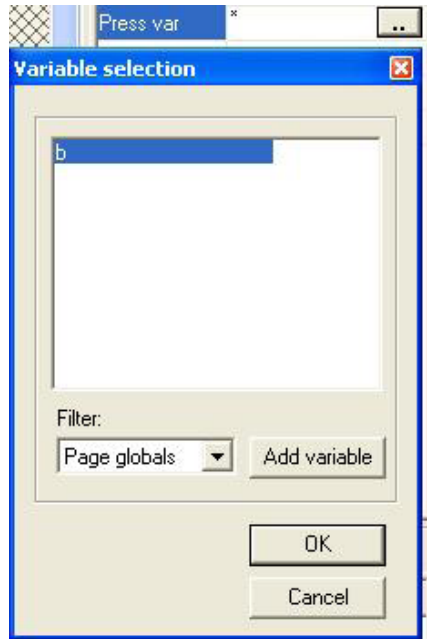
The preview looks like as follows<sup>(1)</sup>.



The *Press variable* property allows the user to associate a boolean variable with a button control. The boolean variable's value corresponds to the pressure state of the button.



For example, associate the button which you have just created with the global variable *b* which has been created paragraph 2.10.1.



At runtime, the LED-button (see paragraph 2.10.1) will be red when pressing the *Press* button. Otherwise it will be green.

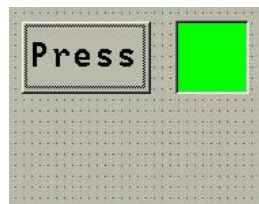
### 2.10.3 INSERTING A BUTTON TO OPEN A CHILD PAGE

Paragraph 2.4.1 showed you how to create a pop-up page.

The following paragraph explains how to invoke the *Pag2* page from the *Init* page by pressing a button.

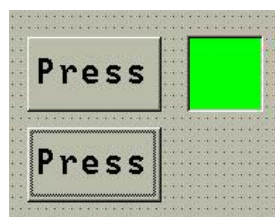
First of all insert a new button in *Init* and set it under the previously created *Press* button (see paragraph 2.10.2). As it should be exactly alike the previous one except the text string and the function, you can copy and paste the *Press* button and afterwards customize its properties.

Select the *Press* button by clicking once: the selection rectangle appears inside the control<sup>(1)</sup>.



Press successively *Ctrl+C* and *Ctrl+V*. A cross + appears. The object will be inserted in the grid near the mouse pointer.

Confirm the insertion point by clicking under *Press*. A copy of the control appears<sup>(1)</sup>; it is the same as the source button except its position and name.

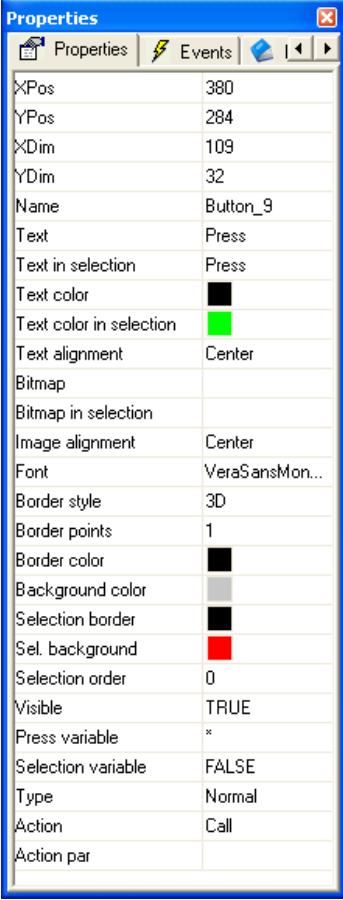






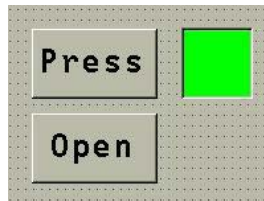
## SoMachine HVAC - UserInterface

You can access this new control's properties and customize them according to the relative purpose<sup>(1)</sup>.



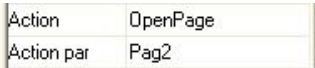
Property	Value
XPos	380
YPos	284
XDim	109
YDim	32
Name	Button_9
Text	Press
Text in selection	Press
Text color	Black
Text color in selection	Green
Text alignment	Center
Bitmap	
Bitmap in selection	
Image alignment	Center
Font	VeraSansMon...
Border style	3D
Border points	1
Border color	Black
Background color	Grey
Selection border	Black
Sel. background	Red
Selection order	0
Visible	TRUE
Press variable	*
Selection variable	FALSE
Type	Normal
Action	Call
Action par	

The preview looks like this<sup>(1)</sup>.



The button control has got a very important attribute, which has not been represented in the properties grid above: the *Action* attribute allows the user to associate an action with the button pressure. Some actions require an additional parameter which you may specify in the *Action par* field.

In this case let us assume that you want that the pressure of the *Open* button opens the *Pag2* page. To obtain this select the *OpenPage* action in the *Action* field; then type the name of the child page *Pag2* in the *Action par* field.



Action	OpenPage
Action par	Pag2

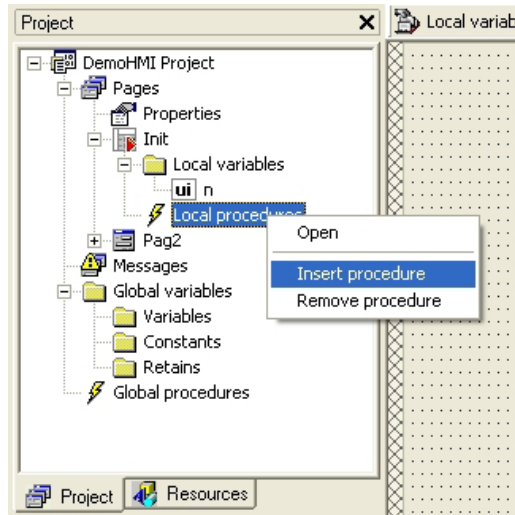


## 2.10.4 INSERTING A BUTTON AIMED AT LAUNCHING A PROCEDURE OF THE USER

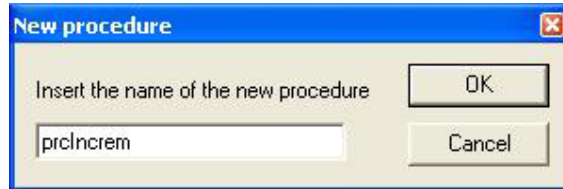
UserInterface enables the user to implement some procedures (see paragraph 4.8.4) through which it is possible to customize the HMI behaviour: this feature makes UserInterface projects very versatile.

Let us suppose that you want to create a procedure to increment the local variable *n* of the *Init* page. As this procedure applies on a local variable, it will be local in the *Init* page, too.

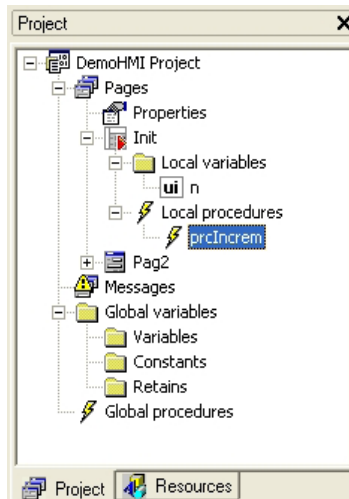
First of all create the procedure: expand the *Init* page tree, right-click on the *Local procedures* item and select the *Insert procedure* command in the contextual menu which appears.



A little dialog window opens. The user is then required to type the new procedure's name. In this case, it may be *prcIncrement*.



Press the *OK* button. Then UserInterface adds a new item in the page tree: it corresponds to the local variable which has been just declared.





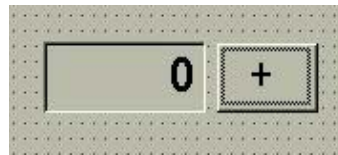
Double-click on the above mentioned new item: the ST language editor opens and lets you either implement or edit the selected procedure's code.

Write a procedure that applies a unit increment to the  $n$  variable.

```
0001 n := n + 1;
```

Then close the document.

Insert a new button beside the edit box associated with the  $n$  variable and type the character + in the *Text* property<sup>(1)</sup>.



Let us suppose that you want to execute the *prcIncrement* procedure by clicking the + button: select the *Call* action in the *Action* field and type the procedure's name in the *Action par* field.

Action	Call
Action par	prcIncrement

Every time the user will press the + button when executing the HMI,  $n$  will increase by one and the edit box will show the up-to-date value.

## 2.11 VISIBILITY AND UPDATING OF CONTROLS

As stated in the previous paragraphs, each control has its own properties which the user may customize through the properties table fields.

Some of these features are specifically related to a single type of control. Others may be included in the properties set of different objects. The following paragraphs concern two important properties which are common to some kinds of control.

### 2.11.1 THE VISIBILITY PROPERTY

Almost all of controls are endowed with the *Visibility* property, which determines whether the object is visible or not. This property can be associated either with a constant value (*FALSE* = the control is always hidden, *TRUE* = the control is always shown) or a boolean variable, whose value dynamically establishes the visibility state.

By following the instructions in paragraph 2.7.1 you have inserted the string: *Stringa di testo* in the *Init* page. At present this string is always visible, as you can deduce from the assigned value to its *Visibility* property.

Visible	TRUE
---------	------

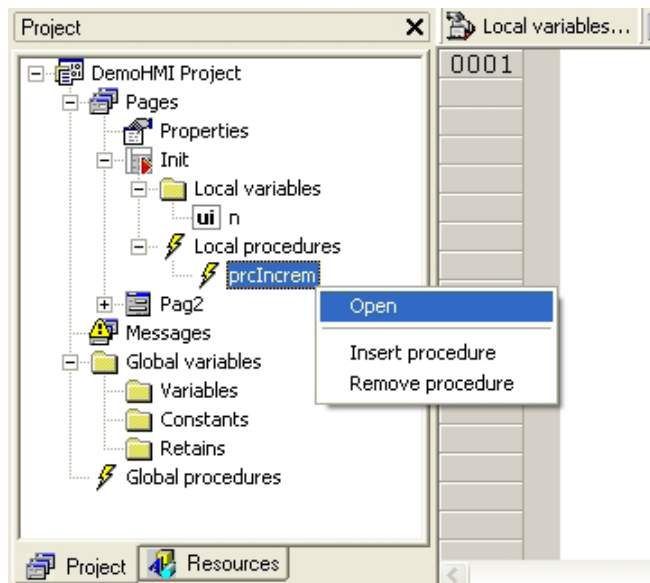


Let us assume that you want to assign this control's visibility to the local variable *n*, which is displayed in the edit box created in paragraph 2.9.2 and managed by the *prcIncrement* procedure, which was implemented in paragraph 2.10.4 and started up by the + button. More precisely, let us suppose that you want the text string visible when *n* is even, whereas hidden when *n* is odd.

To this purpose, it is necessary to declare a new boolean local variable which indicates whether at present *n* is even.

	Name	Type	Array	Init value	Description
1	n	UINT	No	100	Variabile contatore
2	even	BOOL	No	TRUE	Variabile locale n è pari

Then it is necessary to edit the *prcIncrement* procedure so that, when it refreshes the *n* value, it evaluates again whether it is even or odd. In order to access the *prcIncrement* source code, select the corresponding item in the project tree by right-clicking. Afterwards, choose *Open* from the contextual menu which appears.



The ST language editor opens and the procedure's code may be extended as follows.

```

0001 n := n + 1;
0002
0003 even := (n MOD 2) = 0;
0004
0005 |
    
```

In order to associate the string's visibility state with the *even* boolean variable, select the text string and click the *Visibility* field: a button appears.

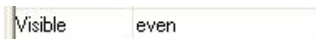




After clicking it, a dialog box opens. Select the *radio button Variable*, which enables the overhead variables list; change the filter *Filter* into *Page locals* and select the only local boolean variable that is *even*.



Confirm your choice by clicking *OK*. The result is the following.



### 2.11.2 THE REFRESH PROPERTY

When available, the *Refresh* property determines if the associated object has to be drawn once (when opening the page or coming back from a child page) or it needs to be constantly refreshed.

This property distinguishes, for example, the edit box and the text box.

With regard to the edit box, the refresh property has to be set when compiling and it cannot be edited at runtime. If you assign *Refresh = TRUE*, the associated variable's value is constantly read and refreshed, otherwise (*Refresh = FALSE*) the value is read and refreshed only when you open the page or when you come back from a child page.

There is another option about text boxes: you can associate a boolean variable that is used as trigger for refresh: when the trigger variable becomes *TRUE*, the control's contents are refreshed then it is automatically reset by UserInterface to *FALSE*.

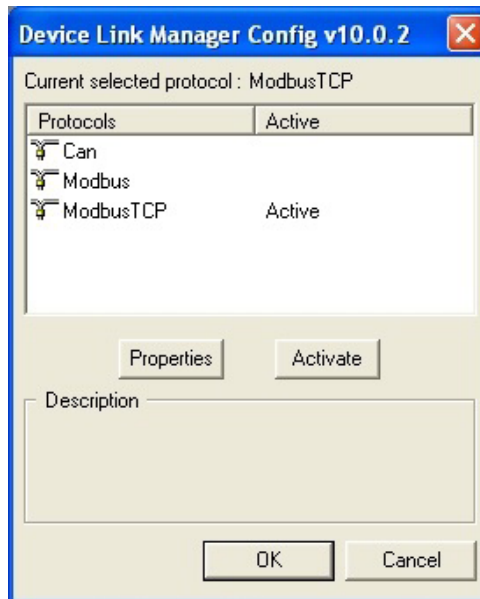
## 2.12 COMPILING AND DOWNLOADING THE PROJECT ON THE TARGET

The following paragraph shows you how to compile and download a HMI project on the target board that runs UserInterface.



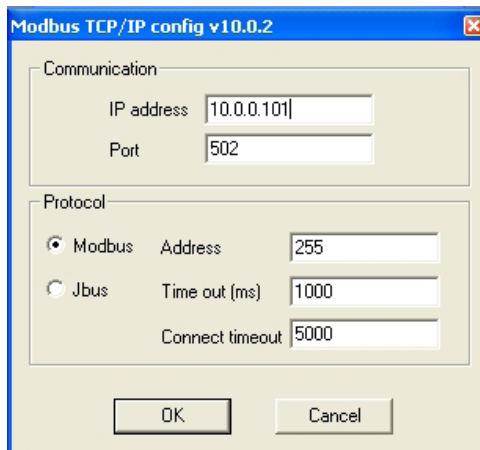
## 2.12.1 CONNECTING TO THE TARGET

Launch the *Communication settings* command from the *Project* menu. This causes the following dialog window to open.



The user is required to select a suitable communication protocol from the left column and to activate it by pressing the *Activate* button.

Then the *Properties* button becomes active: by clicking it the user accesses another dialog window which is different in accordance with the specific selected control and lets set the protocol's parameters. Let us consider the following example.



## 2.12.2 COMPILING PAGES FOR THE TARGET

You can start compiling the HMI project by clicking the corresponding button in the User-Interface's *Project toolbar*.



Compilation is composed of two phases: the first one consists in the PLC code generation which realizes the pages as they have been planned in UserInterface. The program shows in the *Output window* the progress level of the compilation and displays eventual errors.



```
Creating the .ppj file...
Creating the .plc file...
Building the page call tree...
Associating programs with CPU tasks...
Declaring global variables...
Declaring fonts...
Declaring bitmaps...
Declaring constants...
Generating alarm page function blocks...
Generating page-drawing function blocks...
Generating page-refreshing function blocks...
Generating main programs...
PLC code generation completed
```

The second one consists in the compilation of the PLC code which has been generated during the first phase. It can be started only if the first phase has been accomplished without any error.

This process is carried out by an external tool: the PLC command-line compiler *11c*, which UserInterface automatically invokes with the suitable parameters.

### 2.12.3 DOWNLOADING AND EXECUTING THE COMPILED PAGES ON THE TARGET

At the end of the compilation, if all the phases have been successfully accomplished, you will see the downloading button become active in the *Project toolbar*



Clicking it, you activate again the PLC command-line compiler *11c*, which this time just downloads the compiled code in the target.

The downloading permission management depends on the implementation of the on board firmware. Consequently it changes according to the destination target of the download.

### 2.12.4 SIMULATION

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

In order to start the simulation, just click on the appropriate item on the Project toolbar.



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

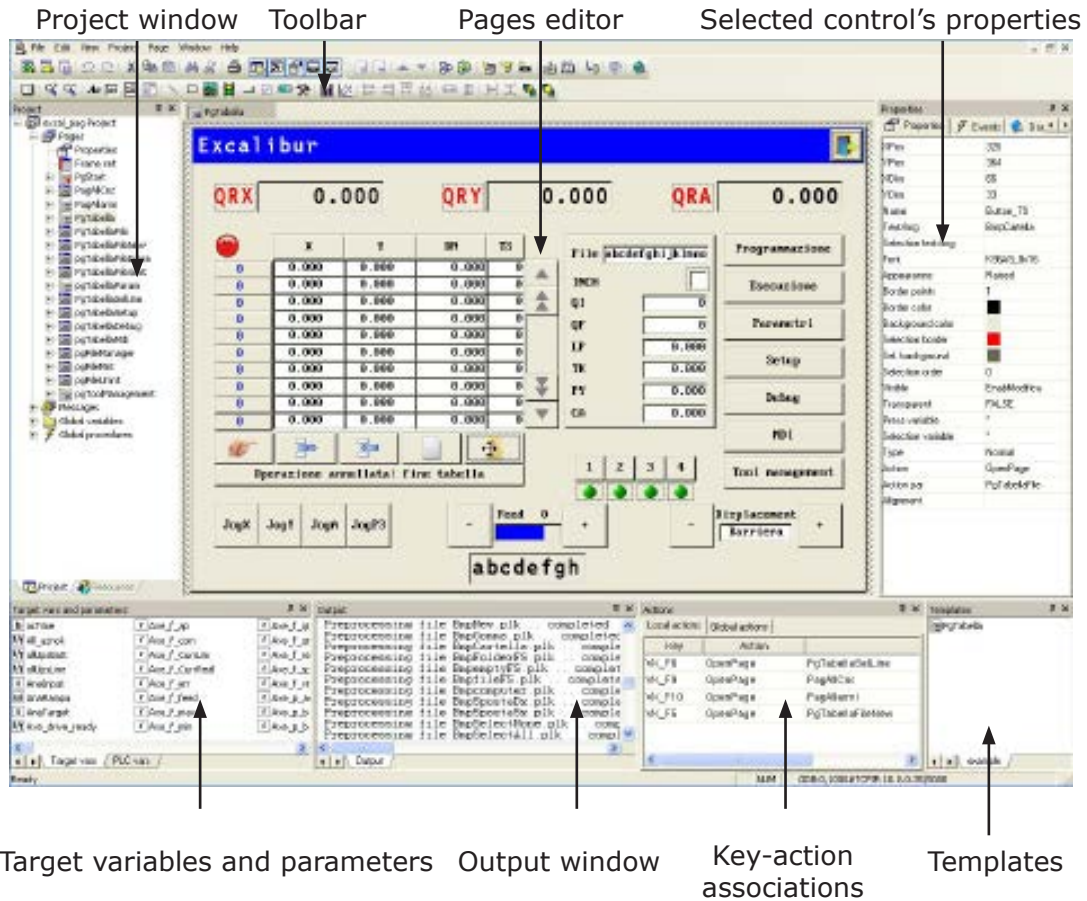






## 3. USERINTERFACE LAYOUT

The following picture shows you the layout and the essential elements of UserInterface.



### 3.1 PROJECT WINDOW

This window includes two pages which are alternatively selectable by pressing the corresponding tab:

- **Project:** it shows the project tree and all the objects the project is composed of, hierarchically arranged. The pages node contains the project properties and the single pages. Each page contains the list of the local variables (visible and usable only in the page where they are declared) and the local procedures, which can be invoked only from the page where they are implemented. Moreover there is the node of the asynchronous messages, the node of the global variables (visible and usable from whatever page) and the node of the global procedures which you can invoke from whatever page.
- **Resources:** it shows the project resources, that is fonts, bitmaps, strings table, enumerated data types, images lists, and sets.



## 3.2 EMBEDDED EDITORS

UserInterface is endowed with three types of editor:

- Pages editor: in order to open this editor, double-click the name of the desired page in the project tree (see paragraph 3.1). This tool shows you a page preview and lets you edit it: you may either add or remove controls (see paragraph 4.4), customize properties, manage the events and the documentation.
- Variables editor: by double-clicking on a local or global variable in the project tree, you can see respectively the declaration table of the local variables or the global variable table.
- Procedures editor: it allows the user in implementing procedures to be associated to the events which are defined for the various project's objects (pages and controls) or generated from the user himself (see paragraph 4.8).

## 3.3 PROPERTIES WINDOW

Each time you select an object in the pages editor, the properties window automatically refreshes and shows the selected object's properties and events.

This window is composed of many pages which you may select alternatively by pressing the corresponding tag above.

- Properties: it shows a table including the selected object's properties either it is a whole page or it is a page's control. The user is enabled to customize this values through the right-hand column of the table.
- Events: it shows a table including the typical events of the currently selected object. The user may associate either a local or variable procedure with each event by typing its name on the corresponding row of the event in the right-hand column.
- Doc: it displays a table which shows the *Description* field of the currently selected object. The user may describe the object and this description will be included in the automatic documentation management (see paragraph 4.10).

## 3.4 TOOLBARS

The user can give commands to UserInterface through some useful toolbars. A toolbar can be defined as a collection of buttons which you may enable by left-clicking them and whose functions are intuitively represented by their icons.

The toolbars support tooltips, too. A tooltip is a small text frame containing a short description of the object which UserInterface automatically displays when you hover with the mouse over a button.

UserInterface is endowed with three essential toolbars:

- Main toolbar: it contains the commands to open and save the project, to cancel/restore the last changes, to print, to display or close other toolbars.
- Project toolbar: it allows you to add new elements to the project as variables, pages, events, actions, as well as to enable or prevent the simulation mode and to compile and download the whole project.
- Page toolbar: it allows you to choose a new type of control to be inserted in the active page, to align or equally space several controls, or to set the vertical order of the elements on the page.



## 3.5 THE OUTPUT WINDOW

UserInterface prints in this window some messages which indicate the progress and the output of the requested processes: opening and compilation of a project, resources importing/exporting, etc..

## 3.6 TARGET VARIABLES AND PARAMETERS

This window shows the list of external variables, available for UserInterface coding.

The window is composed of several pages which you may alternatively select by pressing the corresponding tab. One page contains the list of the available variables (file *.tgt*), another page contains the list of the variables which have been exported from the PLC Application (file *.exp*). Other pages are optional, as many as the number of devices with external parameters linked to the project.

## 3.7 TABLE OF KEYS-ACTIONS ASSOCIATIONS

This table takes primary importance in case of traditional keyboards without touchscreen where the user interact with the system by pressing the relevant keys.

See in paragraph 4.8.5 the list of actions which may be associated to keys.





## 4. HMI PROJECT IN USERINTERFACE

UserInterface manages the creation (development) of pages for a specific application as projects.

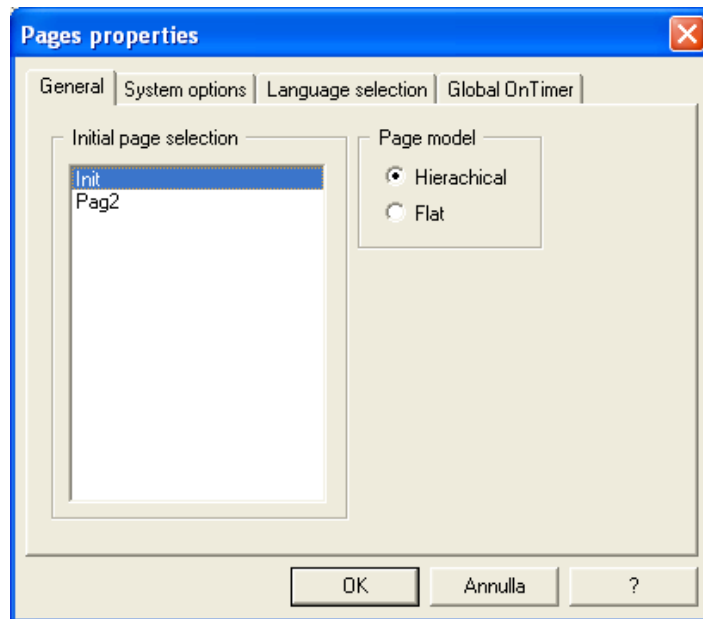
The UserInterface project is composed of several pages where the user may arbitrarily arrange the controls.

In each UserInterface project you have to specify the start page which will be displayed at the start of the system. Other pages will have at least a parent page from which they will be invoked and may have child page to invoke. The invoking/invoked relations implicitly give to the whole project a tree structure.

### 4.1 PROJECT PROPERTIES

In the project tree, click the *Pages* item and access the *Properties* item. By double-clicking the *Properties* item you open a dialog window which is composed of four pages. The following paragraphs show you the features of these pages.

#### 4.1.1 GENERAL



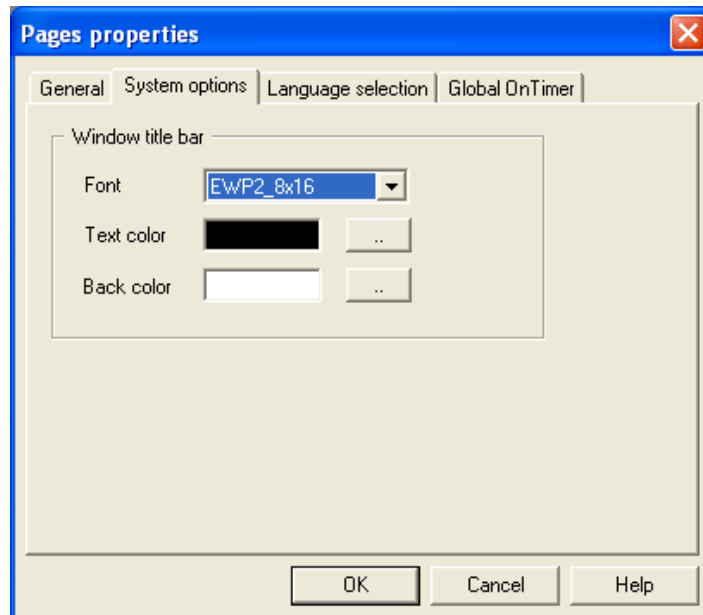
It allows to select the UserInterface project's start page among the implemented pages.

The *Page Model* feature allows to select the type of page model in case of a page calls another page. If the model is hierarchical then a child page cannot recall a parent page. Instead if the model is flat all the pages can call the others without limitations.

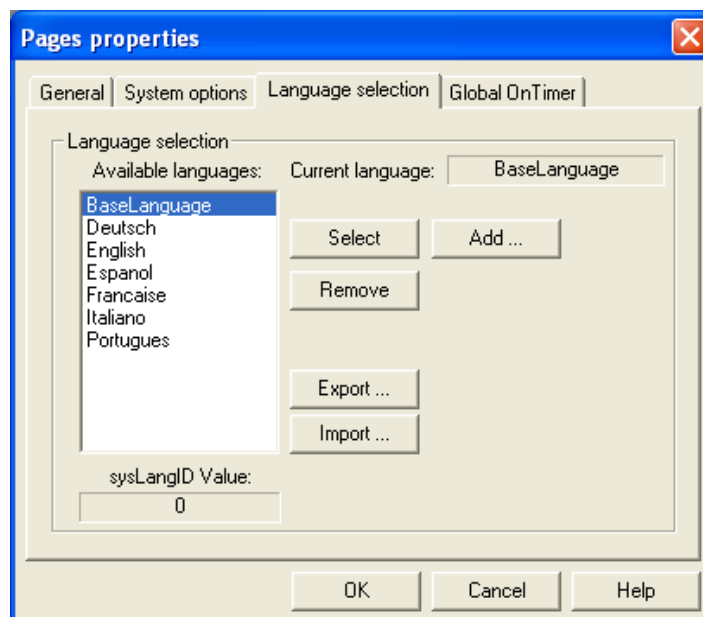


## 4.1.2 SYSTEM OPTIONS

It allows the user to customize the window's title bar features: the font, the text color and the background color.



## 4.1.3 LANGUAGE SELECTION



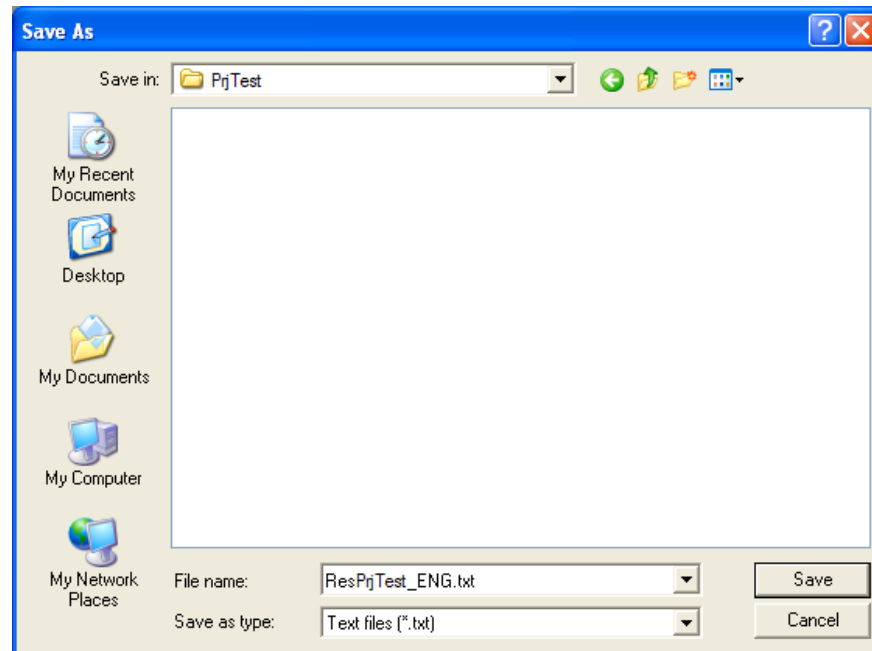
It allows you to add, remove, export, import, and select the resources languages (see paragraph 4.9). The label: *sysLangID Value* indicates the value which the *sysLangID* target variables must take to display the pages in the selected language.

In order to add a language, apply the following procedure.



## SoMachine HVAC - UserInterface

First of all export the language supported by the translator, choose Italian and press the *export...* button, which opens a window requiring the destination folder for the selected language file.

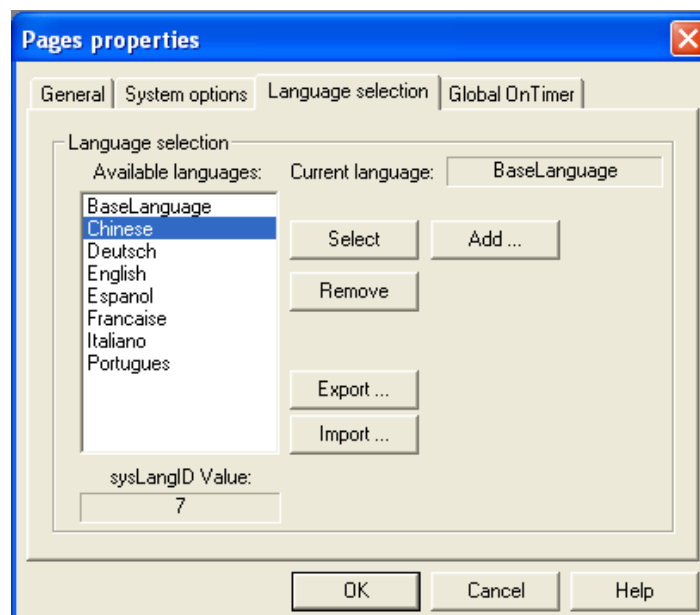


The program suggests a file name: *Res* + project title+'\_' + first three characters of the language + extension *.txt*. At the end of the exportation the file is composed of all the project's resources which have to be translated:

- strings
- enumeratives

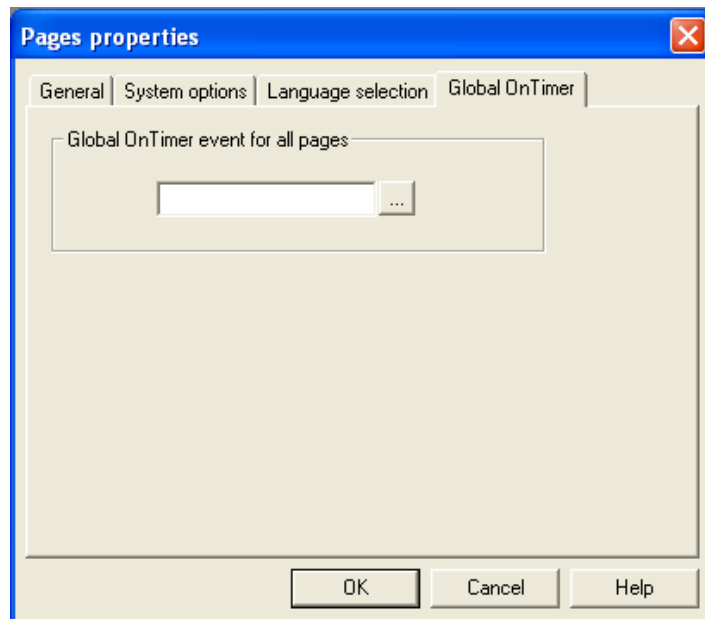
Translate the file and replace the text under the *Language* tag with the one of the new language (for example, in this case change it into *Chinese*). In the *Language selection* panel choose the *Import...* button, then select the suitable file in the PC.

The new language appears in the list.



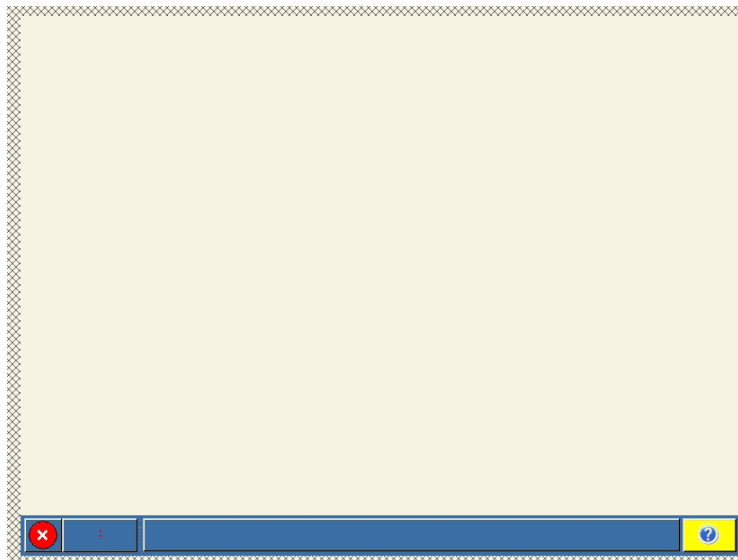


## 4.1.4 GLOBAL PERIODIC PROCEDURE



Global on timer allows you to specify the name of a global procedure to be periodically and independently executed on the active page. Such a procedure may be effectively used to constantly test one or more PLC variables and to emit alarm messages, for example through asynchronous messages (see paragraph 4.3.4).

## 4.2 FRAME SET



UserInterface allows to define areas which are called frames and are placed on the sides of the screen and are always active<sup>(1)</sup>.

The user may set these frames' dimensions and insert some controls which are active whatever the currently loaded page. Consequently frames are useful to host the objects which have to appear in the whole project. In this way the user does not need to duplicate them in each page.

As regards to the above, there are two exceptions: the pop-up pages (see paragraph 4.3.3) when the *Modal* property is set to *Yes* and all the asynchronous messages. When these pages are active, the controls of the frame set are automatically disabled.





## 4.3 PAGES

### 4.3.1 NAVIGATING BETWEEN PAGES

UserInterface manages pages development for a specific application as projects.

UserInterface project is composed of pages where the user can arbitrarily arrange controls.

In each UserInterface project it is necessary to define a start page which will be viewed at system startup. Other pages must have at least a parent page from which they are invoked and may have child page to invoke. The invoking-invoked relations of the pages give the whole project, even though in an implicit way, a multi-node tree structure.

A child page may be invoked in two ways:

- Through an action associated to a key: associate an *OpenPage* action with a physical key (if there is a keyboard) or with a virtual key (whose pressure is an event raised by software);
- Through an action associated with a button: insert in the parent page a *Button* control (see paragraph 4.4.7) and specify in the *Action* property that by pressing it the child page opens.

### 4.3.2 CHILD PAGES

Par041	0.000	Par051	0.000	Par061	0.000	Par071	0.000
Par042	0.000	Par052	0.000	Par062	0.000	Par072	0.000
Par043	0.000	Par053	0.000	Par063	0.000	Par073	0.000
Par044	0.000	Par054	0.000	Par064	0.000	Par074	0.000
Par045	0.000	Par055	0.000	Par065	0.000	Par075	0.000
Par046	0.000	Par056	0.000	Par066	0.000	Par076	0.000
Par047	0.000	Par057	0.000	Par067	0.000	Par077	0.000
Par048	0.000	Par058	0.000	Par068	0.000	Par078	0.000
Par049	0.000	Par059	0.000	Par069	0.000	Par079	0.000
Par050	0.000	Par060	0.000	Par070	0.000	Par080	0.000

**DIA** **UM** **NXT**

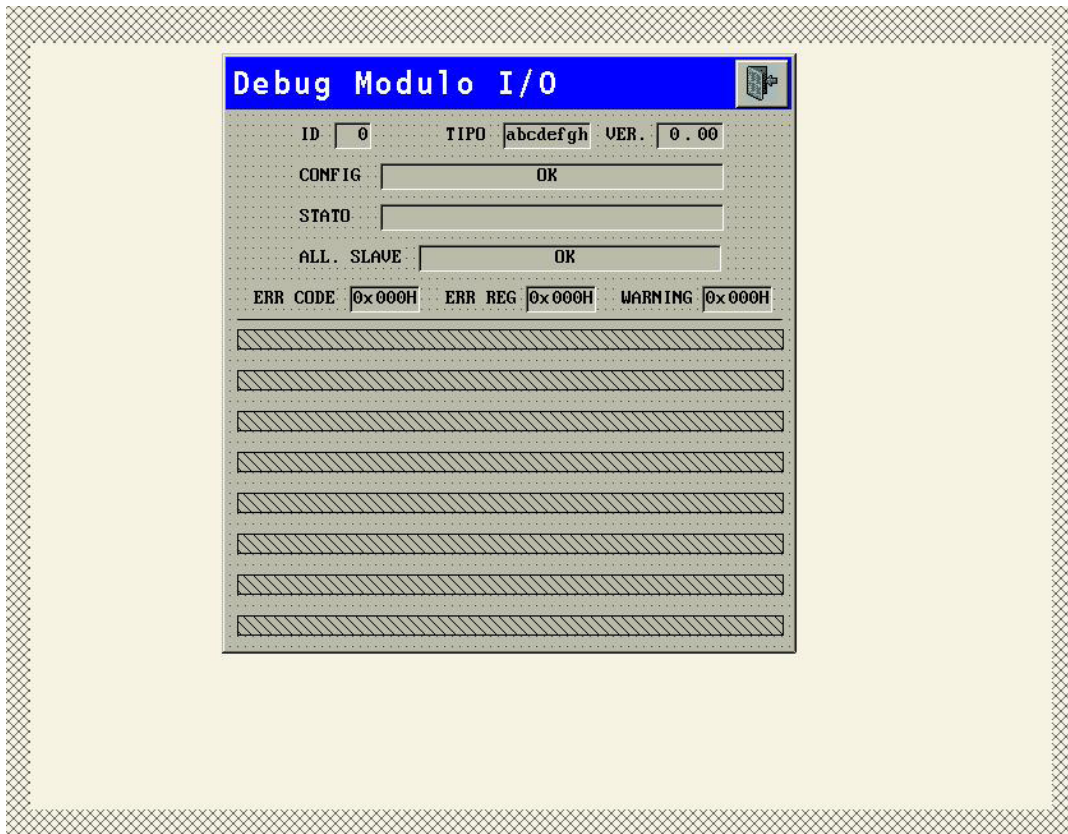
Let us assume that you want to add a page to a project<sup>(1)</sup>.

UserInterface displays a dialog window which requests to insert the name you intend to assign to the new page. This dialog window contains a checkbox with the label: *Pop-up*. If you do not select it, the new page will be a child one.



A child page fits the whole screen or, alternatively if there are defined frames (see paragraph 4.2), it fits the free remaining area. Consequently, the user cannot define position and dimensions of a child page as they are automatically set according to the screen and the frame set.

### 4.3.3 POP-UP PAGES



When creating a new page, if the user selects the aforesaid checkbox with the *Pop-up* label, the new page will be a pop-up one<sup>(1)</sup>.

There are no restrictions about position and dimension. In fact the user may superimpose a pop-up page on the frames: when activating this page, if it is not modal (property: *Modal*), the controls superimposed on the open page will be disabled; otherwise, all the controls will be inactive.

### 4.3.4 ASYNCHRONOUS MESSAGES

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

- They have an additional property, that is the identifier of the associated message (*Msg ID*).
- They cannot contain invocations to child pages.
- They have no defined parent page nor a tree structure (see Introduction 4), but they can be invoked from any other standard page.

An asynchronous message cannot be explicitly invoked; the system displays it whatever the active page when it intercepts a message containing the corresponding *Msg ID*. This message may be launched either by the firmware or by a procedure through the *Video\_SendMessage* function (see paragraph 8.1.8) by using the following syntax:

```
Video_SendEvent( kWM_MSG, Msg ID );
```



## 4.4 CONTROLS

A control is a display element which is contained in a page. The following paragraphs shows you the controls which UserInterface supports.

### 4.4.1 STATIC

It displays a fixed string, whose contents cannot be edited when executing. In fact, you should specify the text of the string directly or by the association of the ID of a string defined as resource to support multi language management. For project resources and multi language support see paragraph 4.9.

In order to insert a *Static* control, press the corresponding button in the *Page toolbar*.



Then click the point where you want to insert the control.

You can get information on properties and events of the *Static* control in paragraph 5.4.

### 4.4.2 GRAPHIC ELEMENT

It displays a static line or rectangle. This means that their properties cannot be edited when executing.

In order to insert a *Line* control, press the corresponding button in the *Page toolbar*.



Then click the point where you want to insert the control.

In order to insert a *Rectangle*, press the corresponding button in the *Page toolbar*.



Then click the point where you want to insert the control.

You can get information on properties and events of the *Line* and *Rectangle* controls in paragraphs 5.5-5.6.

### 4.4.3 EDIT BOX

It displays the contents of an associated variable.

In order to insert an *Edit box*, click the corresponding button in the *Page toolbar*.



Then either click the point where you want to insert the control or drag a variable from the project tree or from the library window.

You can get information on properties and events of the *Edit box* control in paragraph 5.7.



## 4.4.4 TEXT BOX

It displays the contents of an associated string variable. It supports the formatting on several lines of the text which is contained in the string.

To insert a *Text box* control in the page, press the corresponding button in the *Page toolbar*.



Then either click the point where you want to insert the control or drag a variable from the project tree or the library window.

You can get information on properties and events of the *Text box* control in paragraph 5.8.

## 4.4.5 IMAGE

It displays a bitmap image.

In order to insert an *Image* press the corresponding button in the *Page toolbar*



Then click the point where you want to insert the control.

You can get information on properties and events of the *Image* control in paragraph 5.9.

## 4.4.6 ANIMATION

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

In order to insert an *Animation* press the corresponding button in the *Page toolbar*.



Then click the point where you want to insert the control.

You can get information on properties and events of the *Animation* control in paragraph 5.10.

## 4.4.7 BUTTON

You may use the *Button* control either to check a boolean variable's state or (press = *TRUE*, release = *FALSE*) or to send a command to the system.

In order to insert a *Button* press the corresponding button in the *Page toolbar*.



Then either click the point where you want to insert the control or drag a boolean variable from the project tree or the library window.

You can get information on properties and events of the *Button* control in paragraph 5.11.



## 4.4.8 CHART

Chart control draws the static diagram of one or more arrays of values associated. In order to insert a *Chart* control, click the corresponding button in the *Page toolbar*.



Then click the point where you want to place the control.

You can get information on properties and events of the *Chart* control in paragraph 5.14.

## 4.4.9 TREND

After assigning up to 8 numerical variables, the object will automatically and periodically (once every a defined time) acquire their values and will draw the corresponding graphic in a dynamic and automatic way.

In order to insert a *Trend* control press the corresponding button *Page toolbar*.



Then click the point where you want to insert the control.

You can get information on properties and events of the *Trend* control in paragraph 5.15.

## 4.4.10 PROGRESS BAR

It represents the progress of an operation by showing a stained bar in a horizontal or vertical rectangle. The length of the bar, related to the bar's length, shows the percentage of the completed operation.

In order to insert a *Progress bar* control press the corresponding button in the *Page toolbar*.



Then click the point where you want to place the control.

You can get information on properties and events of the *Progress bar* control in paragraph 5.12.

## 4.4.11 COMBO BOX

Shows a list of strings connected to a variable with an enumerator element. In order to insert a Combobox, click the corresponding button in the *Page toolbar*.



Then either click the point where you want to insert the control or drag a variable from the project tree or from the library window. You can get information on properties and events of the Combobox control in paragraph 5.16.

## 4.4.12 CHECKBOX

Displays a check box that allows the user to select a true or false condition identified by a variable.



In order to insert an Checkbox, click the corresponding button in the Page toolbar.



Then either click the point where you want to insert the control or drag a variable from the project tree or from the library window. You can get information on properties and events of the Checkbox control in paragraph 5.17.

### 4.4.13 CUSTOM CONTROL

This control is implemented in the firmware. You can have several types of custom controls which are marked by the *Control ID* property and each type of control may have several instances.

In order to insert a *Custom control*, press the corresponding button in the *Page toolbar*.



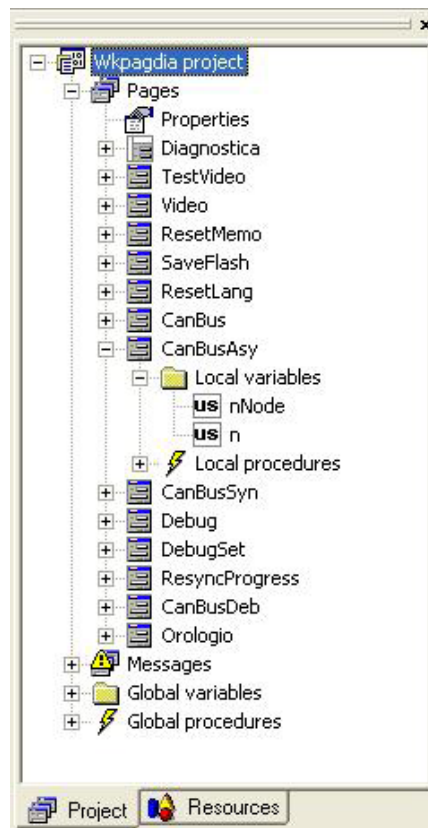
Then click the point where you want to insert the control.

You can get information on properties and events of the *Custom control* in paragraph 5.13.

## 4.5 VARIABLES

In a UserInterface project there are different classes of variables. The following paragraphs show you their features.

### 4.5.1 LOCAL VARIABLES



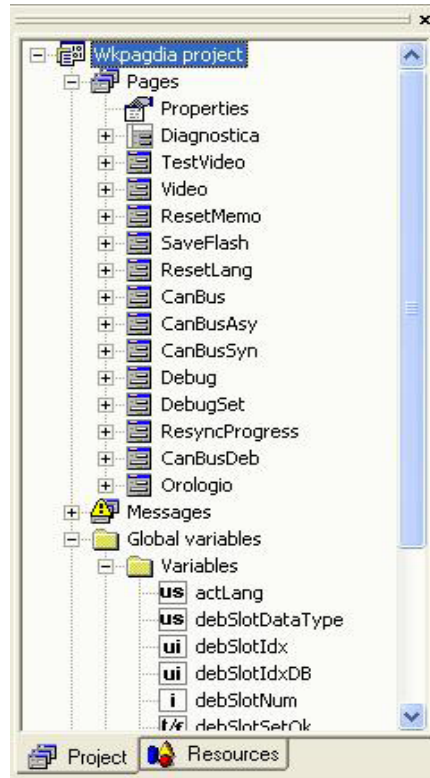




Local variables are variables of the UserInterface project. You can access them only through the page they were declared from.

They are listed in the project tree, under the *Local variables* folder. Local variables can be used to carry out operations on PLC (for example to apply a different scale or to add an offset) or system variables or to implement local procedures.

## 4.5.2 GLOBAL VARIABLES



Global variables are declared in UserInterface and they are accessible from every page of the project. Global variables are listed in the *Global variables* folder in the project tree. The function of the global variables is similar to the local variable's one but the different visibility scope makes them unusable for the implementation of global procedures or for the parameters passing between distinct pages.

## 4.5.3 VARIABLES IMPORTED FROM PLC

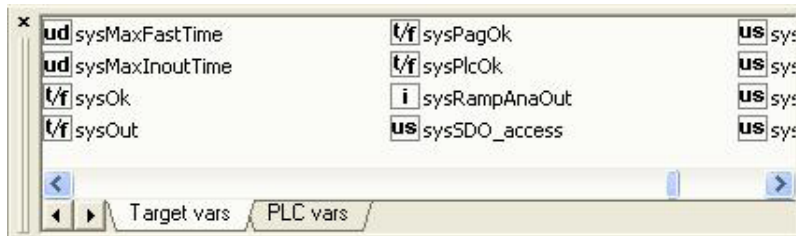


A compiled UserInterface project consists in a PLC that, once downloaded on the target board, is executed by the actual PLC., which is implemented with Application. Variables exported from the Application PLC contained in the *.exp* file enable the interaction between these two distinct components. PLC variables which are not automatic, thus associated to a datablock are exported in the *.exp* file.



In order to include a `.exp` file in the UserInterface project, press the *Link PLC variables file* option from the *Project* menu, then search for the file in the PC resources. After linking a `.exp` file to the UserInterface project, you can get a list update of the exported variables by selecting the *Refresh PLC variables* option from the same menu.

## 4.5.4 SYSTEM VARIABLES



The interaction between UserInterface and target is enabled by system variables which the software publishes outside in a `.tgt` file.

You may access system variables in read/write or in read-only mode; if you try to access a read-only variable in write mode, an error will occur when compiling.

## 4.6 MULTIPLE PAGES MANAGEMENT

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

Sets (see paragraph 4.9.6) can be used with edit boxes or progress bars. Sets are ensemble of variables even of different type. Set definition can be done from the resource tree, they are implemented using a table with a series of variables that are dynamically associated to the control basing on the current index assigned to the page.

Let us see how to use a set.

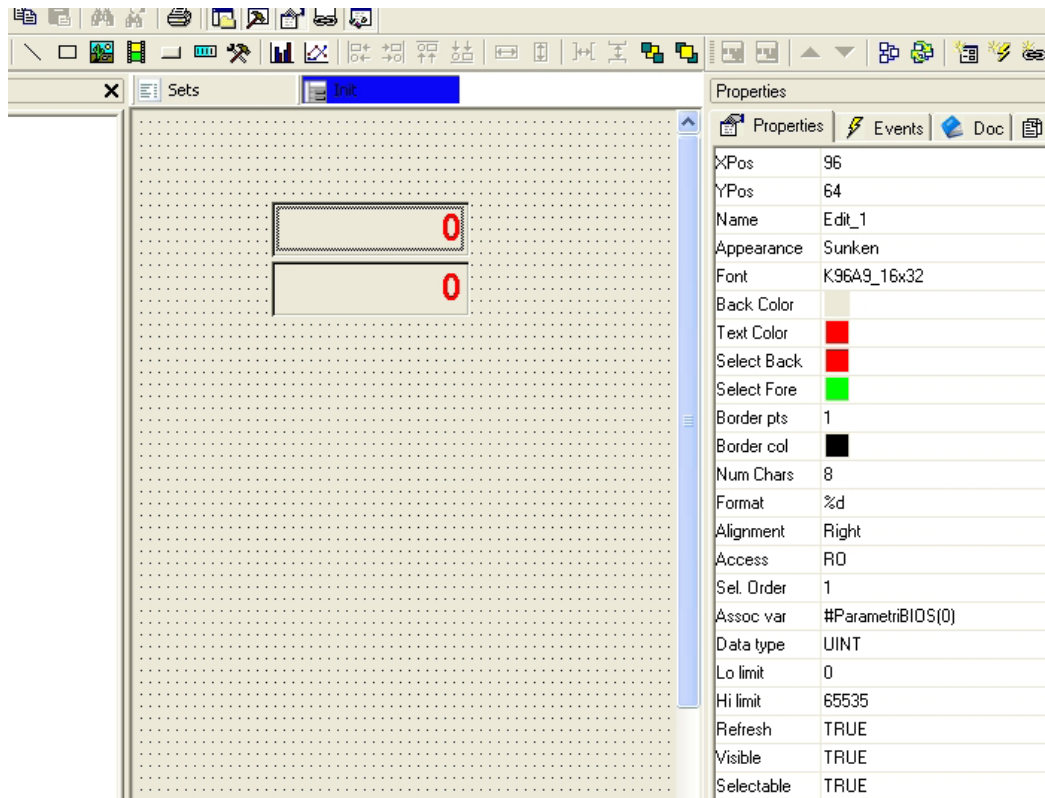
### 4.6.1 ASSOCIATION OF ELEMENTS OF A SET

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

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

A page contains one or more controls based on one (or more) set. At runtime, the page is replied in order to show all the elements contained in the set. In the last page, if any control cannot be filled with element value, that control is hidden<sup>(1)</sup>.





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

## 4.6.2 NAVIGATION OF THE ELEMENTS OF A SET

Navigation of pages that represent a set of elements is automatically done using the *Nex-tEdit* event of the last selectable control of the page and using *PrevEdit* event of the first selectable control of the page.

It is also possible to send special events to force the change of the page in this way:

```
Video_SendEvent( KEV_WM_CHANGESETPAGE, numpage );
```

Where *numpage* is the number of the page of the set.

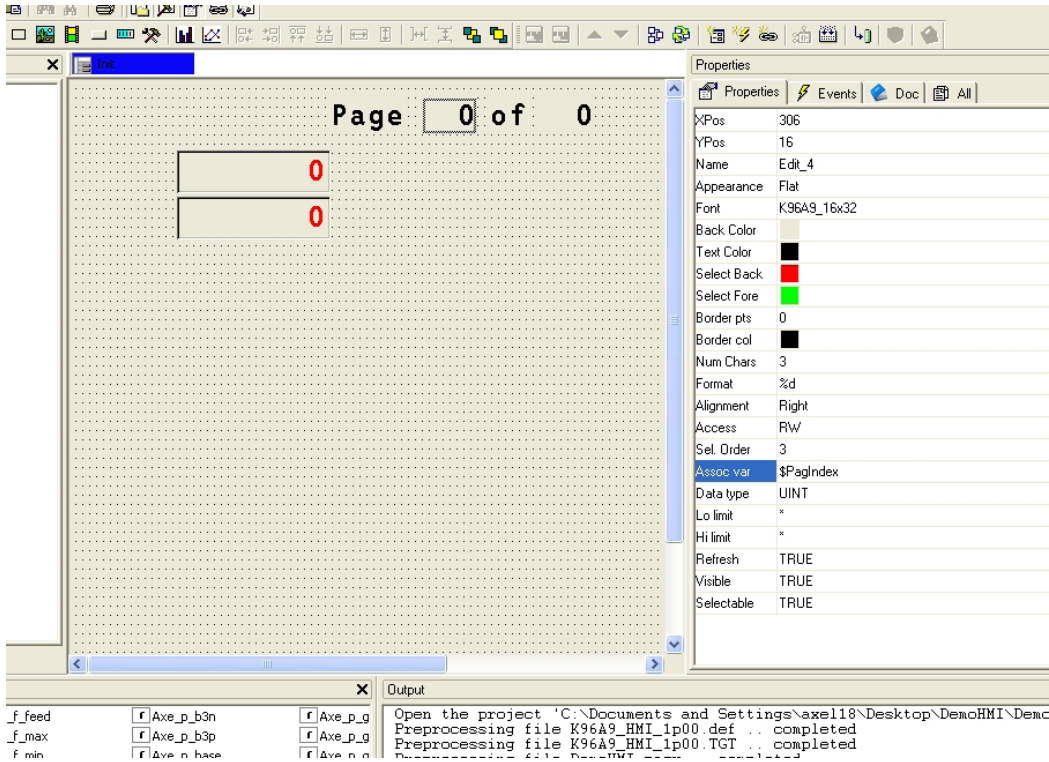
## 4.6.3 PAGES NUMBERING

UserInterface defines two variables related to pages numbering:

- *\$PageIndex* = current index of the page containing controls based on a set;
- *\$PageNumber* = number of pages that complete the visualization of the whole elements of the set.



These variables can be used in the page to show the numeration of the pages. In fact they can be used as variables associated to edit box controls in this way<sup>(1)</sup>:



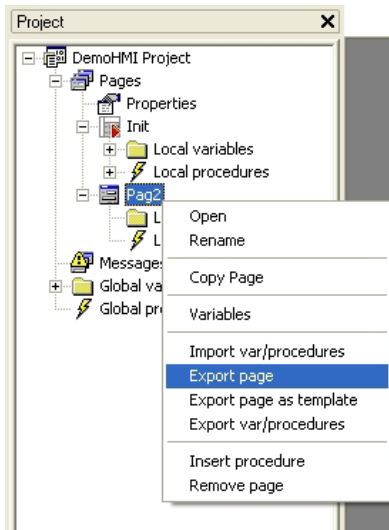
## 4.7 ADVANCED OPERATIONS ON PAGES

Advanced operations such as export/import, copy/paste and page based template management can be done with UserInterface. Next paragraphs show these arguments in details.

### 4.7.1 EXPORT/IMPORT OF PAGES TO/FROM FILES

Each page, even if of a certain complexity, can be saved to be used later in other projects.

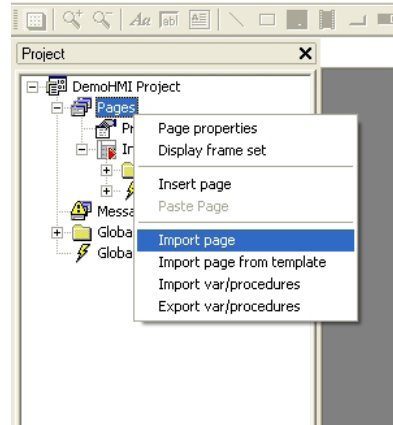
To do so click with the right button on the page node in the project window then select *Export page* from the menu:





Next, application asks user to insert the name of the file in which the page will be saved. This file assumes a `.pex` extension. Export file contains page info and local procedures.

Import operation is quite similar to the export operation. Select *Pages* node, click with right button and select *Import page* from the popup menu. User can then select the file of the page to import. Imported page takes the same name that it had when it was exported.

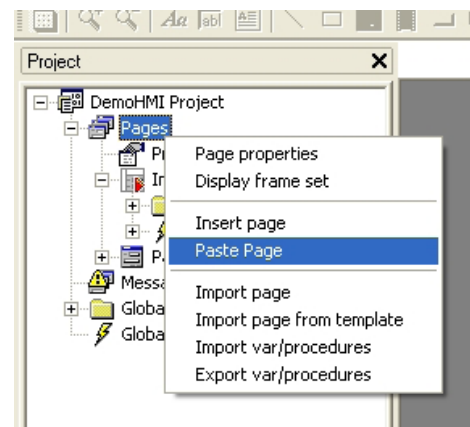
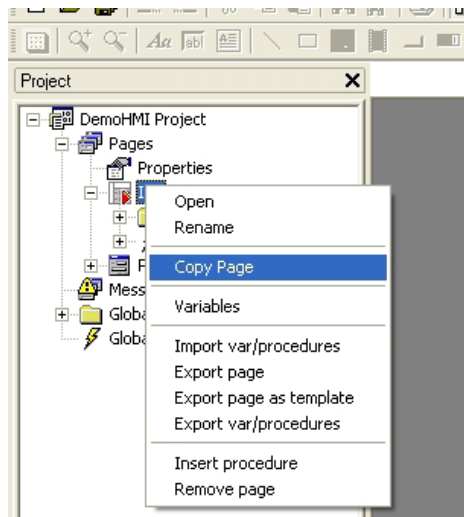


## 4.7.2 EXPORT/IMPORT PROCEDURES AND VARIABLES

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

## 4.7.3 COPY/PASTE OF PAGES IN THE PROJECT

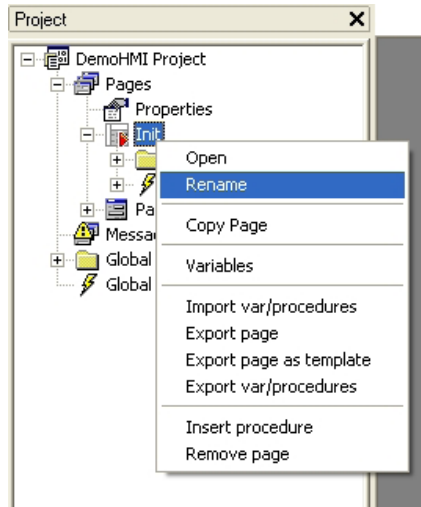
It is possible to copy and to paste a page inside the project. Select desired page, click with right button of the mouse and select *Copy Page* from the menu. Then, to paste page copied, select *Pages* node and select *Paste Page* from the menu.





## 4.7.4 RENAME PAGES

Select desired page from the project tree then click with the right mouse button and select *Rename* from the menu. This allows the user to change the name of the page.



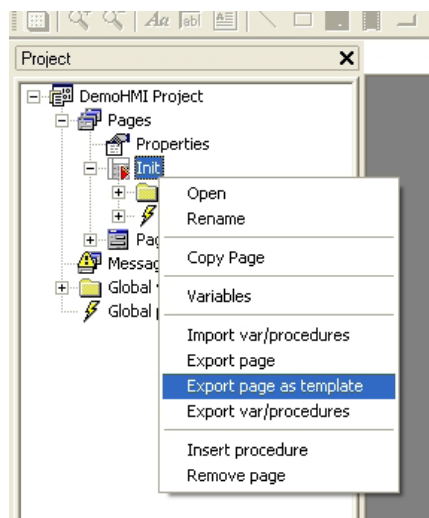
N.B.: this operation changes only the name of the page, project references to the re-named page are not automatically updated.

## 4.7.5 TEMPLATES OF PAGE MANAGEMENT

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

### 4.7.5.1 EXPORT PAGES INTO A TEMPLATE FILE

To export a page into a library of templates follow the procedure paragraph 4.7.1 initial steps then select *Export page as template* from the menu.



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

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



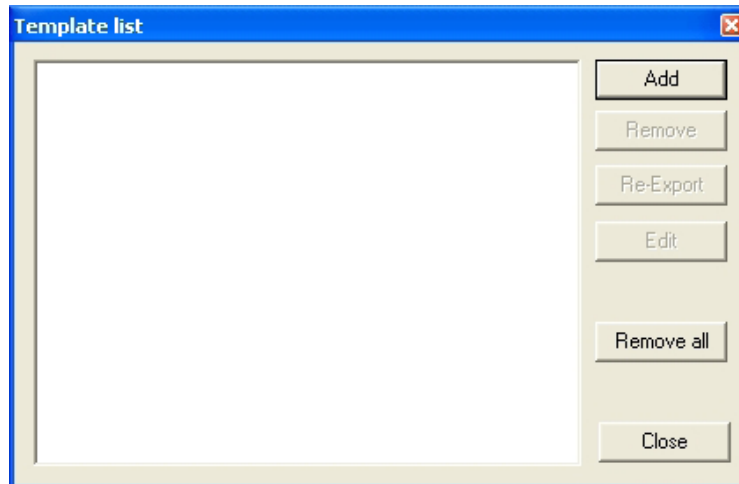
Scripts and local variables are exported without changes. References to variables contained in the scripts are not modified.

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

## 4.7.5.2 USAGE OF THE TEMPLATE LIBRARY IN A PROJECT

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

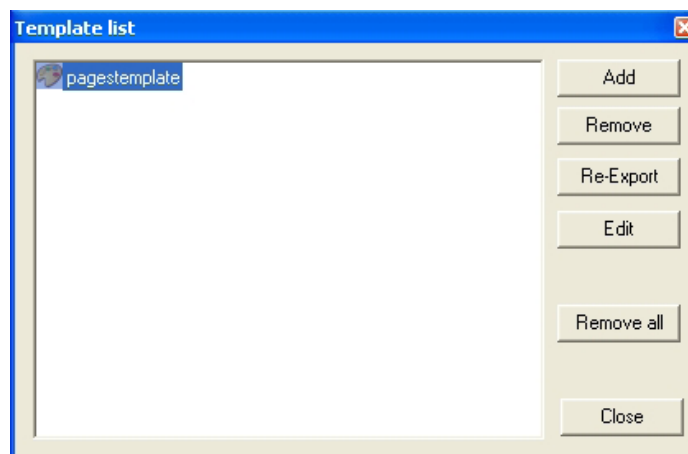
Select *Template Management* menu voice from *Project* menu. Following window will be shown:



Available operations are listed here:

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

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



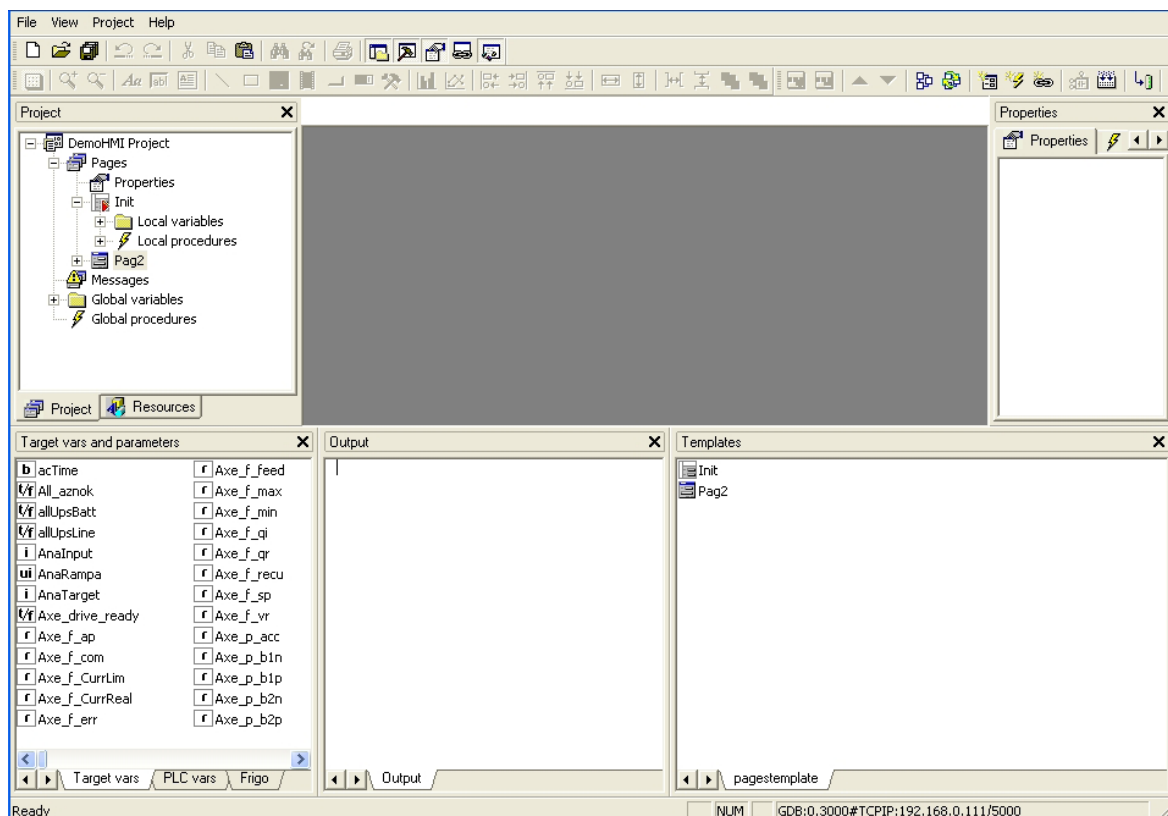
Template library has been included to the project. Press *Close* button, *Templates* window is shown: there is a tab for each library imported in current project.



Each tab shows the list of templates of the corresponding library.

### 4.7.5.3 USING A TEMPLATE

Once a template library has been added it is possible to use its elements simply dragging the chosen one from the template window and dropping it on the project tree.

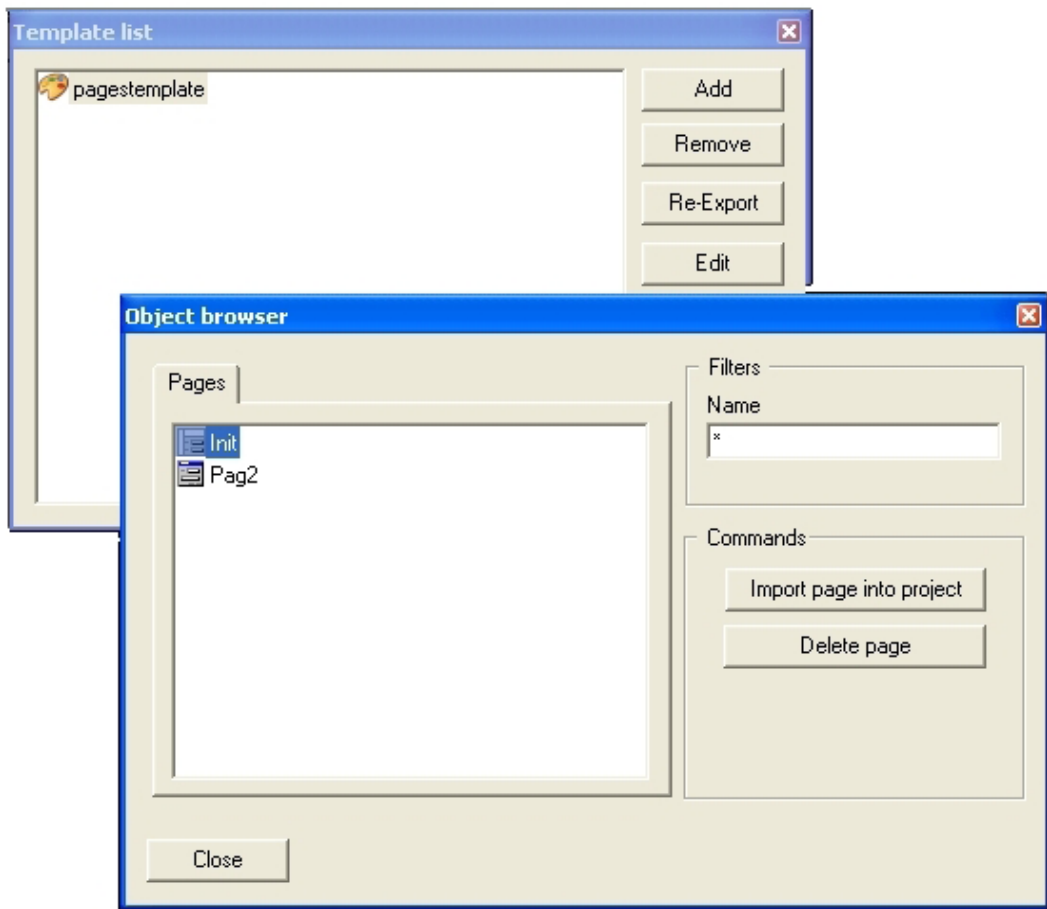


Once the item has been dropped application asks the user for the name of the new page created (based on the template).



## 4.7.5.4 PROJECT TEMPLATE UPDATE

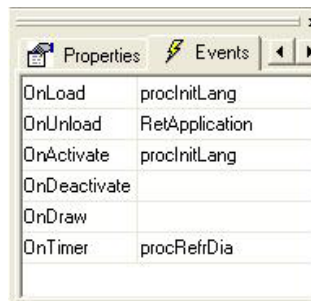
It is possible to delete templates from the (local) template library using *Edit* command in the *Template management* window.



## 4.8 EVENTS

There are different classes of events.

### 4.8.1 PAGE OR CONTROL EVENTS



Each characteristic behaviour of a specific object can raise a specific event.

Each event can be associated to a procedure (see paragraph 4.8.4) that is executed each time the event takes place. The list of all available events for each UserInterface object (page or control) is reported in Chapter 5.



### 4.8.2 KEY PRESSURE EVENTS

These events take place when a key is pressed, the raising of the event starts the execution of the associated action (see paragraph 4.8.5) if it is. The pressure of a key can be also simulated by software, see next paragraph.

### 4.8.3 EVENTS RAISED BY SOFTWARE

```

procOpenSet_7 - [Debug]
0001
0002      (* Apertura finestra impostazione con slot = 7 *)
0003
0004      debSlotNum := 7;
0005      sysVoid := Video_SendEvent( kWM_KEY, kKEY_VK_F1 );
0006
0007
0008
    
```

Programmer can raise events by software using the function *Video\_SendEvent* inside the target software or in the body of the procedure, using following syntax:

```
Video_SendEvent( event_id, param );
```

Where *event\_id* is the identifier of the type of the event and *param* is an integer 16 bit parameter.

UserInterfacesupports software events defined in this table:

Event	Parameter	Description
kWM_NULL	Do not care	No event.
kWM_KEY	Key code	Simulates the pressure of the key specified as parameter then cause the associated action if it is.
kWM_MSG	Window ID	Causes a system message that, once got by the system, causes the instant opening of the alarm page that has <i>Window ID</i> as identifier.
kWM_SELECT	Edit box handle	In touchscreen systems simulates the pressure on the edit box whose handle is passed as parameter, causing its selection or its transition to edit mode.
kWM_PUSH	Button handle	In touchscreen systems simulates the pressure on the button whose handle is passed as parameter, causing the execution of the associated action if it is.
kEV_WM_CHANGESETPAGE	Page number	Shows the page specified by the parameter (if the context is a page in which sets are used).





## 4.8.4 PROCEDURES THAT CAN BE ASSOCIATED TO EVENTS

A procedure is a program that is executed when the event that has been associated to it, takes place. Events have been deeply described in previous paragraphs (see paragraph 4.8).

There are two classes of procedures:

### Local procedures

This kind of procedures can be called only within the scope of the page in which are declared. In particular, they can be associated to the events of the page itself and of all their controls. The same can be said for software events raised when the page they refer to is active. Procedure code can contains references to all the types of variables, with local variables of the page too.

### Global procedures

This kind of procedures can be called from every page and can be also used as periodic asynchronous routine of alarm management. They cannot contain variables references.

Here follow the description of the syntax to get the properties of a control from a procedure; similarly to C language printf it is:

```
"fb%s%s.%s", page_name, ctrl_name, prop_name
```

Where:

- `page_name` is the name of the page that has the control;
- `ctrl_name` is the name of the control;
- `prop_name` is the name of the property of the control.

So if we want to get the property *Foreground color* of the *Static* named *String\_26* in *Main* page, we have to write:

```
fbMainString_26.foreCol
```

N.B.: the name of the property to use in the scripts of the procedures is the name of the functional block exported by the software of the target (see paragraph 8.2), not the name in the properties window (see paragraph 3.3).

## 4.8.5 ACTIONS THAT CAN BE ASSOCIATED TO KEY PRESSURE

In common keyboard, not touchscreen systems, interaction between the user and the system is normally based on keys pressure.

UserInterface shows the following table to the user.

Key	Action	Link
VK_F1	OpenPage	ParAxeTipo1
VK_F2	OpenPage	ParAxeTipo2
VK_F4	OpenPage	ParAxeTipo4

This table permits to associate a code of a key to one of the actions listed in the following table. In this way the pressure of that key causes the specified action.



Action	Link	Description
<i>Call</i>	Procedure name	Causes the invocation of the local or global procedure whose name is indicated in the <i>Link</i> field.
<i>OpenPage</i>	Page name	Causes the opening of the page whose name is indicated in the <i>Link</i> field.
<i>Close</i>	Do not care	Causes the closure of the current page
<i>NextField</i>	Do not care	Move the selection to the next edit box. If the system is not touchscreen moves selection to the buttons to allow their pressure.
<i>PrevField</i>	Do not care	Move the selection to the previous edit box.
<i>Edit</i>	Do not care	Access edit mode for the selected edit box. If the system is not touchscreen allows the user to simulate the pressure of the button.

There are two types of associations key-action:

- Local actions: local associations, valid only for the page currently open in the editor of the pages.
- Global actions: global associations, valid in any point of the project.

If the system has the touchscreen feature, normal interaction with user is made by the pressure of sensible area on the screen. However this table does not loss its meaning because allows the user to define virtual keys and to control their pressure by software causing in this way the dynamic execution of specific actions.

N.B.: if the same action is defined both at local and at global level, system does not give errors nor warnings because local declaration precedes global one.

## 4.9 RESOURCES

A resource is an interface element. User can get informations from resources or can use them to do actions.

UserInterface supports different categories of resources that are managed by *Tab Resources project* window. (see paragraph 3.1). Categories are explained in details in the following paragraphs.

### 4.9.1 FONTS

Fonts are the different kinds of characters supported for the output of text strings on the screen. Fonts had been managed by PageLab old versions as text files with *.plf* extension and structured with the same syntax of the initialization definition of an array variable in IEC. Now, images are saved and loaded in binary format to optimize loading time on images of big size.

At project opening time, if PageLab finds this declaration, it searches in project folder for a file named *font\_name.plk* and loads it in memory.

### 4.9.2 BITMAPS

Bitmaps are pictures to associate to image controls (see paragraph 4.4.5). Bitmaps had been managed by UserInterface old versions as text files with *.plb* extension and structured with the same syntax of the initialization definition of an array variable in IEC. Now,



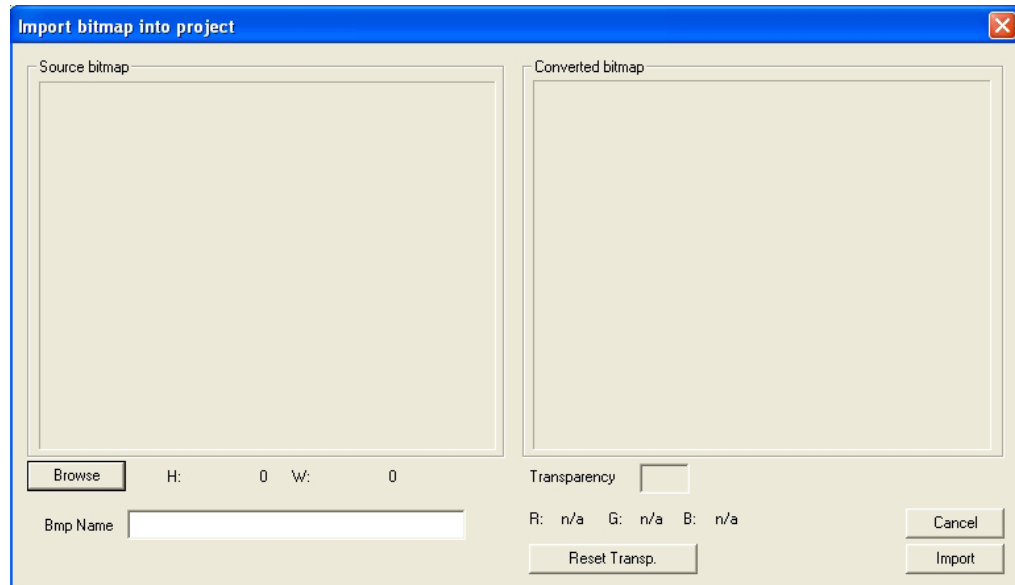
images are saved and loaded in binary format to optimize loading time on images of big size.

At project opening time, if UserInterface finds this declaration, it searches in project folder for a file named *bitmap\_name.plk* and loads it in memory.

UserInterface provides a tool to convert bitmaps from Windows format to UserInterface format.

To start this tool click on *Import resource > Bitmap* from *Project* menu, it is also possible to click on *Import bitmap* from context menu that can be shown by right click on *Bitmaps* node of resources tree.

This dialogue box will be shown as follows.



Click on *Browse* button to navigate computer resources to select desired source file.

In *Bmp Name* field user can personalize bitmap name that will be shown on resource tree; bitmap name is constituted by file name without extension and with *Bmp* prefix by default.

Transparency color field allows the user to specify transparency color, so a color that is not really drawn on the screen but a transparent color zone that does not cover elements previously drawn.

Transparency color can be personalized by choosing it by mouse from *Converted bitmap* window.

RGB indicates transparency color Red, Green, Blue components. *n/a* value indicates that no transparency color has been selected.

*Reset Transp.* button allows the user to undo last selected transparency color.

Once finished these operations it is possible to confirm bitmap importation by clicking on *Import* button.

### 4.9.3 STRINGS TABLE

In a UserInterface project it is always possible to explicitly write the text to show on a text string or on a title of the page. It is also possible to refer to one of the strings of the resources specifying its ID.

In first case text will be always the same, in second case the text that correspond to the active language will be shown.



So, English language string table contains the following record.

ID_GDB_RXNAK	Bad RX packets
--------------	----------------

And Italian language string table contains the following record.

ID_GDB_RXNAK	Pacchetti RX errati
--------------	---------------------

If we refer to the identifier *ID\_GDB\_RXNAK* from a page control or from a page, if current active language is English *Bad RX packets* will be shown, if current active language is Italian *Pacchetti RX errati* will be shown instead.

## 4.9.4 ENUMERATIVES

An enumerative is a data type defined by user, it is a set of constants named by user. Each element of an enumerative is treated as a constant and can be translated in all available languages of the project.

e.g.: we defined *ImpostazTouch* enumerative that is shown on resource tree as follows.



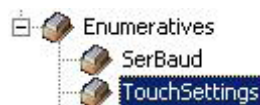
Enumerative records are shown by double clicking on *ImpostazTouch* node.

Value	Description
0	Attesa impostazione ...
1	Memorizzazione in corso ..
2	Touch screen impostato

Now we introduce an *Edit box* control (see paragraph 4.4.3) and insert the name of the enumerative *ImpostazTouch* in its *Format property* field. Control will show the string associated to the value as it is in the table above, not the numeric value of the variable associated to the control.

If the numeric value of the variable does not match with any record of the enumerative table, an error string ##### is shown instead.

Even enumerative are supported by multi-language feature. In fact it is possible to personalize the name of the enumerative.



And its record values.

Value	Description
0	Awaiting settings ...
1	Saving settings ...
2	Touch screen set up

## 4.9.5 IMAGES LISTS

An images list is very similar to an enumerative but with the following differences:

- intervals of constants are supported, not only simple values;
- each value has an image associated;
- a list of images determines the content shown by an *Animation* control, while an enumerative can be associated to an *Edit box*.



e.g.: now we have an images list *ListBulbs* that is shown on the resource tree.



It is possible to see all the records of the list by double-clicking the node.

Init Value	End Value	Bitmap
-10	0	BmpBulb5
0	5	BmpBulb6
5	10	BmpBulb7
*	*	BmpClose

If we introduce an *Animation* control (see paragraph 4.4.6) in the page, and we set its property *Image* list with the name of the enumerative *ListBulbs*, the control will show the image whose specified interval includes the value of a variable associated to the control.

If the numeric value of the associated variable does not match any record in the list a default image (with init and end value set to \*) will be shown if it is. If no default image is specified no image will be drawn.

## 4.9.6 SETS

As it is described above (see paragraph 4.6) sets are ensemble of global variable even of distinct type.

In particular there are two types of set:

- Variable/parameter sets even of not equal type (*VARIANT*);
- Strings sets (*STRINGS*).

The sets of the first type are defined indicating *VARIANT* as type. This kind of set has the following attributes:

- Dynamic: indicates that every n execution cycles target automatically reloads the elements of the set and hides those elements that have no visibility (boolean constant *FALSE* or associated visibility variable set to false at that moment).
- Array: indicates that the unique element of this set is a variable of type array.

N.B.: this kind of set can be assigned only to an edit-box control.

The screenshot shows the 'Sets' table with the following data:

	Name	Type	Dynamic	Array
1	ParametriBIOS	VARIANT	NO	NO
2	Allarmi	VARIANT	YES	NO
3	Descrizioni	STRINGS	YES	NO
4	InputOuput	VARIANT	NO	YES

The project resource tree on the left shows the following structure:

- Resources [Italiano]
  - Fonts
  - Bitmaps
  - String table
  - Enumeratives
  - Image lists
    - ParametriBIOS
    - Allarmi
    - Descrizioni
    - InputOuput
  - Sets
    - ParametriBIOS
    - Allarmi
    - Descrizioni
    - InputOuput

In this example four sets with different characteristics have been defined.

Once defined a set, each element of the set can be added via drag & drop from *Target vars and parameters* or can be manually inserted by user.



Lets see how to manage *ParametriBIOS* set.

	Variable/Parameter	Format	Text align	Min	Max	Visible	Selectable
1	@Frigo.Par_TAB	%d	Right	0	65535	TRUE	TRUE
2	@Frigo.Par_POLI	%d	Right	0	65535	TRUE	TRUE
3	@Frigo.Par_PARMOD	%d	Right	0	1	TRUE	TRUE
4	@Frigo.Gain_PT1000_AI3	%d	Right	0	65535	TRUE	TRUE
5	@Frigo.Gain_mA_AI3	%d	Right	0	65535	TRUE	TRUE

Following attributes can be defined:

- Variable/Parameter = variable/parameter name.
- Format = indicates how to show associated variable value specifying a syntax analogous to C language printf (see paragraph 5.7.2).
- Text Align = the alignment of the text to show.
- Min/Max = minimum and maximum value for the element of the set.
- Visible = boolean variable or constant that defines the visibility of the element. If dynamic feature of the set is active the variable is periodically checked to hide or show the element.
- Selectable = indicates that the element can be selected. In this case a boolean variable or constant can be assigned too.

For a set of type STRING each element of the set is quite simple as it is shown in the next figure:

	Strings	Visible
1	Param1	TRUE
2	ID_MISURA	TRUE

We have to define only two attributes, the string or the ID of a string resource (see paragraph 4.9.3) and the variable/constant of visibility. As we said an element not visible will not be shown on the screen.

N.B.: this kind of set can be used with Static control only.

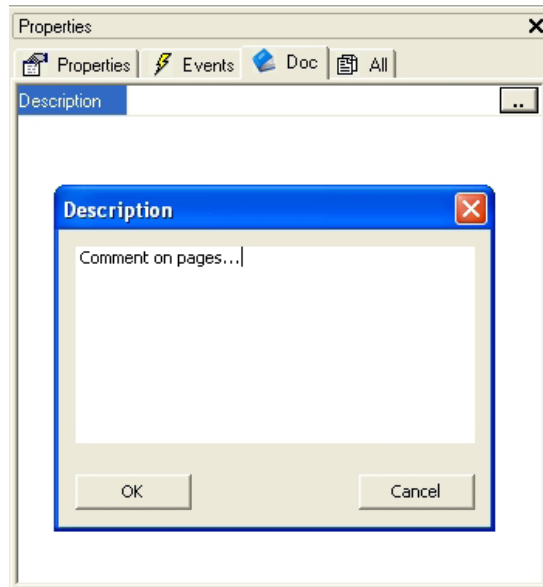
## 4.10 AUTOMATIC DOCUMENTATION

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

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



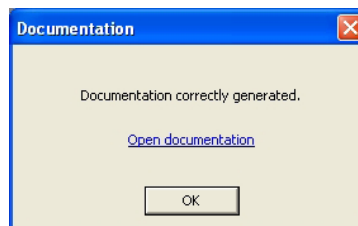
Comments related to controls and pages should be inserted in the *Doc* tab of the properties window.



Documentation is generated when the apposite button is pressed.



At the end of the process the following dialogue is shown. By clicking on the *Open documentation* link it is possible to view the generated report using the browser.



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

N.B.: documentation generation process requires the file *Documentation.xsl* to be in the project folder. This file can be personalized by user to redefine report style.

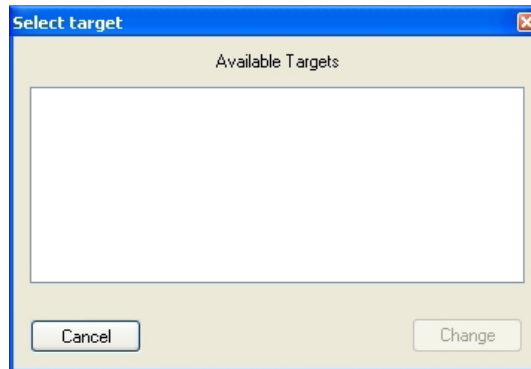
## 4.11 MANAGING PROJECTS

### 4.11.1 SELECTING THE TARGET DEVICE

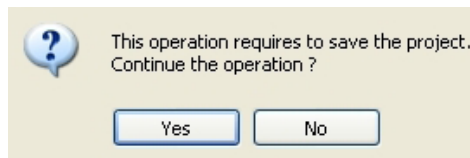
You may need to port a PLC application on a target device which differs from that you originally wrote the code for. Follow the instructions below to adapt your UserInterface project to a new target device.



- 1) Click *Select target* in the *Project* menu of the UserInterface 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, UserInterface displays the following dialog box.



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

If you press *Yes*, UserInterface 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.





## 5. APPENDIX I: PAGE PROPERTIES AND OBJECT PROPERTIES

### 5.1 FRAME SET

#### 5.1.1 PROPERTIES

Properties	Available values	Description
<i>TopDim</i>	$\geq 0$	Top-height of the frame (#pixel).
<i>BottomDim</i>	$\geq 0$	Bottom-height of the frame (#pixel).
<i>LeftDim</i>	$\geq 0$	Left-width of the frame (#pixel).
<i>RightDim</i>	$\geq 0$	Right-width of the frame (#pixel).
<i>CharDimX</i>	$> 0$	Horizontal space among grid points (#pixel).
<i>CharDimY</i>	$> 0$	Vertical space among grid points (#pixel).
<i>Font</i>	Name found in <i>Resources</i>	Default font used when inserting new objects in page.
<i>Background Color</i>	...	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.
<i>Text Color</i>	...	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
<i>Title bar</i>	<i>Yes, No</i>	Title bar, settings can be found in <i>System options</i> dialog: - <i>Yes</i> : page has title; - <i>No</i> : page has not title.
<i>Page Border</i>	<i>Yes, No</i>	- <i>Yes</i> : page with outer border; - <i>No</i> : page without outer border.
<i>Caption</i>	Text otherwise <i>Resource ID</i>	Text on title bar or <i>Resource ID</i> . This property is not sensible if <i>Title Bar</i> field is set to <i>No</i> .
<i>System menu</i>	<i>Yes, No</i>	If <i>Yes</i> denotes that there is a button with 'X' image on it and the behaviour is similar to <i>Windows Dialog</i> : - <i>Yes</i> : page has close button; - <i>No</i> : page has not close button.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- Flat - Raised - Sunken



## 5.2 CHILD PAGE

### 5.2.1 PROPERTIES

Properties	Available values	Description
<i>CharDimX</i>	> 0	Horizontal space among grid points (#pixel).
<i>CharDimY</i>	> 0	Vertical space among grid points (#pixel).
<i>Font</i>	Name found in <i>Resources</i>	Default font used when inserting new objects in page.
<i>Background Color</i>	...	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.
<i>Text Color</i>	...	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
<i>Title bar</i>	<i>Yes, No</i>	Title bar, settings can be found in <i>System options</i> dialog: - <i>Yes</i> : page has title; - <i>No</i> : page has not title.
<i>Page Border</i>	<i>Yes, No</i>	- <i>Yes</i> : page with outer border; - <i>No</i> : page without outer border.
<i>Caption</i>	Text otherwise <i>Resource ID</i>	Text on title bar or <i>Resource ID</i> . This property is not sensible if <i>Title Bar</i> field is set to <i>No</i> .
<i>System menu</i>	<i>Yes, No</i>	If <i>Yes</i> denotes that there is a button with 'X' image on it and the behaviour is similar to <i>Windows Dialog</i> : - <i>Yes</i> : page has close button; - <i>No</i> : page has not close button.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- Flat - Raised - Sunken

### 5.2.2 EVENTS

Event	Description
<i>OnLoad</i>	On loading this page, i.e. when calling from parent page.
<i>OnUnload</i>	On closing this page, when the page returns and the parent page will be restored.



Event	Description
<i>OnDeactivate</i>	On calling a child page and the current page is no more active. This event does not exist in main page.
<i>OnActivate</i>	When the previous opened child page will be closed. This event does not appear in leaf page, i.e in the pages which do not call child pages.
<i>OnDraw</i>	When the page starts drawing all the objects. The page has just drawn border, background, and title.
<i>OnTimer</i>	Asynchronous event. The user can link a procedure and it will be executed cyclically.

## 5.3 POP-UP PAGE

### 5.3.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge of full page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge of full page.
<i>XDim</i>	$> 0$	Width of the page (#pixel).
<i>YDim</i>	$> 0$	Height of the page (#pixel).
<i>CharDimX</i>	$> 0$	Horizontal space among grid points (#pixel).
<i>CharDimY</i>	$> 0$	Vertical space among grid points (#pixel).
<i>Modal</i>	<i>Yes, No</i>	- <i>Yes</i> : all the parent page objects will be disabled; - <i>No</i> : all the parent page objects will be enabled if they are completely visible.
<i>Font</i>	Name found in <i>Resources</i>	Default font used when inserting new objects in page.
<i>Background Color</i>	...	Background color selectable from palette. In addition this color is also set when inserting new objects in the frame.
<i>Text Color</i>	...	Foreground color selectable from palette. This color is set when inserting new objects in the frame.
<i>Title bar</i>	<i>Yes, No</i>	Title bar, the settings can be found in <i>System options</i> dialog: - <i>Yes</i> : page has title; - <i>No</i> : page has not title.
<i>Page Border</i>	<i>Yes, No</i>	- <i>Yes</i> : page with outer border; - <i>No</i> : page without outer border.
<i>Caption</i>	Text otherwise <i>Resource ID</i>	Text on title bar or <i>Resource ID</i> . This property is not sensible if the <i>Title Bar</i> field is set to <i>No</i> .



Properties	Available values	Description
<i>System menu</i>	<i>Yes, No</i>	If <i>Yes</i> denotes that there is a button with <i>X</i> image on it and the behaviour is similar to <i>Windows Dialog</i> : - <i>Yes</i> : page has close button; - <i>No</i> : page has not close button.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	Flat Raised Sunken

### 5.3.2 EVENTS

Event	Description
<i>OnLoad</i>	On loading this page, i.e. when calling from parent page.
<i>OnUnload</i>	On closing this page, when the page returns and the parent page will be restored.
<i>OnDeactivate</i>	On calling a child page and the current page is no more active. This event does not exist in main page.
<i>OnActivate</i>	When the previous opened child page will be closed. This event does not appear in leaf page, i.e in the pages which do not call child pages.
<i>OnDraw</i>	When the page starts drawing all the objects. The page has just drawn border, background and title.
<i>OnTimer</i>	Asynchronous event. The user can link a procedure and it will be executed cyclically.

## 5.4 STATIC

### 5.4.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>Name</i>	Not empty	Name of object.
<i>Text</i>	Text otherwise <i>Resource ID</i>	Text or <i>Resource ID</i> shown in the object.
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the text in object.
<i>Background Color</i>	...	Background color selectable from palette.
<i>Text Color</i>	...	Text color selectable from palette.
<i>Sel. Background</i>	...	Background color selectable from palette when the object is chosen. This property is not sensible if the <i>Select</i> field is constant <i>FALSE</i> .



Properties	Available values	Description
<i>Sel. Foreground</i>	...	Text color selectable from palette when the object is chosen. This property is not sensible if the <i>Select</i> field is constant <i>FALSE</i> .
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- Flat - Raised - Sunken
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Number of Chars</i>	$> 0$	Number of chars that this object can show. If the value is 0 the object will show the complete text. Otherwise with another value it can be truncated or extended.
<i>Alignment</i>	<i>Right, Center, Left</i>	Text alignment in the object.
<i>Refresh</i>	<i>TRUE, FALSE</i>	Continuous redraw of the object: - <i>FALSE</i> : the <i>Text value</i> is read from memory and updated only when opening the page or when a child page is closed; - <i>TRUE</i> : the <i>Text value</i> is read from memory and always updated.
<i>Select</i>	<i>TRUE, FALSE, var_name</i>	Selected status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is selected and so it will show the colors <i>Select Back, Select Fore</i> .
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.

## 5.4.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.

## 5.5 LINE

### 5.5.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.



Properties	Available values	Description
<i>X2Pos</i>	> 0	Bottom-right 'x coordinate' edge relative to page.
<i>Y2Pos</i>	> 0	Bottom-right 'y coordinate' edge relative to page.
<i>Name</i>	Not empty	Name of object.
<i>Thickness pts</i>	> 0	Line thickness (#pixel).
<i>Border col</i>	...	Line color selectable from palette.

## 5.6 RECTANGLE

### 5.6.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	>= 0	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	>= 0	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	> 0	Width (#pixel).
<i>YDim</i>	> 0	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Border points</i>	> 0	Border thickness (#pixel).
<i>Border color</i>	...	Border color selectable from palette.
<i>Background Color</i>	...	Background color selectable from palette. This property is sensible only if <i>Transparent</i> is set to <i>TRUE</i> .
<i>Transparent</i>	<i>TRUE, FALSE</i>	Transparency: - <i>TRUE</i> : transparent background; - <i>FALSE</i> : solid background where color is <i>Back Color</i> .

## 5.7 EDIT BOX

### 5.7.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	>= 0	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	>= 0	Top-left 'y coordinate' edge relative to page.
<i>Name</i>	Not empty	Name of object.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- <i>Flat</i> : plain with use of <i>Border pts</i> and <i>Border col</i> ; - <i>Raised</i> ; - <i>Sunken</i> .
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the text in object.



Properties	Available values	Description
<i>Background Color</i>	...	Background color selectable from palette.
<i>Text Color</i>	...	Text color selectable from palette.
<i>Sel. Background</i>	...	Background color selectable from palette when the object is chosen. This property is not sensible if the <i>Selectable</i> field is constant <i>FALSE</i> .
<i>Sel. Foreground</i>	...	Text color selectable from palette when the object is chosen. This property is not sensible if the <i>Selectable</i> field is constant <i>FALSE</i> .
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Number of Chars</i>	$> 0$	Chars visible in the object. Width of entire object is calculated among this value and the size of <i>Font</i> . If <i>NumChar</i> are less than the value, the object shows this error string: #####.
<i>Format</i>	String as <code>printf</code> or <code>.enum_name</code>	The format can be numeric, to define as <code>printf</code> of C language (see par. 5.7.2), enumerative, if in this field there is <code>enum_name</code> defined in <i>Resources</i> (see par. 4.9).
<i>Alignment</i>	<i>Right, Center, Left</i>	Text alignment in the object.
<i>Access</i>	<i>RO, RW</i>	Accesses variable <i>Assoc var</i> used in object: - <i>RO</i> = read only; - <i>RW</i> = read/write.
<i>Selection Order</i>	$\geq 0$	Selection order of the object. It can be selected by pressing a key or by means of a procedure. In this case the selection moves from the current object to the previous or next <i>Sel. Order</i> object.
<i>Variable</i>	Not empty	Name of the variable that can be shown and edited with this object. It can be any variable of the project, (local, global, imported from PLC or target - see par. 2.9.2), a parameter (see par. 2.9.2) or an element of a set (see par. 4.6).
<i>Data type</i>	<i>UNDEF, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, STRING</i>	Type of <i>Assoc var</i> . If it is a variable, the type is defined automatically. This property is sensible if <i>Assoc var</i> is an explicit parameter.



Properties	Available values	Description
<i>Low limit</i>	<i>CONSTANT, var_name</i>	Name of variable or numeric constant. This is the least number that the object can show. It can be any variable of the project, (local, global, imported from PLC or target - see par. 2.9.2). This object shows an error string (!!!!!!!) if condition does not holds. The * symbol means that there is no low limit.
<i>High limit</i>	<i>CONSTANT, var_name</i>	Name of the variable or numeric constant. This is the maximum number that the object can show. It can be any variable of the project, (local, global, imported from PLC or target - see par. 2.9.2). This object views an error string (!!!!!!!) if condition does not hold. The * symbol means that there is no high limit.
<i>Refresh</i>	<i>TRUE, FALSE</i>	Enables continuous update of the value: - <i>FALSE</i> : the <i>Assoc var</i> value is read from memory and updated only when open page or when a child page is closed; - <i>TRUE</i> : the <i>Assoc var</i> value is read from memory and always updated.
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise hidden.
<i>Selectable</i>	<i>TRUE, FALSE, var_name</i>	Selected status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is selected and so it will show the colors <i>Select Back, Select Fore</i> . If this field is <i>FALSE</i> the <i>Access</i> property is not sensible.

**5.7.2 FORMAT SPECIFICATION - PRINTF**

If the object has not any enumerative format, the format string is composed as follows:

`%[flags][width][.precision]type`

The field has one or more characters, that describe the specification. The simplest format contains only percentage symbol and one char as type (for example: `%s`).

Next table explains in details functions and values.

Field	Available values	Description
<i>flags</i>	- + prints always the sign, even if the number is positive. - 0 prints zeros in head until <i>width</i> (if specified) or <i>NumChar</i> .	This char is an option for chars order, print sign, number of decimal digit. This field may have more than one flag.





Field	Available values	Description
<i>width</i>	$> 0, \leq NumChar$	Maximum chars can be printed. Allows to view values that do not fill <i>NumChar</i> fully.
<i>precision</i>	$\geq 0$	Decimal digits after the point. If the field is an integer and there is a precision the object shows a decimal point. E.g. the value is 102 integer, and precision is 2, with $\%.2d$ , the number is shown as 1.02.
<i>type</i>	<ul style="list-style-type: none"><li>- %d: Integer with sign.</li><li>- %f: Real.</li><li>- %x: Hexadecimal with lowercase chars.</li><li>- %X: Hexadecimal with uppercase chars.</li><li>- %s: String.</li><li>- %@sdf: Password.</li><li>- [%d,u,f,x]: Custom measure unit format.</li></ul>	Mandatory field.

## 5.7.3 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.
<i>OnGotFocus</i>	Whenever object is selected.
<i>OnLostFocus</i>	Whenever object loses the selection.
<i>OnEnter</i>	Whenever the object is selected and receives the command for entering in edit-mode.
<i>OnClick</i>	Whenever HMI receives a pressure on the object, valid only for touchscreen systems.
<i>OnChange</i>	Whenever the user confirms the modifications and the value is different from start.

## 5.8 TEXT BOX

### 5.8.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>Name</i>	Not empty	Name of object.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	<ul style="list-style-type: none"><li>- <i>Flat</i>: plain with use of <i>Border pts</i> and <i>Border col</i>;</li><li>- <i>Raised</i>;</li><li>- <i>Sunken</i>.</li></ul>



Properties	Available values	Description
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the text in the object.
<i>Background Color</i>	...	Background color selectable from palette.
<i>Text Color</i>	...	Text color selectable from palette.
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Number of Chars</i>	$> 0$	Chars visible in the object. Width of entire object is calculated among this value and the size of <i>Font</i> .
<i>Number of Rows</i>	$> 0$	Rows visible in the object. Height of entire object is calculated among this value and the size of <i>Font</i> .
<i>Show line number</i>	<i>TRUE, FALSE</i>	Flag for viewing number of lines.
<i>Access</i>	<i>RO, RW</i>	Access on variable <i>Assoc string</i> used in object: - <i>RO</i> : read only; - <i>RW</i> : read/write.
<i>Selection order</i>	$\geq 0$	Selection order on which the object can be selected with the pressure of a key or with a procedure. In this case the selection moves from the current object to the previous or next <i>Sel.Order object</i> .
<i>String variable</i>	<i>Not empty</i>	Name of variable that can be shown and edited with this object. It can be any string variable of the project, (local, global, imported from PLC or target - see par. 2.9.2).
<i>Refresh trg</i>	<i>TRUE, FALSE, var_name</i>	Enables update of the value: - <i>FALSE</i> : the <i>Assoc string</i> value is read from memory and updated only when opening page or when a child page is closed. - <i>TRUE</i> : the <i>Assoc string</i> value is read from memory and always updated. The runtime sets automatically the value to <i>FALSE</i> .
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise hidden.



## 5.8.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.
<i>OnClick</i>	Whenever HMI receives a pressure on the object, valid only for touchscreen systems.
<i>OnChange</i>	Whenever the user confirms the modifications and the value is different from start.

## 5.9 IMAGE

### 5.9.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	const $\geq 0$ , variable	Top-left 'x coordinate' edge relative to page. It is possible to assign a variable only if <i>Style</i> is set to <i>Floating</i> .
<i>YPos</i>	const $\geq 0$ , variable	Top-left 'y coordinate' edge relative to page. It is possible to assign a variable only if <i>Style</i> is set to <i>Floating</i> .
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Appearance</i>	<i>Flat</i> , <i>Raised</i> , <i>Sunken</i>	- <i>Flat</i> = plain with use of <i>Border pts</i> and <i>Border col</i> ; - <i>Raised</i> ; - <i>Sunken</i> .
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Bitmap</i>	Name found in <i>Resources</i>	Bitmap used for drawing the image in object.
<i>Background image</i>	Image object in the page	Name of another object that is redrawn when <i>Style</i> is set to <i>Floating</i> . It is sensible only if it is overlapped with this image.
<i>Visible</i>	<i>TRUE</i> , <i>FALSE</i> , <i>var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Style</i>	<i>Docking</i> , <i>Floating</i>	- <i>Docking</i> : fixed position; - <i>Floating</i> : variable position, according to <i>XPos</i> variable and <i>Ypos</i> variable.



## 5.10 ANIMATION

### 5.10.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- <i>Flat</i> = plain with use of <i>Border pts</i> and <i>Border col</i> ; - <i>Raised</i> ; - <i>Sunken</i> .
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Image list</i>	Name found in <i>Resources</i>	It contains the images that the object can view and the value range.
<i>Animation variable</i>	<i>var_name</i>	Name of the variable that is compared with value range in <i>Image list</i> .
<i>Data type</i>	<i>SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD</i>	Type of <i>Animation var</i> . If it is a variable, the type is automatically defined.
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise hidden.

### 5.10.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.



## 5.11 BUTTON

### 5.11.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Text/Img</i>	Empty or explicit text or Resource ID or Bitmap	Text or image to view in the button: - string; - Resource ID; - bitmap.
<i>Selection Text/Img</i>	Empty or explicit text or Resource ID or Bitmap	Text or image to view in the button when it is selected: - string; - Resource ID; - bitmap.
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the text in object. This field is not sensible if it shows a bitmap.
<i>Appearance</i>	<i>Flat</i> , <i>Raised</i> , <i>Sunken</i>	- <i>Flat</i> : plain with use of <i>Border pts</i> and <i>Border col</i> ; - <i>Raised</i> ; - <i>Sunken</i> .
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> or <i>Text</i> is not empty.
<i>Background color</i>	...	Background color selectable from palette. This property is sensible only if <i>Transparent</i> is set to <i>TRUE</i> .
<i>Selection border</i>	...	Border color when the object is selected. This property is not sensible if <i>Selection var</i> is <i>FALSE</i> fixed.
<i>Sel. background</i>	...	Background color when the object is selected. This property is not sensible if <i>Selection var</i> is <i>FALSE</i> fixed.



Properties	Available values	Description
<i>Selection order</i>	$\geq 0$	Selection order on which the object can be selected with the pressure of a key or with a procedure. In this case the selection moves from the current object to the previous or next <i>Sel. Order</i> object.
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Transparent</i>	<i>TRUE, FALSE, var_name</i>	Transparency. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is transparent.
<i>Press variable</i>	Empty or <i>var_name</i>	When the button is pressed <i>var_name</i> is set to <i>TRUE</i> . When the button is not pressed, <i>var_name</i> is set to <i>FALSE</i> .
<i>Selection variable</i>	<i>TRUE, FALSE, var_name</i>	Selected status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is selected and so it will show the colors <i>Select Back, SelectBord</i> . If this field is <i>FALSE</i> <i>SelectBord</i> and <i>Select Back</i> properties are not sensible.
<i>Action</i>	<i>Call, OpenPage, Close, NextField, PrevField, Edit</i>	Action executed on button pressure.
<i>Action par</i>	<i>page_name, proc_name</i>	Parameter associated with the action executed on button pressure. It is sensible only if <i>Action</i> is <i>OpenPage</i> ( <i>Action par</i> = name of the page to open) or <i>Call</i> ( <i>Action par</i> = name of the procedure to execute ).
<i>Alignment</i>	<i>Right, Center, Left</i>	Text alignment in the object.

## 5.11.2 EVENTS

Event	Description
<i>OnClick</i>	Whenever HMI receives a pressure on the object, valid only for touchscreen systems.
<i>OnRelease</i>	Whenever HMI releases the pressure on the object, valid only for touchscreen systems.



## 5.12 PROGRESS BAR

### 5.12.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	- <i>Flat</i> : plain with use of <i>Border pts</i> and <i>Border col</i> ; - <i>Raised</i> ; - <i>Sunken</i> .
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> or <i>Text</i> is not empty.
<i>Bar color</i>	...	Color of step bar, selectable from palette.
<i>Background color</i>	...	Background color selectable from palette.
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Refresh trigger</i>	<i>TRUE, FALSE, var_name</i>	Object redraw: - <i>FALSE</i> : the <i>Progress var</i> value is read from memory and updated only when opening page or when a child page is closed. - <i>TRUE</i> : the <i>Progress var</i> value is read from memory and always updated. - <i>var_name</i> : the <i>Progress var</i> value is read from memory and updated only when the variable becomes <i>TRUE</i> . After the update the runtime sets it to <i>FALSE</i> .
<i>Progress variable</i>	Not empty	Step variable. This is the filling percentage of bar in relation with the range assigned by <i>Lo limit</i> and <i>Hi limit</i> . It can be any string variable of the project (local, global, imported from PLC or target) or a parameter (see 2.9.2).
<i>Data type</i>	UNDEF, BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LWORD, REAL, LREAL, STRING	Type of <i>Progress var</i> . If it is a variable, the type is automatically defined. This property is sensible if <i>Progress var</i> is an explicit parameter.



Properties	Available values	Description
<i>Low limit</i>	Constant or <i>var_name</i>	Name of the variable or numeric constant. This is the least value for step bar. It can be any variable of the project, (local, global, imported from PLC or target) with type specified by <i>Data type</i> .
<i>High limit</i>	Constant or <i>var_name</i>	Name of the variable or numeric constant. This is the maximum value for step bar. It can be any variable of the project, (local, global, imported from PLC or target) with type specified by <i>Data type</i> .
<i>Orientation</i>	Horizontal, vertical	Direction of step bar.

## 5.12.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.

## 5.13 CUSTOM CONTROL

### 5.13.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Control ID</i>	$> 0$	Identifier of custom control type.
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Refresh</i>	<i>TRUE, FALSE</i>	Continuous redraw of the object: - <i>FALSE</i> : the body of the runtime object is updated only when opening page or when a child page is closed. - <i>TRUE</i> : the body of the runtime object is always updated.





## 5.13.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.

## 5.14 CHART

### 5.14.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.
<i>Track 1</i>	<i>var_name</i>	Opens a dialog with the following options: <ul style="list-style-type: none"><li>- the array with data of track (<i>Data Source</i>);</li><li>- the visibility condition of the track (<i>TRUE</i> or boolean variable);</li><li>- <i>Color</i> of the track;</li><li>- the scale factor (range among two horizontal divisions);</li><li>- the offset (displacement of the track 0-Y);</li><li>- step of print label for Y axis;</li><li>- three horizontal bars with name, value and colors.</li></ul> If <i>var_name</i> is empty the track is not defined and not drawn.
<i>Track 2</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 2.
<i>Track 3</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 3.
<i>Track 4</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 4.
<i>Track 5</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 5.
<i>Track 6</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 6.
<i>Track 7</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 7.
<i>Track 8</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 8.
<i>Track Left</i>	$\geq 0$ , <i>var_name</i>	Integer value that is the track for Y axis left. It is admitted a constant value (ex 1). If empty means that the chart must not draw the label for Y axis left.
<i>Track Right</i>	$\geq 0$ , <i>var_name</i>	As <i>Track Left</i> for the right side.
<i>Format Left</i>	String as <i>printf</i>	Format of Y axis left as <i>c printf</i> function.
<i>Format Right</i>	String as <i>printf</i>	As <i>Format Left</i> for the right side.
<i>XLabel</i>	$\geq 0$ , <i>var_name</i>	Step for X-axis labels. How many divisions of horizontal bar must have labels.



Properties	Available values	Description
<i>X Scale</i>	<i>var_name</i>	Scale factor of x-Axis. Value range among two divisions of horizontal bars. An empty value indicates that the chart is in autoscale mode.
<i>Len Data</i>	<i>var_name</i>	Name of the variable that contains the array index samples. The chart adds the sample values when the <i>Refresh</i> is <i>TRUE</i> . If the value remains unchanged the chart does not add new values. The runtime maintains the last value of this field. Constant values are not allowed.
<i>X Offset</i>	<i>var_name</i>	Variable name for the deviation of 0 for x-Axis, left or right. A positive value moves the chart values to left.
<i>Grid</i>	<i>Yes, No</i>	Visibility of the grid.
<i>X Div. Grid</i>	value	Number of division on horizontal bar, used with scale factor and offset for drawing the chart tracks.
<i>Y Div. Grid</i>	value	Number of division on vertical bar, used with scale factor and offset for drawing the chart tracks.
<i>Background color</i>	...	Background color selectable from palette.
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	<ul style="list-style-type: none"> <li>- <i>Flat</i>: plain with use of <i>Border pts</i> and <i>Border col</i>;</li> <li>- <i>Raised</i>;</li> <li>- <i>Sunken</i>.</li> </ul>
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> or <i>Text</i> is not empty.
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the label in object.
<i>Refresh</i>	<i>TRUE, FALSE, var_name</i>	<p>Continuous redraw of the object:</p> <ul style="list-style-type: none"> <li>- <i>FALSE</i>: the chart is updated only when opening page or when a child page is closed;</li> <li>- <i>TRUE</i>: the chart object is always updated, synchronized with others objects;</li> <li>- <i>var_name</i>: the chart is drawn on rising edge of boolean variable. This value <i>TRUE</i> of this variable is the moment when the char adds the samples.</li> </ul> <p>(Len Data &lt;&gt; internal HMI index of data).</p>
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Format X</i>	String as <i>printf</i>	Format of X axis as <i>c printf</i> function.



Properties	Available values	Description
<i>X Data</i>	$\geq 0, var\_name$	X-Axis array values. If there is a constant value in this property each Y sample has a X value equal to the product among X Data and the index in array. Ex. $Y = track[3] = 20 \ X = 3 * X \ Data$
<i>X Color</i>	...	Color of X-Axis label.
<i>Grid Step</i>	Value	Space among two points of grid in pixel. The property is sensible if the grid is visible.
<i>Sample Buffer</i>	Value	Number of samples that the runtime can store. The older are deleted if the size has exceeded.
<i>Grid Color</i>	...	Color of grid if it is visible.
<i>Bord. Color</i>	...	Color of border grid.
<i>Vertical Bar 1</i>	$\geq 0, var\_name$	Name of variable for drawing a vertical fixed bar on chart. It is allowed also a constant value. The * symbol means that there is not this vertical bar.
<i>Color bar 1</i>	...	Color of vertical bar 1 if different from *.
<i>Vertical Bar 2</i>		As <i>Vert. Bar 1</i> , but relative to bar 2.
<i>Color bar 2</i>		As <i>Color bar 1</i> , but relative to bar 2.
<i>Vertical Bar 3</i>		As <i>Vert. Bar 1</i> , but relative to bar 3.
<i>Color bar 3</i>		As <i>Color bar 1</i> , but relative to bar 3.
<i>Clear Data</i>	<i>var_name</i>	Boolean variable. If it is <i>TRUE</i> and <i>Refresh</i> is <i>TRUE</i> the chart deletes all the previous data.

## 5.14.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.

## 5.15 TREND

### 5.15.1 PROPERTIES

Properties	Available values	Description
<i>XPos</i>	$\geq 0$	Top-left 'x coordinate' edge relative to page.
<i>YPos</i>	$\geq 0$	Top-left 'y coordinate' edge relative to page.
<i>XDim</i>	$> 0$	Width (#pixel).
<i>YDim</i>	$> 0$	Height (#pixel).
<i>Name</i>	Not empty	Name of object.



Properties	Available values	Description
<i>Track 1</i>	<i>var_name</i>	Open a dialog with the following options: <ul style="list-style-type: none"> <li>- the variable will be sampled each <i>Sampling Time</i> seconds;</li> <li>- the visibility condition of the track (<i>TRUE</i> or boolean variable);</li> <li>- color of the track;</li> <li>- the scale factor (range among two horizontal divisions);</li> <li>- the offset (displacement of the track 0-Y);</li> <li>- step of print label for Y axis;</li> <li>- three horizontal bars with name, value and colors.</li> </ul> If <i>var_name</i> is empty the track is not defined and not drawn.
<i>Track 2</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 2.
<i>Track 3</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 3.
<i>Track 4</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 4.
<i>Track 5</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 5.
<i>Track 6</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 6.
<i>Track 7</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 7.
<i>Track 8</i>	<i>var_name</i>	As <i>Track 1</i> , but for track 8.
<i>Track Left</i>	$\geq 0$ , <i>var_name</i>	Integer value that is the track for Y axis left. It is admitted a constant value (ex. 1). If empty means that the chart must not draw the label for Y axis left.
<i>Track Right</i>	$\geq 0$ , <i>var_name</i>	As <i>Track Left</i> for the right side.
<i>Format Left</i>	String as <i>printf</i>	Format of Y axis left as <i>c printf</i> function.
<i>Format Right</i>	String as <i>printf</i>	As <i>Format Left</i> for the right side.
<i>XLabel</i>	$\geq 0$ , <i>var_name</i>	Step for X-axis labels. How many divisions of horizontal bar must have labels.
<i>X Scale</i>	<i>var_name</i>	Scale factor of x-Axis. Value range among two divisions of horizontal bars. An empty value indicates that the chart is in autoscale mode.
<i>Sampling Time</i>	$> 0$	Sampling time measured in seconds. Every <i>Sampling Time</i> seconds the trend sample the value even if the chart is not shown.
<i>X Offset</i>	<i>var_name</i>	Variable name for the deviation of 0 for x-Axis, left or right. A positive value moves the chart values to left.
<i>Grid</i>	<i>Yes, No</i>	Visibility of the grid.
<i>X Div. Grid</i>	Value	Number of division on horizontal bar, used with scale factor and offset for drawing the chart tracks.
<i>Y Div. Grid</i>	Value	Number of division on vertical bar, used with scale factor and offset for drawing the chart tracks.
<i>Background color</i>	...	Background color selectable from palette.



Properties	Available values	Description
<i>Appearance</i>	<i>Flat, Raised, Sunken</i>	<ul style="list-style-type: none"> <li>- <i>Flat</i>: plain with use of <i>Border pts</i> and <i>Border col</i>;</li> <li>- <i>Raised</i>;</li> <li>- <i>Sunken</i>.</li> </ul>
<i>Border points</i>	$\geq 0$	Border thickness (#pixel). This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> .
<i>Border color</i>	...	Border color selectable from palette. This property is sensible only if <i>Appearance</i> is set to <i>Flat</i> or <i>Text</i> is not empty.
<i>Font</i>	Name found in <i>Resources</i>	Font used for drawing the label in object.
<i>Refresh</i>	<i>TRUE, FALSE, var_name</i>	Continuous redraw of the object: <ul style="list-style-type: none"> <li>- <i>FALSE</i>: the chart is updated only when opening page or when a child page is closed;</li> <li>- <i>TRUE</i>: the chart object is always updated, synchronized with others objects;</li> <li>- <i>var_name</i>: the chart is drawn on rising edge of boolean variable. This value <i>TRUE</i> of this variable is the moment when the char adds the samples.</li> </ul> (Len Data <> internal HMI index of data)
<i>Visible</i>	<i>TRUE, FALSE, var_name</i>	Visible status of the object. It can be constant ( <i>TRUE</i> or <i>FALSE</i> ) or linked with a boolean variable <i>var_name</i> : if <i>var_name</i> is <i>TRUE</i> the object is visible, otherwise it is hidden.
<i>Format Time</i>	<i>Default list</i>	Format for X-axis label. The choices are: <ul style="list-style-type: none"> <li>- <i>ss</i>: seconds;</li> <li>- <i>mm.ss</i>: minutes, seconds;</li> <li>- <i>hh.mm</i>: hours, minutes;</li> <li>- <i>hh.mm.ss</i>: hours, minutes, seconds.</li> </ul>
<i>X Color</i>	...	Color of X-Axis label.
<i>Grid Step</i>	Value	Space among two points of grid in pixel. The property is sensible if the grid is visible.
<i>Sample buffer</i>	Value	Number of samples that the runtime can store. The older are deleted if the size has exceeded.
<i>Grid Color</i>	...	Color of grid if it is visible.
<i>Bord. Color</i>	...	Color of border grid.
<i>Vertical Bar 1</i>	$\geq 0, var\_name$	Name of variable for drawing a vertical fixed bar on chart. It is allowed also a constant value. The * symbol means that there is not this vertical bar.
<i>Color bar 1</i>	...	Color of vertical bar 1 if different from *.
<i>Vertical Bar 2</i>		As <i>Vert. Bar 1</i> , but relative to bar 2.
<i>Color bar 2</i>		As <i>Color bar 1</i> , but relative to bar 2.



Properties	Available values	Description
<i>Vertical Bar 3</i>		As <i>Vert. Bar 1</i> , but relative to bar 3.
<i>Color bar 3</i>		As <i>Color bar 1</i> , but relative to bar 3.
<i>Clear Data</i>	<i>var_name</i>	Boolean variable. If it is <i>TRUE</i> and <i>Refresh</i> is <i>TRUE</i> the chart deletes all the previous data.

### 5.15.2 EVENTS

Event	Description
<i>BeforeUpdate</i>	Before the object is redrawn.
<i>AfterUpdate</i>	Immediately after the object is redrawn.



## 6. APPENDIX II: FILE FOR TARGET DESCRIPTION

The *.def* files contain some definitions of target environment. UserInterface uses this information for generating custom code.

The *.def* file consists of two sections. It is allowed comment, that starts with a semicolon(;).

This file is included in *pajx* file.

### 6.1 TARGET PROPERTIES

#### 6.1.1 DESCRIPTION

This section consist of five records, which support one or more parameters. Each record is on new line and the elements must be separated with spaces or tabs.

Record Structure			
Header	Param. 1	Param. 2	Description
<i>SCREEN</i>	<i>dimX</i>	<i>dimY</i>	Screen dimension of target measured in pixel: - <i>DimX</i> : width; - <i>DimY</i> : height.
<i>SAVESCREEN</i>	0/1	---	Target board can save and restore video memory: - 0: no save; - 1: save and restore.
<i>TOUCHSCREEN</i>	0/1	---	Target board has touchscreen, i.e. can use the pressure events: - 0: no touchscreen; - 1: exists touchscreen.
<i>REFRESH</i>	msec	---	Refresh time of all objects in page, measured in milliseconds.
<i>FONT_FORMAT</i>	"HH"/"VH"	---	Font encoding.
<i>ColorSET</i>	"RGB"	---	
<i>BMP_FORMAT</i>	"SIMULAB"	---	Image encoding.
<i>UNICODE</i>	0/1	---	Target board has support for unicode fonts.
<i>JOYPAD</i>	0/1	---	Target board has a joypad that can be used for moving among elements of page and can be connected to actions.
<i>INIT</i>	0/1	---	If set to 1 says that HMI run-time has <i>Video_InitHMI()</i> , invoked on target start-up. Typically it is used for custom commands on start-up.
<i>BMPFULL</i>	0/1	---	If set to 1 generates PLC code extended for bitmap instead binary bitmap.



## 6.2 OBJECT VERSION

The graphical objects (editbox, textbox, static, bitmap, etc.) can have a version, or cannot exist. The syntax is:

```
CTRL "Name" "Version"
```

where:

- *Name*: name of graphical object. Ex. *Editbox*;
- *Version*: version of HMI run-time objects.

If this value is set to -1, UserInterface does not make available this object.

## 6.3 SYSTEM ENUMERATIVES

Enumeratives of *.def* file are maps for binding among numeric values and strings, or other numeric values.

Each enumerative has an identifier, that specifies a function in the map with this syntax:

```
ENUM id en_key en_val
```

where:

- *id*: enumerative identifier;
- *en\_key*: value-key of record, must be a number;
- *en\_val*: value of value-key, can be a number or string.

### 6.3.1 DESCRIPTIONS

This paragraph describes the values for system enumeratives.

- Enumerative 100

With this key you can define new buttons (the names will be shown in the *Key* field of actions table (see paragraph 4.8.5).

The number of lines is not limited. But the user must define at least all the elements of 102 enumerative.

- *id*: 100;
- *en\_key*: key encoding, one byte;
- *en\_val*: string with key name.

- Enumerative 101

With this key you can define new actions (the names will be shown in *Action* field of actions table).

- *id*: 101;
- *en\_key*: action identifier;
- *en\_val*: string with action name.





This enumerative has a well defined number of lines. The following table shows you the corresponding actions.

en_key	Action
0	Calls local or global procedures.
1	Opens child page.
2	Closes current page.
3	Selects next object <i>Edit Box, Button, etc..</i>
4	Selects previous object <i>Edit Box, Button, etc..</i>
9	Enters editing-mode ( <i>Edit Box, Button</i> ).
10	Leaving (not implemented).

The string `en_val` is arbitrary.

- Enumerative 102

Selection and edit functions:

- `id: 102;`
- `en_key`: identifier of edit function;
- `en_val`: string with the name of associated string.

The name of this field `en_val` must be the same of `en_val` of 100 enumerative, so that UserInterface associates an edit function with a key.

This enumerative has a well defined number of lines. See the actions in the table below:

en_key	Edit function
0	Confirms modifications and leaves editing-mode.
1	Loses modification and leaves editing-mode.
2	Deletes selected character.
3	Moves cursor left.
4	Moves cursor right.
5	Selects the previous element of an enumerative associated with an Editbox.
6	Selects the next element of an enumerative associated with an Editbox.
7	Deletes the first character on the left.
8	Inserts tab character.
9	Switching to uppercase alphanumeric characters for a single character.
10	Transition to permanent uppercase alphanumeric characters.

- Enumerative 103

Define a color palette, the encoding is RGB:

- `id: 103;`
- `en_key`: index of the color inside palette;
- `en_val`: RGB color encoding.

RGB encoding represents 24 bit of colors: `0x00bbggrr` where `bb` (1 byte) intensity of blue, `gg` (1 byte) the green and `rr` (1 byte) the red. The intensity is at least 0 and at most `0xff`.

The number of lines is not limited. The user can define which colors he wants.



- Enumerative 104

Names of object styles (shown on *Appearance* property):

- *id*: 104;
- *en\_key*: style;
- *en\_val*: string with the name of style.

This enumerative contains at most 3 records, supported by UserInterface.

en_key	Style
0	Flat, plane.
1	Raised.
2	Sunken.

### 6.3.2 EXAMPLE

```

;
;   Target properties
;
SCREEN      128   64
SAVESCREEN 1
REFRESH    50
FONT_FORMAT    "VH"
JOYPAD    1
INIT      1
BMPFULL   1
UNICODE   1
;
;   Versions of controls
;
CTRL"Static"      1
CTRL"EditBox"     1
CTRL"TextBox"     -1
CTRL"Button"      2
CTRL"Progress"    0
CTRL"Animation"   0
CTRL"Image"       0
CTRL"CustomCtrl"-1
CTRL"Chart"       -1
CTRL"Trend"       -1

;
;   Enumeratives
;
;   ENUM 100: key codes

```



```
;
ENUM100  13  "Enter"
ENUM100  8   "Left"
ENUM100  12  "Right"
ENUM100  11  "Up"
ENUM100  10  "Down"
ENUM100  19  "LongEnter"
ENUM100  15  "LongLeft"
ENUM100  16  "LongRight"
ENUM100  17  "LongUp"
ENUM100  18  "LongDown"
ENUM100  30  "VK_F1"
ENUM100  31  "VK_F2"
ENUM100  32  "VK_F3"
ENUM100  33  "VK_F4"
ENUM100  34  "VK_F5"
ENUM100  35  "VK_F6"
ENUM100  36  "VK_F7"
ENUM100  37  "VK_F8"
ENUM100  38  "VK_F9"
ENUM100  39  "VK_F10"
;
; ENUM 101: key-related actions
;
ENUM101  0   "Call"
ENUM101  1   "OpenPage"
ENUM101  2   "Close"
ENUM101  3   "NextField"
ENUM101  4   "PrevField"
ENUM101  9   "Edit"
;
; ENUM 102: editing-mode keys
;
ENUM102  0   "Enter"
ENUM102  1   "LongLeft"
ENUM102  3   "Left"
ENUM102  4   "Right"
ENUM102  5   "Up"
ENUM102  6   "Down"
;
; ENUM 103: color codes
;
ENUM103  0   "0x00000000" ; Bianco
ENUM103  1   "0x00FFFFFF" ; Nero
```



```
;  
; ENUM 104: controls appearance  
;  
ENUM104 0 "Flat"  
ENUM104 1 "Raised"  
ENUM104 2 "Sunken"
```



## 7. APPENDIX III: DESCRIPTION OF PARAMETER FILE

As described in section 2.8.4 it is possible to link in UserInterface some variables from external device.

In some objects you can define an explicit or implicit syntax in order to use the parameter mode.

To use the implicit syntax, @Device.Parametro, UserInterface requires a .PARX file in xml format.

For example:

```
<parameters>
<par ipa="10100" name="Par_TAB" descr="Tab (map code)" defval="0" min="0"
max="65535" um="num" typetarg="unsignedShort">
<protocol name="Modbus" commaddr="15716" commsubindex="0"/>
<protocol name="CanOpen" commaddr="15716" commsubindex="0"/>
</par>
<par ipa="10001" name="Gain_Ntc_AI2" descr="NTC calibration gain AI2" de
fval="32768" min="0" max="65535" um="num" typetarg="unsignedShort">
<protocol name="Modbus" commaddr="15617" commsubindex="0"/>
<protocol name="CanOpen" commaddr="15617" commsubindex="0"/>
</par>
<par ipa="11308" readonly="false" name="Modem_InitStr1" defval="" descr="Init
String (1st part)" typetarg="string" strsize="19">
<protocol name="Modbus" commaddr="15821" commsubindex="0"/>
<protocol name="CanOpen" commaddr="15821" commsubindex="0"/>
</par>
</parameters>
```

Where each parameter has these fields.

- *ipa*: parameter index used as input value of *Video\_SetParam()*, *Video\_GetParam()*. If there are nodes with protocol type, they have more priority than *ipa*, so UserInterface uses them.
- *Name*: parameter name.
- *descr*: complete description of parameter.
- *defval*: default value of parameter.
- *min*: minimum value of parameter.
- *max*: maximum value of parameter.
- *um*: measure unit of parameter.
- *typetarg*: type of parameter.

The available values with the translation in PLC are:

- *char*: SINT;
- *unsignedChar*: USINT;
- *short*: INT;
- *unsignedShort*: UINT;
- *int*: DINT;
- *unsignedInt*: UDINT;
- *boolean*: BOOL;



- *digitalInput*: BOOL;
  - *digitalOutput*: BOOL;
  - *float*: REAL;
  - *double*: REAL;
  - *string*: STRING.
- *strsize*: number of character if it is a string type.



## 8. APPENDIX IV: ELEMENTS OF HMI RUNTIME

### 8.1 FUNCTIONS

This chapter lists all the functions that HMI run-time exports to UserInterface and so the user can use them into script and procedures.

These functions are divided into several categories which are shown in details in the following paragraphs.

#### 8.1.1 SYSTEM FUNCTIONS: HARDWARE AND OPERATING SYSTEM

```
unsigned char Video_InitHMI (unsigned char dmy)
```

Function of initialization for HMI runtime

Parameter	Description
dmy	Reserved. Set 0.
Return Value	Description
Video_InitHMI	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_Switch (unsigned char on);
```

Turn on/off the display

Parameter	Description
on	<i>TRUE</i> : turns on the display. <i>FALSE</i> : turns of the display.
Return Value	Description
Video_Switch	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_LCDContrast( unsigned char more );
```

Display contrast

Parameter	Description
more	<i>TRUE</i> : increases display contrast. <i>FALSE</i> : decreases display contrast.
Return Value	Description
Video_LCDContrast	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_SaveRect( unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2 );
```

Save display area to memory

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.



Return Value	Description
Video_SaveRect	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_WriteFromBuff( unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2 );
```

Restore display area from memory (previously saved with Video\_SaveRect).

Parameter	Description
x1	Not sensible (saved area has the original coordinates).
y1	Not sensible (saved area has the original coordinates).
x2	Not sensible (saved area has the original coordinates).
y2	Not sensible (saved area has the original coordinates).
Return Value	Description
Video_WriteFromBuff	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_Lock( unsigned char res );
```

Lock the display resources for exclusive access

Parameter	Description
res	Reserved. Set 0.
Return Value	Description
Video_Lock	Not sensible (return input parameter <i>res</i> ).

```
unsigned char Video_Unlock( unsigned char res );
```

Unlock the display resource after exclusive access

Parameter	Description
res	Reserved. Set 0.
Return Value	Description
Video_Unlock	Not sensible (return input parameter <i>res</i> ).

```
unsigned char Video_Sleep( unsigned short msec );
```

Suspend the task where the function is used

Parameter	Description
msec	Suspends time measured in milliseconds.
Return Value	Description
Video_Unlock	Not sensible (always <i>TRUE</i> ).





## 8.1.2 FUNCTION FOR MANAGING PROJECT RESOURCES AND COMMON PROPERTIES

```
unsigned char Video_SetWndSysProps( unsigned long pFont, unsigned long colFore, unsigned long colBack );
```

Set common properties for all pages in the project

Parameter	Description
pFont	Address of font for printing text in title bar (the font must be added with <code>Video_AddFont</code> function ).
colFore	Text color of <i>Title Bar</i> .
colBack	Background color of <i>Title Bar</i> .
Return Value	Description
<code>Video_SetWndSysProps</code>	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_SetEditKey( unsigned char id, unsigned char code );
```

Set key-code for editing functions

Parameter	Description
id	Identifier of editing function (see. Enumerative table 102, par. 6.3.1).
code	Key code associated with editing function.
Return Value	Description
<code>Video_SetEditKey</code>	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_AddFont( unsigned long pFont, unsigned char charLen, unsigned char charHei, unsigned char offs );
```

Publish a new font in HMI run-time

Parameter	Description
pFont	Address of first byte of font.
charLen	Character width of font (#pixel).
charHei	Character height of font (#pixel).
offs	Byte offset of a font that starts with ASCII 0x00 (subset of characters).
Return Value	Description
<code>Video_AddFont</code>	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.



```
unsigned char Video_AddFontUnicode( unsigned long pFont, unsigned char charLen, unsigned char charHei );
```

Publish a new unicode font in HMI run-time

Parameter	Description
pFont	Address of first byte of font.
charLen	Character width of font (#pixel).
charHei	Character height of font (#pixel).
Return Value	Description
Video_AddFontUnicode	TRUE if successful, FALSE otherwise.

```
unsigned char Video_LoadLanguage( unsigned long pResStrings, unsigned long pEnums );
```

Load strings and enumeratives of any language

Parameter	Description
pResStrings	Address of first resources string for current language.
pEnums	Address of first resources string for current language.
Return Value	Description
Video_LoadLanguage	TRUE if successful, FALSE otherwise.

```
unsigned char Video_DrawFrames(unsigned short left, unsigned short top, unsigned short right, unsigned short bottom, unsigned long colBack, unsigned char fBar, unsigned long pTitle, unsigned char fResStr, unsigned char fSysBtn, unsigned char style );
```

Function for draw frame-set

Parameter	Description
left	Width of left frame (#pixel).
top	Height of top frame (#pixel).
right	Width of right frame (#pixel).
bottom	Height of bottom frame (#pixel).
colBack	Background color.
fBar	- TRUE: shows title bar; - FALSE: hides title-bar.
pTitle	Text of title bar: NULL: No string in title.
fResStr	- TRUE: pTitle is a resource string; - FALSE: pTitle is an address of constant string.
fSysBtn	- TRUE: shows system; - FALSE: hides system button.
style	- 0: Flat; - 1: Raised; - 2: Sunken.



Return Value	Description
Video_DrawFrames	Not sensible (always <i>TRUE</i> ).

## 8.1.3 FUNCTIONS FOR OPERATING WITH PAGES

```
unsigned char Video_InitPage( unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, unsigned long pTitle, unsigned short wData );
```

Show a page on display

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
pTitle	Address of <i>Text</i> of title bar: - <i>NULL</i> : no text in title bar.
wData	Feature declaration: b0..b7: - 0: Flat; - 1: Raised; - 2: Sunken. b8: - 0: no title bar; - 1: shows title bar. b9: - 0: pTitle is an address of constant string; - 1= pTitle is a resource string. b10: - 0: no system button; - 1: shows system button. b11: - 0: window not modal; - 1: modal window (sensible only for pop-ups windows).
Return Value	Description
Video_InitPage	Not sensible (always <i>TRUE</i> ).



```
unsigned char Video_SetPageColors( unsigned long colFore, unsigned long col-
Back );
```

Assign all colors for current page

Parameter	Description
colFore	Color of the text of page.
colBack	Background color of page.
Return Value	Description
Video_SetPageColors	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_ClrScreen( );
```

Delete entire display area and fill with background color defined with Video\_SetPageColors

Return Value	Description
Video_ClrScreen	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_ClrRect( unsigned short x1, unsigned short y1, unsigned
short x2, unsigned short y2 );
```

Delete only a portion of display and fill with background color defined with Video\_SetPageColors

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
Return Value	Description
Video_ClrRect	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_SetFont( unsigned long fontPtr );
```

Load a font as current font for drawing objects. To correctly execute this function, the font must be declared with Video\_AddFont.

Parameter	Description
fontPtr	Address of first byte of font.
Return Value	Description
Video_SetFont	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.



```
unsigned char Video_SetColors( unsigned long colForeTxt, unsigned long col-  
BackTxt, unsigned long colForeSel, unsigned long colBackSel );
```

Assign the current colors for drawing objects

Parameter	Description
colForeTxt	Text color.
colBackTxt	Background color.
colForeSel	Text color for selection.
colBackSel	Background color for selection.
Return Value	Description
Video_SetColors	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_ResetMaps( unsigned char res );
```

Delete the maps saved for every object. The maps are created adding an object at once, with access mode `kACS_INIT`.

Parameter	Description
res	Reserved. Set 0.
Return Value	Description
Video_ResetMaps	Not sensible (return input parameter <i>res</i> ).

## 8.1.4 FUNCTION FOR OBJECTS

```
unsigned char Video_NextEdit( unsigned char fRWOnly );
```

Enable selection for next objects identified by *Set*. *Order* attribute.

Parameter	Description
fRWOnly	Limit for selecting the next edit-box: - <i>FALSE</i> : next edit-box must be selectable; - <i>TRUE</i> : the next edit-box must be selectable and writable.
Return Value	Description
Video_NextEdit	Handle of selected objects; if -1 the function has an error.

```
unsigned char Video_PrevEdit( unsigned char fRWOnly );
```

Enable selection for previous objects identified by *Set*. *Order* attribute.

Parameter	Description
fRWOnly	Limit for selecting the next edit-box: - <i>FALSE</i> : the next edit-box must be selectable; - <i>TRUE</i> : the next edit-box must be selectable and writable.
Return Value	Description
Video_PrevEdit	Handle of selected objects; if -1 the function has an error.



```
unsigned char Video_EnterEdit( unsigned short wHnd );
```

Enter edit-mode of an Edit Box otherwise execute the action for a button. The object holds the task until exits from edit-mode.

Parameter	Description
wHnd	Handle of object that must be edited or execute his action.
Return Value	Description
Video_EnterEdit	Return pressed key code for exiting edit-mode. If return -1 is an error only if the object is an edit-box.

```
unsigned char Video_EnterEditSel( unsigned short wHnd, unsigned char onlySelect )
```

Select object or enter edit-mode of an Edit box otherwise execute the action for a button. The object holds the task until exit from edit-mode.

Parameter	Description
wHnd	Handle of object that must be edited or execute his action.
OnlySelect	- <i>FALSE</i> : as VideoEnterEdit(); - <i>TRUE</i> : enables only the selection without entering edit-mode.
Return Value	Description
Video_EnterEditSel	Return pressed key code for exiting edit-mode. If return -1 is an error only if the object is an edit-box.

```
unsigned char Video_PushButton( unsigned short wHnd );
```

Enter press-mode for buttons. The object holds the task until exit from press-mode. This function is sensible only for touchscreen systems.

Parameter	Description
wHnd	Handle of button.
Return Value	Description
Video_PushButton	- <i>TRUE</i> : last pressure event was in button area; - <i>FALSE</i> : last pressure event was outside button area; - <i>-1</i> : error.

```
short Video_FirstLastEdit( unsigned char rwReq, unsigned char last )
```

Return the handle of first or last selectable controls.

Parameter	Description
rwReq	Boolean parameter. It indicates if the function checks for the objects that have read-write access mode.
last	- <i>TRUE</i> : last selectable object; - <i>FALSE</i> : first selectable object.
Return Value	Description
Video_FirstLastEdit	Handle of the object; -1 if errors or do not exist selectable objects



## 8.1.5 DRAWING FUNCTIONS

```
unsigned char Video_Line( unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, unsigned char pts, unsigned long color );
```

Draw a line

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
pts	Thickness.
color	Line color.
Return Value	Description
Video_Line	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_Rectangle( unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, unsigned char pts, unsigned char transp, unsigned long bordCol, unsigned long fillCol );
```

Draw a rectangle

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
pts	Border thickness.
transp	- <i>TRUE</i> : transparent square; - <i>FALSE</i> : solid square.
bordCol	Border color.
fillCol	Fill color. The value is not sensible if <i>transp</i> is <i>TRUE</i> .
Return Value	Description
Video_Rectangle	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_DrawBorder( unsigned char style, unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, unsigned char pts, unsigned char color );
```

Draw a border outside the rectangle area

Parameter	Description
style	- 0: flat; - 1: raised; - 2: sunken.
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.



y2	Bottom-down 'y coordinate' edge relative to full page.
pts	Border thickness. It is sensible only if <i>style</i> = 0.
color	Border color. It is sensible only if <i>style</i> = 0.
Return Value	Description
Video_DrawBorder	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_DelBorder( unsigned char style, unsigned short x1, unsigned short y1, unsigned short x2, unsigned short y2, unsigned char pts );
```

Delete a border outside the rectangle area. The color of fill is the page color assigned with Video\_SetPageColors

Parameter	Description
style	- 0: flat; - 1: raised; - 2: sunken.
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
pts	Border thickness It's sensible only if <i>style</i> = 0
Return Value	Description
Video_DelBorder	<i>TRUE</i> if successful, <i>FALSE</i> otherwise.

```
unsigned char Video_PrintBitmap( unsigned long ptrBmp, unsigned short x, unsigned short y );
```

Print a bitmap coded with run-time HMI format

Parameter	Description
ptrBmp	Address of first byte of bitmap.
x	Top-left 'x coordinate' edge relative to full page.
y	Top-left 'y coordinate' edge relative to full page.
Return Value	Description
Video_PrintBitmap	Not sensible (always <i>TRUE</i> ).

```
unsigned char Video_DelBitmap( unsigned long ptrBmp, unsigned short x, unsigned short y );
```

Delete a bitmap where it is not transparent, coded with run-time HMI format

Parameter	Description
ptrBmp	Address of first byte of bitmap.
x	Top-left 'x coordinate' edge relative to full page.
y	Top-left 'y coordinate' edge relative to full page.
Return Value	Description
Video_DelBitmap	Not sensible (always <i>TRUE</i> ).





```
unsigned long Video_InitBmpTreeRefresh( unsigned short x1,  
unsigned short y1, unsigned short x2, unsigned short y2 );
```

Switch context of drawing area. With this call all the next drawing functions uses the invisible device context

Parameter	Description
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
Return Value	Description
Video_InitBmpTreeRefresh	Address of invisible device context.

```
unsigned long Video_EndBmpTreeRefresh( unsigned short pDC,  
unsigned short x1, unsigned short y1,  
unsigned short x2, unsigned short y2 );
```

Restore original device context and copy the area from invisible context to display context

Parameter	Description
pDC	Address of invisible device context.
x1	Top-left 'x coordinate' edge relative to full page.
y1	Top-left 'y coordinate' edge relative to full page.
x2	Bottom-down 'x coordinate' edge relative to full page.
y2	Bottom-down 'y coordinate' edge relative to full page.
Return Value	Description
Video_EndBmpTreeRefresh	Not sensible (always <i>TRUE</i> ).

## 8.1.6 FUNCTIONS FOR TEXT

```
unsigned char Video_PrintStr( char * str, unsigned short x, unsigned short  
y );
```

Print a string using the current font set with *SetFont* and current colors set with *SetColor()*

Parameter	Description
str	Text to print.
x	Top-left 'x coordinate' edge relative to full page.
y	Top-left 'y coordinate' edge relative to full page.
Return Value	Description
Video_PrintStr	Number of chars printed.



```
unsigned char Video_PrintResStr( unsigned short idRes, unsigned short x,
unsigned short y );
```

Print a resources string using the current font set with *SetFont* and current colors set with *SetColors()*

Parameter	Description
idRes	Identifiers of resource.
x	Top-left 'x coordinate' edge relative to full page.
y	Top-left 'y coordinate' edge relative to full page.
Return Value	Description
Video_PrintResStr	Number of chars printed.

```
unsigned char Video_PrintNChar( char * str, unsigned char accMode, unsigned
short x, unsigned short y, unsigned char nChar, unsigned long format );
```

Print at most *nChar* characters of a string, using the current font set with *SetFont* and current colors set with *SetColors()*. It uses also a format for drawing the text.

If *nChar* is less than string length, it truncates the string; otherwise apply the alignment.

Parameter	Description
str	Text to print.
accMode	- kACS_PRINT: print with colForeTxt and colBackTxt colors. - kACS_SELECT: print with colForeSel and colBackSel colors.
x	Top-left 'x coordinate' edge relative to full page.
y	Top-left 'y coordinate' edge relative to full page.
nChar	Maximum number of chars to print.
format	Alignment of text. It is sensible only if <i>nChar &gt; length of str</i> : - 0x08 = right alignment; - 0x10 = center alignment; - 0x20 = left alignment.
Return Value	Description
Video_PrintNChar	Number of chars of truncated string.

## 8.1.7 FUNCTIONS FOR PARAMETER ACCESS

```
unsigned short Video_GetParam( unsigned char idxDevice, unsigned short idx-
Param, unsigned char subIdxParam, unsigned long pVal, unsigned char type )
```

Read a parameter from a device

Parameter	Description
idxDevice	Index of device connected.
idxParam	Index of parameter.
subIdxParam	Sub-index of parameter.



pVal	Address of variable that contains the read value.
type	Parameter type. Available values: tyBool, tySInt, tyUSInt, tyByte, tyInt, tyUInt, tyWord, tyDInt, tyUDInt, tyDWord, tyReal, tyString.
Return Value	Description
Video_GetParam	Integer values: - 0 = successful; - 1 = index of parameter not found; - 2,8,9 = system errors; - 3 = type not valid.

```
unsigned short Video_SetParam( unsigned char idxDevice, unsigned short idx-  
Param, unsigned char subIdxParam, unsigned long pVal, unsigned char type )
```

Write a parameter to a device.

Parameter	Description
idxDevice	Index of device connected.
idxParam	Index of parameter.
subIdxParam	Sub-index of parameter.
pVal	Address of variable that contains the value to write.
type	Parameter type. Available values: tyBool, tySInt, tyUSInt, tyByte, tyInt, tyUInt, tyWord, tyDInt, tyUDInt, tyDWord, tyReal, tyString.
Return Value	Description
Video_SetParam	Integer values: - 0 = successful; - 1 = index of parameter not found; - 2,8,9 = system errors; - 3 = type not valid; - 4 = read-only parameter; - 5 = cannot write now; - 6 = the value is less than the min value; - 7 = the value is more than the max value.



## 8.1.8 FUNCTIONS FOR EVENTS

```
unsigned char Video_SendEvent( unsigned short msgID, unsigned short wParam );
```

Send an event from code

Parameter	Description
msgID	Available values: <ul style="list-style-type: none"> <li>- kWM_NULL = no event;</li> <li>- kWM_KEY = key pressure;</li> <li>- kWM_MSG = open message;</li> <li>- kWM_SELECT = select an edit-box, a button;</li> <li>- kWM_PUSH = pressure on button.</li> </ul>
wParam	Event parameter. It has a different meaning according to msgID: <ul style="list-style-type: none"> <li>- if kWM_NULL= not sensible;</li> <li>- if kWM_KEY= pressed key.</li> </ul> For the key a constant value exists. The syntax is: kKEY_<key> Ex. LongLeft -> kKEY_LongLeft <ul style="list-style-type: none"> <li>- if kWM_MSG =ID of message page to open;</li> <li>- if kWM_SELECT= handle of selected edit-box, button;</li> <li>- if kWM_PUSH= handle of pressed button.</li> </ul>
Return Value	Description
Video_SendEvent	TRUE if successful, FALSE otherwise.

```
unsigned long Video_GetEvent( unsigned char dmy );
```

Pop an event from queue

Parameter	Description
dmy	Reserved. Set 0.
Return Value	Description
Video_GetEvent	Double word with inside the encoding.                     16 low bit = type of event: <ul style="list-style-type: none"> <li>- kWM_NULL = no event;</li> <li>- kWM_KEY = key pressure;</li> <li>- kWM_MSG = open message;</li> <li>- kWM_SELECT = select an edit-box, a button;</li> <li>- kWM_PUSH = pressure on button.</li> </ul> 16 high bit = event parameter: <ul style="list-style-type: none"> <li>- if kWM_NULL= not sensible;</li> <li>- if kWM_KEY= pressed key;</li> <li>- if kWM_MSG= ID of message page to open;</li> <li>- if kWM_SELECT= handle of selected edit-box, button;</li> <li>- if kWM_PUSH= handle of pressed button.</li> </ul>



## 8.2 FUNCTION BLOCKS

FUNCTION BLOCK: Video\_GetPageColors

Get the page colors of the page where called

Frame structure: FB_VIDEO_GETPAGEColorS		
Local variables	Type	Description
---		
Input variables	Type	Description
---		
Output variables	Type	Description
color	word32	Text color in the page.
back	word32	Background of the page.

FUNCTION BLOCK: Static01

Text strings with variable visibility

Frame structure: FB_STATIC01		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among static objects.
x	word16	Top-left 'x coordinate' edge relative to full page.
y	word16	Top-left 'y coordinate' edge relative to full page.
accMode	byte	- kACS_IDLE = no effect; - kACS_INIT = first draw on display; - kACS_PRINT = update draw on display.
fResStr	byte	Boolean value: - FALSE = pString is the address of string to draw; - TRUE = pString is the identifier of resource string.
pString	word32	Text to draw. It is different according to fResStr.
pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
foreCol	word32	Text color.
bckCol	word32	Background color.
pVisVar	word32	Visibility. Available values: - FALSE = text not visible; - TRUE = text always visible; - var_addr = address of boolean variable.



Frame structure: FB_STATIC01		
format	word16	Format for numeric values, encoded in 32 bit: b3 1= right alignment b4 1= center alignment b5 1= left alignment
style	byte	- 0 = flat - 1 = raised - 2 = sunken
bordPts	byte	Border thickness. It is sensible only if <i>style</i> = 0
bordCol	word32	Border color. It is sensible when <i>style</i> = 0 <i>bordPts</i> > 0 and not <i>pSelVar</i> = 1 fixed.
selBackCol	word32	Background color when object is selected. It is not sensible if <i>pSelVar</i> = 0 fixed.
selForeCol	word32	Text color when selected. It is not sensible if <i>pSelVar</i> = 0 fixed.
pRefrVar	word32	Variable for update: - FALSE = the object is redrawn only when the page is opening or when returning from child page; - TRUE = the object is always redrawn.
pSelVar	word32	Selection flag for the object. Suggest if the object must uses { 'selBackCol' } and { 'selForeCol' }. Available values: - FALSE = object is never selected; - TRUE = object is always selected; - <i>var_addr</i> = address of boolean variable.
numChars	word16	Number of max characters. 0 indicates that the string is drawn with the entire value of <i>pString</i> .
Output variables	Type	Description
---		

FUNCTION BLOCK: Image

Image object

Frame structure: FB_IMAGE		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
memSel	byte	Selection status of the previous execution.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among image objects.



Frame structure: FB_IMAGE		
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
px1	word32	Address of variable for moving image on X-Axis. It is sensible only if <i>floating = TRUE</i>
py1	word32	Address of variable for moving image on Y-Axis. It is sensible only if <i>floating = TRUE</i>
type_x	byte	Type for px1. Available values: tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord. It is sensible only if <i>floating = TRUE</i> and <i>px1 &lt;&gt; NULL</i>
type_y	byte	Type for py1. Available values: tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord. It is sensible only if <i>floating = TRUE</i> and <i>py1 &lt;&gt; NULL</i>
dx	word16	Width (#pixel).
dy	word16	Height (#pixel).
style	byte	- 0 = flat - 1 = raised - 2 = sunken
floating	byte	Position of object: - FALSE = docking - TRUE = floating
bordPts	byte	Border thickness. It is sensible only if <i>style = 0</i>
bordCol	word32	Border color. It is sensible when <i>style = 0</i> and <i>bordPts &gt; 0</i> and not <i>pSelVar = 1</i> fixed.
bordSelCol	word32	Border color for selected object. It is sensible when <i>style = 0</i> and <i>bordPts &gt; 0</i> and not <i>pSelVar = 0</i> fixed
accMode	byte	- kACS_IDLE = no effect - kACS_INIT = first draw on display - kACS_PRINT = update draw on display - kACS_QUERY = request for updating output variables - kACS_BCKQUERY = request for updating output variables when the object is in background pages - kACS_DELETE = delete object
pBmp	word32	Address of first byte of bitmap to view. It is not sensible if <i>pSelBmp = 1</i> fixed.



Frame structure: FB_IMAGE		
Output variable	Type	Description
pSelBmp	word32	Address of first byte of bitmap to view when selected. It is not sensible if pSelBmp = 0 fixed.
pSelVar	word32	Selection flag for the object. Suggest if the object must uses { 'bordCol', 'pBmp' } or { 'bordSelCol', 'pSelBmp' }. Available values: <ul style="list-style-type: none"> <li>- FALSE = object is never selected</li> <li>- TRUE = object is always selected</li> <li>- var_addr = address of boolean variable</li> </ul>
pVisVar	word32	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- FALSE = image not visible</li> <li>- TRUE = image always visible</li> <li>- var_addr = address of boolean variable</li> </ul>
reqRefr	byte	Request refresh, updated when the object is called with accMode = kACS_QUERY or accMode = kACS_BCKQUERY.
abs_x1	word16	Top-left 'x coordinate' edge relative to full page obtained with the sum among 'x1' and 'px1'. The value is updated when the object is called with accMode = kACS_INIT or accMode = kACS_QUERY.
abs_y1	word16	Top-left 'y coordinate' edge relative to full page obtained with the sum among 'y1' and 'py1'. The value is updated when the object is called with accMode = kACS_INIT or accMode = kACS_QUERY.
mem_x1	word16	Value read from abs_x1 when the object is called with accMode = kACS_INIT or accMode = kACS_PRINT.
mem_y1	word16	Value read from abs_y1 when the object is called with accMode = kACS_INIT or accMode = kACS_PRINT.

FUNCTION BLOCK: Animation

Animation object

Frame structure: FB_ANIMATION		
Local variables	Type	Description
memBmp	word32	Address of bitmap of the previous execution
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among animation objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.





Frame structure: FB_ANIMATION		
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page
style	byte	- 0 = flat - 1 = raised - 2 = sunken
bordPts	byte	Border thickness. It is sensible only if style = 0
bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0 and not pSelVar = 1 fixed
accMode	byte	- kACS_IDLE = no effect - kACS_INIT = first draw on display - kACS_PRINT = update draw on display
pBmpArr	word32	Address of first image to view.
pCaseArr	word32	Address of first element of selection.
nArrEl	byte	Number of elements in image list.
pBmpDef	word32	Address of bitmap to view pSelVar not in pCaseArr.
pSelVar	word32	Address of variable for selection.
type	byte	Type of pSelVar. Available values: tyBool; tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord.
pVisVar	word32	Flag of visibility. Available values: - FALSE = image not visible - TRUE = image always visible - var_addr = address of boolean variable
Output variable	Type	Description
---		

FUNCTION BLOCK: Button02

Button object

Frame structure: FB_BUTTON02		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
memTransp	byte	Transparency status of the previous execution.
memSel	byte	Selection status of the previous execution.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among buttons objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.



Frame structure: FB_BUTTON02		
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
fResStr	byte	Boolean value: <ul style="list-style-type: none"> <li>- <i>FALSE</i> = pString is the address of string to draw</li> <li>- <i>TRUE</i> = pString is the identifier of resource string</li> </ul>
pText	word32	Text to draw on the button. It has different meaning according to fResStr. If this field is <i>NULL</i> , no text is drawn.
pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
style	byte	<ul style="list-style-type: none"> <li>- 0 = flat</li> <li>- 1 = raised</li> <li>- 2 = sunken</li> </ul>
bordPts	byte	Border thickness. It is sensible only if style = 0
bordCol	word32	Border color and text color. It is sensible only if style = 0 and bordPts > 0, or pString different as <i>NULL</i> , and not pSelVar = 1 fixed.
fillCol	word32	Color of button area. It is sensible only if pTransp different as 1 fixed, and not pSelVar = 1 fixed.
bordSelCol	word32	Border color and text color when selected. It is sensible only if style = 0 and bordPts > 0, or pString different as <i>NULL</i> , and not pSelVar = 0 fixed.
fillSelCol	word32	Color of button area when selected. It is sensible only if pTransp different as 1 fixed, and not pSelVar = 0 fixed.
accMode	byte	<ul style="list-style-type: none"> <li>- kACS_IDLE = no effect</li> <li>- kACS_INIT = first draw on display</li> <li>- kACS_PRINT = update draw on display</li> </ul>
pVisVar	word32	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- <i>FALSE</i> = image not visible</li> <li>- <i>TRUE</i> = image always visible</li> <li>- var_addr = address of boolean variable</li> </ul>
pTransp	word32	Flag of transparency. Available values: <ul style="list-style-type: none"> <li>- <i>FALSE</i> = button always solid</li> <li>- <i>TRUE</i> = button always transparent</li> <li>- var_addr = address of boolean variable</li> </ul>



Frame structure: FB_BUTTON02		
pPressVar	word32	Address of a boolean variable. Pressed button= *pPressVar = <i>TRUE</i> Released button= *pPressVar = <i>FALSE</i> . If the field is <i>NULL</i> there is no variable.
pSelVar	word32	Selection flag for the object. Suggest if the object must uses {'bordCol', 'fillCol'} or {'bordSelCol', 'fillSelCol'}. Available values: - <i>FALSE</i> = object is never selected - <i>TRUE</i> = object is always selected - var_addr = address of boolean variable
format	word16	Format of numeric values encoded with 16 bit: b4 1= right alignment b5 1= center alignment b6 1=left alignment
order	word16	Number for establishing a sequential selection
Output variable	Type	Description
---		

FUNCTION BLOCK: EditBox01

Edit object

Frame structure: FB_EDITBOX01		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among edit-box objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
pFont	word32	Address of font for drawing text. The font must be initialized with <i>Video_AddFont</i> .
style	byte	- 0 = flat - 1 = raised - 2 = sunken
foreCol	word32	Text color.



Frame structure: FB_EDITBOX01		
bckCol	word32	Background color.
foreSelCol	word32	Text color when selected. It is sensible only if pCanSel is not 0 fixed.
bckSelCol	word32	Background color when selected. It is sensible only if pCanSel is not 0 constant.
bordPts	byte	Border thickness. It is sensible only if style = 0
bordCol	word32	Border color. It is sensible when style = 0 bordPts > 0
rw	byte	- FALSE = read-only mode - TRUE = read-write mode
refr	byte	Request refresh: - FALSE = the object is redrawn only when the page is opening or return from child page - TRUE = the object is always redrawn
pVar	word32	Address of variable or parameter according to format. It cannot be NULL. If it is a parameter is encoded in this way: b0..b7 = Subindex parameter b8..b23 = IPA parameter b24..b32 = Device address
type	byte	Type of data. Available values: tyBool; tySInt; tyUSInt; tyByte; tyInt; tyUInt; tyWord; tyDInt; tyUDInt; tyDWord, tyReal
pVarMin	word32	Min value for edit-box variable. If bit b16-b17 (LSB) of field format contains 0 the limit is not set, if contains 1 is a constant limit, if contains 2 it is a variable limit.
pVarMax	word32	Max value for edit-box variable. If bit b14-b15 (LSB) of field format contains 0 the limit is not set, if contains 1 is a constant limit, if contains 2 it's a variable limit.
enumId	int16	Identifier of enumerative. If 0 no enumerative associated with this field exists.



Frame structure: FB_EDITBOX01		
format	word32	View format encoded in 32 bit: b0 - 0 = draw sign only if number is negative - 1 = draw sign also for positive numbers b1 - 0 = does not print most significant null digits - 1 = draw zeroes on most significant null digits b2 - 0 = 'pVar' is a variable - 1 = 'pVar' is a parameter b3 1 = right alignment b4 1 = center alignment b5 1 = left alignment b10 Exadecimal format, with a..f lowercase b11 Exadecimal format, with A..F uppercase b14..b15 - 0=no max limit - 1=constant max limit - 2=variable max limit b16..b17 - 0=no min limit - 1=constant min limit - 2=variable min limit b24..b26 Precision (real numbers) b27..b31 Width (cfr. § 1.7.2)
pVisVar	word32	Flag of visibility. Available values: - <i>FALSE</i> = object not visible - <i>TRUE</i> = object always visible - <i>var_addr</i> = address of boolean variable
pCanSel	word32	Available values: - <i>FALSE</i> = object not selected - <i>TRUE</i> = object always selected - <i>var_addr</i> = address of boolean variable
order	byte	Number for establish a sequential selection.



Frame structure: FB_EDITBOX01		
accMode	byte	<ul style="list-style-type: none"> <li>- kACS_IDLE = no effect</li> <li>- kACS_INIT = first draw on display</li> <li>- kACS_PRINT = update draw on display</li> <li>- kACS_SELECT = update draw on display when selected</li> <li>- kACS_MODIFY = enter in editing mode</li> </ul>
Output variable	Type	Description
outKey	char	Key code for exiting editing-mode.

FUNCTION BLOCK: TextBox

Text box object

Frame structure: FB_TEXTBOX		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
base	word16	Number of first line seen in object.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among textbox objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
style	byte	<ul style="list-style-type: none"> <li>- 0 = flat</li> <li>- 1 = raised</li> <li>- 2 = sunken</li> </ul>
foreCol	byte	Text color.
bckCol	byte	Background color.
bordPts	byte	Border thickness It is sensible only if style = 0
bordCol	byte	Border color. It is sensible when style = 0 bordPts > 0
LineNr	byte	<ul style="list-style-type: none"> <li>- FALSE = hide line number</li> <li>- TRUE = show line number</li> </ul>
rw	byte	<ul style="list-style-type: none"> <li>- FALSE= read-only mode</li> <li>- TRUE= read-write mode</li> </ul>
pVar	word32	Address of string variable. It cannot be NULL.
szpVar	word32	Size of pVar.



Frame structure: FB_TEXTBOX		
pVisVar	word32	Flag of visibility. Available values: - <i>FALSE</i> = object not visible - <i>TRUE</i> = object always visible - var_addr= address of boolean variable
order	byte	Number for establishing a sequential selection.
accMode	byte	Access mode. Available values: - kACS_IDLE = no effect - kACS_INIT = first draw on display - kACS_PRINT = update draw on display - kACS_SELECT = update draw on display when selected - kACS_MODIFY = enter editing mode - kACS_SCROLLUP= scroll up one line - kACS_SCROLLDW= scroll down one line
rqCursPos	word16	Char Index where move the cursor.
rqCursRow	word16	Row to select.
dispCurs	byte	- <i>TRUE</i> = the cursor is always visible even if it is not enabled editing mode - <i>FALSE</i> = the cursor is visible only if it is enabled editing mode.
dispRow	byte	- <i>TRUE</i> = the row selection is always visible even if it is not enabled editing mode - <i>FALSE</i> = the row selection is visible only if it is enabled editing mode
bckSelCol	word32	Future developments.
wParam	word32	Future developments.
lParam	word32	Future developments.
Output variable	Type	Description
outKey	char	Key code for exiting editing-mode.
outCursPos	word16	Char index where there is the cursor.
outCursRow	word16	Index of selected row.

FUNCTION BLOCK: Progress

Progress bar object

Frame structure: FB_PROGRESS		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution
memVal	word32	Progress status of the previous execution
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among progress objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.



Frame structure: FB_PROGRESS		
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
style	byte	<ul style="list-style-type: none"> <li>- 0 = flat</li> <li>- 1 = raised</li> <li>- 2 = sunken</li> </ul>
barCol	word32	Color of step bar.
bckCol	word32	Background color.
bordPts	byte	Border thickness. It is sensible only if <code>style = 0</code>
bordCol	word32	Border color. It is sensible when <code>style = 0</code> <code>bordPts &gt; 0</code>
pVar	word32	Step variable. This is the filling percentage of bar in relation with the range assigned by <code>pMin</code> and <code>pMax</code> .
type	byte	Type of <code>pVar</code> . Assigned values: <code>tyBool</code> ; <code>tySInt</code> ; <code>tyUSInt</code> ; <code>tyByte</code> ; <code>tyInt</code> ; <code>tyUInt</code> ; <code>tyWord</code> ; <code>tyDInt</code> ; <code>tyUDInt</code> ; <code>tyDWord</code>
pMin	word32	Min value for edit-box variable. If bit b0 (LSB) of field <code>format</code> contain 0 is a constant limit, if contain 1 it is a variable limit.
pMax	word32	Min value for edit-box variable. If bit b1 (LSB) of field <code>format</code> contain 0 is a constant limit, if contain 1 it is a variable limit.
format	word32	View format encoded in bit: b0 <ul style="list-style-type: none"> <li>- 0 = <code>pMin</code> contains a constant value of Type 'type'</li> <li>- 1 = <code>pMin</code> contains the address of variable of Type 'type'</li> </ul> b1 <ul style="list-style-type: none"> <li>- 0 = <code>pMax</code> contains a constant value of Type <code>type</code></li> <li>- 1 = <code>pMax</code> contains the address of variable of Type <code>type</code></li> </ul> b2 <ul style="list-style-type: none"> <li>- 0 = horizontal orientation</li> <li>- 1 = vertical orientation</li> </ul>
pVisVar	word32	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- <code>FALSE</code> = object not visible</li> <li>- <code>TRUE</code> = object always visible</li> <li>- <code>var_addr</code> = address of boolean variable</li> </ul>
accMode	byte	Access mode. Available values: <ul style="list-style-type: none"> <li>- <code>kACS_IDLE</code> = no effect</li> <li>- <code>kACS_INIT</code> = first draw on display</li> <li>- <code>kACS_PRINT</code> = update draw on display</li> </ul>





Frame structure: FB_PROGRESS		
Output variable	Type	Description
---		

FUNCTION BLOCK: CustomCtrl

Embedded function block which implements custom control

Frame structure: FB_CUSTOMCTRL		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
ptrFunct	word32	Address of function that implements <i>Type</i> wCtrlID.
data0	word32	Local variable.
data1	word32	Local variable.
data2	word32	Local variable.
data3	word32	Local variable.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among custom control objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
wCtrlID	word16	Identifier of custom control.
pVisVar	word32	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- FALSE = object not visible</li> <li>- TRUE = object always visible</li> <li>- var_addr = address of boolean variable</li> </ul>
refr	byte	Request refresh: <ul style="list-style-type: none"> <li>- FALSE = the object is redrawn only when the page is opening or return from child page</li> <li>- TRUE = the object is always redrawn</li> </ul>
accMode	byte	Access mode. Available values: <ul style="list-style-type: none"> <li>- kACS_IDLE = no effect</li> <li>- kACS_INIT = first draw on display</li> <li>- kACS_PRINT = update draw on display</li> </ul> The value greater than 200 can be used for custom purpose.
wParam	word16	16 bit data without sign, used for custom purpose
lParam	int32	32 bit data with sign, used for custom purpose
rParam	float	32 bit real data with sign, used for custom purpose
Output variable	Type	Description
---		



FUNCTION BLOCK: Chart

Chart object

Frame structure: FB_CHART		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution
grx1	word16	Top-left 'x coordinate' edge relative to full page.
gry1	word16	Top-left 'y coordinate' edge relative to full page.
grx2	word16	Bottom-right 'x coordinate' edge relative to full page.
gry2	word16	Bottom-right 'y coordinate' edge relative to full page.
pChart	word32	Handle of the chart created after ACS_INIT.
lastIdxSamples	word32	Actual index of inserted track data.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among chart objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
style	byte	- 0 = flat - 1 = raised - 2 = sunken
bordPts	byte	Border thickness. It is sensible only if <i>style</i> = 0
bordCol	byte	Border color. It is sensible when <i>style</i> = 0 <i>bordPts</i> > 0
backCol	byte	Background color.
pNSamples	word32	Address of the number of available samples to add in the chart. This value is used only when refresh is <i>TRUE</i> .
tyNSamples	byte	Type of the number of samples.
tyXOffset	byte	Type of the offset of X-axis.
pXOffset	word32	Address of the offset of X-axis. (move right-left the chart in order to 0 ).
tyTrackRight	word16	Type of default track for right Y-Axis
pTrackRight	word32	Address of the track of right Y-Axis. (if 0 the right label will not drawn).
tyTrackLeft	word16	Type of default track for left Y-Axis.



Frame structure: FB_CHART		
pTrackLeft	word32	Address of the track of left Y-Axis. (if 0 the left label will not drawn).
formatLeft	word32	Label format of left Y-Axis.
formatRight	word32	Label format of right Y-Axis.
format	Word32	Label format of X-Axis.
iDivGridX	word16	Number of division on horizontal bar , used with scale factor and offset for drawing the chart tracks (Ex. scale X=1, iDivGridX = 5 value between 0 and 5 ). Sensible even if the grid is not visible.
iDivGridY	word16	Number of division on vertical bar , used with scale factor and offset for drawing the chart tracks (Ex. scale Y=1, iDivGridY = 5 value between 0 and 5 ). Sensible even if the grid is not visible.
fGrid	byte	Draw grid: - <i>FALSE</i> = grid not visible - <i>TRUE</i> = grid visible
iXLabelDiv	word16	Step for X-axis labels. How many division of horizontal bar must have labels.
tyXScaleType	byte	Type of X-Axis scale.
pXScale	word32	Address of Scale factor of x-Axis. Value range among two division of horizontal bars. 0 value indicate that the chart is in auto-scale mode.
pClearVar	Word32	Address of boolean variable. If it is <i>TRUE</i> the chart delete all the previous data.
accMode	byte	Access mode. Available values: - <i>kACS_IDLE</i> = no effect - <i>kACS_INIT</i> = first draw on display - <i>kACS_PRINT</i> = update draw on display - <i>kACS_CLOSE</i> = close the chart and delete all the data
pXData	word32	Address or constant for X-Axis definition. Available values: - constant: number of samples * constant start with 0 - variable = array that contains pNSamples samples with X-axis value
tyXData	byte	Type of 'pXData' array. If tyXData = tyUndefined is a constant.
XlabelCol	word32	Color of X-Axis label.
iDotStep	word16	Space among two points of grid in pixel. The property is sensible if the grid is visible.
iSampleBuffer	word16	Number of samples that the run-time can store. The older ones are deleted if the size is exceeded.



Frame structure: FB_CHART		
arXBars	word32[3]	Array of addresses of vertical bars. If 0 the vertical bar is not defined, otherwise the address of variable or constant value.
arXBarsType	word16[3]	Type of variable that indicates the value of vertical bars. If <code>arXBarsType[n] = tyUndefined</code> and <code>arXBars[n]</code> is not <code>NULL</code> , the value of <code>arXBars[n]</code> is a numeric constant.
arXBarsCol	word32[3]	Colors of vertical bars.
GridCol	word32	Color of grid.
BorderGridColor	word32	Color of border of grid.
pVisVar	word32	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- <code>FALSE</code> = object not visible</li> <li>- <code>TRUE</code> = object always visible</li> <li>- <code>var_addr</code> = address of boolean variable</li> </ul>
arTrkData	word32[8]	Array of addresses of data. The nth of <code>arTrkData</code> contains the address of first elements of array of nth track. If address is <code>NULL</code> the track is not define.
arTrkType	byte[8]	Array of data. The nth of <code>arTrkType</code> contains the type of nth elements of <code>arTrkData</code> . This value is sensible only if the element of <code>arTrkData</code> is not <code>NULL</code> .
arTrkCol	byte[8]	Array of track colors. This value is sensible only if the element of <code>arTrkData</code> is not <code>NULL</code> .
arTrkVis	word32[8]	Array of visibility flags. The nth element of <code>arTrkMinY</code> determines the visibility of the track: <ul style="list-style-type: none"> <li>- <code>FALSE</code> = track not visible</li> <li>- <code>TRUE</code> = track always visible</li> <li>- <code>var_addr</code> = address of boolean variable</li> </ul> This value is sensible only if the element of <code>arTrkData</code> is not <code>NULL</code> .
arTrkScaleY	word32[8]	Array of Y-axis scale. The range of samples for every horizontal division.
arTrkScaleType	word16[8]	Type of variable of Y-Axis scale. If constant value <code>arTrkScaleType[n] = tyUndefined</code> .
arTrkOffset	word32[8]	Array of offset of Y-Axis for every track. The displacement of the track from 0 high and low.
arTrkOffsetType	word16[8]	Type array of offset of Y-Axis for every track. If constant value <code>arTrkOffsetType[n] = tyUndefined</code> .
iYLabelDiv	word16[8]	Array that contains on every step draw the Y-Axis label.
arTrkBarValue	word32[8*3]	Array of addresses of variables for horizontal bars.
arTrkBarValueType	word16[8*3]	Array of types of variable for horizontal bars.



Frame structure: FB_CHART		
Output variable	Type	Description
arTrackBarName	word32[8*3]	Array of names for horizontal bars.
arTrkBarCol	word32[8*3]	Array of colors for horizontal bars.
---		

FUNCTION BLOCK: Trend

Trend object

Frame structure: FB_TREND		
Local variables	Type	Description
memVis	byte	Visibility status of the previous execution.
grx1	word16	Top-left 'x coordinate' edge relative to full page.
gry1	word16	Top-left 'y coordinate' edge relative to full page.
grx2	word16	Bottom-right 'x coordinate' edge relative to full page.
gry2	word16	Bottom-right 'y coordinate' edge relative to full page.
pChart	word32	Handle of the chart created after ACS_INIT.
FirstSamplingTimeS	word32	Sampling time in seconds take on ACS_INIT or when cleared.
FirstSamplingTimeMS	word16	Sampling time in milli-seconds take on ACS_INIT or when cleared.
LastSamplingTimeS	word32	Sampling time in seconds take every acquisition.
LastSamplingTimeMS	word16	Sampling time in millisecond take every acquisition.
InitDraw	Byte	If TRUE the trend is just drawn.
Input variables	Type	Description
wHnd	word16	Handle of the object. Must be unique among chart objects.
x1	word16	Top-left 'x coordinate' edge relative to full page.
y1	word16	Top-left 'y coordinate' edge relative to full page.
x2	word16	Bottom-right 'x coordinate' edge relative to full page.
y2	word16	Bottom-right 'y coordinate' edge relative to full page.
pFont	word32	Address of font for drawing text. The font must be initialized with Video_AddFont.
style	byte	- 0 = flat - 1 = raised - 2 = sunken



Frame structure: FB_TREND		
bordPts	byte	Border thickness It is sensible only if <i>style</i> = 0
bordCol	byte	Border color. It is sensible when <i>style</i> = 0 bordPts > 0
backCol	byte	Background color.
pNSamples	word32	Inherited from chart but contains acquisition time in seconds.
tyNSamples	byte	Not used.
tyXOffset	byte	Type of the offset of X-axis.
pXOffset	word32	Address of the offset of X-axis (move right-left the chart in order to 0 )
tyTrackRight	word16	Type of default track for right Y-Axis.
pTrackRight	word32	Address of the track of right Y-Axis (if 0 the right label will not drawn).
tyTrackLeft	word16	Type of default track for left Y-Axis.
pTrackLeft	word32	Address of the track of left Y-Axis (if 0 the left label will not drawn).
formatLeft	word32	Label format of left Y-Axis.
formatRight	word32	Label format of right Y-Axis.
format	Word32	Label format of X-Axis. Available values: - 0 = ss - 1 = mm.ss - 2 = hh.mm - 3 = hh.mm.ss
iDivGridX	word16	Number of division on horizontal bar, used with scale factor and offset for drawing the chart tracks. (Ex. scale X=1, iDivGridX = 5 value between 0 and 5). Sensible even if the grid is not visible.
iDivGridY	word16	Number of division on vertical bar, used with scale factor and offset for drawing the chart tracks (Ex. scale Y=1, iDivGridY = 5 value between 0 and 5). Sensible even if the grid is not visible.
fGrid	byte	Draw grid: - <i>FALSE</i> = grid not visible - <i>TRUE</i> = grid visible
iXLabelDiv	word16	Step for X-axis labels. How many division of horizontal bar must have labels.
tyXScaleType	byte	Type of X-Axis scale.
pXScale	word32	Address of Scale factor of x-Axis. Value range among two division of horizontal bars. 0 value indicate that the chart is in auto-scale mode.
pClearVar	Word32	Address of boolean variable. If it is <i>TRUE</i> the chart delete all the previous data.



Frame structure: FB_TREND		
accMode	byte	Access mode. Available values: <ul style="list-style-type: none"> <li>- kACS_IDLE = no effect</li> <li>- kACS_INIT = first draw on display</li> <li>- kACS_PRINT = update draw on display</li> <li>- kACS_CLOSE = close the chart and delete all the data</li> </ul>
XlabelCol	word32	Address or constant for X-Axis definition. Available values: <ul style="list-style-type: none"> <li>- constant: number of samples * constant start with 0</li> <li>- variable: array that contains pNSamples samples with X-axis value</li> </ul>
iDotStep	word16	Type of pXData array. If tyXData = tyUndefined is a constant.
iSampleBuffer	word16	Color of X-Axis label.
arXBars	word32[3]	Space among two points of grid in pixel. The property is sensible if the grid is visible.
arXBarsType	word16[3]	Number of samples that the run-time can store. The older ones are deleted if the size exceeds.
arXBarsCol	word32[3]	Array of addresses of vertical bars. If 0 the vertical bar is not defined, otherwise the address of variable or constant value.
GridCol	word32	Type of variable that indicates the value of vertical bars. If arXBarsType[n] = tyUndefined and arXBars[n] is not NULL, the value of arXBars[n] is a numeric constant.
BorderGridColor	word32	Colors of vertical bars.
pVisVar	word32	Color of grid.
arTrkData	word32[8]	Color of broder of grid.
arTrkType	byte[8]	Flag of visibility. Available values: <ul style="list-style-type: none"> <li>- FALSE = object not visible</li> <li>- TRUE = object always visible</li> <li>- var_addr = address of boolean variable</li> </ul>
arTrkCol	byte[8]	Array of addresses of data. The nth of arTrkData contains the address of first elements of array of nth track. If address is NULL the track is not define.
arTrkVis	word32[8]	Array of data. The nth of arTrkType contains the type of nth elements of arTrkData. This value is sensible only if the element of arTrkData is not NULL.
arTrkScaleY	word32[8]	Array of track colors. This value is sensible only if the element of arTrkData is not NULL.



Frame structure: FB_TREND		
arTrkScaleType	word16[8]	Array of visibility flags. The nth element of arTrkMinY determines the visibility of the track: - <i>FALSE</i> = track not visible - <i>TRUE</i> = track always visible - <i>var_addr</i> = address of boolean variable This value is sensible only if the element of arTrkData is not <i>NULL</i> .
arTrkOffset	word32[8]	Array of Y-axis scale. The range of samples for every horizontal division.
arTrkOffsetType	word16[8]	Type of variable of Y-Axis scale. If constant value arTrkScaleType[n] = tyUndefined
iYLabelDiv	word16[8]	Array of offset of Y-Axis for every track. The displacement of the track from 0 high and low.
arTrkBarValue	word32[8*3]	Type array of offset of Y-Axis for every track. If constant value arTrkOffsetType[n] = tyUndefined
arTrkBarValueType	word16[8*3]	Array that contains on every step draw the Y-Axis label.
arTrackBarName	word32[8*3]	Array of addresses of variables for horizontal bars.
arTrkBarCol	word32[8*3]	Array of types of variable for horizontal bars.
Output variable	Type	Description
---		

<sup>(1)</sup> Available figures and colors depend on target's features.







## Contents

<b>1.</b>	<b>Overview</b>	449
<b>2.</b>	<b>Environment components</b>	451
<b>3.</b>	<b>Operating modes</b>	453
3.1	Full simulation	453
3.2	Simplified simulation	453
<b>4.</b>	<b>Using the simulator</b>	455
4.1	Simulation with Application	455
4.2	Simplified simulation with Application	457
4.3	Simulation with UserInterface	457
4.4	Simulation with both Application and UserInterface	459
<b>5.</b>	<b>Program interface</b>	461
5.1	Control panel	461
5.1.1	Manual workspace editing	462
5.2	Target panel	462
5.3	I/O panels	463
5.3.1	Adding elements to panels	463
5.3.2	Editing panel elements	464
5.3.3	Removing elements from panels	464
5.4	I/O panels list	464
5.4.1	Adding a new panel	464
5.4.2	Editing a panel	464
5.4.3	Removing a panel	465



## 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](http://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.<sup>(1)</sup>
- 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. OVERVIEW

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

Simulation can simulate execution of:

- PLC applications, IEC 61131-3 (made with Application).
- HMI pages (made with UserInterface).

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

NOTE: The simulation is not intended as a substitute for real, empirical testing during commissioning. It is a means for the programmer to submit his application, or parts of application, to unit testing and verification.

Only empirical testing with live equipment in the complete application can be considered a valid mechanism for validation.

### **⚠ WARNING**

#### **UNINTENDED EQUIPMENT OPERATION**

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

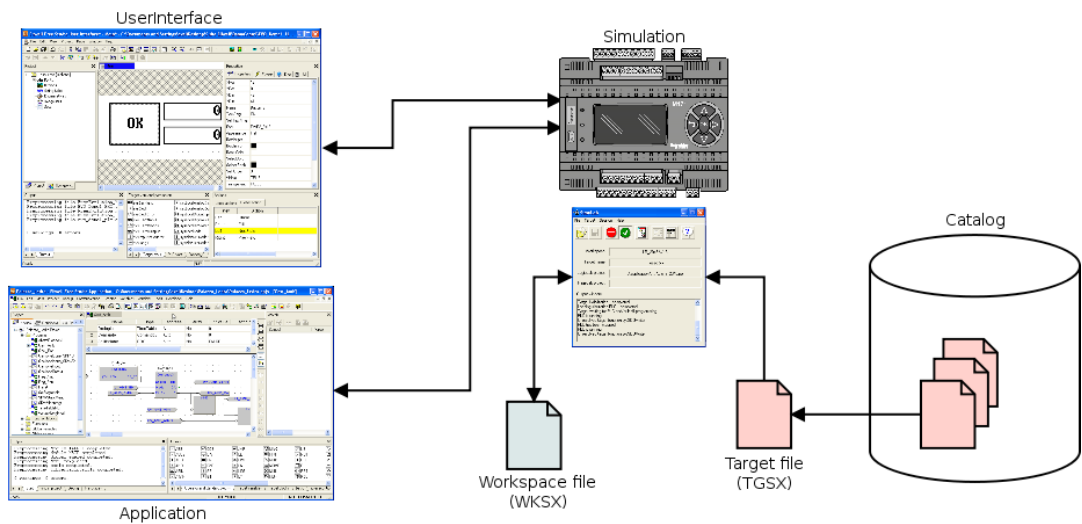
**Failure to follow these instructions can result in death, serious injury, or equipment damage.**





## 2. ENVIRONMENT COMPONENTS

The following paragraph shows you the main components of the simulated environment.



- Simulation: program that runs the simulator on the PC.
- Application: PLC development environment connected to the simulator.
- UserInterface: HMI development environment connected to the simulator.
- TCP/IP connection: localhost connection between the development environments.
- Catalog: repository of all target definitions, used by all software components.
- Simulation targets: Catalog components that define the targets to simulate; these files have TGSX extension. All targets supporting the simulation have a TGSX file linked in their PCT.
- Workspace: user file with WKSX extension that contains all the elements of a working session of the simulator (I/O panels, source PLC and HMI project, etc.). Each PLC and HMI project can have multiple simulation workspace files, and the user can manage them freely.







## 3. OPERATING MODES

Simulation can work with the two following operating modes.

### 3.1 FULL SIMULATION

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

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

The full simulation has the following features:

- simultaneous simulation of both PLC application and HMI pages;
- availability of the target panel, to have a visual and realistic representation of the target to run and interact with HMI pages;
- execution of the simulated application tasks handled by a scheduler that can reproduce the real target scheduler policy;
- the simulated target can have some parts implemented in C and/or IEC to implement the real target behaviour and characteristics, to react to PLC application as the real target would do;
- use of the I/O panels, that the user can configure to view and/or modify the simulated status and I/O variables of the target.

### 3.2 SIMPLIFIED SIMULATION

Simplified simulation is available only for PLC projects made with Application.

It lets you simulate the application immediately without having a prepared TGSX target file; in brief it is a simple cyclic execution of all the PLC tasks on the development PC, without the more advanced and graphical features of the full simulation.

Application will automatically create a temporary TGSX file, to be used with the simulator.

It will be possible to view and/or modify the simulated status and I/O variables of the target with the I/O panels, as in the full simulation mode.





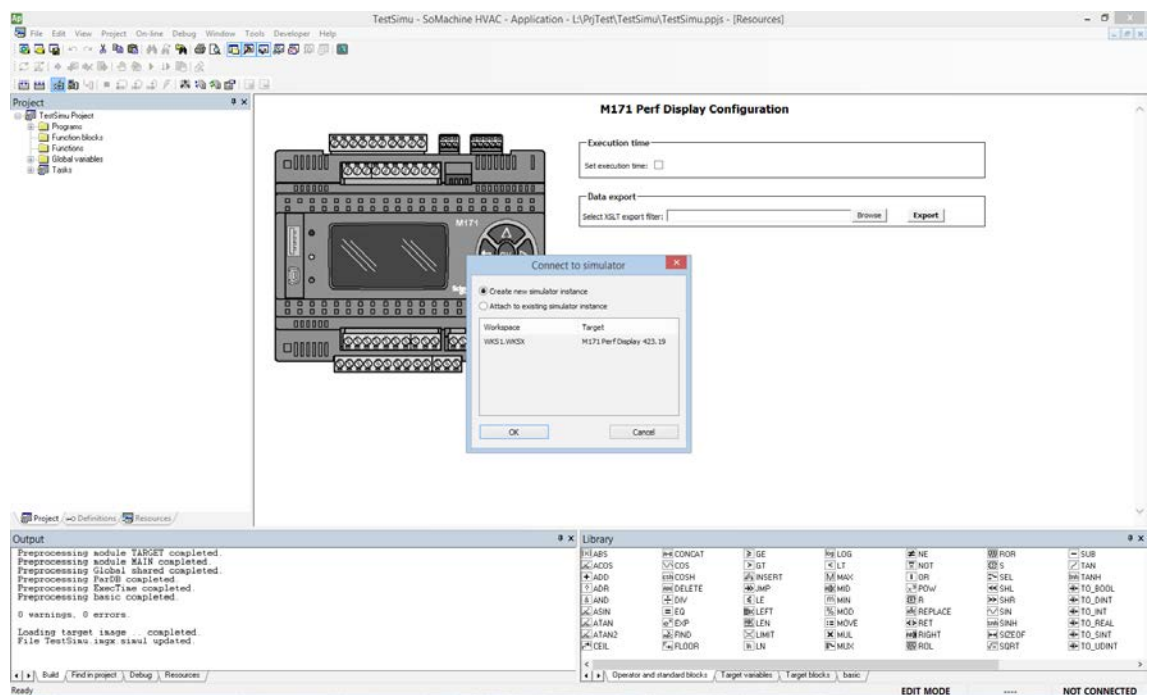
## 4. USING THE SIMULATOR

### 4.1 SIMULATION WITH APPLICATION

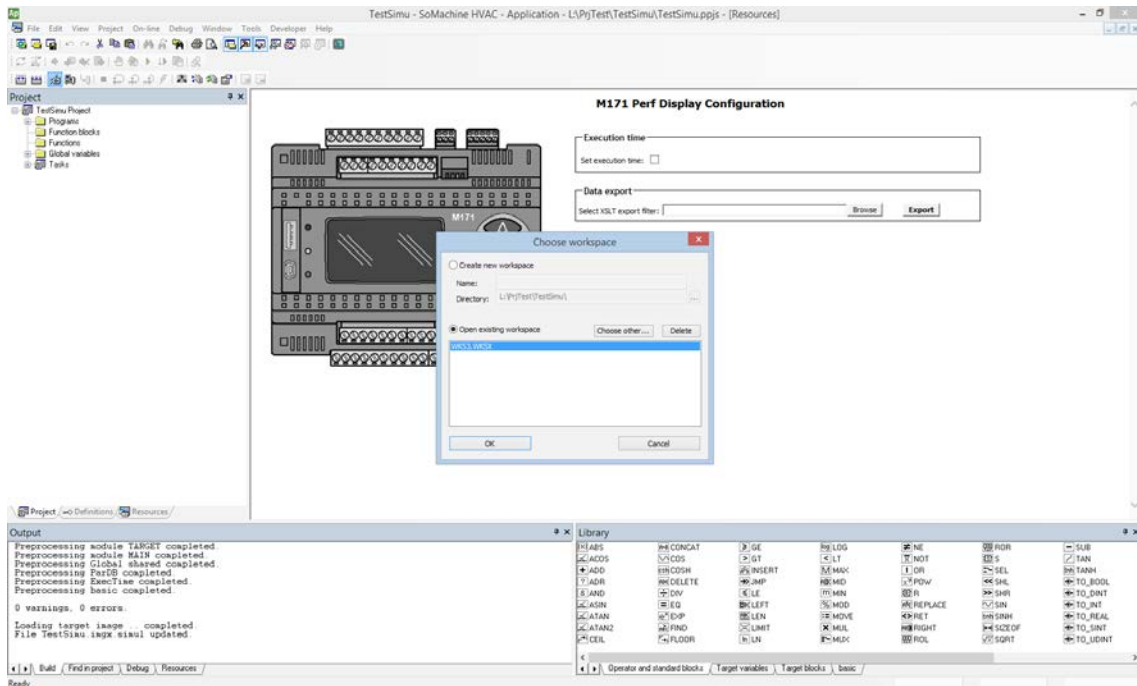
Application has a button in the toolbar that lets you activate a simulation session.

Follow the instructions below in order to carry out a simulation session:

- 1) Write your PLC code with Application.
- 2) Check the correctness of the code by compiling the project.
- 3) Activate the simulation with the appropriate button in the toolbar.
- 4) If there are already running instances of the simulator, Application will ask you if you want to create a new instance or attach to an existing compatible instance, showing a list of them; if there are no running instances, a new instance will be automatically run.



- 5) You can choose to open a recently used simulator workspace (WKSX) or create a new one if it is the first simulation session with this PLC project; the last used workspace will be then proposed as the default choice. The list of all used workspaces is saved inside the PLC project itself.



- 6) Application will choose the right simulation target file (TGSX) from the Catalog, depending on the target of the current PLC project.
- 7) Application can now activate the simulation status, that will be similar to the normal connection to a physical target device, with a different connection status indicator. While in simulation status, the PLC project will be built for the x86 processor and the connection will take place using the GDB protocol over TCP/IP on the localhost (127.0.0.1).
- 8) Then you can compile and download the code inside the simulated target.
- 9) In Application you can debug the code as if you were connected the real target (watchwindow, triggers, breakpoints); it is worth to note that you will be able to debug with all Application debugging features, independently of the real target capabilities.
- 10) In Simulation you can operate in the target panel (if there is one) to simulate the local I/O.
- 11) In Simulation you can operate with the I/O panels to change values of the application parameters.
- 12) The simulation session is terminated when the user deactivates the simulation mode inside Application (and the simulator will be automatically closed) or the user manually closes Simulation (in this case the communication in Application will go in the timeout state, as in the real situation when the physical target is powered off or disconnected).
- 13) When Simulation is closed everything will be saved inside the current workspace (I/O panels, window positions, etc.).
- 14) Application will save the list of recently used workspaces inside the PLC project for further use.



## 4.2 SIMPLIFIED SIMULATION WITH APPLICATION

The simplified simulation is almost identical to the full simulation, except for the following points:

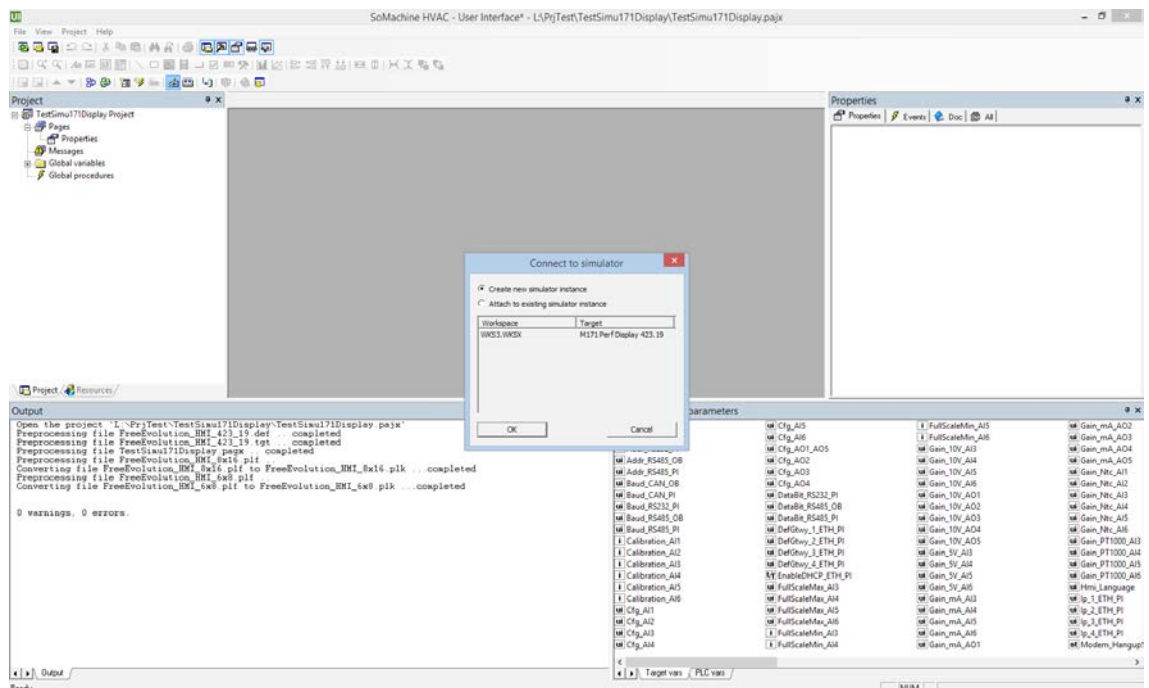
- 6) Because the TGSX file is not available in the Catalog, Application will self-generate a temporary one by examining the PLC project and its TGT and IMG files, and pass it to the simulator.
- 10) The target panel will not be available, you can interact with I/O panels and/or watch-window only.

## 4.3 SIMULATION WITH USERINTERFACE

UserInterface has a button in the toolbar that lets you activate a simulation session, like in Application.

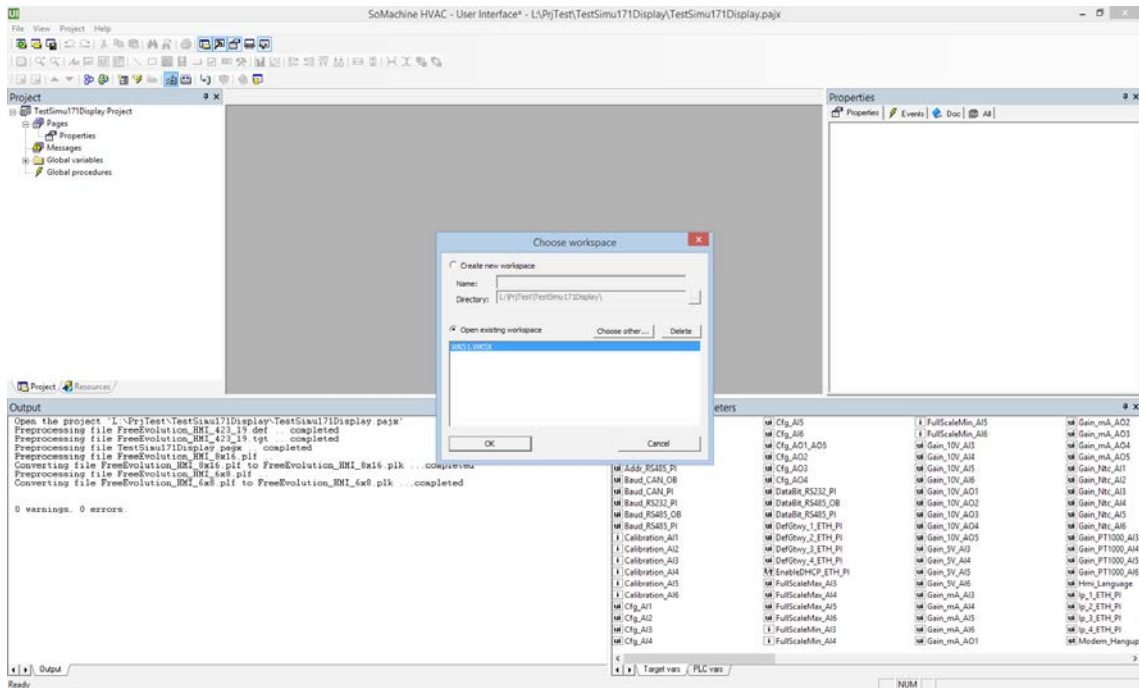
Perform the following steps for a simulation session, very similar to Application ones:

- 1) Write your HMI pages with UserInterface.
- 2) Check the correctness of the code by compiling the project.
- 3) Activate the simulation with the appropriate button in the toolbar.
- 4) If there are already running instances of the simulator, UserInterface will ask you if you want to create a new instance or attach to an existing compatible instance, showing a list of them; if there are no running instances, a new instance will be automatically run.





- The user can choose to open a recently used simulator workspace (WKSX) or create a new one if it is the first simulation session with this HMI project; the last used workspace will be then proposed as the default choice. The list of all used workspaces is saved inside the HMI project itself.



- UserInterface will choose the right simulation target file (TGSX) from the Catalog, depending on the target of the current HMI project.
- UserInterface can now activate the simulation status. While in simulation status, the HMI project will be built for the x86 processor and the connection will take place using the GDB protocol over TCP/IP on the localhost (127.0.0.1).
- You can then compile and download the code inside the simulated target.
- In Simulation you can operate on the target panel and work on the pages with the mouse and the keyboard, and operate on the local I/O.
- In Simulation you can operate with the I/O panels to change values of the application parameters.
- The simulation session is terminated when the user deactivates the simulation mode inside UserInterface (and the simulator will be automatically closed) or the user manually closes Simulation (in this case next downloads will go in the timeout state, as in the real situation when the physical target is powered off or disconnected).
- When Simulation is closed everything will be saved inside the current workspace (I/O panels, window positions, etc.).
- UserInterface will save the list of recently used workspaces inside the HMI project for further use.



### 4.4 SIMULATION WITH BOTH APPLICATION AND USERINTERFACE

It is possible to use Simulation with a simultaneous connection to both Application and UserInterface; to do so it is required that the target is the same for both PLC and HMI projects: in this case the second program will connect to the same instance without running a new simulator.

Otherwise, if the targets are different it will be necessary to run two different simulator instances, each one connected to a different program (one to Application and one to UserInterface).

When all the connected clients (Application and/or UserInterface) will be disconnected from the simulator, it will be automatically closed.



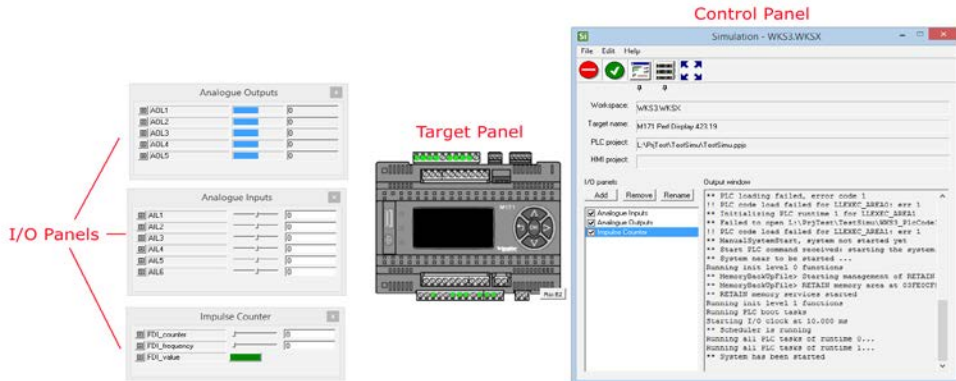




## 5. PROGRAM INTERFACE

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

The following picture shows you the main windows.

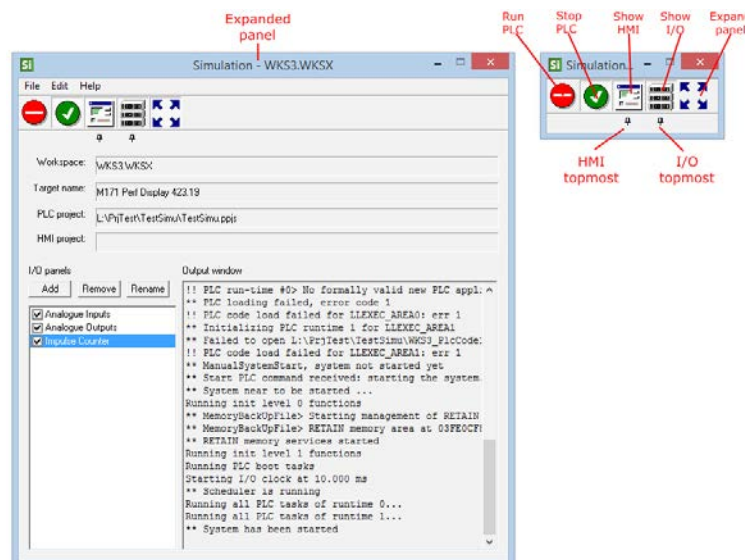


### 5.1 CONTROL PANEL

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

When you click the *Expand* button, it will be expanded to show the *Menu* bar with the standard *new/load/save/exit* commands, a central panel showing the main characteristics of the current workspace, an output window showing execution logs, and the I/O panels list.

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



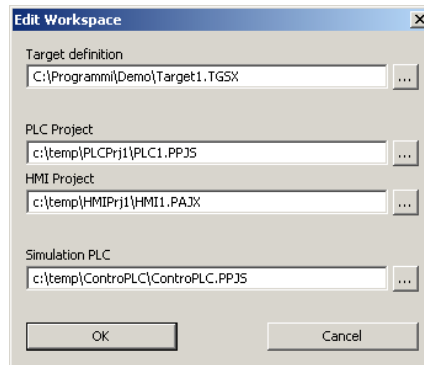


## 5.1.1 MANUAL WORKSPACE EDITING

With the menu command *Edit/Edit workspace...* you can manually edit your workspace, explicitly by inserting which TGSX file to use by choosing it from disk, which PLC and HMI projects should be simulated, and eventually a "simulation PLC".

This is a special type of PLC application used to simulate the behaviour of the real hardware, that shares the same data-blocks of the PLC to simulate to react to its actions as the real target would do.

Each time the simulator is run, the "simulation PLC" is compiled and downloaded along with the main application.



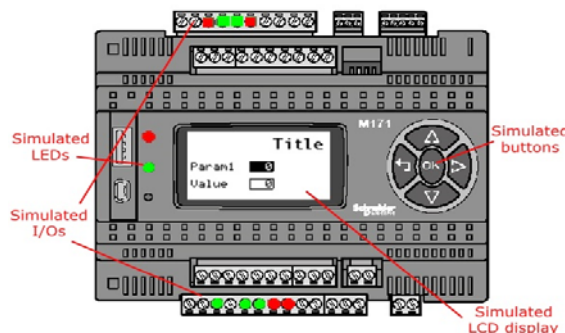
## 5.2 TARGET PANEL

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

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

The user can interact with this window with the mouse or with the PC keyboard, that emulates the real device keys.

This window is activated with the proper button in the control panel; you can right-click on it to open its context menu, by which you can activate the *topmost* state (it will stay always above any other window) or close it. Finally, you can drag and move it around the screen anywhere you want.





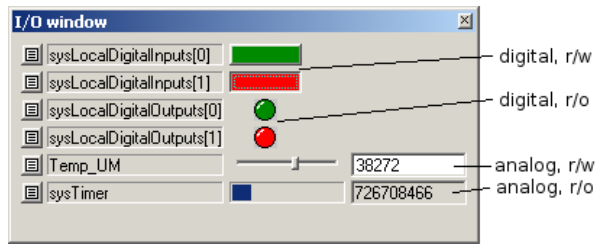
## 5.3 I/O PANELS

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

The user can create how many panels he wants, and decide which objects to put on each panel freely; they are complementary to the *target* panel, because with them you can watch and edit the I/Os that are not already visible there.

The I/O panels can be put in *topmost* mode, that is always above all visible windows; this is useful for example while debugging with Application at full screen; all the configuration is then saved inside the workspace file.

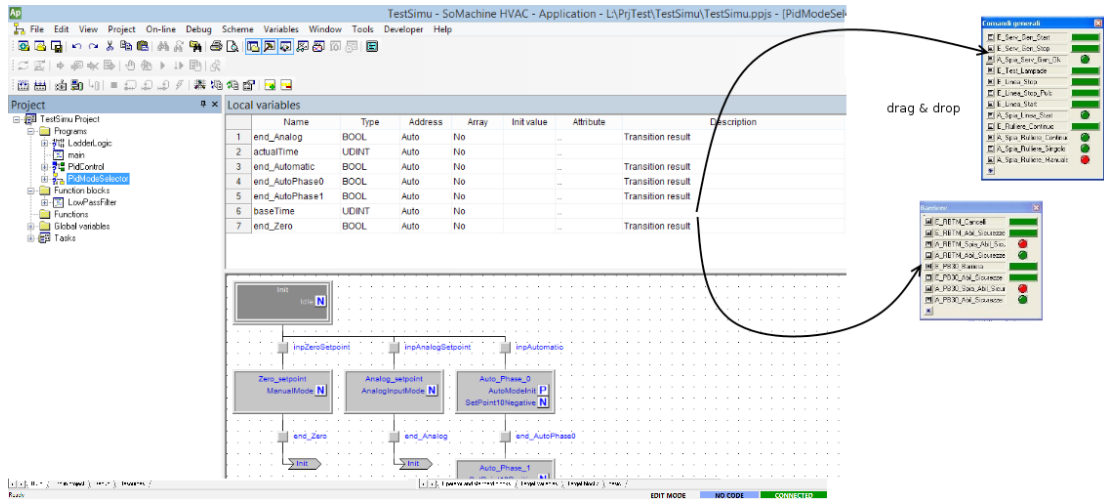
See the section I/O panels list to see how to add and delete panels.



### 5.3.1 ADDING ELEMENTS TO PANELS

To add an element (or "signal") to an existing panel to watch or edit its value, you can drag&drop it from Application inside the panel itself. You can drag it from the *Target variables* panel, from the *Workspace* tree or from a *Variables* grid inside an editor.

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



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



## 5.3.2 EDITING PANEL ELEMENTS

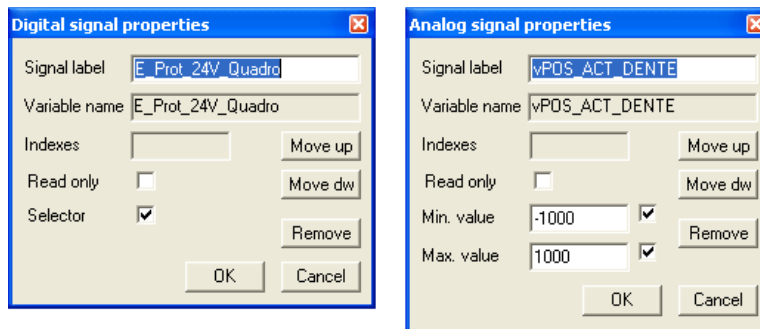
You can edit the more advanced options of each signal by clicking the small icon on the left of each name.

For a digital I/O the options are:

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

For an analog I/O the options are:

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



## 5.3.3 REMOVING ELEMENTS FROM PANELS

To remove a signal, click on the small button described above for the editing and then choose the *Remove* button to delete it (see the above images).

## 5.4 I/O PANELS LIST

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

### 5.4.1 ADDING A NEW PANEL

To add a new empty panel click the apposite *Add* button in the *Control* panel. A new empty panel without name will be created and placed next to the button.

### 5.4.2 EDITING A PANEL

In order to rename the panel select it in the list and click the *Rename* button; you will be asked for the name to give to the window, that will be shown in its title bar.

Any panel can be drag around the screen in any place; to temporary hide it you can toggle



the check next to its name in the list.

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

### 5.4.3 REMOVING A PANEL

In order to remove a panel select it in the list and click the *Remove* button; the panel and all its signals and settings will be permanently removed.



