USER MANUAL

LONGO Programmable Controller
LPC Manager

Version 7

User manual
Document Version: 7
June, 2025

Index

LONGO Programmable Controller LPC Manager

1 ABOUT THIS DOCUMENT....1

1.1 Who Should Read This Document....1
1.2 Terminology....2
1.2.1 LPC Family products based terminology....2
1.2.2 LPC Manager based terminology....3
1.2.3 Conventions used in this document....3

2 LPC MANAGER SOFTWARE....4

2.1 Introduction....4
2.2 LPC Manager editor....5
2.3 Main menu....5
2.3.1 File....5
2.3.2 Edit....6
2.3.3 Display....6
2.3.4 Help....6
2.4 Toolbars....7
By pressing the "Stop" button, the connected controller will stop all controller processes. 8
2.5 Project window....12
2.5.1 Build - Transfer procedure....12
2.5.2 Project (program structure window)....13
Add new (DataTypes, Functions, Function Blocks, Programs, Resources and Extensions)....13
Right-click menu (Project name)....14
Right-click menu (DataTypes, Functions, Function Blocks, Programs and Resources)....14
Right-click menu (Function, Function block and Program)....15
Double-click (Project name, DataTypes, Functions, Function Blocks, Programs and Resources)....15
2.5.3 Project (instances window)....15
2.5.4 Editor workspace....16
2.6 POUs and extensions....17
2.6.1 Datatype....17
2.6.2 Function....18
2.6.3 Function block....18
2.6.4 Program....19

2.6.5 Resources....................20
2.6.6 C extension....................21
2.6.7 Python file....................21
2.6.8 Smarteh HMI....................22
2.6.9 Bacnet support....................23
2.6.10 Modbus....................24
2.6.11 MQTT....................28
2.6.12 CanOpen....................29
2.6.13 SVGHMI....................30
2.7 Library....................36
BLOCK PROPERTIES....................36
2.7.1 Standard function blocks....................37
SR....................37
RS....................37
SEMA....................38
R TRIG....................38
F TRIG....................38
CTU CTU_DINT, CTU_LINT, CTU_UDINT, CTU_ULINT....................39
CTD CTD_DINT, CTD_LINT, CTD_UDINT, CTD_ULINT....................39
CTUD CTUD_DINT, CTUD_LINT, CTUD_UDINT, CTUD_ULINT....40
TP....................40
TON....................41
TOF....................41
2.7.2 Additional function block....................43
RTC....................43
INTEGRAL....................43
DERIVATIVE....................43
PID....................44
RAMP....................44
HYSTERESIS....................44
2.7.3 Type conversion....................45
TYPE[A]_TO_TYPE[B]....................45
2.7.4 Numerical....................46
ABS....................46
SQRT....................46
LN....................46
LOG....................46
EXP....................46
SIN....................46
COS....................48
TAN....................48
ASIN....................48
ACOS....................48
ATAN....................48

2.7.5 Arithmetic....49
ADD....49
MUL....49
SUB....49
DIV....49
MOD....50
EXPT....50
MOVE....50
2.7.6 Time....51
ADD....51
ADD_TIME....51
ADD....51
ADD_TOD_TIME....51
ADD....52
ADD_DT_TIME....52
MUL....52
MULTIME....52
SUB_TIME....52
SUB....52
SUB....52
SUB_DATE_DATE....53
SUB....53
SUB_TOD_TIME....53
SUB....53
SUB_TOD_TOD....53
SUB....53
SUB_DT_TIME....53
SUB....54
SUB_DT_TIME....54
DIV....54
DIVTIME....54
2.7.7 Bit-shift....55
SHL....55
SHR....55
ROR....55
ROL....55
2.7.8 Bitwise....56
AND....56
OR....56
XOR....56
NOT....56
2.7.9 Selection....57
SEL....57
MAX....57

MIN........57
LIMIT........58
MUX........58
2.7.10 Comparison........59
GT........59
GE........59
EQ........59
LT........60
LE........60
NE........60
2.7.11 Character string........61
LEN........61
LEFT........61
RIGHT........61
MID........61
CONCAT........62
CONCAT_DAT_TOD........62
INSERT........62
DELETE........62
REPLACE........62
FIND........63
2.7.12 Native POUs........63
LOGGER........63
2.7.13 Python POUs........64
csv_read_by_string........64
csv_reload........64
csv_read_by_int........65
python_eval........65
python_poll........65
python_gear........65
2.7.14 RTC POUs........66
getRTCString........66
getRTC........67
setRTC........67
2.7.15 User-defined POUs........68

2.8 Debugger....68
2.9 Search....69
2.10 Console....69
2.11 PLC Log....69

3 PROGRAMMING LANGUAGES....70

3.1 IL - Instruction List....70
3.2 ST - Structured Text....71
3.3 FBD - Function Block Diagram....73
3.4 LD - Ladder Diagram....74
3.5 SFC - Sequential Function Chart....75

APPENDIX A – ERROR REPORTING....77

APPENDIX B – LICENSE....78

APPENDIX C – DOCUMENT HISTORY....79

1 ABOUT THIS DOCUMENT

LPC Manager user manual describes how to use application *LPC Manager*.

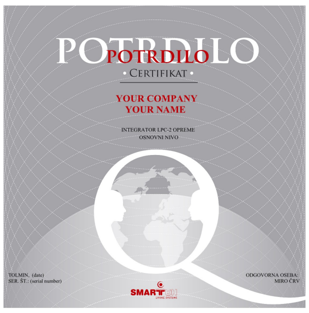
1.1 Who Should Read This Document

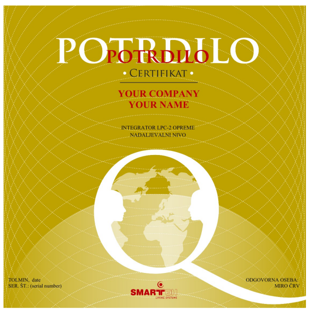
If you are new to the *LPC Manager* software and want to get started with it or upgrading from previous versions then You should read this document.

In order to properly understand this document, it is necessary to understand at least basics of the *LPC* hardware. For this it is highly recommended to view *LPC Getting Started* video (installed with *Smarteh IDE setup*) or to go through the proper training to a certified SMARTEH d.o.o. trainer. On this short course you will learn all the basics and have the opportunity to become a certified SMARTEH d.o.o. integrator. Not only you will get a head start, but you will also become a part of the growing group of LPC users, get a chance to ask direct questions to experts, get contacts with other integrators and companies all over the world...

NOTE: Since *LPC Manager* is based on *IEC 61131-3 international standard,* please refer to *PLC* (programming logic controllers) programming languages *International Standard IEC 61131-3 f*or more detailed information.

For more information, call *+386 5 388 44 00* or e-mail to *info@smarteh.si* and reserve your ticket for the LPC certificate training:


POTRDILO
POTRDILO
• CERTIFIKAT •
YOUR COMPANY
YOUR NAME
INTEGRATOR LPC-2 OPREME
OSNOVNI NIVO
TOLMIN, (date)
SER. ŠT.: (serial number)
ODGOVORNA OSEBA:
MIRO ČRV


POTRDILO
POTRDILO
• CERTIFIKAT •
YOUR COMPANY
YOUR NAME
INTEGRATOR LPC-2 OPREME
NADALJEVALNI NIVO
TOLMIN, date
SER. ŠT.: (serial number)
ODGOVORNA OSEBA:
MIRO ČRV

1.2 Terminology

Throughout this manual, various phrases are used. Here is a description of some of them.

1.2.1 LPC Family products based terminology

LONGO™

... is a family of products (hardware and software) and is a trademark of SMARTEH d.o.o.

LPC™

... Longo Programming Controller and is a trademark of SMARTEH d.o.o.

IDE

… is a **integrated development environment.**

LHC-2 Programmable Controller

... is a hardware module MW1

LPC-2 Programmable Controller

... is a hardware module MC9

LPC-3 Programmable Controller

... is a family of hardware modules GOT

Smarteh IDE, LPC Manager
... are members of LPC family software.

1.2.2 LPC Manager based terminology

Beremiz

… is a free software framework for automation (http://www.beremiz.org)

IEC 61131-3

… is an international standard for programmable controllers, programmable languages

PLC

… Programmable Logic Controller

MCU

… Main Control Unit

POU

… Program Organization Unit

Programming Languages

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

True

… Logical 1, on, active, high state

False

… Logical 0, off, not active, low state

1.2.3 Conventions used in this document

Items appearing in this document are sometimes given a special appearance to set them apart from the regular text. Here's how they look:

Italic
Used for marking important keywords.

NEW:
Used to mark sections which changed most from previous versions.

2 LPC MANAGER SOFTWARE

2.1 Introduction

LPC Manager software (LONGO Programmable Controller Manager software) is a product that is used for programming LPC-2, LHC-2 and LPC-3 family of Smarteh controllers.

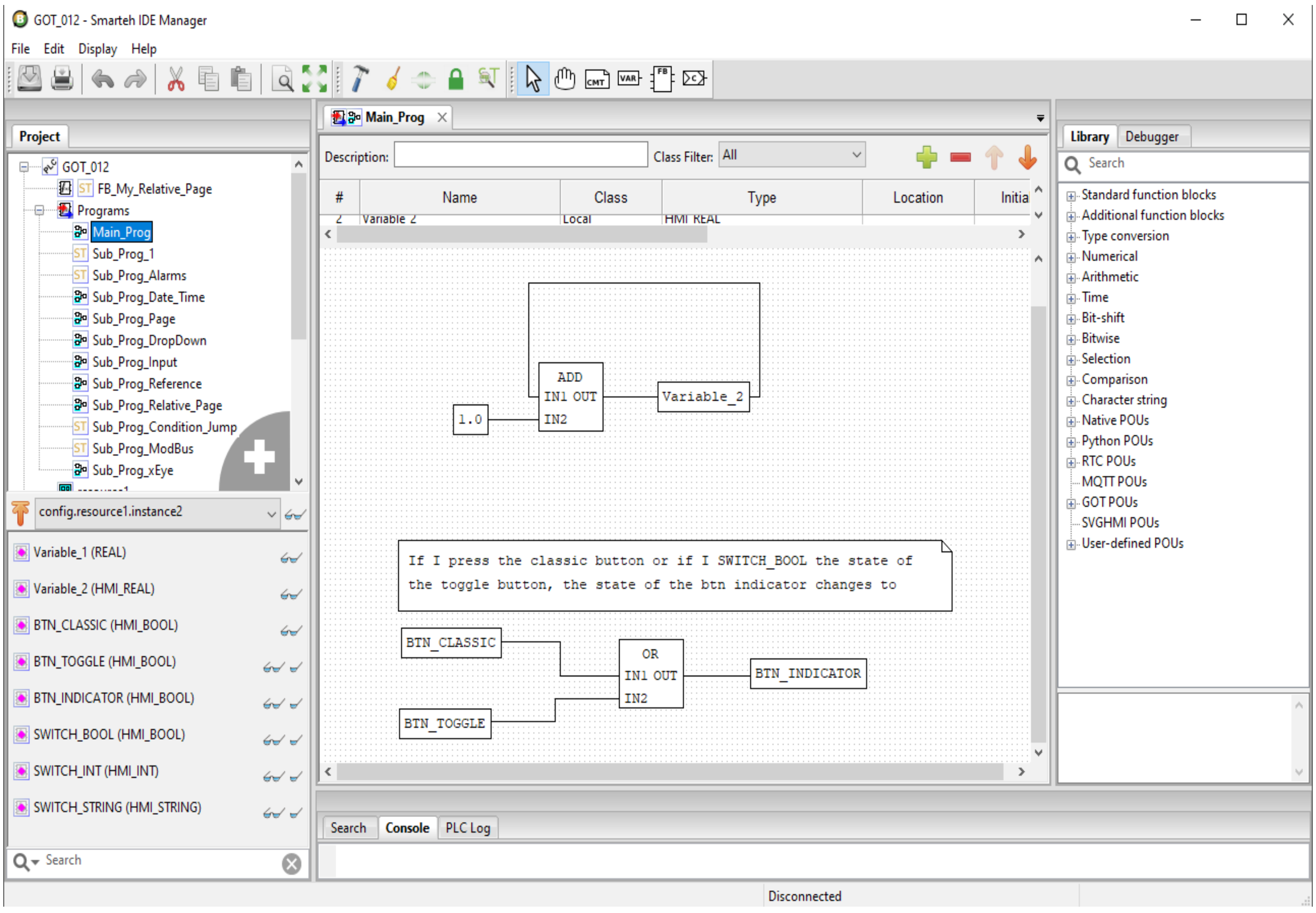
After creating a configuration in Smarteh IDE, LPC Manager can be started by:

– click on *Program* button, or

– select a project configuration and then click on *Program in LPC Manager* command line.

Software is based on Beremiz open source software adapted to Smarteh LPC controllers, which supports IEC 61131-3 standard programming languages IL(instruction list), ST(structured text), LD(ladder diagram), FBD(function block diagram) and SFC(sequential function chart).

LPC manager software is easy to use and offers many possibilities in programming, debugging, monitoring and trending LPC controllers application software.



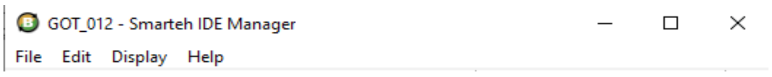
```
If I press the classic button or if I SWITCH_BOOL the state of
the toggle button, the state of the btn indicator changes to
```

2.2 LPC Manager editor

LPC Manager software consists of:

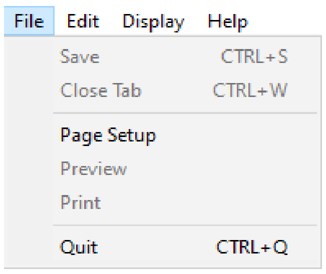
- *Main menu:* File, Edit, Display, Help
- *Tool bars:* Save, Print, Undo, Redo, Cut, Copy, Paste, Search in Project, Toggle full screen mode; Build, Clean project, Connect to controller; Select and object, Move the view, Create a new comment, Create a new variable, Create a new block, Create a new connection
- Project window
- *Variables and* Editor workspace window
- *Search, Console and PLC Log window*
- *Library and Debugging window*

2.3 Main menu



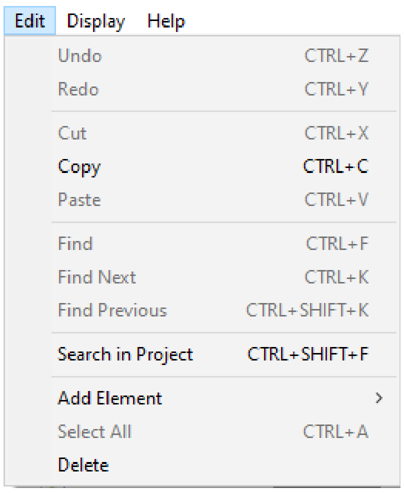
In main menu you can find options *File*, *Edit*, *Display*, and *Help*

2.3.1 File



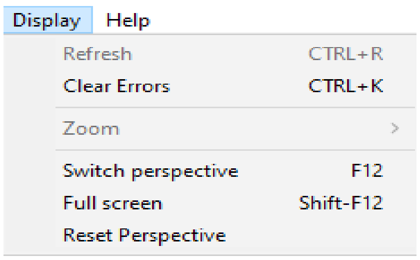
Save	-Save currently open workspace.
Close Tab	-Close currently open workspace.
Page Setup	-Setup page for printing.
Preview	-Printing preview of workspace.
Print	-Print currently open workspace.
Quit	-Exit *LPC Manager*.

2.3.2 Edit



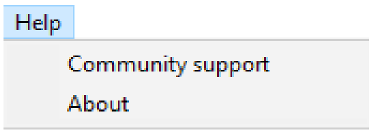
Undo	-Undo last change in workspace.
Redo	-Redo reverts the effects of the undo action.
Cut	-Cut the selected element in workspace.
Copy	-Copy the selected element in workspace.
Paste	-Paste (previously copied) element(s) in the workspace.
Find	-Search elements
Find Next	-Search next element
Find Previous	-Search previous element
Search in Project	-Search elements in the project.
Add Element	-Add element (Data Type, Function, Function Block, Program, Configuration) into an appropriate item under *Types* window.
Select All	-Select all elements in workspace.
Delete	-Delete element from workspace.

2.3.3 Display



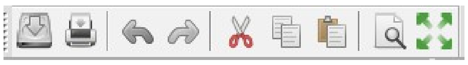
Refresh	-Refresh workspace.
Clear Errors	-Clear program errors.
Zoom	-Window zoom settings (12 .. 283%).
Switch perspective	-Hides/shows windows.
Full screen	-Enters/exits full screen.
Reset Perspective	-Reset program windows to default.

2.3.4 Help



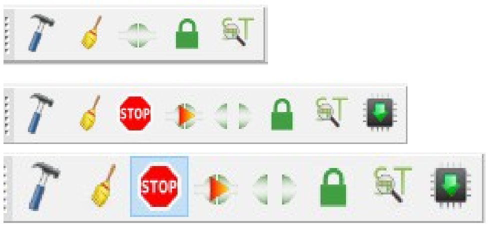
Community support	-Community support information.
About	-Main information about Beremiz.

2.4 Toolbars



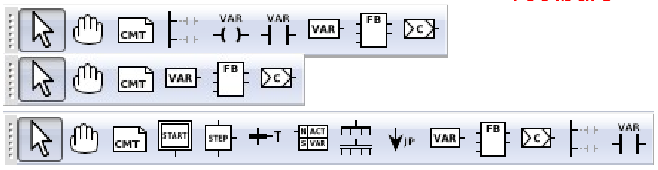
Toolbar1

-Standard windows icons (Save, Print, Undo, Redo, Cut, Copy, Paste, Search in Project, Toggle full screen mode).



Toolbar2

-PLC related execution functions (set of elements depends on the connection and controller state).



Toolbar3

-Main graphical icons for editor workspace elements (set of elements depends on the selected programming language).

Toolbar2

Connect

When the URI location is set you can connect to the controller by pressing the "*connect*" button.

Disconnect

Press the "*Disconnect*" button to disconnect from the current connected controller.

Build project into build folder

Press the "*Build*" button to start building the project. The "Log Console" displays different building steps. The build results in an executable code, named as the project name. It's located in the build directory of the project.

Clean

The "*Clean*" button clears the current build directory of the project. If you try to clean already cleared project a warning will appear and nothing will happen.

Manage secure connections

Opens a new window for managing previous secure connections and adding certificates.

Show IEC code

Shows IEC code generated by PLCGenerator, only for inspection.

Transfer PLC

This and following commands are only available when PC is connected to the controller.

Press the “*Transfer*” button to transfer executable application code to the controller.

Start PLC

This command is active, when the connected controller is in STOP mode. By pressing the “*Run*” button, the controller will start to execute the application.

Stop running PLC

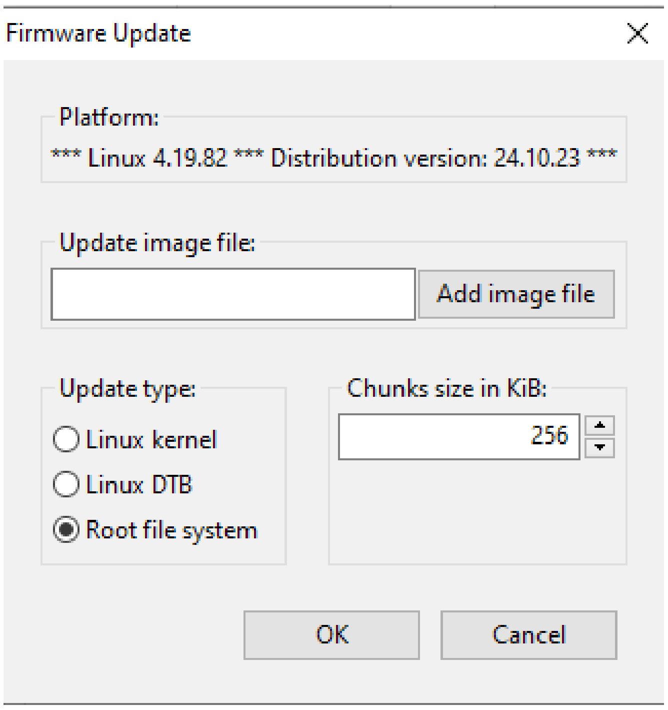
By pressing the “Stop” button, the connected controller will stop all controller processes.

Repair PLC

The repair button clears the project that is on the controller and then reboots it. Use this in case a corrupted project was transferred.

Update PLC firmware

Opens a new window for updating the connected controller.


Firmware Update

Platform:
*** Linux 4.19.82 *** Distribution version: 24.10.23 ***

Update image file:

Update type:
- Linux kernel
- Linux DTB
- ● Root file system

Chunks size in KiB: 256

Under Platform current distribution version is shown. Updating the firmware version requires selecting update file and corresponding update type. Chunks size should not be changed. After the update controller needs to be rebooted.

Toolbar3

Select an object

Standard tool to select one or more objects inside POU.

Move the view

Move current view inside POU to the desired direction.

Available for LD, FBD and SFC.

Create a new comment

Insert a new comment into POU.

Available for LD, FBD and SFC.

Create a new power rail

Insert a power rail (left or right) into POU.

Available for LD and SFC.

Create a new coil

Insert a coil into POU.

Available for LD.

Create a new contact

Insert a contact into POU.

Available for LD and SFC.

Create a new variable

Insert a variable into POU.

Available for LD and SFC.

Create a new block

Insert a block into POU.

Available for LD, FBD and SFC.

Create a new connection

Insert a connection into POU.

Available for LD, FBD and SFC.

Create a new initial step

Insert an initial step into POU.

Available for SFC.

Create a new step

Insert a new step into POU.

Available for SFC.

Create a new transition

Insert a transition into POU.

Available for SFC.

Create a new action block

Insert an action block into POU.

Available for SFC.

Create a new divergence

Insert a divergence into POU.

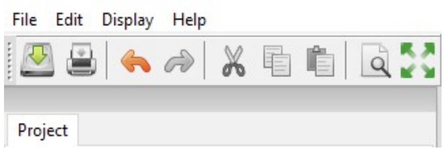
Available for SFC.

Create a new jump

Insert a jump into POU.

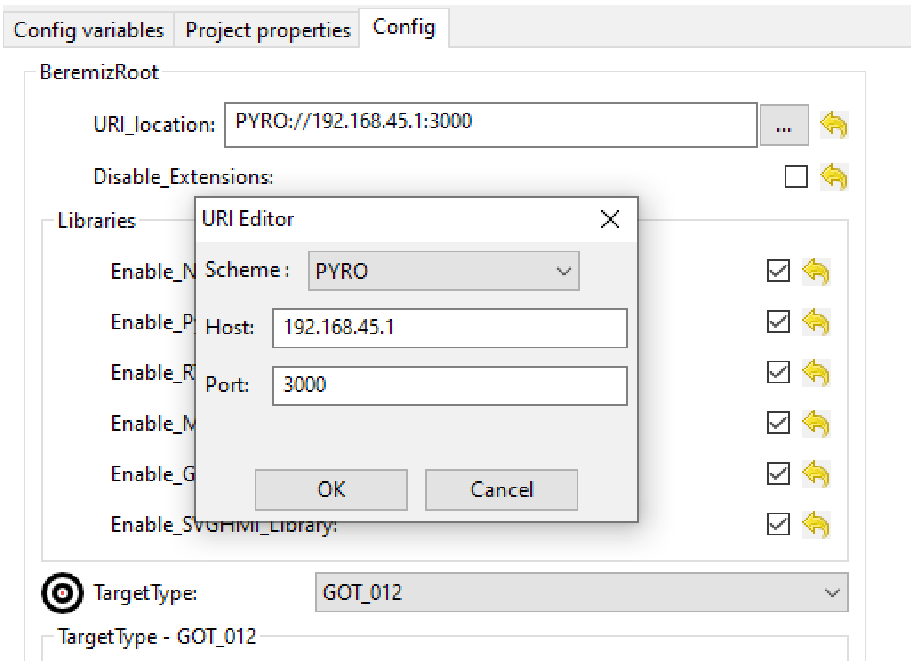
Available for SFC.

2.5 Project window



Project window -Group of windows which consist of Project described below.

2.5.1 Build - Transfer procedure


Config variables | Project properties | Config

BeremizRoot

URI_location: PYRO://192.168.45.1:3000

Disable_Extensions:

Libraries

Enable_N
Enable_P
Enable_R
Enable_M
Enable_G
Enable_SVGHMI_Library:

URI Editor

Scheme : PYRO

Host: 192.168.45.1

Port: 3000

OK Cancel

TargetType: GOT_012

TargetType - GOT_012

When the *LPC Manager* is opened for the first time (for the correspondent Smarteh IDE configuration) there are three buttons available inside Toolbar 2: *Connect*, *Build* and *Clean*. After the required application is created inside POU, then this application must be built. This procedure can be observed inside Console window. If the code is error free then the correspondent files are generated inside project folder.

The next step is to transfer the generated binary code inside a controller which must be connected. There are several connection types. To chose one select project(name) in Project tree and then the config tab. Double click on URI Location edit box and a window appears where you set the connection options.

LOCAL connection: Not used

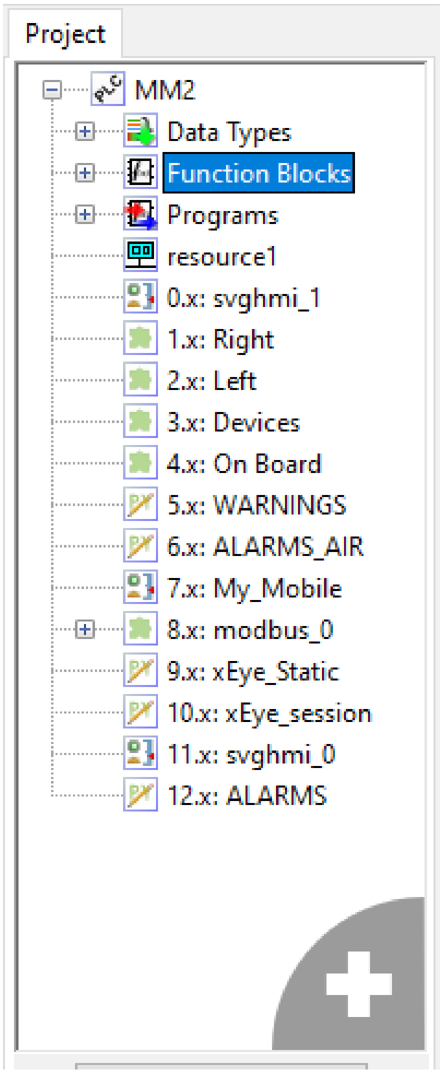
PYRO connection: Connect the USB port via USB programming cable (USB type A male to USB type B male). Select type ERPC connection and enter Host: 192.168.45.1 with Port: 3000.

PYROS connection: Secure connection to the controller in your local area network. You will need to know the IP, port and ID of the controller.

WAMP connection: Not used

WAMPS connection: Secure remote connection to the controller. You will need to know Crossbar router IP, port and controller WAMP ID. Realm must be set to Automation.

2.5.2 Project (program structure window)

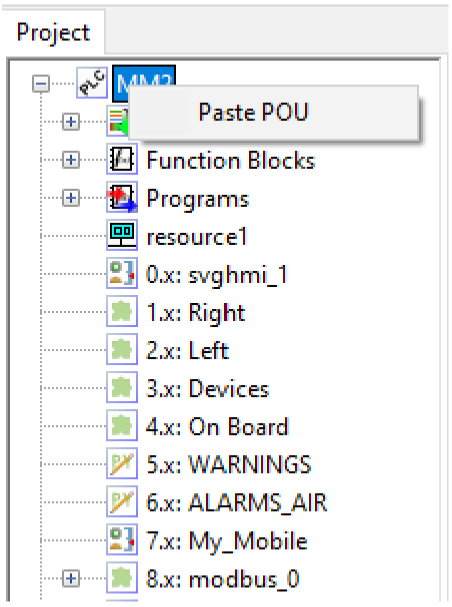


Project	-Main project properties and descriptions (*Project, Author, Graphics, Miscellaneous)*
Data Types	-User defined data types *(Directly, Subrange, Enumerated, Array, Structure).*
Functions	-User defined POU function *(IL, ST, LD, FBD)*
Function Blocks	-User defined POU function blocks *(IL, ST, LD, FBD, SFC).*
Programs	-POU programs *(IL, ST, LD, FBD, SFC).*
Resources	-Contains variable list (global variables), tasks and instances (for execution of the POU programs in the project).
Extensions	-C extenstion and Python files are used to create a file in corresponding language.
Smarteh HMI	-lpc_hmi is a GUI editor that only GOT.xx1 modules use it.
Bacnet support	-Bacnet variables and communication settings.
Modbus	-Modbus TCP/RTU variables and communication settings.
MQTT Client	-MQTT variables and settings
CanOpen	-CANopen variables and communication settings.
SVGHMI	-svg_hmi is a GUI editor that GOT.xx2 and MMx modules use it
Right	-Input and output modules variables.
Left	-Networking modules variables.
Devices	-Intelligent peripheral modules variables
On Board	-MCU on board variables.

Add new (DataTypes, Functions, Function Blocks, Programs, Resources and Extensions)

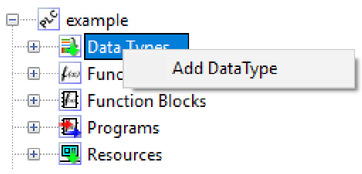
New POU	-New POU can be added by clicking on the big plus in the bottom right corner. POU type and programming language must be defined. A POU name can be changed.

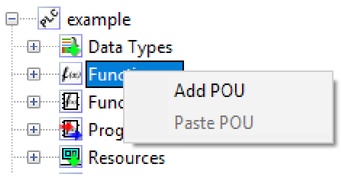
Right-click menu (Project name)



Paste POU	-Copied POU (e.g. Function block) can be pasted inside correspondent *Types* section. POU can also be imported from a text file (previously exported POU).

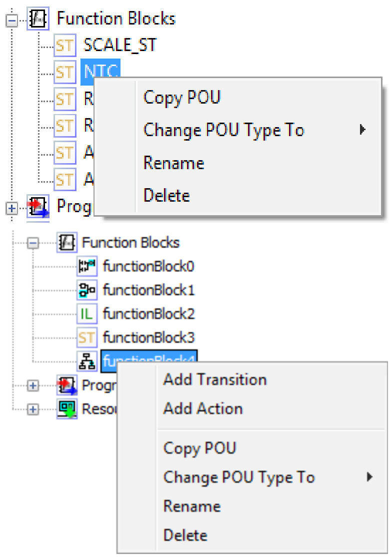
Right-click menu (DataTypes, Functions, Function Blocks, Programs and Resources)





Add DataType	-New *DataType* can be created.
Add POU and *Paste POU*	-New POU can be created. Programming language must be defined. A POU name POU type can be changed. -Copied POU (e.g. Function block) can be pasted inside correspondent *Types* section. POU can also be imported from a text file (previously exported POU).

Right-click menu (Function, Function block and Program)



Contains of the Right-click menu(Function block) depends on a programming language and type of POU.

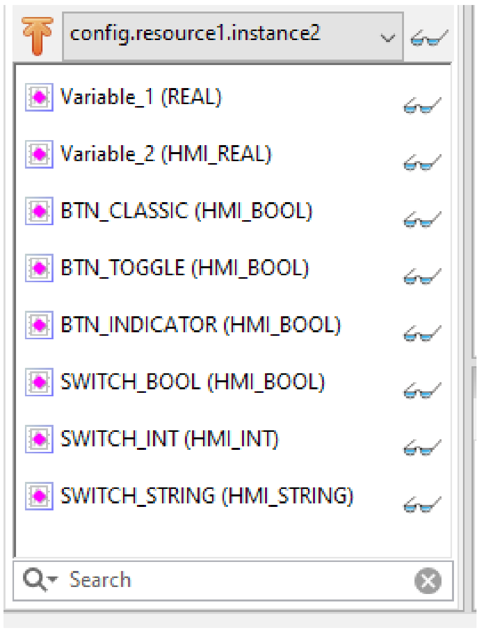
Add transition	-*(only for SFC language)*
Add action	-*(only for SFC language)*
Copy POU inside	-Function block can be copied (pasted) *Function Blocks* section.
Change POU Type To	-POU type of the selected Function can be changed to a Function block or *Program and Function block can be changed to Program*.
Rename	-Rename selected Function block.
Delete	-Delete selected Function block.

HINT: *The content of the copied Function block (clipboard) can be saved into a text file for backup or use in other LPC Manager applications.*

Double-click (Project name, DataTypes, Functions, Function Blocks, Programs and Resources)

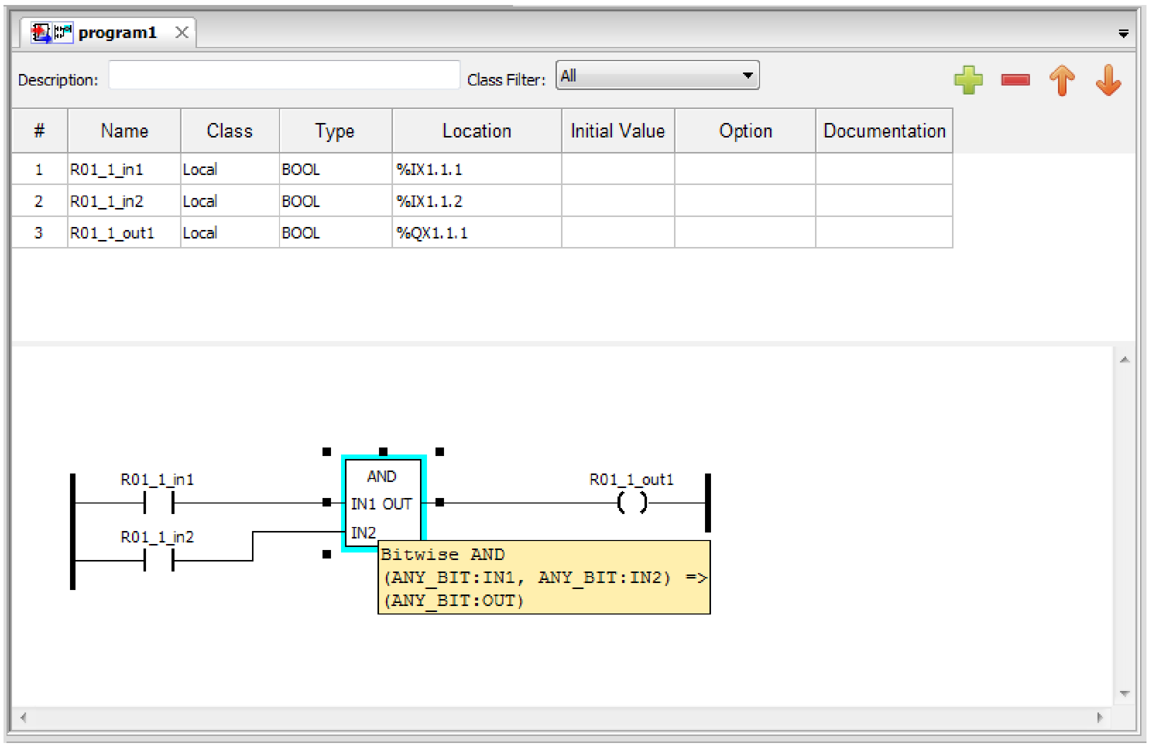
- Double-click on any data type or POU opens selected data type or POU in editor workspace.
- Double-click on project name opens a *Config variables* and *Project properties* in editor workspace.

2.5.3 Project (instances window)



Instances	-Contains image of built and transferred application to the target controller. It is used for graphical presentation in on line graphical debugging mode. All variables and internal logic connections can be animated in editing work space as logic structure, value in the list inside debugging window or in real time graphical trend window. Debugging can be started by clicking on glasses. Variables can be forced to the desired value.

2.5.4 Editor workspace



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	R01_1_in1	Local	BOOL	%IX1.1.1			
2	R01_1_in2	Local	BOOL	%IX1.1.2			
3	R01_1_out1	Local	BOOL	%QX1.1.1			

```
Bitwise AND
(ANY_BIT:IN1, ANY_BIT:IN2) =>
(ANY_BIT:OUT)
```

Editor workspace is used for editing, setting, programming and debugging of all elements in the project window (POUs, data types, configurations, resources, topology and other variables, instances,...). Edit elements are opened in a separate window from those that are listed in the workspace.

SHORTCUTS:	**CTRL+Scroll Up**	Zoom In
	CTRL+Scroll Down	Zoom Out
	CTRL+Down Arrow	Scroll toward bottom
	CTRL+Up Arrow	Scroll toward top
	CTRL+Right Arrow	Scroll toward right
	CTRL+Left Arrow	Scroll toward left

Double click the connection line between elements to optimize the connection line length (shortest line)

CTRL+C, CTRL+V Copy - Paste element(s) (click on the selected position before pressing CTRL+V; cursor has a shape as cross)

WARNING! *Usage of non-standard characters (e.g. č, š, ž,...) in editing fields can cause editor functioning problems.*

HINT: *By placing the cursor on the desired block then a name of the block and correspondent variables are shown inside a small pop-up window beside cursor.*

VARIABLES -Variables window contains all used variables of related POUs, providing a means of identifying data objects with its' elementary data type declaration.

SHORTCUT: **Double click** on the Type field opens a pop-up bar to select Base type or User Data type of the variable.

2.6 POUs and extensions

2.6.1 Datatype

Allows to create custom, more complex types for variables. The created type is then found under user data types when selecting type for a variable. There are different derivation types:

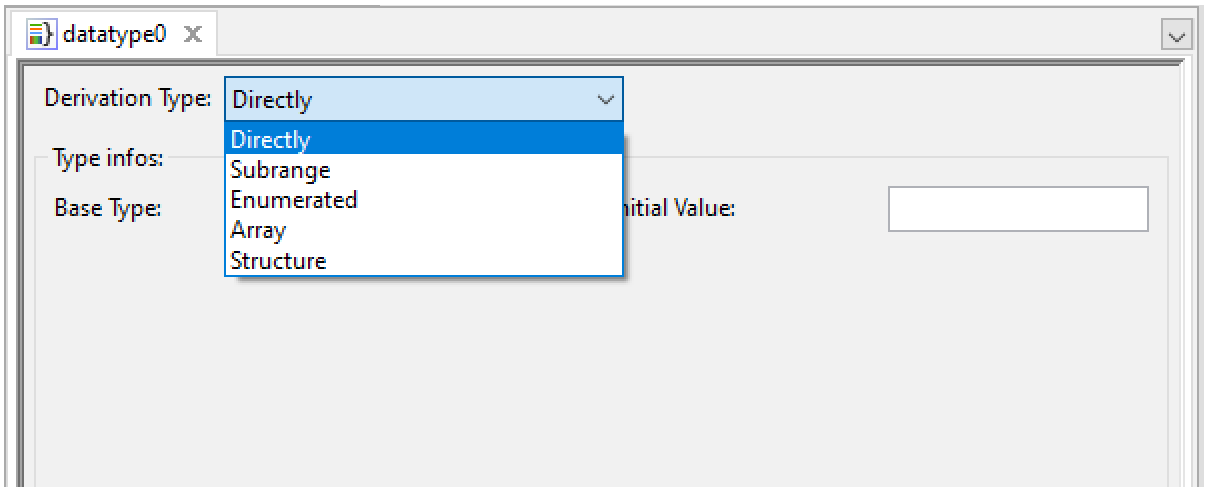
-*Directly*. Can set base type and initial value

-*Subrange*. Can set base type and initial value. Also allows to set range with minimum and maximum parameters.

-*Enumerated*. Can set multiple values which are enumerated and an initial value.

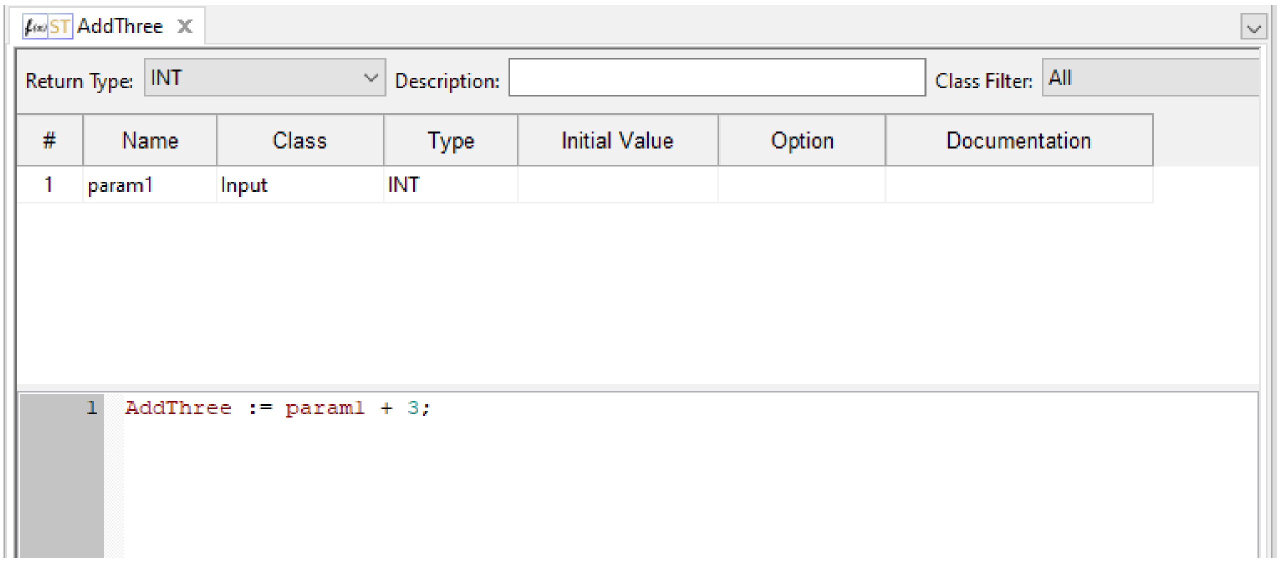
-*Array*. Can set the base type of values in the array, the dimensions (e.g. 1..10 is an array of indexes 1 to 10) and the initial value

-*Structure*. Can create a structure of different types of variables



2.6.2 Function

Allows the creation of custom functions. The output must have the same name as the function. The type of the output is set under return type. Supported languages are IL, ST, LD and FBD. The created function is found in the library under User-defined POUs.



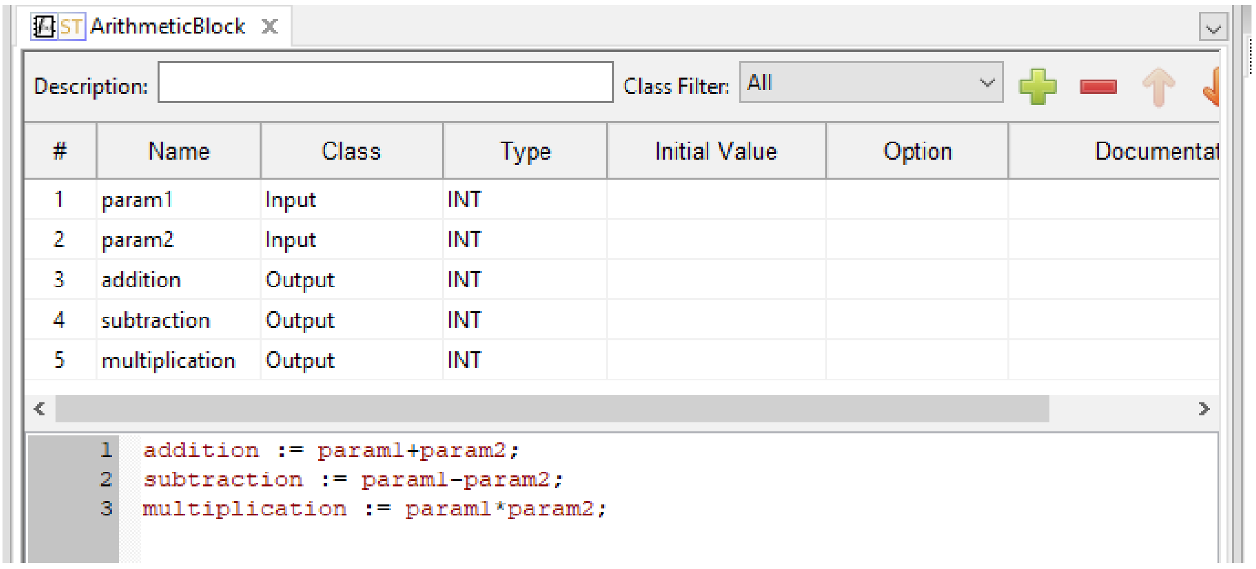
Return Type: INT Description: Class Filter: All

#	Name	Class	Type	Initial Value	Option	Documentation
1	param1	Input	INT			

```
AddThree := param1 + 3;
```

2.6.3 Function block

Allows the creation of custom function blocks, with multiple inputs and outputs. Supported languages are IL, ST, LD, FBD and SFC. The created function block is found in the library under User-defined POUs.



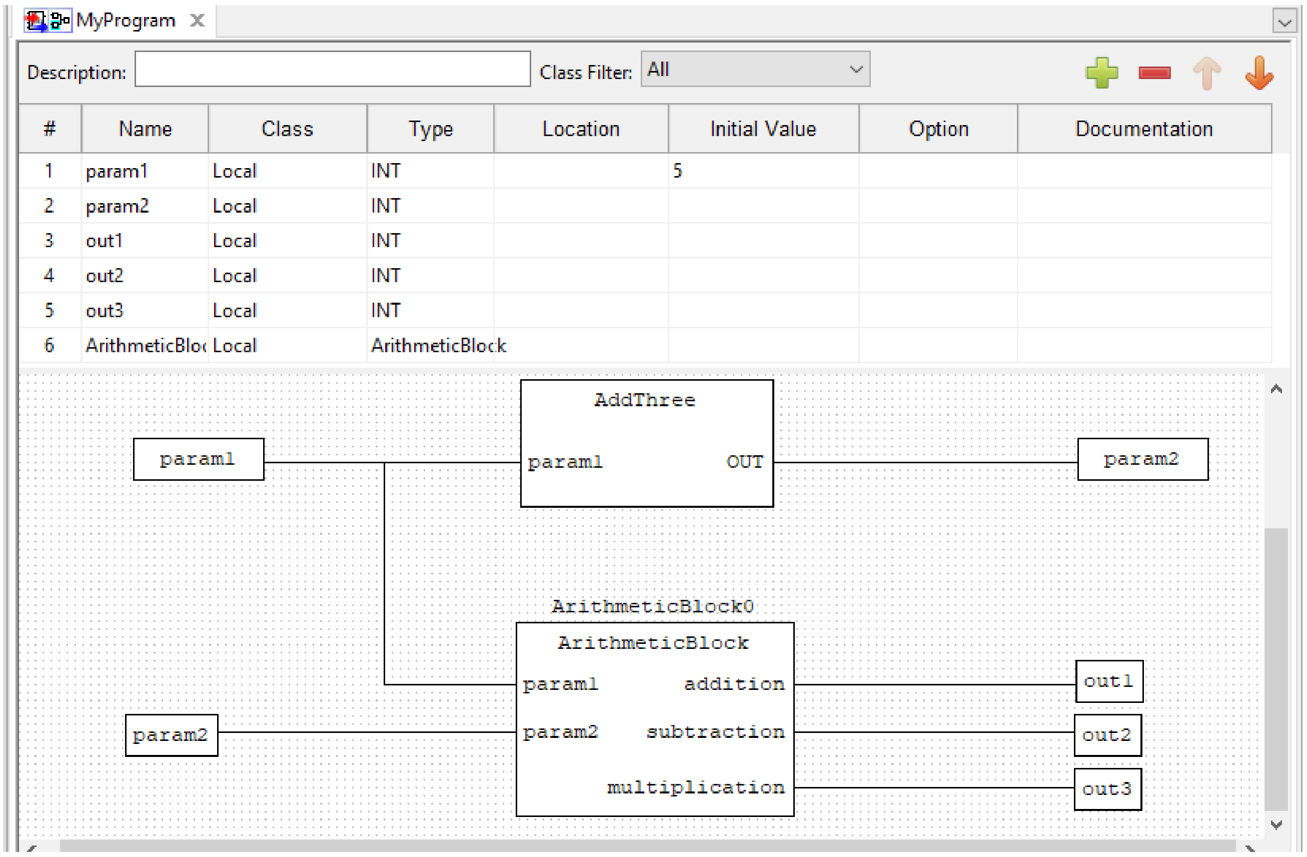
Description: Class Filter: All

#	Name	Class	Type	Initial Value	Option	Documentat
1	param1	Input	INT			
2	param2	Input	INT			
3	addition	Output	INT			
4	subtraction	Output	INT			
5	multiplication	Output	INT			

```
addition := param1+param2;
subtraction := param1-param2;
multiplication := param1*param2;
```

2.6.4 Program

Allows the creation of different programs that run on controllers. Supported languages are IL, ST, LD, FBD and SFC. Created programs must be configured in resources so that are processed on the controllers during runtime.



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	param1	Local	INT		5		
2	param2	Local	INT				
3	out1	Local	INT				
4	out2	Local	INT				
5	out3	Local	INT				
6	ArithmeticBloc	Local	ArithmeticBlock				

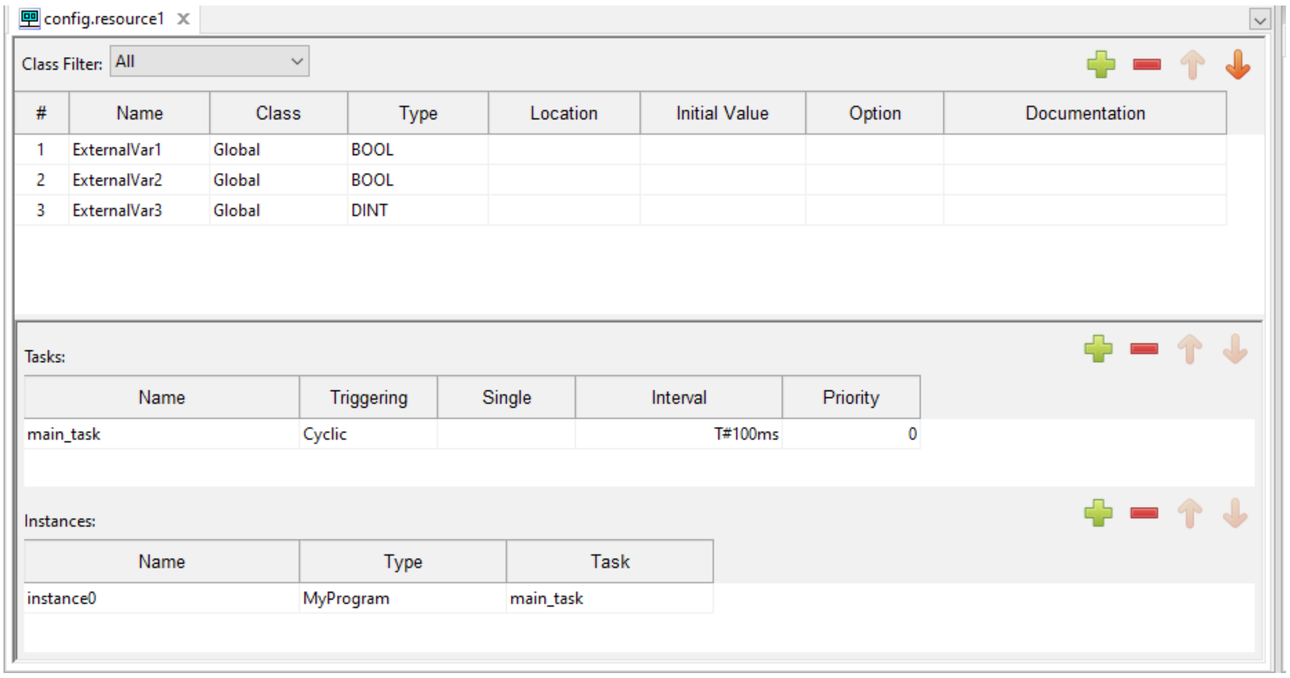
2.6.5 Resources

Resources are used for defining global variables and configuring programs. They are divided intro 3 sections.

The top section is for defining global variables. Variables defined here can be used in all POUs. The class must be set to Global

In the middle section tasks are defined. Their name can be defined along with the type of triggering, the interval and priority.

Instances are defined in the bottom section. Here you can set the tasks of individual programs.



config.resource1

Class Filter: All

#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	ExternalVar1	Global	BOOL				
2	ExternalVar2	Global	BOOL				
3	ExternalVar3	Global	DINT				

Tasks:

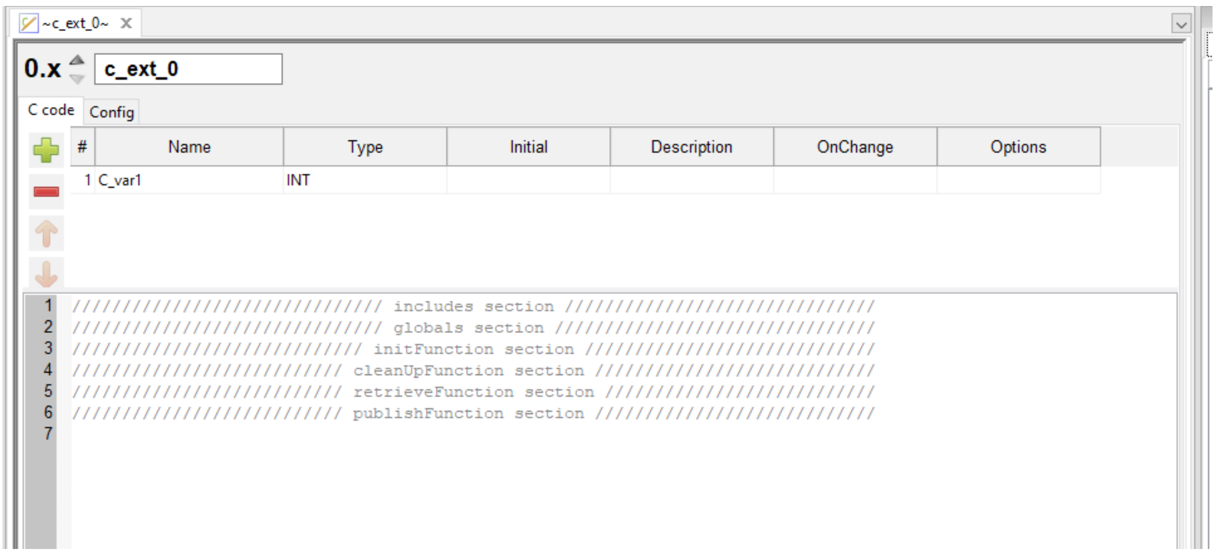
Name	Triggering	Single	Interval	Priority
main_task	Cyclic		T#100ms	0

Instances:

Name	Type	Task
instance0	MyProgram	main_task

2.6.6 C extension

C extension is used to create custom code using C programming language. All variables defined are treated as class global. Code execution depends on the section it is written in. C FLAGSs and LDFLAGS are defined in the config tab.

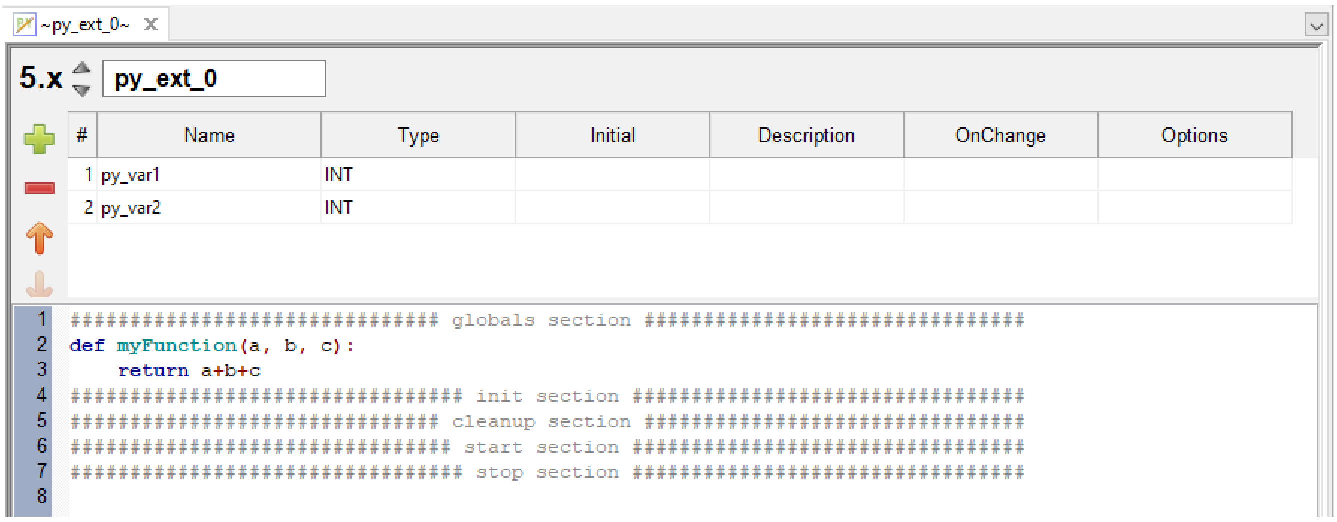


#	Name	Type	Initial	Description	OnChange	Options
1	C_var1	INT				

```
/////////////////////////////// includes section //////////////////////////////
/////////////////////////////// globals section ///////////////////////////////
////////////////////////////// initFunction section ////////////////////////////
//////////////////////////// cleanUpFunction section ///////////////////////////
/////////////////////////// retrieveFunction section ///////////////////////////
/////////////////////////// publishFunction section ////////////////////////////
```

2.6.7 Python file

Python extension is used to create custom code using python programming language. All variables defined are treated as class global. Code execution time depends on the section it is written in. Functions defined in the globals section can be called in PLC programs using python functions found in Python POUs in the library.



#	Name	Type	Initial	Description	OnChange	Options
1	py_var1	INT				
2	py_var2	INT				

```
############################### globals section ################################
def myFunction(a, b, c):
    return a+b+c
################################# init section #################################
############################### cleanup section ################################
################################ start section #################################
################################# stop section #################################
```

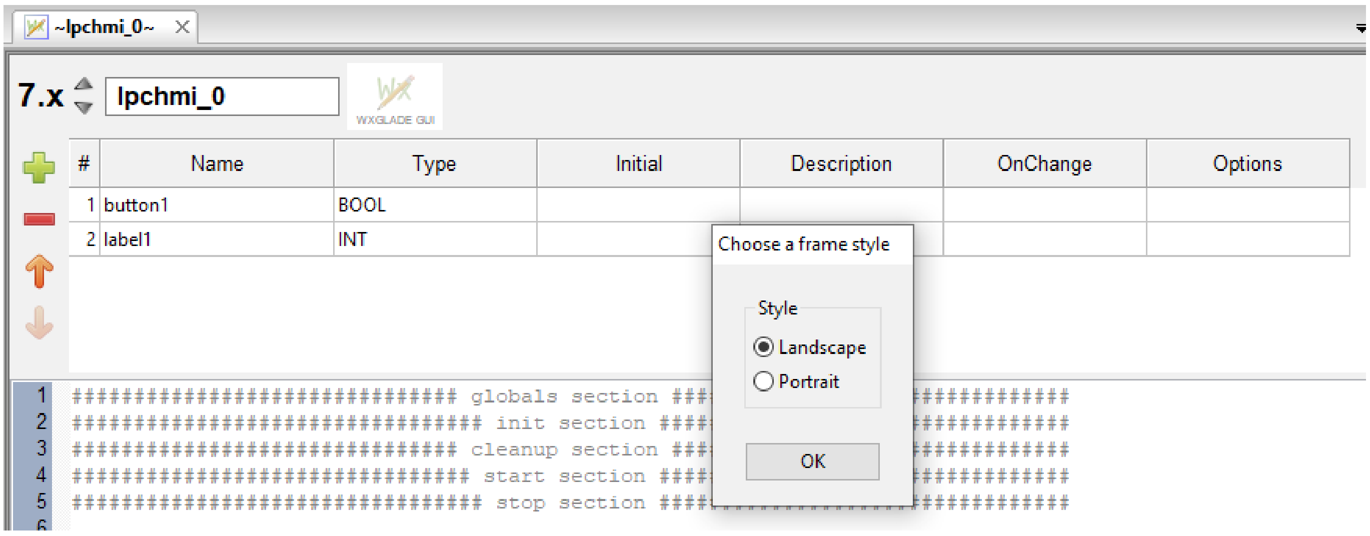
2.6.8 Smarteh HMI

Smarteh HMI extension is used for designing GUI on LPC-3.GOT.xx1 and LHC-2.GOT.111 modules. Variables can be defined in the top section of the extension, they are used to link and manage graphical components. Their class type is external so they can be used in other programs.

Bottom section can be used to create custom code using python programming language. Code execution time depends on the section it is written in.

WXGLADE GUI button at the top, opens the LPC GUI manager. When starting LPC GUI manager for the first time a choice to select the frame style appears. This option manages the orientation of the screen.

WARNING! Frame style can only be chosen on the first opening of LPC GUI manager. To change it later a new Smarteh HMI extension needs to be created and the current one deleted.



#	Name	Type	Initial	Description	OnChange	Options
1	button1	BOOL				
2	label1	INT				

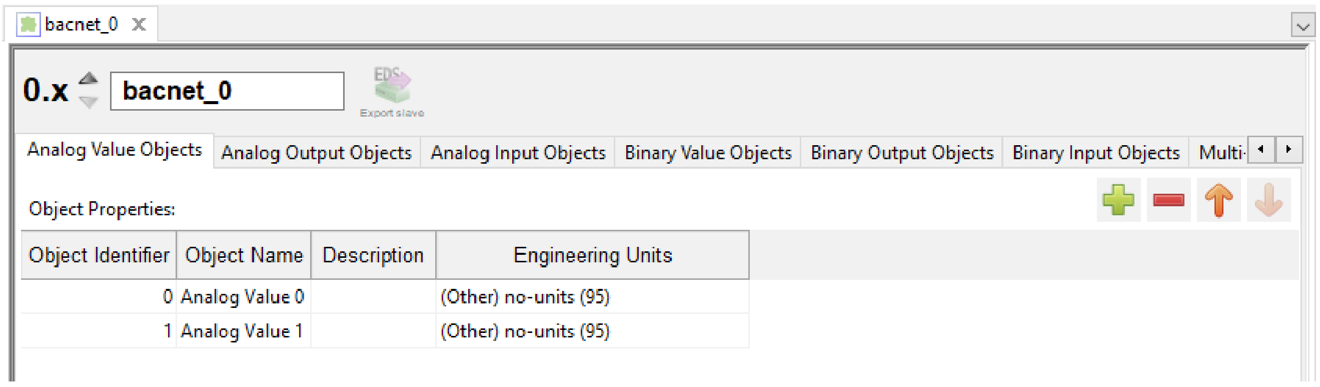
Style: [● Landscape] ○ Portrait

```
################################ globals section ###
################################## init section ####
################################ cleanup section ###
################################# start section ####
################################## stop section ####
```

For more on how to use LPCHMI, video tutorials can be found on Smarteh main page - support&downloads - video - tutorials - LPC-3.GOT.XX1 LCD PLC HMI programming tutorials.

2.6.9 Bacnet support

BACnet support extension is used for communication via BACnet protocol. Analog, binary and multi-state objects can be defined within multiple tabs. Variables can also be exported with the Export slave button. Defined variables are linked with variables in programs with the location column. Server configuration can be set in the Config tab.

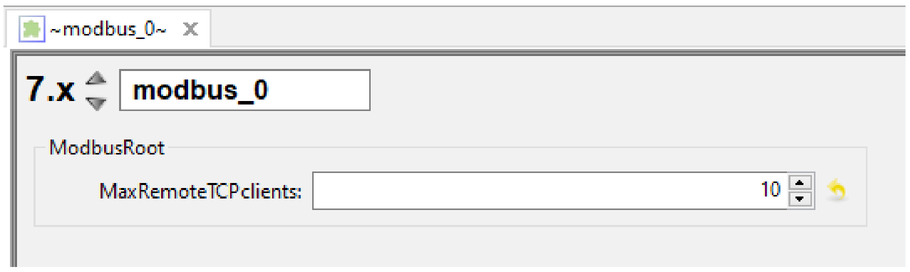


Object Identifier	Object Name	Description	Engineering Units
0	Analog Value 0		(Other) no-units (95)
1	Analog Value 1		(Other) no-units (95)

2.6.10 Modbus

Modbus extension allows the modbus TCP and modbus RTU communication protocol.

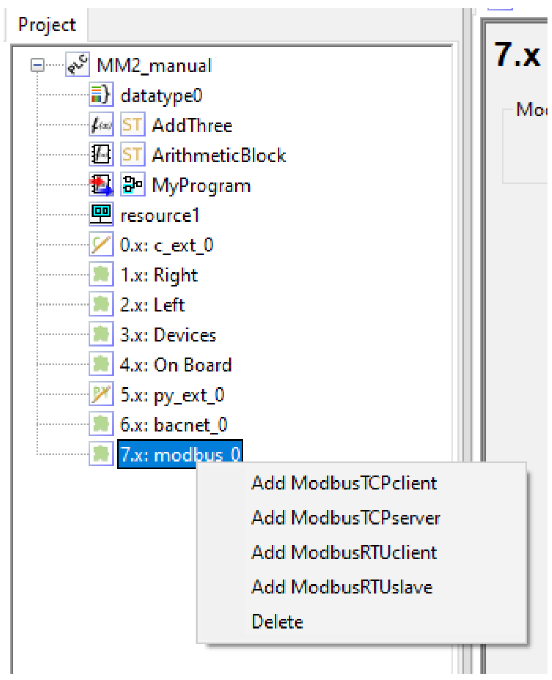
Maximum number of remote TCP clients is set on the main modbus screen.


~modbus_0~

7.x modbus_0

ModbusRoot

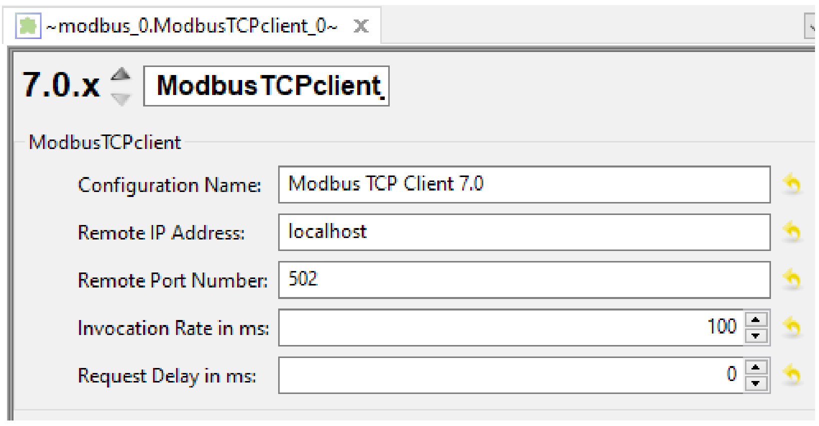
MaxRemoteTCPclients: 10



Right-click on modbus extension shows options to add a new modbus TCP client/server or modbus RTU client/slave.

Modbus TCP client can request data from modbus TCP server.

Remote IP Address and port number are the modbus TCP server address. Invocation rate and request delay can also be set.



~modbus_0.ModbusTCPclient_0~

7.0.x ModbusTCPclient

ModbusTCPclient

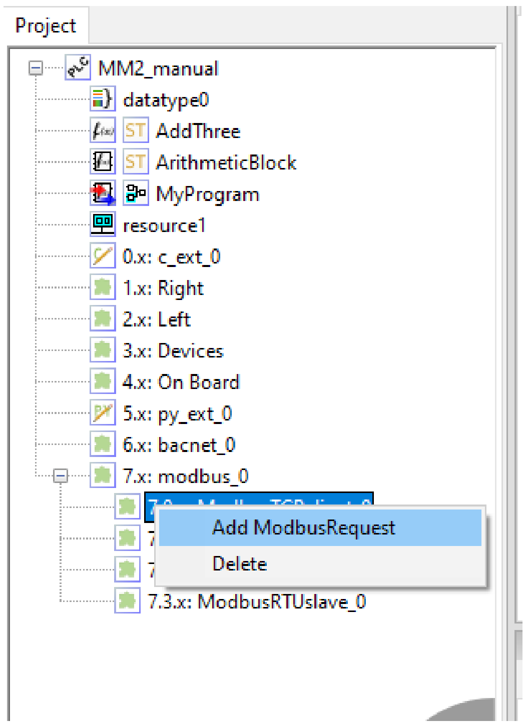
Configuration Name: Modbus TCP Client 7.0

Remote IP Address: localhost

Remote Port Number: 502

Invocation Rate in ms: 100

Request Delay in ms: 0



Right click on modbus TCP client in the project tree to add a modbus request.

There is no limit to the number of modbus requests and each request can only be one type of function

Functions:

-Read coils

-Read input discretes

-Read holding registers

-Read input registers

-Write single coil

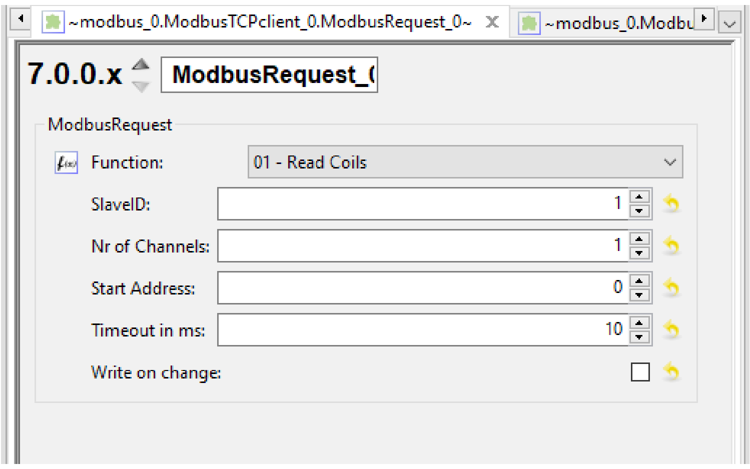
-Write single register

-Write multiple coils

-Write multiple registers

SlaveID is ID of the server

Nr of channels is the number of modbus registers to be read in this request.

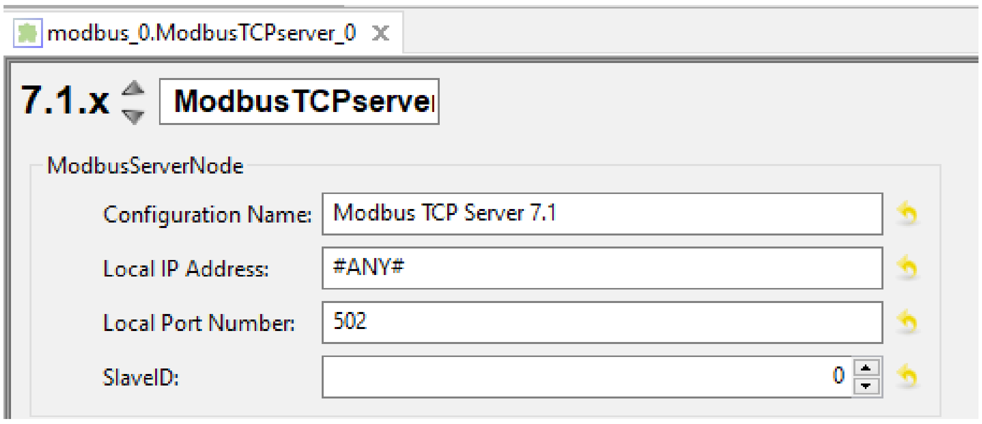

~modbus_0.ModbusTCPclient_0.ModbusRequest_0~ | ~modbus_0.Modbu

7.0.0.x ModbusRequest_(

ModbusRequest

Function:	01 - Read Coils
SlaveID:	1
Nr of Channels:	1
Start Address:	0
Timeout in ms:	10
Write on change:	☐

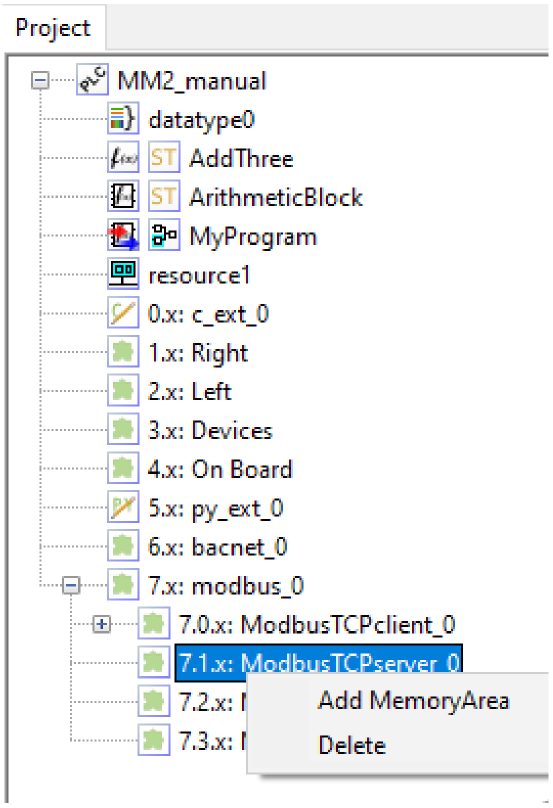
Modbus TCP server serves data to other modbus TCP clients. Here the address, port and slave ID are set. *#ANY#* is the default settings for the address and so the controller IP will be used.


modbus_0.ModbusTCPserver_0

7.1.x ModbusTCPserve

ModbusServerNode

Configuration Name:	Modbus TCP Server 7.1
Local IP Address:	#ANY#
Local Port Number:	502
SlaveID:	0



Right click on modbus TCP server in the project tree to add a modbus memory area.

There is no limit to the number of modbus memory areas, and each can have one type

Memory area types:

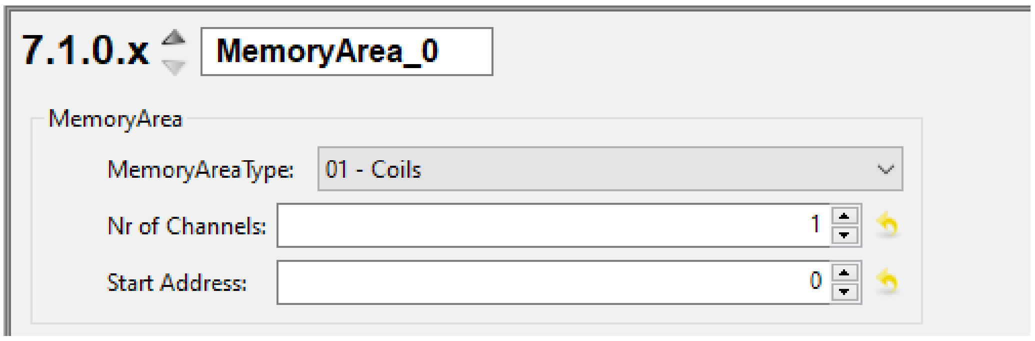
-Coils

-Input disccretes

-Holding registers

-Input registers

Nr of channels is the number of modbus registers to be read in this request.

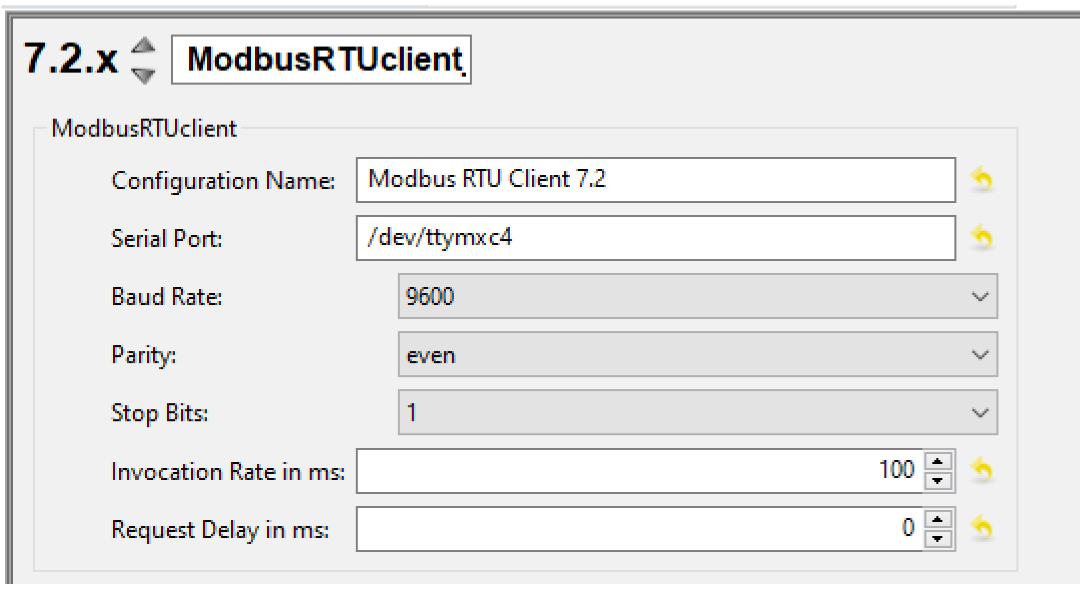


7.1.0.x MemoryArea_0

MemoryArea

MemoryAreaType:	01 - Coils
Nr of Channels:	1
Start Address:	0

Modbus RTU client requests data from modbus RTU slave. For use of LPC and LHC controllers serial port should be set to default /dev/ttymxc4. Baud rate, parity and stop bits depends on the slave we are connecting to. There is also an option to set invocation rate and request delay. Right-click on the extension to add requests. Requests are the same for modbus TCP and RTU



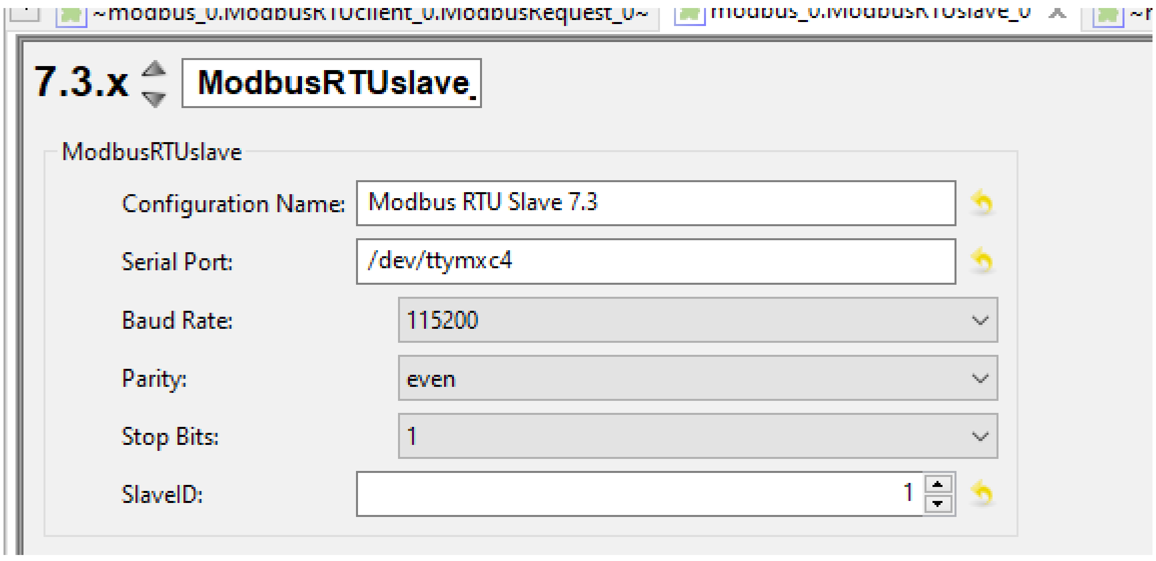
7.2.x ModbusRTUclient

ModbusRTUclient

Configuration Name:	Modbus RTU Client 7.2
Serial Port:	/dev/ttymxc4
Baud Rate:	9600
Parity:	even
Stop Bits:	1
Invocation Rate in ms:	100
Request Delay in ms:	0

Modbus RTU slave serves data to clients. For use of LPC and LHC controllers serial port should be set to default /dev/ttymxc4. Recommended settings for controllers are Baud Rate: 115200, Parity: even, Stop Bits: 1. Slave ID can also be set. Right-click on the extension to add memory area. The settings for memory area is the same for modbus TCP and RTU.

All set variables can be linked in programs by selecting *Location* column in variable definitions.



~modbus_0.ModbusRTUclient_0.ModbusRequest_0~ | modbus_0.ModbusRTUslave_0 | ~r

7.3.x ModbusRTUslave

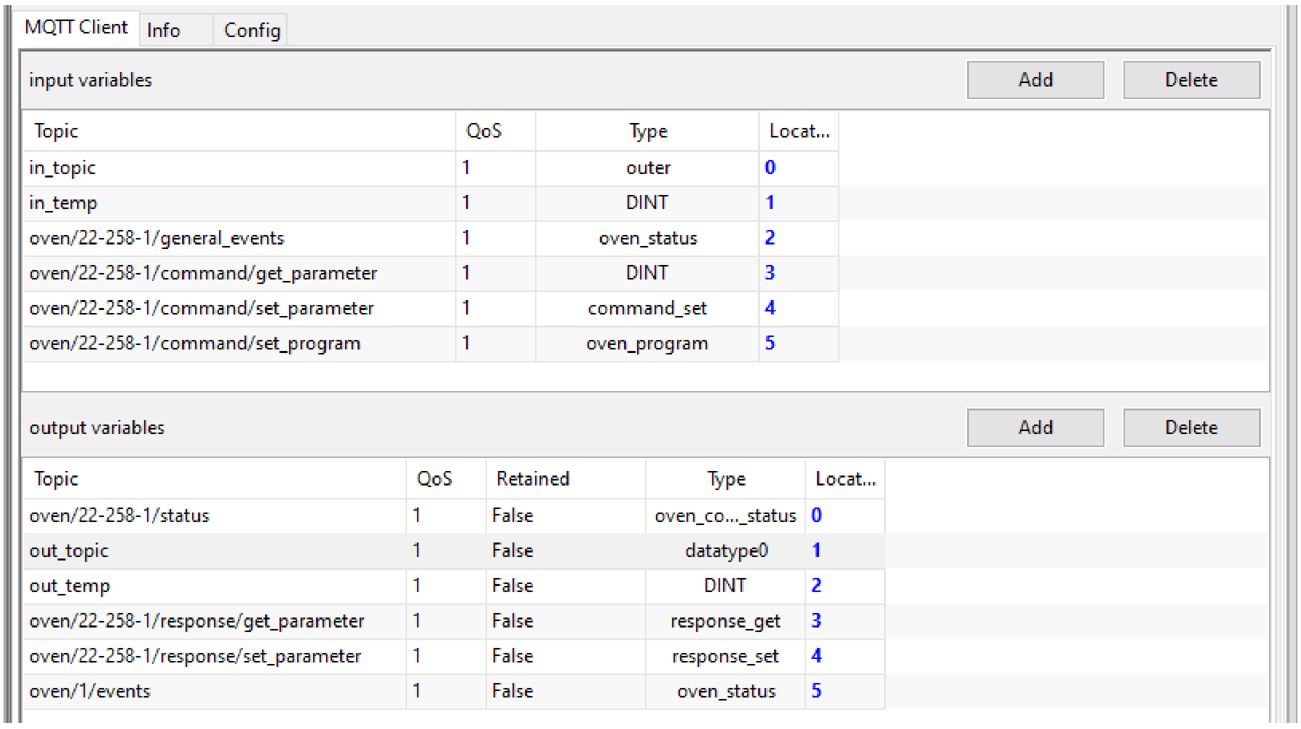
ModbusRTUslave

Configuration Name:	Modbus RTU Slave 7.3
Serial Port:	/dev/ttymxc4
Baud Rate:	115200
Parity:	even
Stop Bits:	1
SlaveID:	1

2.6.11 MQTT

MQTT extension is used for communication to a broker via MQTT protocol. Variables are defined in the MQTT Client tab. Input variables are used to subscribe to the broker, while output variables are used for publishing. Each variables has a topic, Quality of Service(QoS), type and location.

MQTT extension supports standard MQTT payload encoding using base types for variables(i.e. INT, UINT, REAL, etc.) and also JSON payload encoding (data is first transformed into JSON string and then sent over MQTT). To use defined variables in the extension, they have to be linked in PLC program with the location column. Any topic update on the broker will appear on input variable. Similarly any change to the output cariable will be sent to the broker to be published.



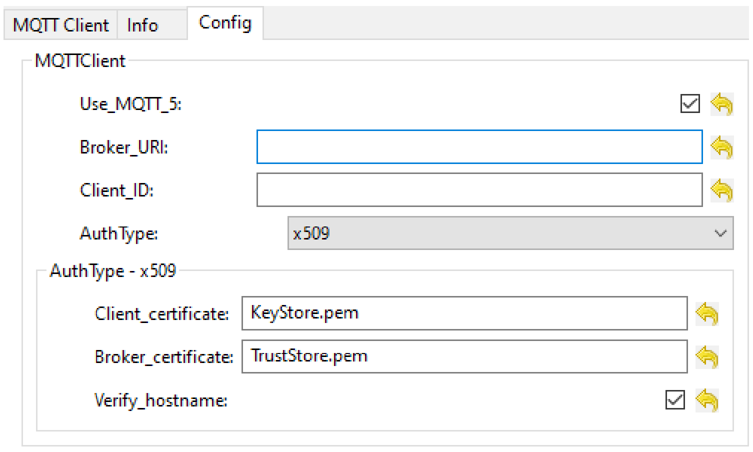
input variables

Topic	QoS	Type	Locat...
in_topic	1	outer	0
in_temp	1	DINT	1
oven/22-258-1/general_events	1	oven_status	2
oven/22-258-1/command/get_parameter	1	DINT	3
oven/22-258-1/command/set_parameter	1	command_set	4
oven/22-258-1/command/set_program	1	oven_program	5

output variables

Topic	QoS	Retained	Type	Locat...
oven/22-258-1/status	1	False	oven_co..._status	0
out_topic	1	False	datatype0	1
out_temp	1	False	DINT	2
oven/22-258-1/response/get_parameter	1	False	response_get	3
oven/22-258-1/response/set_parameter	1	False	response_set	4
oven/1/events	1	False	oven_status	5

The info tab shows the name of connection status global variable. Each client has its


MQTT Client | Info | Config

MQTTClient

Use_MQTT_5:	☑
Broker_URI:	
Client_ID:	
AuthType:	x509

AuthType - x509

Client_certificate:	KeyStore.pem
Broker_certificate:	TrustStore.pem
Verify_hostname:	☑

own connection status variable.

Broker, client and authentication settings are found in the Config tab. MQTT currently supports 4 types of authentication.

SmartehCloud: Uses client and broker certificates found on cloud.smarteh.com

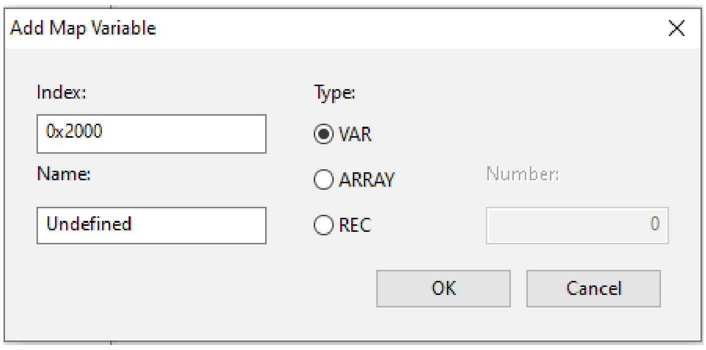
x509: requires client and broker certificate to be loaded in project files of the project.

PSK: requires secret file to be loaded in project files and also its ID. Both can be found on web interface of the controller.

UserPassword: Not used

2.6.12 CanOpen

CanOpen extension allows CANopen communication protocol. It uses master/slave relationship. To add CANopen master or slave, right-click on the CanOpen extension in the project tree.


Add Map Variable

Index: 0x2000

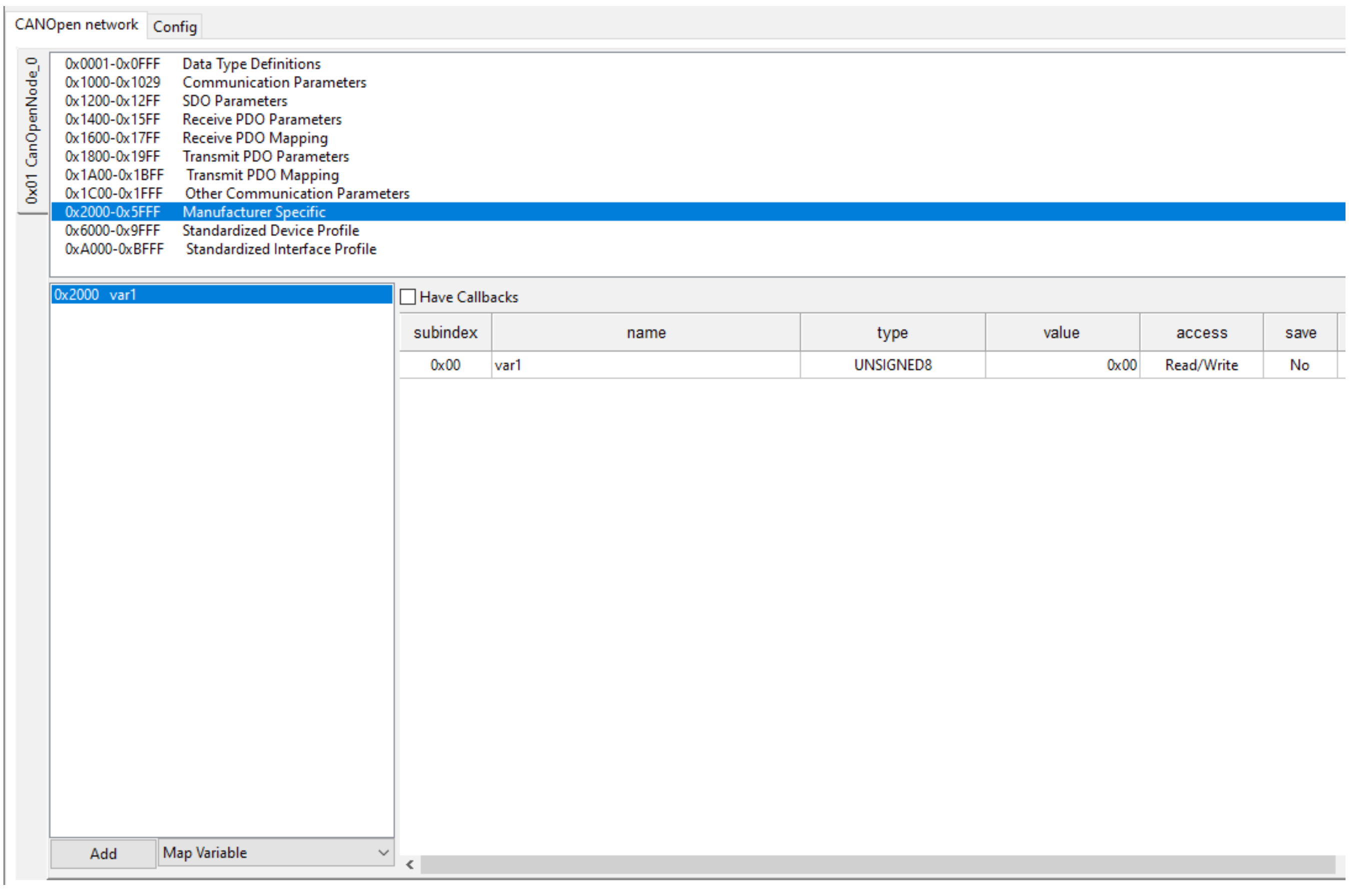
Name: Undefined

Type: [✓ VAR] ARRAY REC

Number: 0

To add a variable, first a range index needs to be selected. Then click Add to add new variable. New window will appear where index, name and type can be set. Index must stay in selected range. possible types are VAR(single variable) or ARRAY and REC. Selecting ARRAY or REC, number (number of variables in ARRAY and REC) must be set.

Other options to change are available by selecting created variable. Here name, type, initial value access and save can be set. Variables can be used in programs by selecting *Location* column in variable definitions. Or by by dragging the number under subindex to the program.

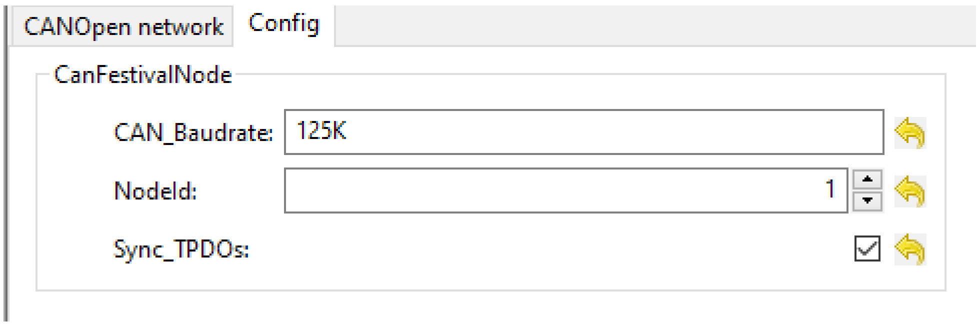


0x0001-0x0FFF Data Type Definitions
0x1000-0x1029 Communication Parameters
0x1200-0x12FF SDO Parameters
0x1400-0x15FF Receive PDO Parameters
0x1600-0x17FF Receive PDO Mapping
0x1800-0x19FF Transmit PDO Parameters
0x1A00-0x1BFF Transmit PDO Mapping
0x1C00-0x1FFF Other Communication Parameters
0x2000-0x5FFF Manufacturer Specific
0x6000-0x9FFF Standardized Device Profile
0xA000-0xBFFF Standardized Interface Profile

0x2000 var1

subindex	name	type	value	access	save
0x00	var1	UNSIGNED8	0x00	Read/Write	No

Under Config tab CAN_Baudrate and Node ID can be set. Allowed CANopen baud rate are 50K, 125K and 250K. Node ID can be from 1 to 127.



CAN_Baudrate: 125K
NodeId: 1
Sync_TPDOs: ✓

2.6.13 SVGHMI

SVGHMI extension is used for designing SVG GUI on GOT.xx2 and MMx modules.

In the main toolbar are options

Import SVG – Import already created graphics in svg format

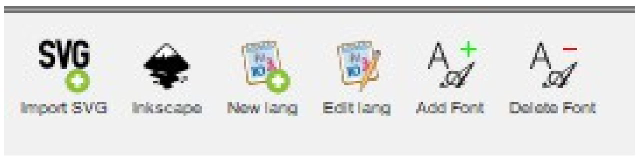
Inkscape – Open the editor(Inkscape) with currently working file or create a new file and open it if the file does not exists.

New language – Create a new language for translations

Edit language – Edit an existing language

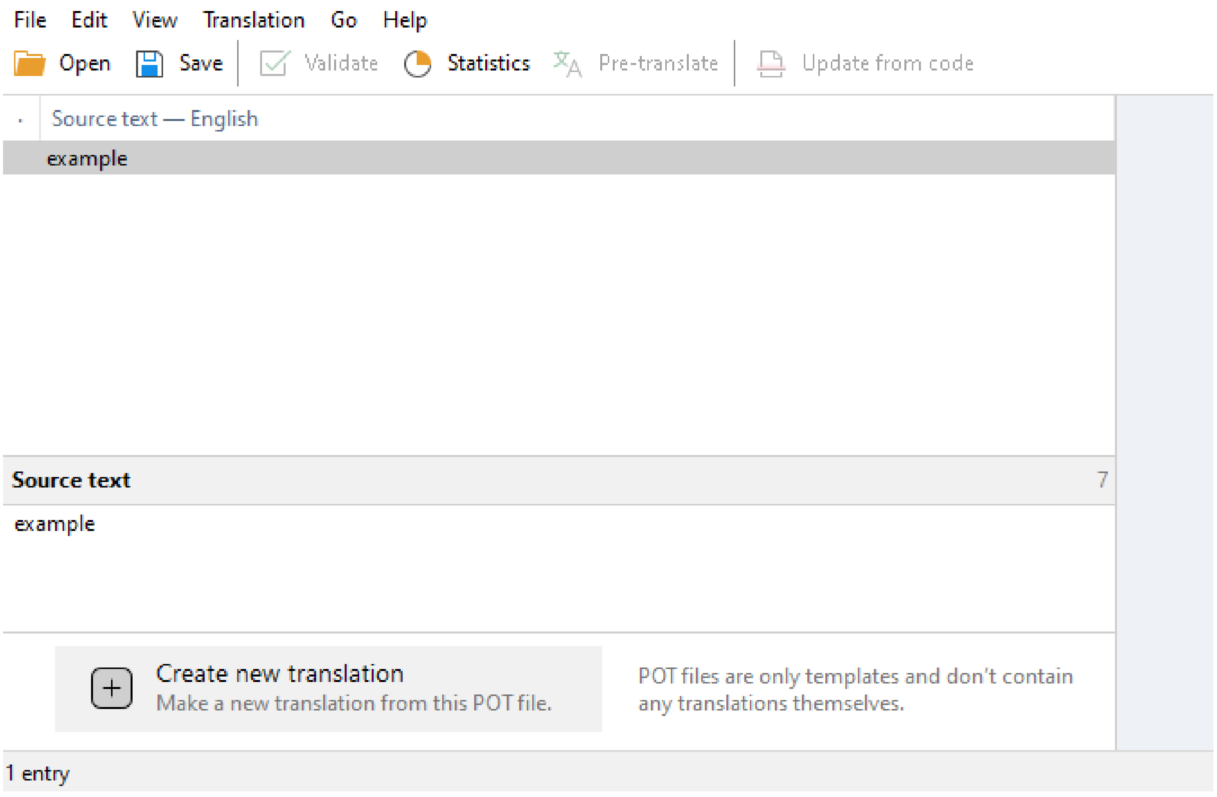
Add font – Import a new font to the controller

Delete font – Delete a font from the controller

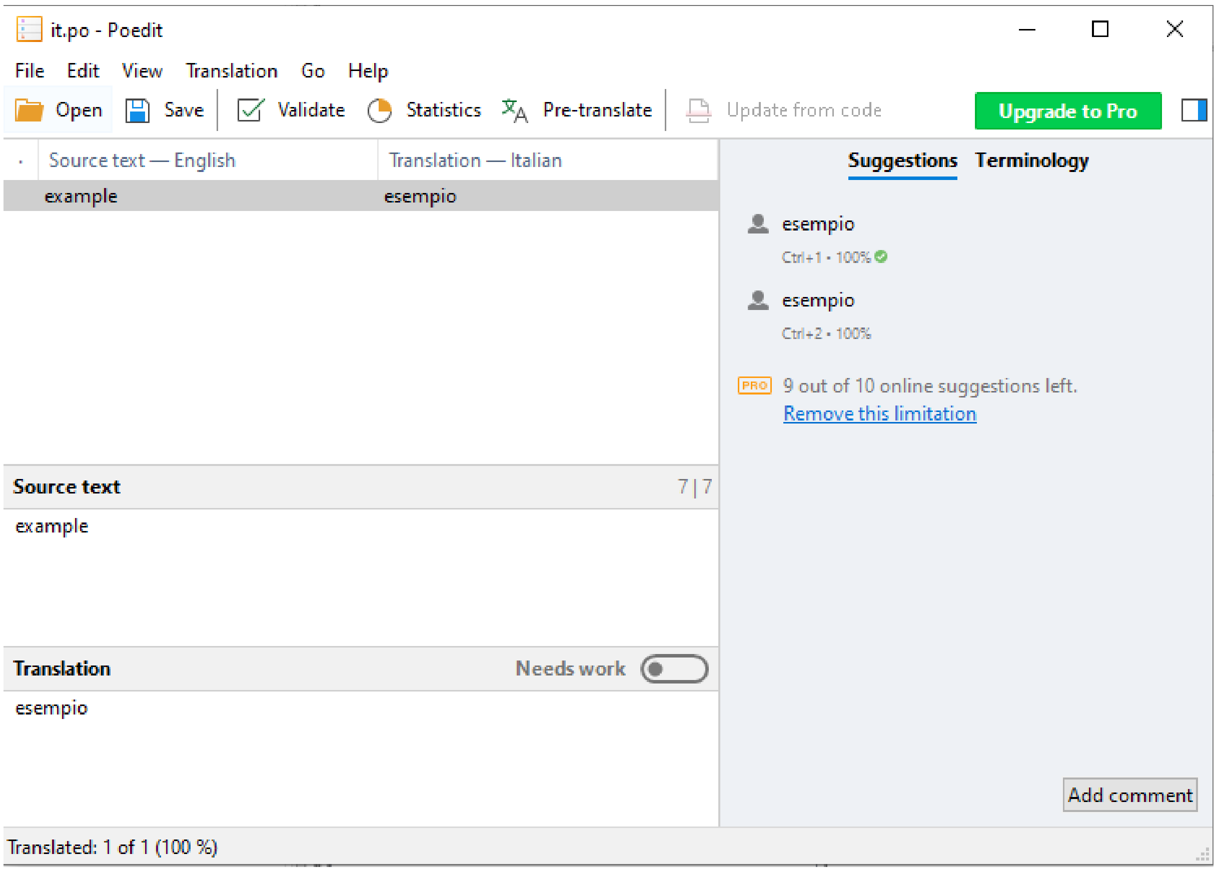


To create an SVG application *Inkscape* is required, an open source vector graphics editor. Inkscape must be installed on the PC to be able to create graphics for the program.

Creating/editing languages requires *poedit* utility to be installed on the PC. Creating a translation requires some translatable text (labels with an _ sign in front of the name, e.g. “_text1”) and the project must be built. Selecting *New language* button opens poedit with all the translatable text shown. Select *Create new translation* to choose the language.



Text to translate is shown in the source text field and translated text must be written in the translation field. Translated file must be saved to recommended svghmi folder of the project with default name (e.g. Italian – it.po)


it.po - Poedit
File Edit View Translation Go Help
Open Save Validate Statistics Pre-translate Update from code Upgrade to Pro
Suggestions Terminology

Source text — English	Translation — Italian
example	esempio

esempio
Ctrl+1 • 100%
esempio
Ctrl+2 • 100%
PRO 9 out of 10 online suggestions left.
Remove this limitation

Source text 7 | 7
example

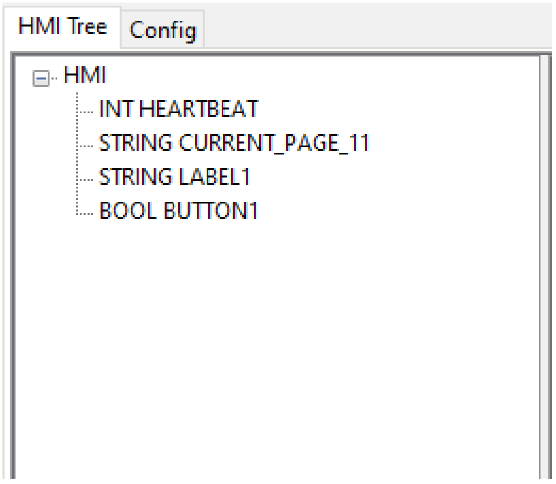
Translation Needs work
esempio

Add comment

Translated: 1 of 1 (100 %)

Adding a font saves it in the project files of the project and are transferred to the controller when transferring the project.

Tab HMI tree consists of list of all HMI variables defined in the project and the SVG widget library.

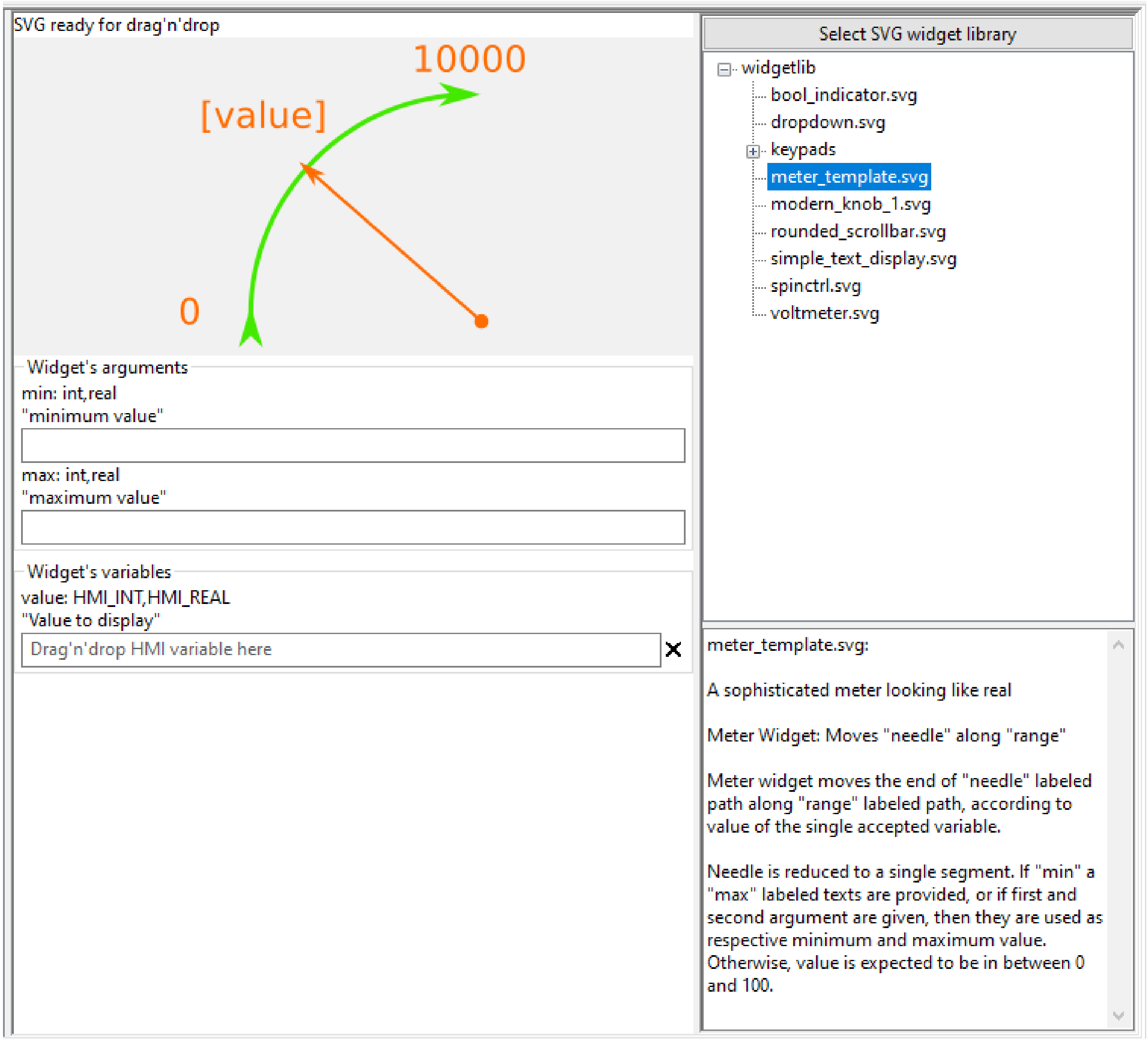


```
HMI
  INT HEARTBEAT
  STRING CURRENT_PAGE_11
  STRING LABEL1
  BOOL BUTTON1
```

Variables are linked with graphical widgets by setting variable type to SVGHMI data type (e.g. HMI_INT, HMI_REAL, HMI_BOOL, etc.). When the project is built their type and name are shown in the HMI tree. HEARTBEAT and CURRENT_PAGE_x are predefined external variables.

SVG widget library is a library of widgets ready to be used. Library has some predefined widgets. To create a widget you must first choose it in the list on the right side. When a widget is selected a preview is shown. Underneath the preview are possible properties of the widget.

Property of a widget that requires a variable can be set by selecting the variable in the HMI Tree and drag it in the box of the desired property. If the property is set correctly a check mark will appear next to the property. When all compulsory variables are set the widget is ready to be used. To use a widget first the editor must be open. Then select the preview of the widget and drag it onto the editor. The widget will appear in the editor with the desired structure. The widget can then also be modified.



meter_template.svg:

A sophisticated meter looking like real

Meter Widget: Moves "needle" along "range"

Meter widget moves the end of "needle" labeled path along "range" labeled path, according to value of the single accepted variable.

Needle is reduced to a single segment. If "min" a "max" labeled texts are provided, or if first and second argument are given, then they are used as respective minimum and maximum value. Otherwise, value is expected to be in between 0 and 100.

Config tab has various settings for managing SVGHMI.

OnStart -Function is called when starting PLC. Default *{LPCBrowserStart}* starts graphics on GOT modules. If using multiple SVGHMI extensions, only one should call this function.

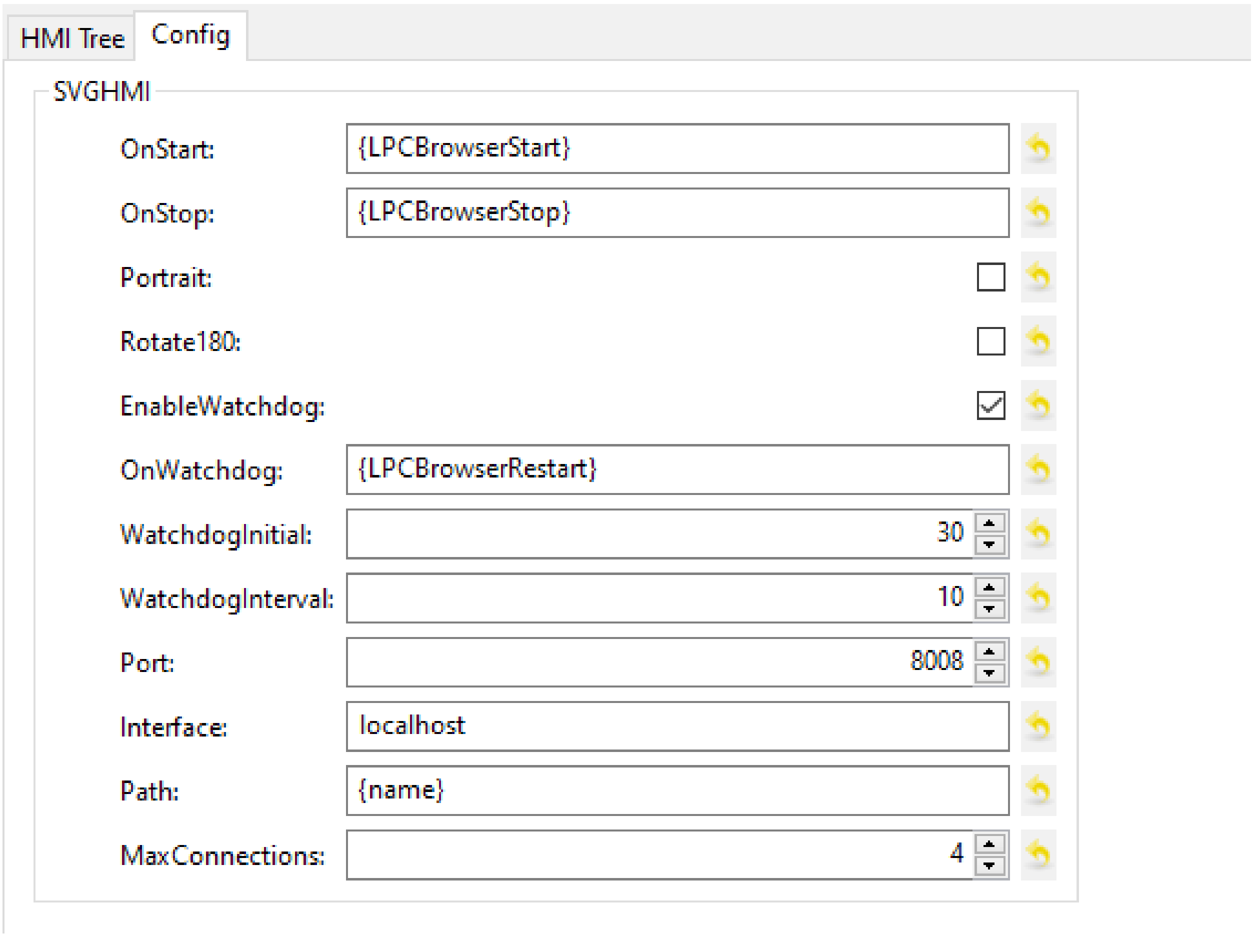
OnStop -Function is called when stopping PLC. Default *{LPCBrowserStop}* stops graphics on GOT modules. If using multiple SVGHMI extensions, only one should call this function.

Portrait and *Rotate180* -Manage orientation of the screens.

EnableWatchdog, *OnWatchdog*, *WatchdogInitial* and *WatchdogInterval* -Configure the function should watchdog trigger and the timers of timeout. Function *{LPCBrowserRestart}* restarts only the graphics part of PLC. If using multiple SVGHMI extensions, only one watchdog can be active.

Port, *Interface* and *Path* -Set URL where the graphics is broadcasted. If interface is set to
-localhost: graphics is broadcasted on GOT module.
-vpn: graphics is broadcasted over VPN.
-192.168.45.1: graphics is broadcasted over USB cable connected to the controller.
Path is the path part of URL with {name} being the name of SVGHMI extension.

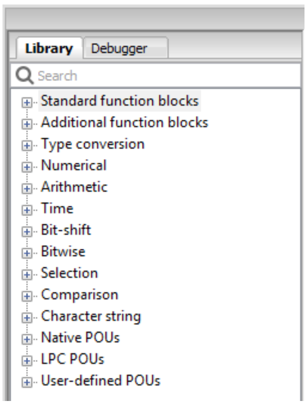
MaxConnections -Sets the max number of clients able to connect over VPN.



OnStart:	{LPCBrowserStart}
OnStop:	{LPCBrowserStop}
Portrait:	☐
Rotate180:	☐
EnableWatchdog:	✓
OnWatchdog:	{LPCBrowserRestart}
WatchdogInitial:	30
WatchdogInterval:	10
Port:	8008
Interface:	localhost
Path:	{name}
MaxConnections:	4

For more on how to use SVGHMI video tutorials can be found on smarteh main page - support&downloads - video - tutorials - LPC-3.GOT.XX2 LCD PLC HMI programming tutorials.

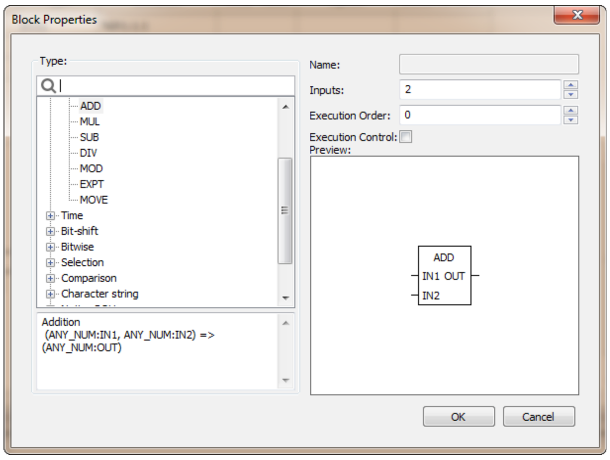
2.7 Library



Library

-Library contains various groups of standard and user defined functions. They support the usage in different programmable controller programming languages inside POUs.

BLOCK PROPERTIES

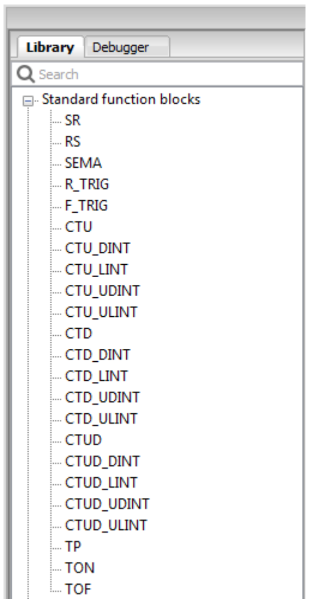


Inputs: 2
Execution Order: 0

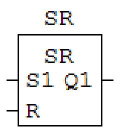
Addition
(ANY_NUM:IN1, ANY_NUM:IN2) =>
(ANY_NUM:OUT)

Block Properties pop-up window can be opened by double-click on function block. Some of the blocks can have more inputs than default. This is selectable in the *Inputs* field (e.g. ADD block). Also an *Execution Order* of the blocks can be programmer-defined. All blocks have an additional *Execution Control* check-box. If it is checked than two new pins are added (EN – input and ENO – output) to control dynamically their execution.

2.7.1 Standard function blocks


Library Debugger
Search
Standard function blocks
SR
RS
SEMA
R_TRIG
F_TRIG
CTU
CTU_DINT
CTU_LINT
CTU_UDINT
CTU_ULINT
CTD
CTD_DINT
CTD_LINT
CTD_UDINT
CTD_ULINT
CTUD
CTUD_DINT
CTUD_LINT
CTUD_UDINT
CTUD_ULINT
TP
TON
TOF

SR



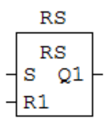
SR bistable

The SR bistable is a latch where the Set dominates.

(BOOL:S1, BOOL:R) => (BOOL:Q1)

This function represents a standard set-dominant set/reset flip flop. The Q1 output become TRUE when the input S1 is TRUE and the R input is FALSE. In the same way, the Q1 output become FALSE when the input S1 is FALSE and the R input is TRUE. After one of these transitions, when both the S1 and R signals return to FALSE, the Q1 output keeps the previous state until a new condition occurs. If you apply a TRUE condition for both the signals, the Q1 output is forced to TRUE (set-dominant).

RS



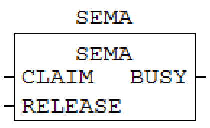
RS bistable

The RS bistable is a latch where the Reset dominates.

(BOOL:S, BOOL:R1) => (BOOL:Q1)

This function represents a standard reset-dominant set/reset flip flop. The Q1 output become TRUE when the input S is TRUE and the R1 input is FALSE. In the same way, the Q1 output become FALSE when the input S is FALSE and the R1 input is TRUE. After one of these transitions, when both the S and R1 signals return to FALSE, the Q1 output keeps the previous state until a new condition occurs. If you apply a TRUE condition for both the signals, the Q1 output is forced to FALSE (reset-dominant).

SEMA



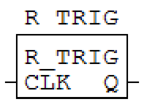
Semaphore

The semaphore provides a mechanism to allow software elements mutually exclusive access to certain resources.

(BOOL:CLAIM, BOOL:RELEASE) => (BOOL:BUSY)

This function block implements a semaphore function. Normally this function is used to synchronize events. The BUSY output is activated by a TRUE condition on the CLAIM input and it is de-activated by a TRUE condition on the RELEASE input.

R TRIG



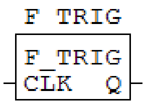
Rising edge detector

The output produces a single pulse when a rising edge is detected.

(BOOL:CLK) => (BOOL:Q)

This function is a rising-edge detector. The Q output becomes TRUE when a 0 to 1 (or FALSE to TRUE or OFF to ON) condition is detected on the CLK input and it sustains this state for a complete scan cycle.

F TRIG



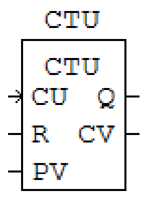
Falling edge detector

The output Q produces a single pulse when a falling edge is detected.

(BOOL:CLK) => (BOOL:Q)

This function is a falling-edge detector. The Q output becomes TRUE when a 1 to 0 (or TRUE to FALSE or ON to OFF) condition is detected on the CLK input and it sustains this state for a complete scan cycle.

CTU
CTU_DINT, CTU_LINT, CTU_UDINT, CTU_ULINT



Up-counter

The up-counter can be used to signal when a count has reached a maximum value.

CTU: (BOOL:CU, BOOL:R, INT:PV) => (BOOL:Q, INT:CV)

CTU_DINT: (BOOL:CU, BOOL:R, DINT:PV) => (BOOL:Q, DINT:CV)

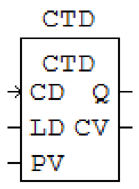
CTU_LINT: (BOOL:CU, BOOL:R, LINT:PV) => (BOOL:Q, LINT:CV)

CTU_UDINT: (BOOL:CU, BOOL:R, UDINT:PV) => (BOOL:Q, UDINT:CV)

CTU_ULINT: (BOOL:CU, BOOL:R, ULINT:PV) => (BOOL:Q, ULINT:CV)

The CTU function represents an up-counter. A rising-edge on CU input will increment the counter by one. When the programmed value, applied to the input PV, is reached, the Q output becomes TRUE. Applying a TRUE signal on R input will reset the counter to zero (Asynchronous reset). The CV output reports the current counting value.

CTD
CTD_DINT, CTD_LINT, CTD_UDINT, CTD_ULINT



Down-counter

The down-counter can be used to signal when a count has reached zero, on counting down from a pre-set value.

CTD: (BOOL:CD, BOOL:LD, INT:PV) => (BOOL:Q, INT:CV)

CTD_DINT: (BOOL:CD, BOOL:LD, DINT:PV) => (BOOL:Q, DINT:CV)

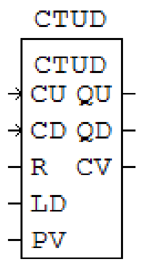
CTD_LINT: (BOOL:CD, BOOL:LD, LINT:PV) => (BOOL:Q, LINT:CV)

CTD_UDINT: (BOOL:CD, BOOL:LD, UDINT:PV) => (BOOL:Q, UDINT:CV)

CTD_ULINT: (BOOL:CD, BOOL:LD, ULINT:PV) => (BOOL:Q, ULINT:CV)

The CTD function represents a down-counter. A rising-edge on CD input will decrement the counter by one. The Q output becomes TRUE when the current counting value is equal or less than zero. Applying a TRUE signal on LD (LOAD) input will load the counter with the value present at input PV (Asynchronous load). The CV output reports the current counting value.

CTUD
CTUD_DINT,
CTUD_LINT,
CTUD_UDINT,
CTUD_ULINT



Up-down counter

The up-down counter has two inputs CU and CD. It can be used to both count up on one input and down on the other.

CTUD: (BOOL:CU, BOOL:CD, BOOL:R, BOOL:LD, INT:PV) => (BOOL:QU, BOOL:QD, INT:CV)

CTUD_DINT: (BOOL:CU, BOOL:CD, BOOL:R, BOOL:LD, DINT:PV) => (BOOL:QU, BOOL:QD, DINT:CV)

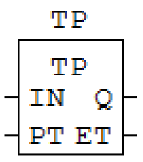
CTUD_LINT: (BOOL:CU, BOOL:CD, BOOL:R, BOOL:LD, LINT:PV) => (BOOL:QU, BOOL:QD, LINT:CV)

CTUD_UDINT: (BOOL:CU, BOOL:CD, BOOL:R, BOOL:LD, UDINT:PV) => (BOOL:QU, BOOL:QD, UDINT:CV)

CTUD_ULINT: (BOOL:CU, BOOL:CD, BOOL:R, BOOL:LD, ULINT:PV) => (BOOL:QU, BOOL:QD, ULINT:CV)

This function represents an up-down programmable counter. A rising-edge on the CU (COUNT-UP) input increments the counter by one while a rising-edge on the CD (COUNT-DOWN) decreases the current value. Applying a TRUE signal on R input will reset the counter to zero. A TRUE condition on the LD signal will load the counter with the value applied to the input PV (PROGRAMMED VALUE). QU output becomes active when the current counting value is greater or equal to the programmed value. The QD output becomes active when the current value is less or equal to zero.
The CV output reports the current counter value.

TP



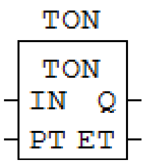
Pulse timer

The pulse timer can be used to generate output pulses of a given time duration.

(BOOL:IN, TIME:PT) => (BOOL:Q, TIME:ET)

This kind of timer has the same behaviour of a single-shot timer or a monostable timer.
When a rising-edge transition is detected on the IN input, the Q output becomes TRUE immediately. This condition continues until the programmed time PT, applied to the relative pin, is elapsed. After that the PT is elapsed, the Q output keeps the ON state if the input IN is still asserted else the Q output returns to the OFF state. This timer is not re-triggerable. This means that after that the timer has started it can't be stopped until the complete session ends. The ET output reports the current elapsed time.

TON



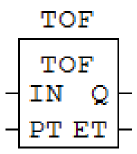
On-delay timer

The on-delay timer can be used to delay setting an output true, for fixed period after an input becomes true.

(BOOL:IN, TIME:PT) => (BOOL:Q, TIME:ET)

Asserting the input signal IN of this function starts the timer. When the programmed time, applied to the input PT, is elapsed and the input IN is still asserted, the Q output becomes TRUE. This condition will continue until the input IN is released. If the IN input is released before time elapsing, the timer will be cleared. The ET output reports the current elapsed time.

TOF



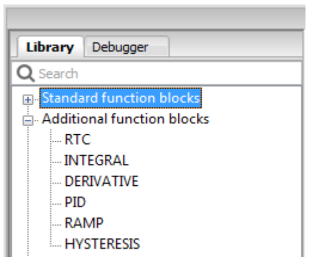
Off-delay timer

The off-delay timer can be used to delay setting an output false, for fixed period after input goes false.

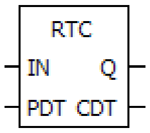
(BOOL:IN, TIME:PT) => (BOOL:Q, TIME:ET)

Asserting the input signal IN of this function immediately activates the Q output. At this point, releasing the input IN will start the time elapsing. When the programmed time, applied to the input PT, is elapsed and the input IN is still released, the Q output becomes FALSE. This condition will be kept until the input IN is released. If the IN input is asserted again before time elapses, the timer will be cleared and the Q output remains TRUE. The ET output reports the current elapsed time.

2.7.2 Additional function block



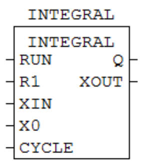
RTC



RTC

Functioning is not supported by our PLC.

INTEGRAL



Integral

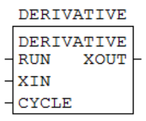
The integral function block integrates the value of input XIN over time.

(BOOL:RUN, BOOL:R1, REAL:XIN, REAL:X0, TIME:CYCLE) =>
(BOOL:Q, REAL:XOUT)

When input RUN is True and override R1 is False, XOUT will change for XIN value depends on CYCLE time value sampling period. When RUN is False and override R1 is True, XOUT will hold the last output value. If R1 is True, XOUT will be set to the X0 value.

XOUT = XOUT + (XIN * CYCLE)

DERIVATIVE



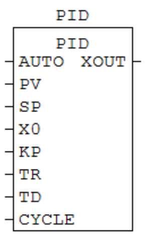
Derivative

The derivative function block produces an output XOUT proportional to the rate of change of the input XIN.

(BOOL:RUN, REAL:XIN, TIME:CYCLE) => (REAL:XOUT)

When RUN is True, XOUT will change proportional to the rate of changing of the value XIN depends on CYCLE time value sampling period.
XOUT = ((3 * (XIN - $XIN_{(to-3)}$)) + $XIN_{(to-1)}$ - $XIN_{(to-2)}$) / (10 * CYCLE)

PID



PID

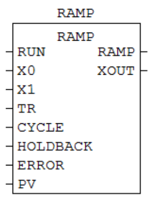
The PID (Proportional, Integral, Derivative) function block provides the classical three term controller for closed loop control. It does not contain any output limitation parameters (dead-band, minimum, maximum, …) or other parameters normally used for real process control (see also PID_A).

(BOOL:AUTO, REAL:PV, REAL:SP, REAL:X0, REAL:KP, REAL:TR, REAL:TD, TIME:CYCLE) => (REAL:XOUT)

When AUTO is False, PID function block XOUT will follow X0 value. When AUTO is True, XOUT will be calculated from error value (PV process variable – SP set point), KP proportional constant, TR reset time, TD derivative constant and CYCLE time value sampling period.

XOUT = KP * ((PV-SP) + (I_OUT/TR) + (D_OUT * TD))

RAMP



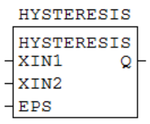
Ramp

The RAMP function block is modelled on example given in the standard but with the addition of a 'Holdback' feature.

(BOOL:RUN, REAL:X0, REAL:X1, TIME:TR, TIME:CYCLE, BOOL:HOLDBACK, REAL:ERROR, REAL:PV) => (BOOL:RAMP, REAL:XOUT)

When RUN and HOLDBACK are False, XOUT will follow X0 value. When RUN is True and HOLDBACK value is False, XOUT will change for OUT$_{(to-1)}$ + (X1 – XOUT$_{(to-1)}$) every CYCLE time value sampling period.

HYSTERESIS



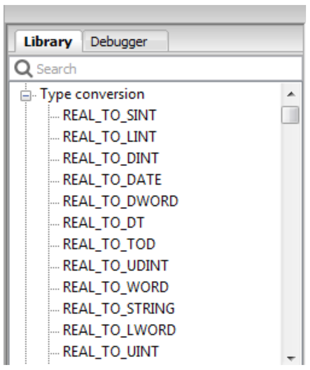
Hysteresis

The hysteresis function block provides a hysteresis boolean output driven by the difference of two floating point (REAL) inputs XIN1 and XIN2.

(REAL:XIN1, REAL:XIN2, REAL:EPS) => (BOOL:Q)

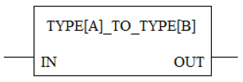
When XIN1 value will be grater than XIN2 + EPS value, Q becomes True. When XIN1 value will be less than XIN2 - EPS value, Q becomes False.

2.7.3 Type conversion



Type conversion
- REAL_TO_SINT
- REAL_TO_LINT
- REAL_TO_DINT
- REAL_TO_DATE
- REAL_TO_DWORD
- REAL_TO_DT
- REAL_TO_TOD
- REAL_TO_UDINT
- REAL_TO_WORD
- REAL_TO_STRING
- REAL_TO_LWORD
- REAL_TO_UINT

TYPE[A]_TO_TYPE[B]

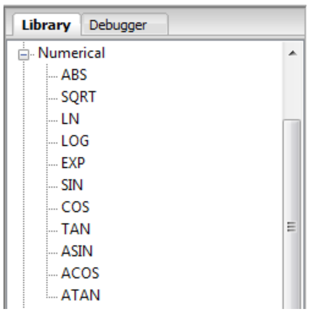


Data type conversion

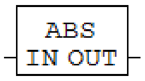
(TYPE[A]:IN) => (TYPE[B]:OUT)

ST syntax example: `OUT := TYPE[A]_TO_TYPE[B](IN1);`

2.7.4 Numerical



ABS

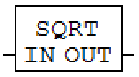


Absolute number

(ANY_NUM:IN) => (ANY_NUM:OUT)

ST syntax example: `OUT := ABS(IN1);`

SQRT

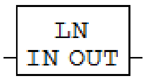


Square root (base 2)

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := SQRT(IN1);`

LN

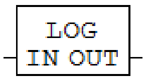


Natural logarithm

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := LN(IN1);`

LOG

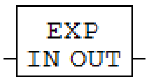


Logarithm to base 10

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := LOG(IN1);`

EXP

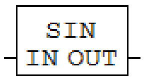


Exponentiation

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := EXP(IN1);`

SIN

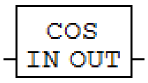


Sine

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := SIN(IN1);`

COS

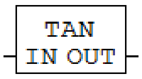


Cosine

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := COS(IN1);`

TAN

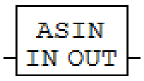


Tangent

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := TAN(IN1);`

ASIN

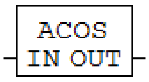


Arc sine

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := ASIN(IN1);`

ACOS

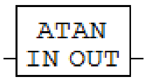


Arc cosine

(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := ACOS(IN1);`

ATAN

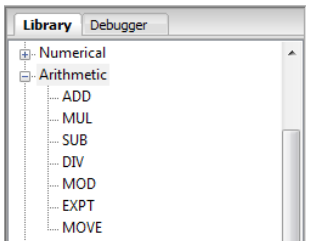


Arc tangent

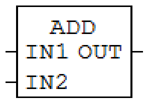
(ANY_REAL:IN) => (ANY_REAL:OUT)

ST syntax example: `OUT := ATAN(IN1);`

2.7.5 Arithmetic



ADD



Addition

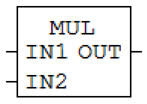
(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

OUT = IN1 + IN2.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 + IN2;`

MUL



Multiplication

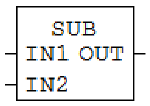
(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

OUT = IN1 * IN2.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 * IN2;`

SUB



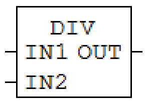
Subtraction

(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

OUT = IN1 - IN2.

ST syntax example: `OUT := IN1 - IN2;`

DIV



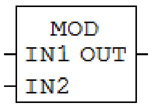
Division

(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

OUT = IN1 / IN2.
For example 1234 / 10 = 3.

ST syntax example: `OUT := IN1 / IN2;`

MOD



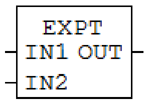
Remainder (modulo)

(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

OUT = IN1 modulo IN2.
For example 1234 modulo 10 = 4.

ST syntax example: `OUT := IN1 MOD IN2;`

EXPT



Exponent

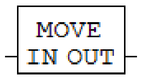
(ANY_REAL:IN1, ANY_NUM:IN2) => (ANY_REAL:OUT)

OUT = $IN1^{IN2}$.

For example 2^3= 8.

ST syntax example: `OUT := IN1 ** IN2;`

MOVE



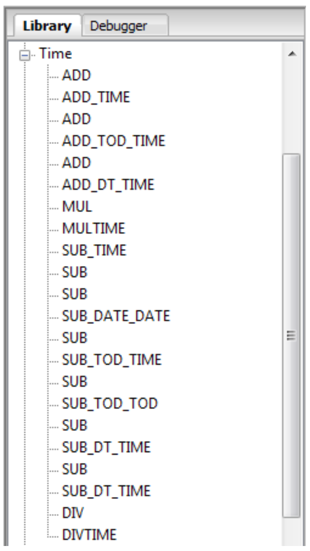
Assignment

(ANY:IN) => (ANY:OUT)

OUT = IN.

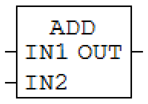
ST syntax example: `OUT := IN1;`

2.7.6 Time



Time
- ADD
- ADD_TIME
- ADD
- ADD_TOD_TIME
- ADD
- ADD_DT_TIME
- MUL
- MULTIME
- SUB_TIME
- SUB
- SUB
- SUB_DATE_DATE
- SUB
- SUB_TOD_TIME
- SUB
- SUB_TOD_TOD
- SUB
- SUB_DT_TIME
- SUB
- SUB_DT_TIME
- DIV
- DIVTIME

ADD

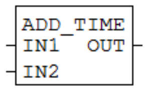


Time addition

(TIME:IN1, TIME:IN2) => (TIME:OUT)

Number of inputs can be expanded.

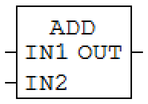
ADD_TIME



Time addition

(TIME:IN1, TIME:IN2) => (TIME:OUT)

ADD

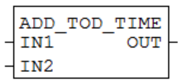


Time-of-day addition

(TOD:IN1, TIME:IN2) => (TOD:OUT)

Number of inputs can be expanded.

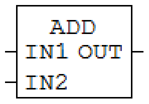
ADD_TOD_TIME



Time-of-day addition

(TOD:IN1, TIME:IN2) => (TOD:OUT)

ADD

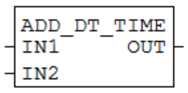


Addition

(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

Number of inputs can be expanded.

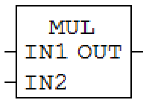
ADD_DT_TIME



Date addition

(DT:IN1, TIME:IN2) => (DT:OUT)

MUL

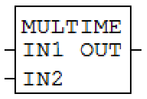


Multiplication

(ANY_NUM:IN1, ANY_NUM:IN2) => (ANY_NUM:OUT)

Number of inputs can be expanded.

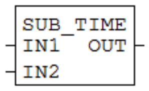
MULTIME



Time multiplication

(TIME:IN1, ANY_NUM:IN2) => (TIME:OUT)

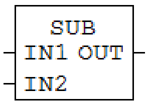
SUB_TIME



Time subtraction

(TIME:IN1, TIME:IN2) => (TIME:OUT)

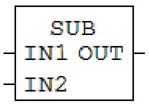
SUB



Time subtraction

(TIME:IN1, TIME:IN2) => (TIME:OUT)

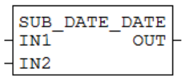
SUB



Date subtraction

(DATE:IN1, DATE:IN2) => (TIME:OUT)

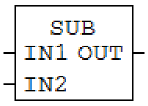
SUB_DATE_DATE



Date subtraction

(DATE:IN1, DATE:IN2) => (TIME:OUT)

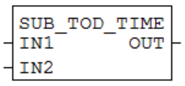
SUB



Time-of-day subtraction

(TOD:IN1, TIME:IN2) => (TOD:OUT)

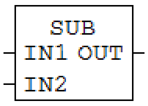
SUB_TOD_TIME



Time-of-day subtraction

(TOD:IN1, TIME:IN2) => (TOD:OUT)

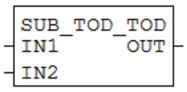
SUB



Time-of-day subtraction

(TOD:IN1, TOD:IN2) => (TIME:OUT)

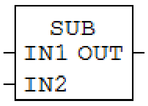
SUB_TOD_TOD



Time-of-day subtraction

(TOD:IN1, TOD:IN2) => (TIME:OUT)

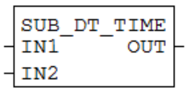
SUB



Date and time subtraction

(DT:IN1, TIME:IN2) => (DT:OUT)

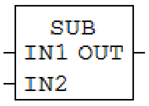
SUB_DT_TIME



Date and time subtraction

(DT:IN1, TIME:IN2) => (DT:OUT)

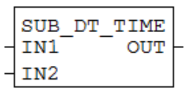
SUB



Date and time subtraction

(DT:IN1, DT:IN2) => (TIME:OUT)

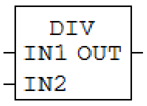
SUB_DT_TIME



Date and time subtraction

(DT:IN1, DT:IN2) => (TIME:OUT)

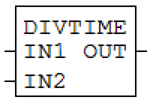
DIV



Time division

(TIME:IN1, ANY_NUM:IN2) => (TIME:OUT)

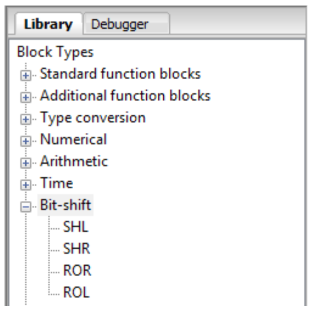
DIVTIME



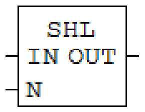
Time division

(TIME:IN1, ANY_NUM:IN2) => (TIME:OUT)

2.7.7 Bit-shift



SHL



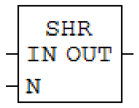
Shift left

(ANY_BIT:IN, ANY_INT:N) => (ANY_BIT:OUT)

OUT represents IN variable shifted left by N bits. Zeros are filled on the right side of the OUT variable.

ST syntax example: `OUT := SHL(IN := IN1, N := IN2);`

SHR



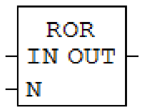
Shift right

(ANY_BIT:IN, ANY_INT:N) => (ANY_BIT:OUT)

OUT represents IN variable shifted right by N bits. Zeros are filled on the left side of the OUT variable.

ST syntax example: `OUT := SHR(IN := IN1, N := IN2);`

ROR



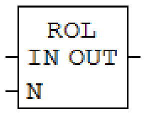
Rotate right

(ANY_NBIT:IN, ANY_INT:N) => (ANY_NBIT:OUT)

OUT represents IN variable right rotated by N bits. Each rotation most right bit is filled into most left bit of the OUT variable.

ST syntax example: `OUT := ROR(IN := IN1, N := IN2);`

ROL



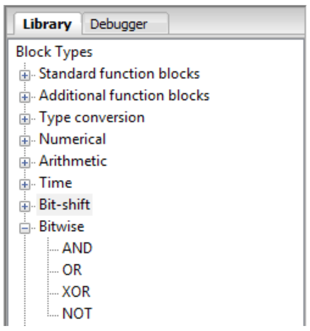
Rotate left

(ANY_NBIT:IN, ANY_INT:N) => (ANY_NBIT:OUT)

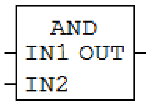
OUT represents IN variable left rotated by N bits. Each rotation most left bit is filled into most right bit of the OUT variable.

ST syntax example: `OUT := ROL(IN := IN1, N := IN2);`

2.7.8 Bitwise



AND



Bitwise AND

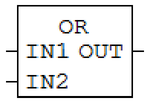
(ANY_BIT:IN1, ANY_BIT:IN2) => (ANY_BIT:OUT)

OUT = IN1 *AND* IN2.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 AND IN2;`

OR



Bitwise OR

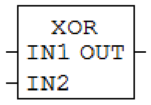
(ANY_BIT:IN1, ANY_BIT:IN2) => (ANY_BIT:OUT)

OUT = IN1 *OR* IN2.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 OR IN2;`

XOR



Bitwise XOR

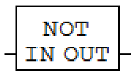
(ANY_BIT:IN1, ANY_BIT:IN2) => (ANY_BIT:OUT)

OUT = IN1 *EXCLUSIVE OR* IN2.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 XOR IN2;`

NOT



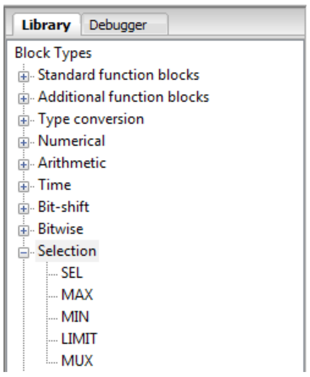
Bitwise inverting

(ANY_BIT:IN) => (ANY_BIT:OUT)

OUT = *NOT* IN.

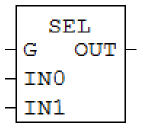
ST syntax example: `OUT := IN1 NOT IN2;`

2.7.9 Selection



Block Types
- Standard function blocks
- Additional function blocks
- Type conversion
- Numerical
- Arithmetic
- Time
- Bit-shift
- Bitwise
- Selection
  - SEL
  - MAX
  - MIN
  - LIMIT
  - MUX

SEL

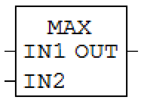


Binary selection (1 of 2)

(BOOL:G, ANY:IN0, ANY:IN1) => (ANY:OUT)

If G is False, OUT will follow IN0 value. If G is True, OUT will follow IN1 value.

MAX



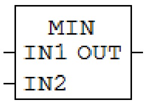
Maximum

(ANY:IN1, ANY:IN2) => (ANY:OUT)

This function block compares the magnitude of the values present at input IN1 and IN2 reporting on its output the largest value (maximum value).

Number of inputs can be expanded.

MIN



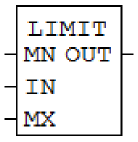
Minimum

(ANY:IN1, ANY:IN2) => (ANY:OUT)

This function block compares the magnitude of the values present at input IN1 and IN2 reporting on its output the smallest value (minimum value).

Number of inputs can be expanded.

LIMIT

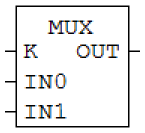


Limitation

(ANY:MN, ANY:IN, ANY:MX) => (ANY:OUT)

OUT will follow IN value between minimum MN and maximum MX value. If IN will be less than minimum MN value, OUT will represent MN value and if IN will be grater than maximum MX value, OUT will represent MX value.

MUX



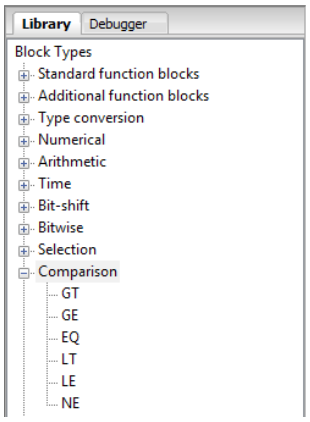
Multiplexer (select 1 of N)

(ANY_INT:K, ANY:IN0, ANY:IN1) => (ANY:OUT)

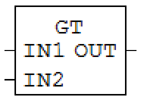
Depends on K value, one of IN1, IN2. INn is selected and OUT will represent the selected input value.

Number of inputs can be expanded.

2.7.10 Comparison



GT



Greater than

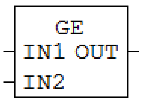
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 is greater than IN2, else OUT will be False.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 > IN2;`

GE



Greater than or equal to

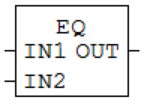
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 is greater or equal than IN2, else OUT will be False.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 >= IN2;`

EQ



Equal to

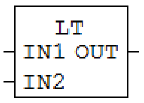
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 and IN2 are equal, else OUT will be False.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 = IN2;`

LT



Less than

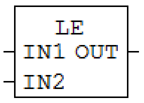
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 is less than IN2, else OUT will be False.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 < IN2;`

LE



Less than or equal to

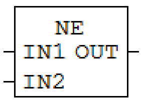
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 is less or equal than IN2, else OUT will be False.

Number of inputs can be expanded.

ST syntax example: `OUT := IN1 <= IN2;`

NE



Not equal to

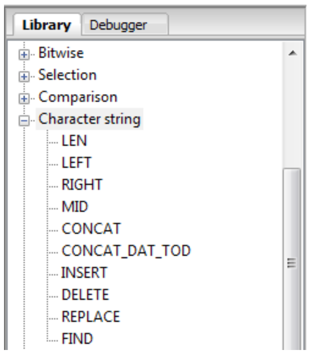
(ANY:IN1, ANY:IN2) => (BOOL:OUT)

OUT will become True if IN1 and IN2 are NOT equal, else OUT will be False.

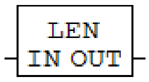
Number of inputs can be expanded.

ST syntax example: `OUT := IN1 <> IN2;`

2.7.11 Character string



LEN

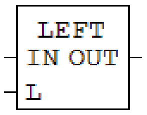


Length of string

(STRING:IN) => (INT:OUT)

OUT represents number of characters in IN string. For example IN string is 'ABCDEFGH', OUT will be 8.

LEFT

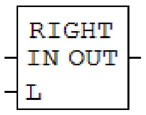


String left of

(STRING:IN, ANY_INT:L) => (STRING:OUT)

OUT represents string of L number of characters, leftmost of IN string. For example IN string is 'ABCDEFGH', L is 3, OUT string will be 'ABC'.

RIGHT

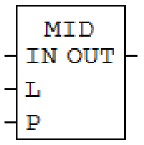


String right of

(STRING:IN, ANY_INT:L) => (STRING:OUT)

OUT represents string of L number of characters, rightmost of IN string. For example IN string is 'ABCDEFGH', L is 3, OUT string will be 'FGH'.

MID

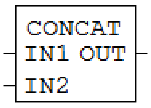


String from the middle

(STRING:IN, ANY_INT:L, ANY_INT:P) => (STRING:OUT)

OUT represents string of L characters of string IN, beginning at position P from left side of IN string. For example IN string is 'ABCDEFGH', L is 4 and P is 2. OUT string will be 'DE'.

CONCAT



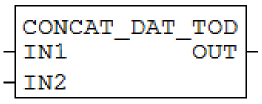
Concatenation

(STRING:IN1, STRING:IN2) => (STRING:OUT)

OUT represents concatenated string of IN1 and IN2. For example IN1 string is 'ABCD' and IN2 string is 'EFG', OUT string will be 'ABCDEFG'.

Number of inputs can be expanded.

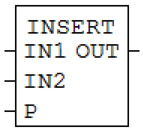
CONCAT_DAT_TOD



Time concatenation

(DATE:IN1, TOD:IN2) => (DT:OUT)

INSERT

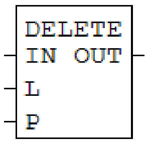


Insertion (into)

(STRING:IN1, STRING:IN2, ANY_INT:P) => (STRING:OUT)

OUT represents inserted string of IN2 to the IN1 string, after P position from left side of string IN1. For example if P is 2, IN1 string is 'ABC' and IN2 string is '12', OUT string will be 'AB12C'.

DELETE

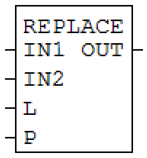


Deletion (within)

(STRING:IN, ANY_INT:L, ANY_INT:P) => (STRING:OUT)

OUT represents deleted string for number of L characters, beginning at IN string P position from left side of string IN1. For example L is 3, P is 2 and IN2 string is 'ABCDEFG, OUT string will be 'AEFG'.

REPLACE

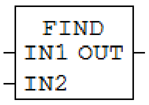


Replacement (within)

(STRING:IN1, STRING:IN2, ANY_INT:L, ANY_INT:P) =>
(STRING:OUT)

OUT represents replaced string for number of L characters, beginning at P position from left side of string IN1. For example L is 3, P is 2 and IN2 string is 'ABCDEFG, OUT string will be 'AEFG'.

FIND

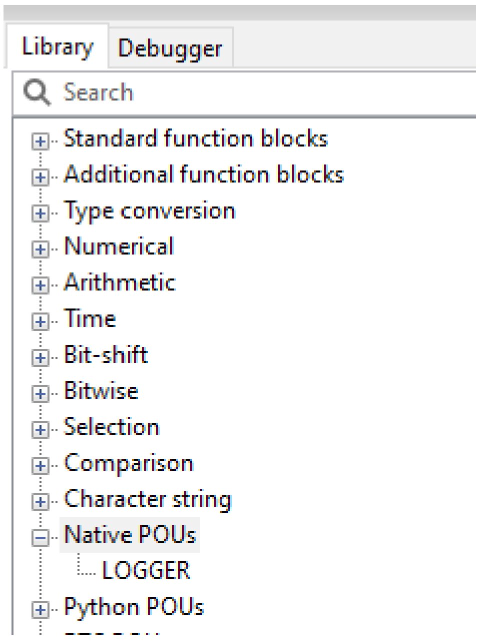


Find position

(STRING:IN1, STRING:IN2) => (INT:OUT)

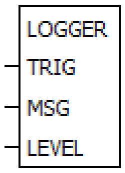
OUT represents first left position in string IN1 where string IN2 starts. If string IN2 is not found in string IN1, OUT will be 0. For example string IN1 is 'ABCDEFG', string IN2 is 'DEF', OUT will be 4.

2.7.12 Native POUs



- Standard function blocks
- Additional function blocks
- Type conversion
- Numerical
- Arithmetic
- Time
- Bit-shift
- Bitwise
- Selection
- Comparison
- Character string
- Native POUs
  - LOGGER
- Python POUs

LOGGER

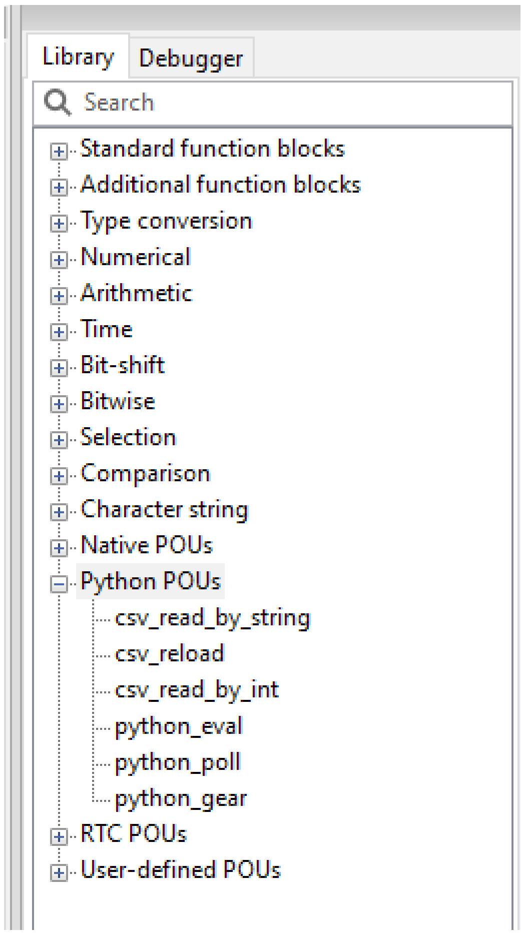


Logger

(BOOL:TRIG, STRING:MSG, LOGLEVEL:LEVEL) => ()

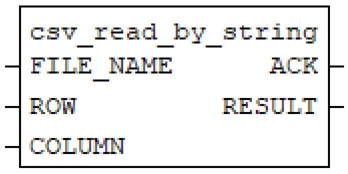
Logger is used for off-line logging. Log data defined by trig, message and log level are saved in MCU database. Log data can be read from PLC Log (2.10 PLC Log). Level must be a number between 0 and 3 written as Expression.

2.7.13 Python POUs



- Standard function blocks
- Additional function blocks
- Type conversion
- Numerical
- Arithmetic
- Time
- Bit-shift
- Bitwise
- Selection
- Comparison
- Character string
- Native POUs
- Python POUs
  - csv_read_by_string
  - csv_reload
  - csv_read_by_int
  - python_eval
  - python_poll
  - python_gear
- RTC POUs
- User-defined POUs

csv_read_by_string

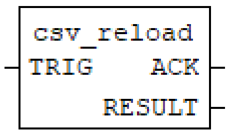


Read csv file by string

(STRING:FILE_NAME, STRING:ROW, STRING:COLUMN) => (BOOL:ACK, STRING:RESULT)

When CSV file FILE_NAME contains headers (rows/columns names) it preforms a search using ROW and COLUMN and returns found value RESULT (or error if row/column doesn't exist) and ACK is set to TRUE when output is ready.

csv_reload

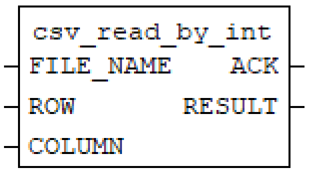


Reload csv

(BOOL:TRIG) => (BOOL:ACK, STRING:RESULT)

When new file is uploaded (or existing is changed), TRIG needs needs to be called to refresh all variables with new data. ACK is set to TRUE when output is ready and success in RESULT.

csv_read_by_int

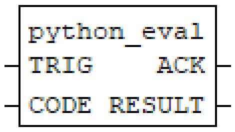


Read csv file by intiger

(STRING:FILE_NAME, INT:ROW, INT:COLUMN) => (BOOL:ACK, STRING:RESULT)

When CSV file FILE_NAME contains only integers, if preforms a search using indexes ROW and COLUMN and returns found value RESULT and ACK is set to TRUE when output is ready.

python_eval

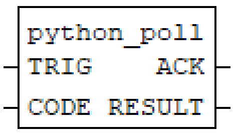


Python eval

(BOOL:TRIG, STRING:CODE) => (BOOL:ACK, STRING:RESULT)

Run python script CODE when TRIG is detected. ACK is set to TRUE when output is ready and RESULT is return of the code. Script runs until TRIG is set to False.

python_poll

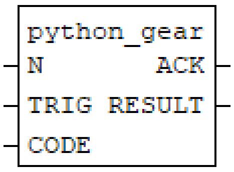


Python poll

(BOOL:TRIG, STRING:CODE) => (BOOL:ACK, STRING:RESULT)

Run python script CODE when TRIG is detected. ACK is set to TRUE when output is ready and RESULT is return of the code. Script executes only once.

python_gear

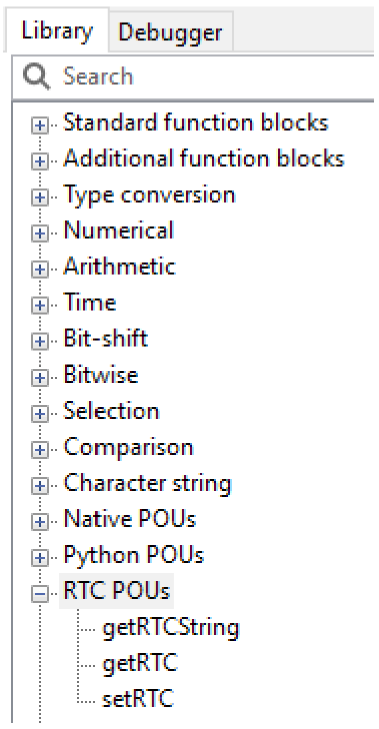


Python gear

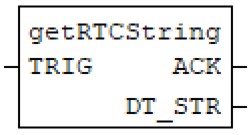
(UINT:N, BOOL:TRIG, STRING:CODE) => (BOOL:ACK, STRING:RESULT)

Run python script CODE when TRIG is detected. ACK is set to TRUE when output is ready and RESULT is return of the code. Script executes N times.

2.7.14 RTC POUs



getRTCString



Get RTC in string

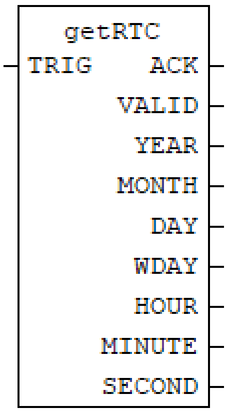
(BOOL:TRIG) => (BOOL:ACK, STRING:DT_STR)

Sample time from RTC on raising edge of TRIG. Outputs a formated string in DT_STR and sets ACK to TRUE when output is ready.

Format:

YYYY-mm-dd HH:MM:SS w v

getRTC

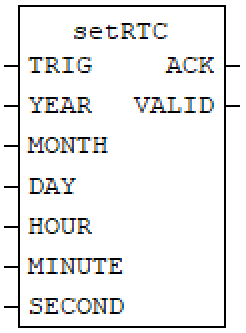


Get RTC

(BOOL:TRIG) => (BOOL:ACK, BOOL:VALID, UINT:YEAR, UINT:MONTH, UINT:DAY, UINT:WDAY, UINT:HOUR, UINT:MINUTE, UINT:SECOND)

Sample time from RTC on raising edge of TRIG. Outputs time as separate values and sets ACK to TRUE when output is ready. VALID is FALSE if hardware clock had a power failure and time is incorrect, TRUE otherwise.

setRTC

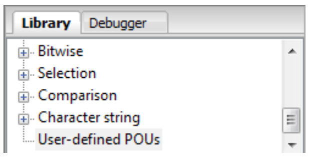


Set RTC

(UINT:N, BOOL:TRIG, STRING:CODE) => (BOOL:ACK, STRING:RESULT)

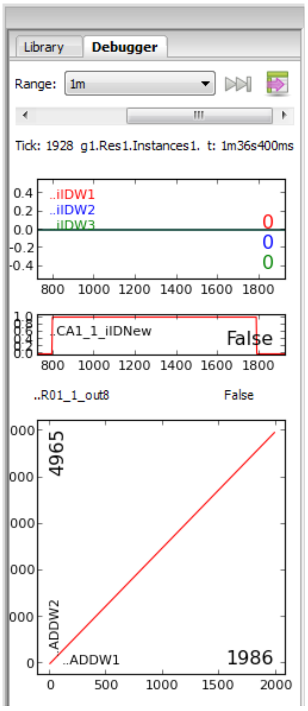
Set RTC time on raising edge of TRIG. VALID is FALSE if set date is invalid, TRUE otherwise.

2.7.15 User-defined POUs



User-defined POUs -All user-defined functions and function blocks are added in this library group.

2.8 Debugger



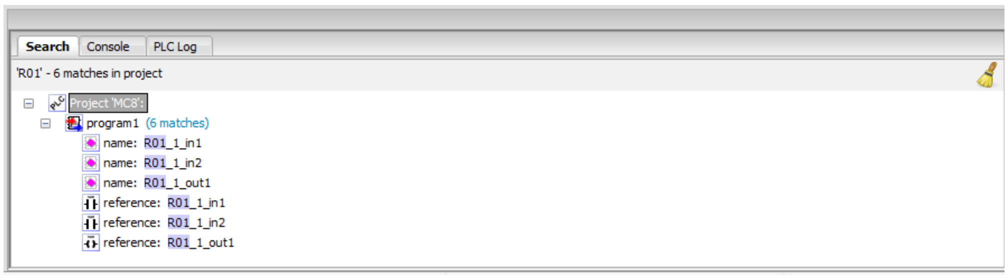
Debugger can display actual-online values of selected variables in numerical and graphical presentation. Variables to be displayed can be added here by clicking on its glasses icon in Project tab (instances window) or double-click on variable in editor workspace.
Actual values of the Variables can be presented as:
- numerical value display,
- one dimension graph with one or more variables (drag-and-drop variable to the right side of existing graph) and
- multi dimensional graph (drag-and-drop variable to the left side of existing graph).

Values-trends are displayed inside the selectable time range (10ms,..,1s,..1m,..,1h,.., 24h). The range is common for all observed variables.

Variables can also be exported to the clipboard.

Forcing (releasing) of these variables is also possible inside this window by clicking on padlock icon (move mouse pointer over the value and click on the padlock locked).

2.9 Search

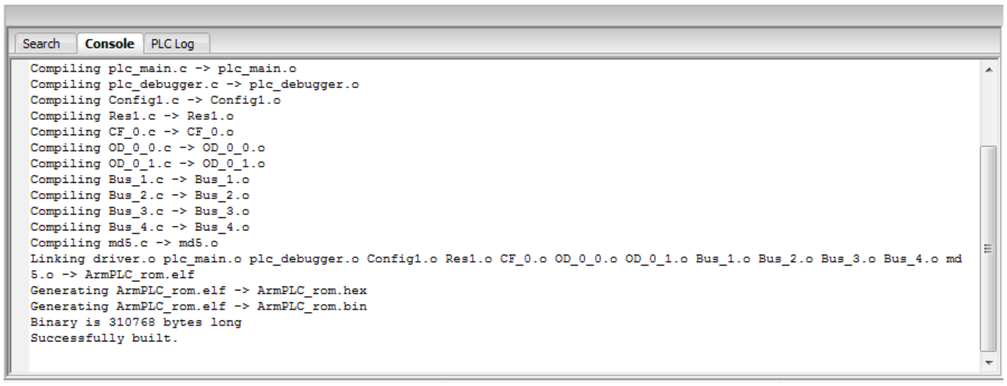


'R01' - 6 matches in project

```
Project 'MC8':
  program1 (6 matches)
    name: R01_1_in1
    name: R01_1_in2
    name: R01_1_out1
    reference: R01_1_in1
    reference: R01_1_in2
    reference: R01_1_out1
```

Search -Search window shows the result(s) of the Search in Project request from the Edit toolbar. Custom search pattern and different scope can be selected.

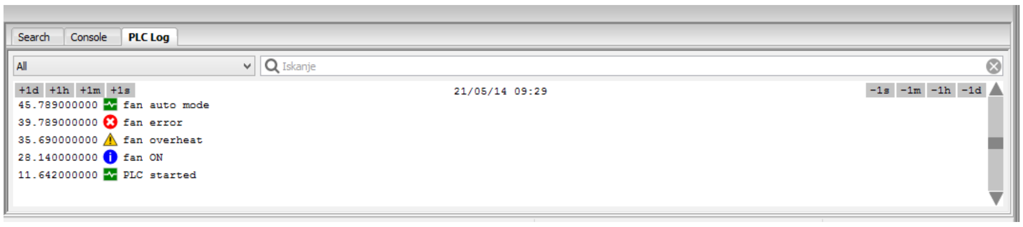
2.10 Console



```
Compiling plc_main.c -> plc_main.o
Compiling plc_debugger.c -> plc_debugger.o
Compiling Config1.c -> Config1.o
Compiling Res1.c -> Res1.o
Compiling CF_0.c -> CF_0.o
Compiling OD_0_0.c -> OD_0_0.o
Compiling OD_0_1.c -> OD_0_1.o
Compiling Bus_1.c -> Bus_1.o
Compiling Bus_2.c -> Bus_2.o
Compiling Bus_3.c -> Bus_3.o
Compiling Bus_4.c -> Bus_4.o
Compiling md5.c -> md5.o
Linking driver.o plc_main.o plc_debugger.o Config1.o Res1.o CF_0.o OD_0_0.o OD_0_1.o Bus_1.o Bus_2.o Bus_3.o Bus_4.o md
5.o -> ArmPLC_rom.elf
Generating ArmPLC_rom.elf -> ArmPLC_rom.hex
Generating ArmPLC_rom.elf -> ArmPLC_rom.bin
Binary is 310768 bytes long
Successfully built.
```

Console -Contains log of *LPC Manager* activities during processing system program activities (build, transfer, debugging, communication,...)

2.11 PLC Log



```
                                   21/05/14 09:29
45.789000000 fan auto mode
39.789000000 fan error
35.690000000 fan overheat
28.140000000 fan ON
11.642000000 PLC started
```

PLC Log -Contains log data logged by logger function which are stored on MCU. Log data are filtered by level type. Level type are indicate by signs (error sign = level 0, warning sign = level 1, information sign = level 2 and status sign = level 3).

3 PROGRAMMING LANGUAGES

LPC Manager is based on *PLC (programmable logic controller)* programming languages *International Standard IEC 61131-3*.

Following types of PLC programming languages can be used:

- Textual:

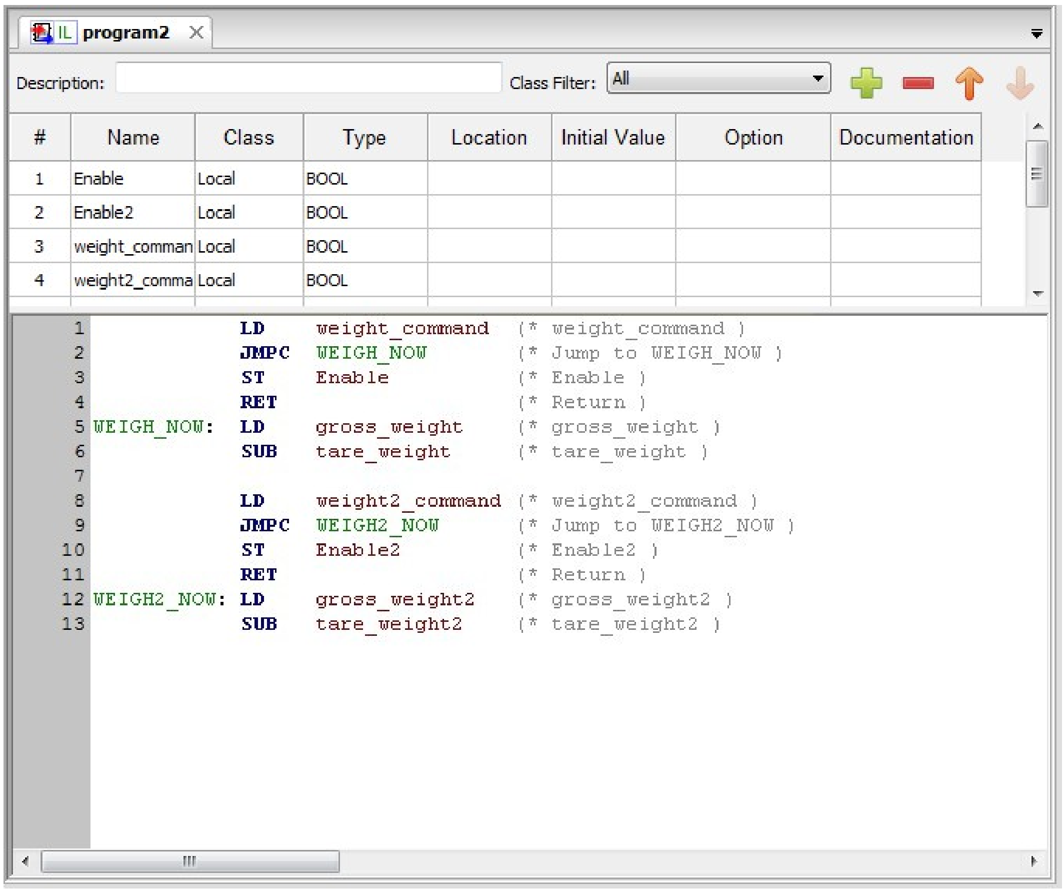
- IL	Instruction List
- ST	Structured Text

- Graphic:

- FBD	Function Block Diagram
- LD	Ladder Diagram
- SFC	Sequential Function Chart

3.1 IL - Instruction List

Instruction list programmable language is composed of a sequence of instructions. It is similar as assembler language. Each instruction shall begin on a new line and shall contain an operator with optional modifiers and, if necessary for the particular operation, one or more operands separated by commas. Operands can be literals, enumerated values and variables.

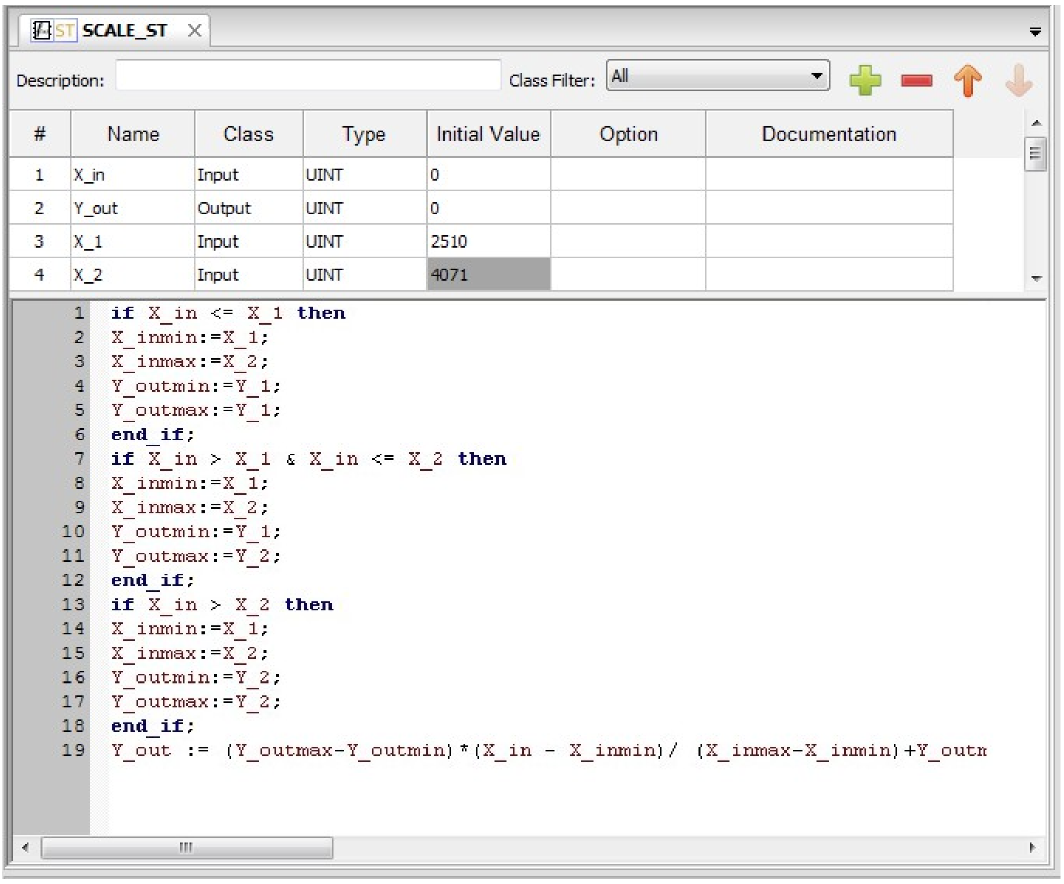


#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	Enable	Local	BOOL				
2	Enable2	Local	BOOL				
3	weight_comman	Local	BOOL				
4	weight2_comma	Local	BOOL				

```
           LD     weight_command   (* weight_command )
           JMPC   WEIGH_NOW        (* Jump to WEIGH_NOW )
           ST     Enable           (* Enable )
           RET                     (* Return )
WEIGH_NOW: LD     gross_weight     (* gross_weight )
           SUB    tare_weight      (* tare_weight )

           LD     weight2_command  (* weight2_command )
           JMPC   WEIGH2_NOW       (* Jump to WEIGH2_NOW )
           ST     Enable2          (* Enable2 )
           RET                     (* Return )
WEIGH2_NOW: LD    gross_weight2    (* gross_weight2 )
           SUB    tare_weight2     (* tare_weight2 )
```

3.2 ST - Structured Text

Structured text programmable language is composed of a sequence of instructions. It is higher level language similar as C. End of a textual line shall be treated the same as a space (SP) character.



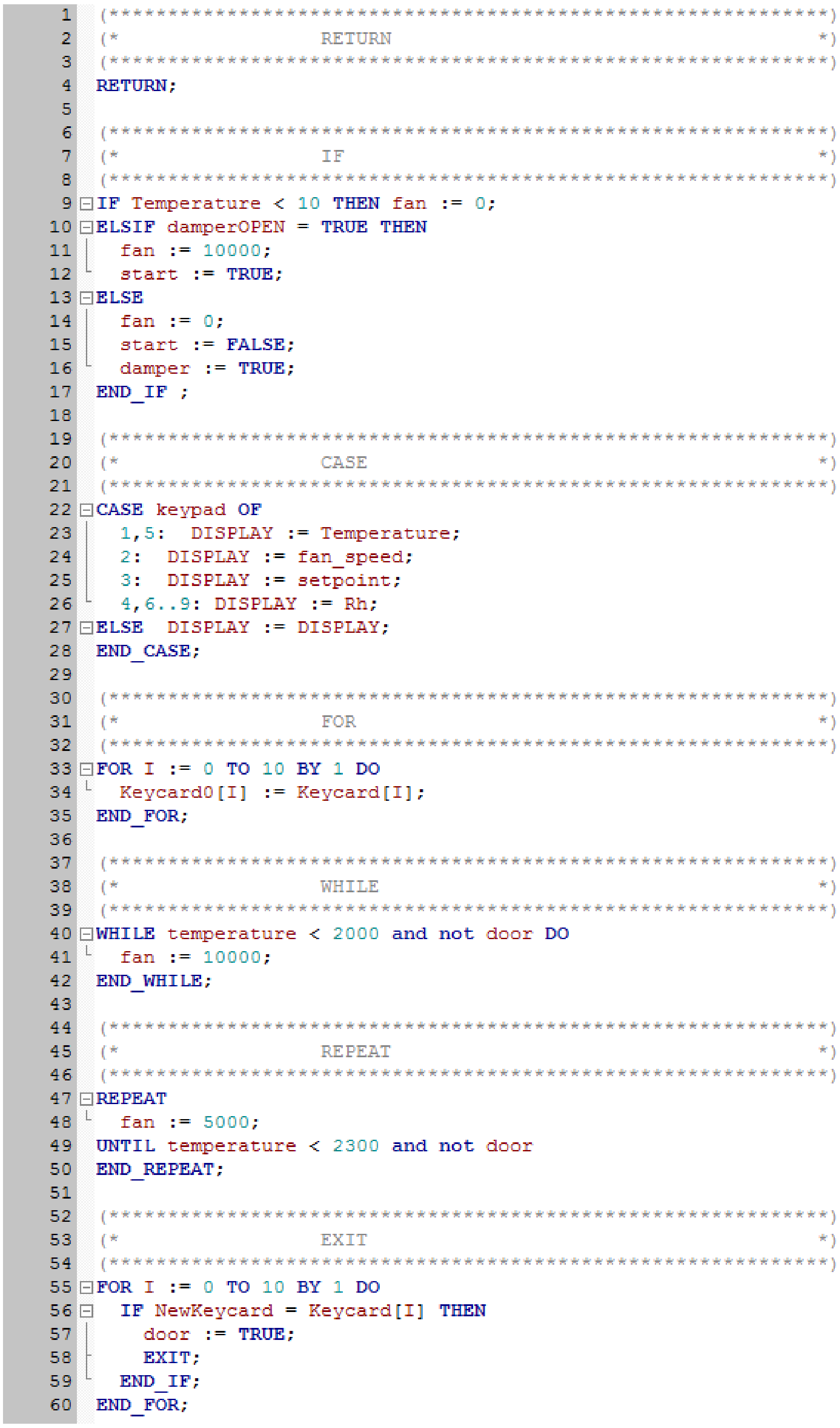
SCALE_ST

Description: Class Filter: All

#	Name	Class	Type	Initial Value	Option	Documentation
1	X_in	Input	UINT	0		
2	Y_out	Output	UINT	0		
3	X_1	Input	UINT	2510		
4	X_2	Input	UINT	4071		

```
if X_in <= X_1 then
X_inmin:=X_1;
X_inmax:=X_2;
Y_outmin:=Y_1;
Y_outmax:=Y_1;
end_if;
if X_in > X_1 & X_in <= X_2 then
X_inmin:=X_1;
X_inmax:=X_2;
Y_outmin:=Y_1;
Y_outmax:=Y_2;
end_if;
if X_in > X_2 then
X_inmin:=X_1;
X_inmax:=X_2;
Y_outmin:=Y_2;
Y_outmax:=Y_2;
end_if;
Y_out := (Y_outmax-Y_outmin)*(X_in - X_inmin)/ (X_inmax-X_inmin)+Y_outn
```

Structured text syntax examples:

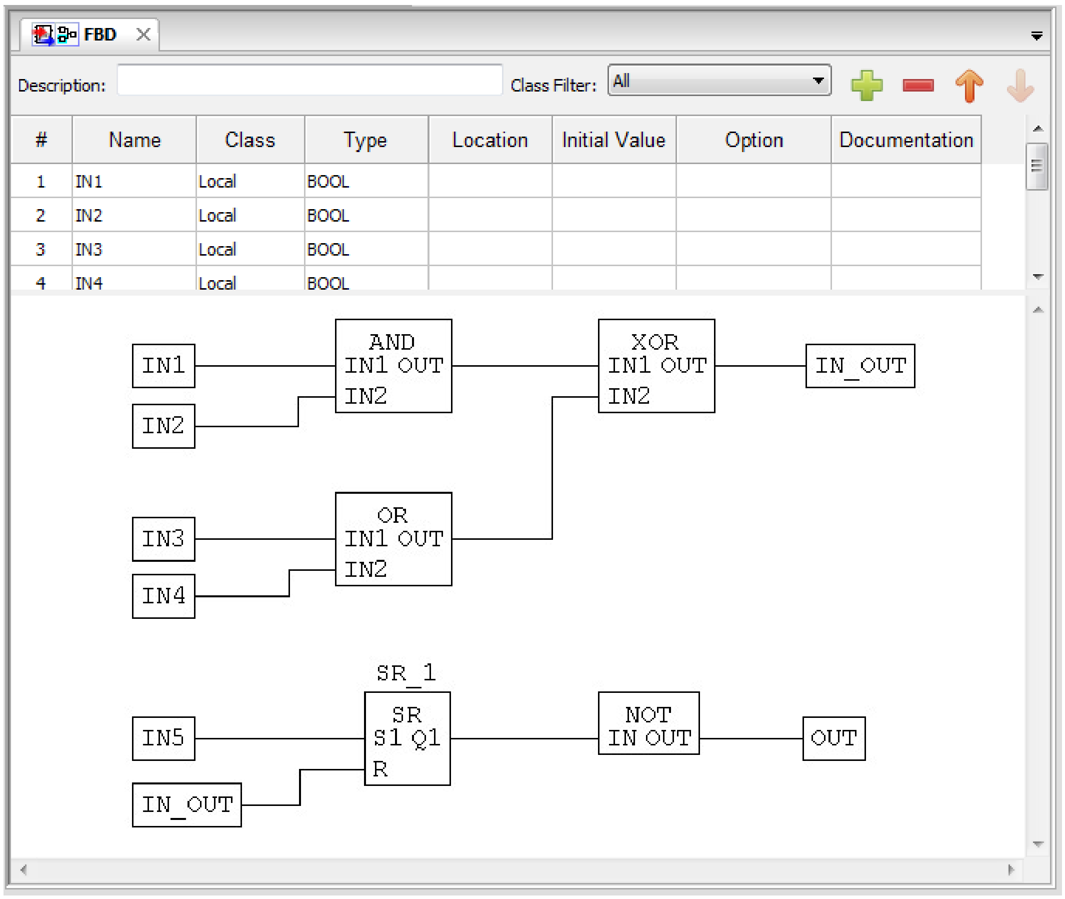


```
(*****************************************************************)
(*                    RETURN                                     *)
(*****************************************************************)
RETURN;

(*****************************************************************)
(*                    IF                                         *)
(*****************************************************************)
IF Temperature < 10 THEN fan := 0;
ELSIF damperOPEN = TRUE THEN
  fan := 10000;
  start := TRUE;
ELSE
  fan := 0;
  start := FALSE;
  damper := TRUE;
END_IF ;

(*****************************************************************)
(*                    CASE                                       *)
(*****************************************************************)
CASE keypad OF
  1,5:  DISPLAY := Temperature;
  2:  DISPLAY := fan_speed;
  3:  DISPLAY := setpoint;
  4,6..9: DISPLAY := Rh;
ELSE  DISPLAY := DISPLAY;
END_CASE;

(*****************************************************************)
(*                    FOR                                        *)
(*****************************************************************)
FOR I := 0 TO 10 BY 1 DO
  Keycard0[I] := Keycard[I];
END_FOR;

(*****************************************************************)
(*                    WHILE                                      *)
(*****************************************************************)
WHILE temperature < 2000 and not door DO
  fan := 10000;
END_WHILE;

(*****************************************************************)
(*                    REPEAT                                     *)
(*****************************************************************)
REPEAT
  fan := 5000;
UNTIL temperature < 2300 and not door
END_REPEAT;

(*****************************************************************)
(*                    EXIT                                       *)
(*****************************************************************)
FOR I := 0 TO 10 BY 1 DO
  IF NewKeycard = Keycard[I] THEN
    door := TRUE;
    EXIT;
  END_IF;
END_FOR;
```

3.3 FBD - Function Block Diagram

Function block diagram programmable language is a graphical language for the programming of programmable controllers.

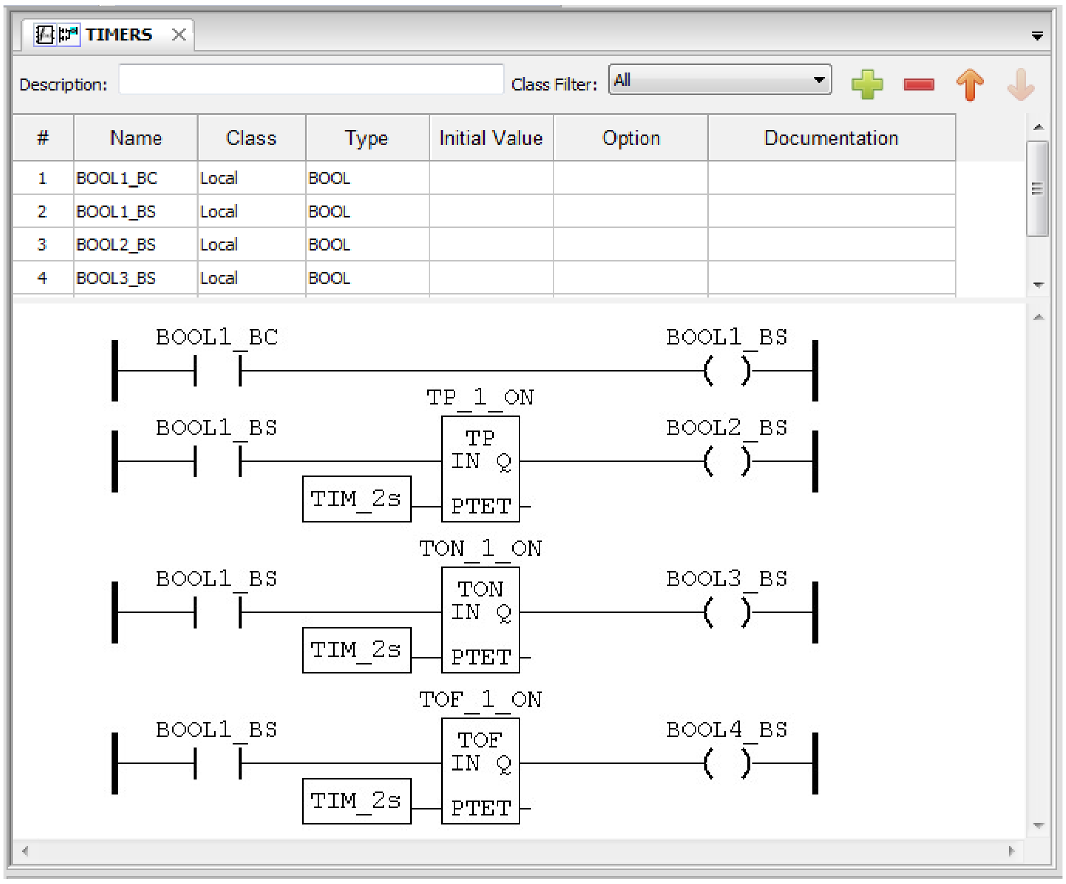
Elements of the FBD language shall be interconnected by signal flow lines. Outputs of function blocks shall not be connected together. In particular, the "wired-OR" construct of the LD language is not allowed in the FBD language. An explicit Boolean "OR" block is required instead.



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	IN1	Local	BOOL				
2	IN2	Local	BOOL				
3	IN3	Local	BOOL				
4	IN4	Local	BOOL				

3.4 LD - Ladder Diagram

Ladder diagram programmable language is a graphical language for the programming of programmable controllers. It enables the programmable controller to test and modify data by means of standardized graphic symbols. These symbols are laid out in networks in a manner similar to a "rung" of a relay ladder logic diagram. LD networks are bounded on the left and right by power rails.



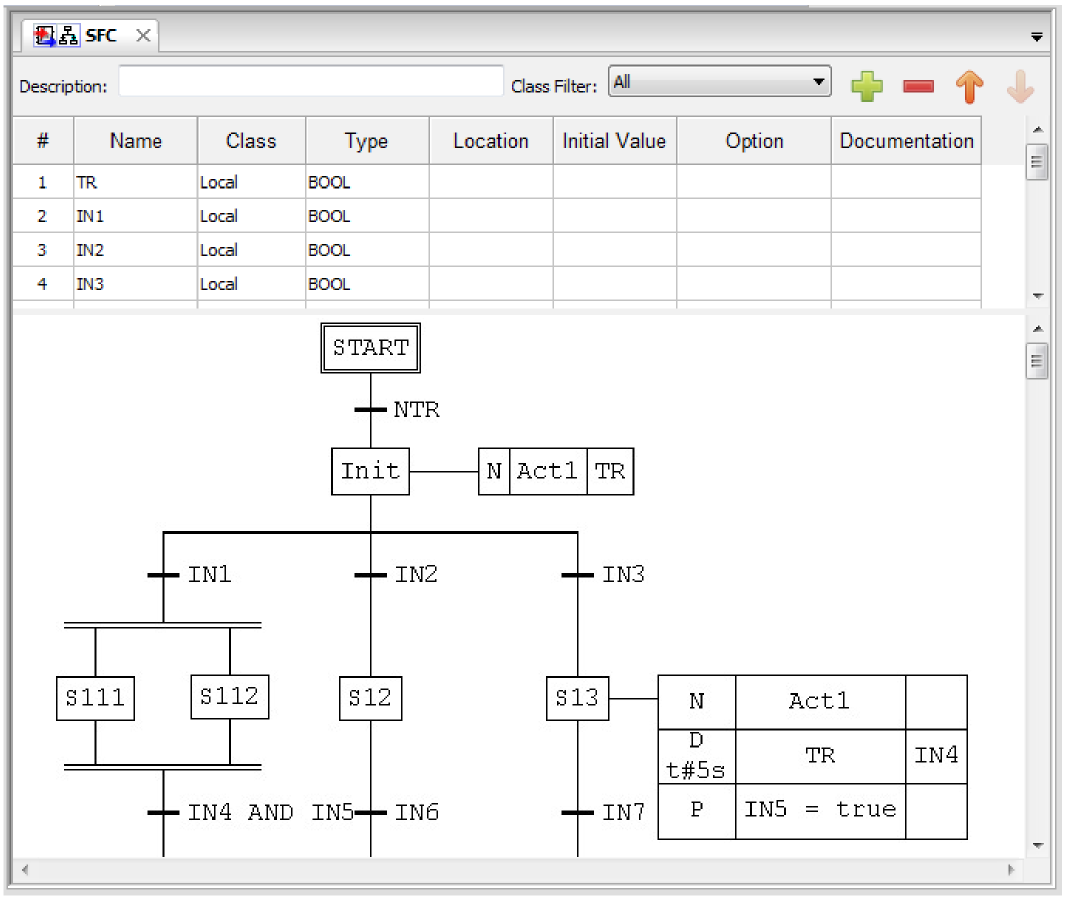
#	Name	Class	Type	Initial Value	Option	Documentation
1	BOOL1_BC	Local	BOOL			
2	BOOL1_BS	Local	BOOL			
3	BOOL2_BS	Local	BOOL			
4	BOOL3_BS	Local	BOOL			

3.5 SFC - Sequential Function Chart

Sequential function chart programmable language is a graphical language for the programming of programmable controllers. Sequential function chart elements are used to structure the internal organization of a programmable controller program organization unit, written in one of the *PLC* (programming logic controller) programming languages explained in this document, for the purpose of performing sequential control functions.

The SFC elements provide a means of partitioning a programmable controller 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 a transition condition.

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 shall be so partitioned. If no SFC partitioning is given for a program organization unit, the entire program organization unit shall be considered to be a single action which executes under the control of the invoking entity.



#	Name	Class	Type	Location	Initial Value	Option	Documentation
1	TR	Local	BOOL				
2	IN1	Local	BOOL				
3	IN2	Local	BOOL				
4	IN3	Local	BOOL				

APPENDIX A – ERROR REPORTING

If you think you found a bug in our software or you have an idea of what can be improved or added, you are most welcome to share your thoughts with us (support@smarteh.si). We will consider the possibilities and try to include them in our next release.

You should contact your vendor with the description. The following information should be included:

- Software version.
- Detailed description of the bug or idea.
- If possible, steps that will recreate the problem (if bug is being reported).
- Your contact information (e-mail, phone, fax).

In case we need more information we may need to contact you before we can determine the exact solution. And remember: the only software without a need for maintenance is the software not being used!

APPENDIX B - LICENSE

APPENDIX C – DOCUMENT HISTORY

The following table describes all the changes to the document.

Date	V.	Description
30.09.2011	-	The preliminary version, issues as *LPC Manager User Manual*.
30.01.2012	001	First release.
30.06.2012	002	Changes from previous version.
25.05.2014	003	Change according new release of LPC Composer 5.0.1.32
22.01.2016	004	Change according new release of LPC Smarteh IDE 5.1.4.2
25.08.2018	005	Change according new release of LPC Smarteh IDE 5.4.0.0
14.05.2024	006	Added new extensions and library POUs
12.06.2025	007	Added Appendix B - License