



akYtec ALP v2.11

Programming software

User manual

akYtec ALP v2.11_3-EN-130795-1.26
© All rights reserved
Subject to technical changes and misprints

Contents

1. Introduction.....	5
1.1. System requirements	5
1.2. Terms and abbreviations	5
2. Installation	7
3. User interface.....	10
3.1. Main menu	10
3.2. Toolbars	12
3.3. Library Box.....	13
3.4. Property box.....	16
3.5. Variable Box	16
3.6. Workspace	18
3.7. Status bar	19
3.8. Display manager.....	20
4. General information	24
4.1. Program execution.....	24
4.2. Project creation	24
4.2.1. Protecting a project with a key	25
4.3. Program creation	27
4.3.1. Text field	29
4.3.2. Standard variable block	30
4.3.3. Constant block	32
4.3.4. Delay line.....	33
4.3.5. Network variable	34
4.3.6. Read from/Write to FB	35
4.3.7. Conversion block.....	36
4.3.8. Arrange elements.....	37
4.3.9. Execution sequence	37
4.4. Display programming	38
4.4.1. Monochrome text LCD	40
4.4.2. Graphic color LCD	43
4.5. Simulation	47
4.5.1. Visualization Simulation	50
4.5.2. ST code simulation	51
4.6. Connection to device.....	54
4.7. Upload project to device	55
4.8. Online debugging	56
4.9. Project information.....	58
4.10. Component manager	59
4.11. Macro development.....	61
4.12. Using ST function.....	69
4.13. ST function blocks.....	74
5. Device configuration	79
5.1. Display.....	79
5.2. Clock	80
5.3. Data exchange	81

5.3.1. Interfaces	81
5.3.2. Modbus	83
5.4. Extension modules	94
5.5. Inputs and outputs	95
5.6. Password	96
5.7. Connecting to akYtec Cloud	97
6. Variables	99
6.1. Data types	104
6.2. Service variables	105
6.3. Network variables	105
6.4. Binding variables to parameters	106
6.5. Copy-paste variable block	107
7. Library	109
7.1. Functions	109
7.1.1. Logical operators	109
7.1.2. Mathematical operators	112
7.1.3. Relational operators	115
7.1.4. Bitshift operators	117
7.1.5. Bit operators	119
7.2. Function blocks	121
7.2.1. Triggers	121
7.2.2. Timers	123
7.2.3. Generators	126
7.2.4. Counters	127
7.2.5. Analog	129
7.3. Project macros	136
7.4. ST functions	137
7.5. ST function block	141
7.6. Display elements	145
7.6.1. Text box	146
7.6.2. I/O box (INT/REAL)	146
7.6.3. I/O box (BOOL)	147
7.6.4. Dynamic box	148
7.6.5. ComboBox	149
7.7. Basic graphic elements	150
7.7.1. Text	151
7.7.2. Indicator	152
7.7.3. Progress bar	154
7.7.4. Dynamic box	155
7.7.5. REAL/INT input/output	157
7.7.6. Time input/output	160
7.7.7. Input/output of IP address	161
7.7.8. Line	163
7.7.9. Polygon	164
7.7.10. Circle	165
7.7.11. Button	166
7.7.12. Switch	168

7.7.13. Switch group	170
7.7.14. Image	173
7.7.15. Menu	174
7.7.16. Chart	176
8. Device	181
8.1. Device information	181
8.2. Cycle time	182
8.3. Firmware update / repair	182
8.4. Calibration.....	183
8.4.1. Input calibration.....	184
8.4.2. Output calibration	185
8.5. Change target device	185
9. Plugins	187
9.1. Replication master	188
9.1.1. Writing a program to the device via the Replication master on Windows	192
9.1.2. Writing a program to a device via the Replication master on Linux	195
9.2. Exporting the device to akYtec Cloud	196
10. Keyboard shortcuts	198
11. Program examples.....	200
11.1. Task 1: Light switch with automatic switch-off	200
11.2. Task 2: Mixer control.....	202
11.3. Task3. Direct connection of PR103 to akYtec Cloud	205
12. ST language	210
12.1. Syntax	210
12.1.1. Bitwise access to variables.....	210
12.1.2. Using functions in other ST program elements	210
12.1.3. Using one function block in another.....	211
12.1.4. Comments in ST editor	212
12.1.5. Copying ST elements between projects	212
12.2. Documentation in the ST editor	213
12.3. Data types	214
12.3.1. Reserved keywords	215
12.3.2. Arrays.....	216
12.4. Language structures	217
12.4.1. Operations	217
12.4.2. Assignment operation	220
12.4.3. IF statement.....	220
12.4.4. CASE statement.....	221
12.4.5. RETURN statement.....	222
12.4.6. FOR statement.....	222
12.4.7. WHILE statement	224
12.4.8. REPEAT UNTIL statement	224
12.5. System functions	225
12.6. System Function Blocks	226
12.6.1. Triggers	227
12.6.2. Timers	231
12.6.3. Generators.....	237

12.6.4. Counters.....	238
-----------------------	-----

1 Introduction

ALP is a programming software for programmable devices of akYtec GmbH. ALP uses graphical language FBD (Function block diagram) and ST (Structure Language) as programming languages, according to IEC 61131-3. The software enables simulation and debugging of the created program and its upload to the non-volatile memory of the device.

Created project contains minimum one circuit program and device configuration. The first workspace contains the main circuit program. In separate workspaces user can create macros as circuit programs. If the target device has a display, it can be programmed using display forms in separate workspaces. Only one project can be opened in ALP at a time.

akYtec ALP has different functions for the following groups of devices:

- first generation devices (PR100, PR102, PR200 and SMI200)
- second generation devices (PR103, PR205 and newer)

Basic akYtec ALP functionality is available for all devices, functionality and interfaces for devices on the second generation are not available for first generation devices.

1.1 System requirements

Operation systems:

- Windows 7 (SP1+)
- Windows 8.1
- Windows 10
- Windows 11

System libraries:

- Microsoft .NET Framework 4.8
- Microsoft .NET Desktop Runtime 6.0.8
- Microsoft Visual C++ 2015-2022

Recommended hardware requirements:

- 3.2 GHz processor
- 4 GB RAM
- 700 MB available hard disk space
- Free USB port
- Keyboard and mouse
- Screen resolution 1280×800

Internet connection is required for:

- Software update
- Device firmware update
- Macros download in Component manager

1.2 Terms and abbreviations

- **ALP** — programming software for programmable devices of akYtec GmbH.
- **EEPROM** — electrically erasable programmable read-only memory of the device.
- **FBD (Function Block Diagram)** — graphical programming language supported by IEC 61131-3.
- **Function** — Structural program unit with one return value. The function does not store information about its internal state, i.e. if the function is called with the same input values, it returns the same output value.
- **Function block** — Structural program unit with internal memory and one or more output values. It is used in program as an instance, i. e. a copy with its own memory.
- **Macro** — user function block.
- **Program cycle** — execution time of the circuit program, which depends of its complexity.
- **Project** — user application created for a specific device with ALP software, includes the circuit program.
- **ST (Structured Text)** — a text programming language supported by IEC 61131-3.
- **Target device** — device type for which the project is created.

- **Workspace** — graphic area in the user interface for the program creation, modification and debugging.

2 Installation

1. Download and run the akYtecALP.exe file on your PC.
2. Select setup language.

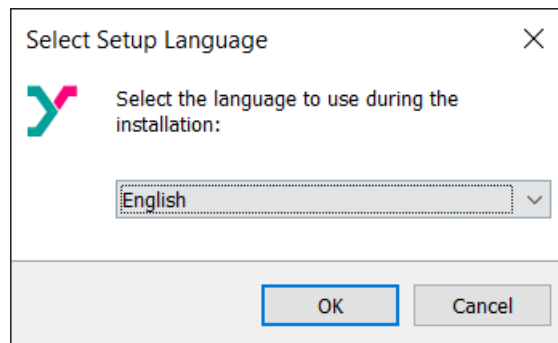


Fig. 2.1 Setup language selection

3. The installation wizard window will open. Review the information and click the **Next** button.

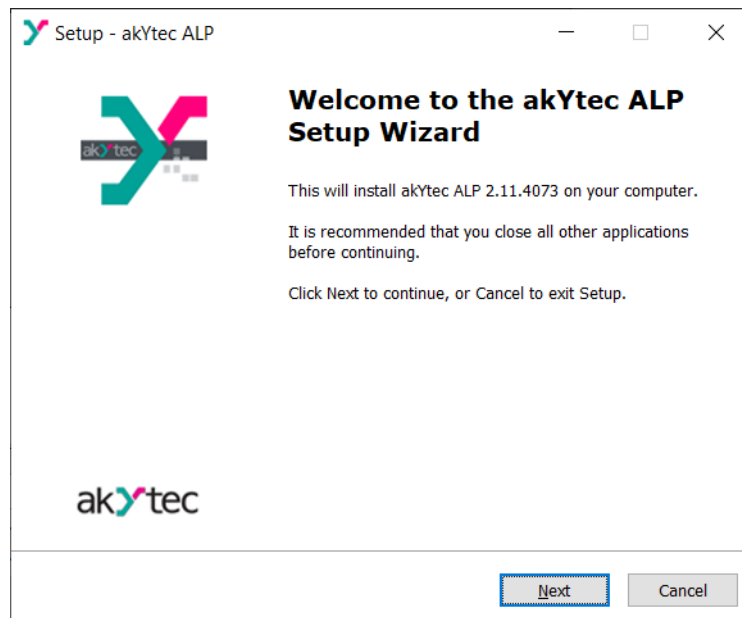


Fig. 2.2 Installation Wizard Window

4. Select the installation folder for akYtec ALP and the folder for the Start menu shortcut. Click the **Next** button.
5. If necessary, enable the corresponding checkboxes to create a desktop shortcut, create a quick launch panel shortcut, associate files with the .owl and .owle extensions with akYtec ALP, and install the .NET6 SDK library for creating a Replication Wizard for Linux OS. Click the **Next** button.

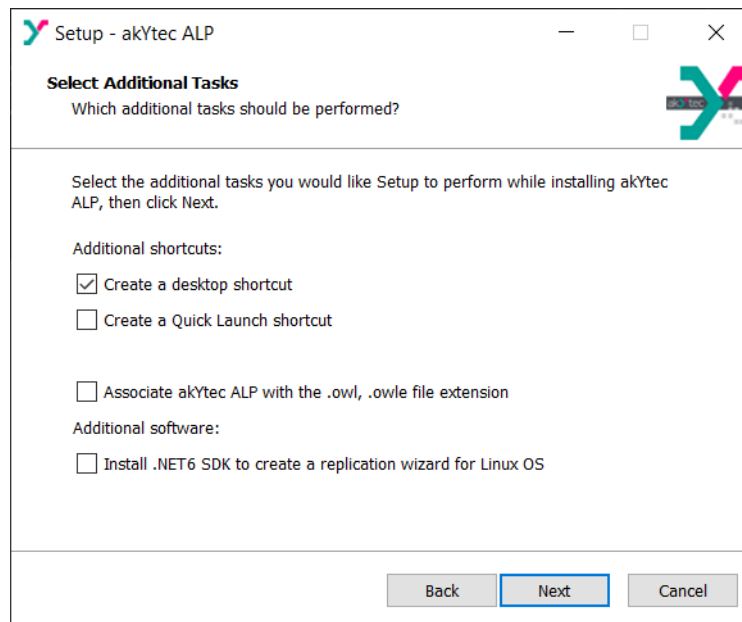


Fig. 2.3 Selecting Additional Tasks

6. Review the installation information and click the **Install** button.

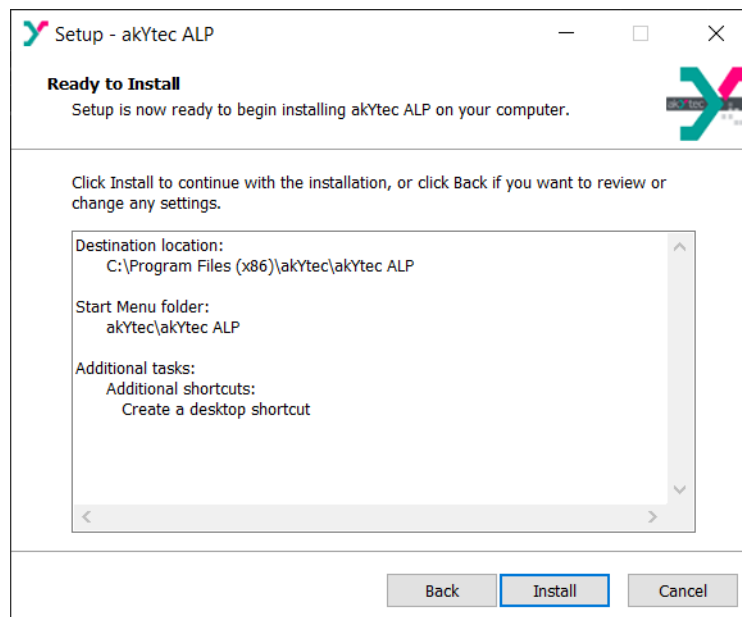


Fig. 2.4 Installation launch window

7. A window will open displaying the installation progress.

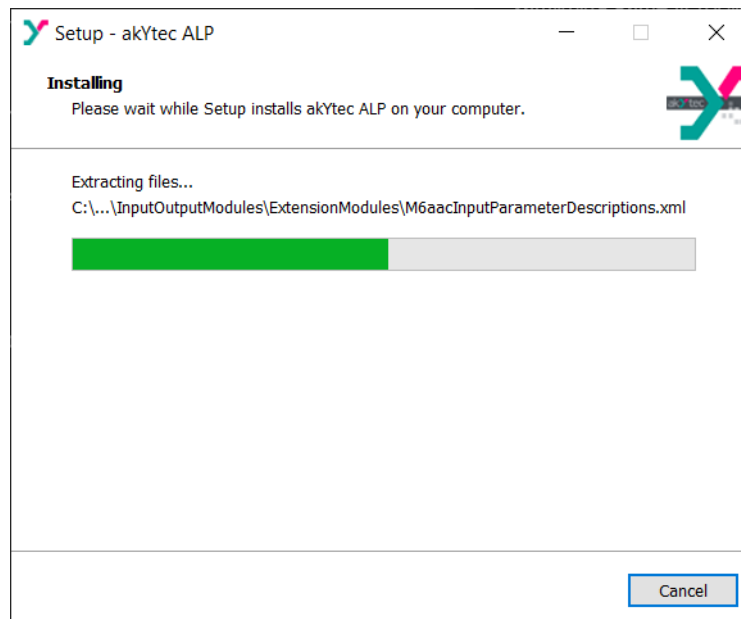


Fig. 2.5 Program installation process

8. Wait for the installation to complete and, if necessary, check the **Launch akYtec ALP** checkbox.

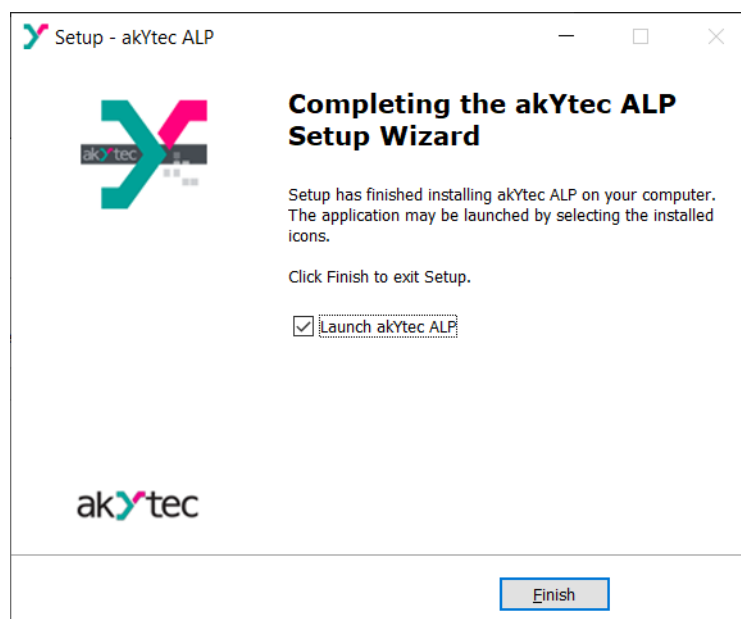
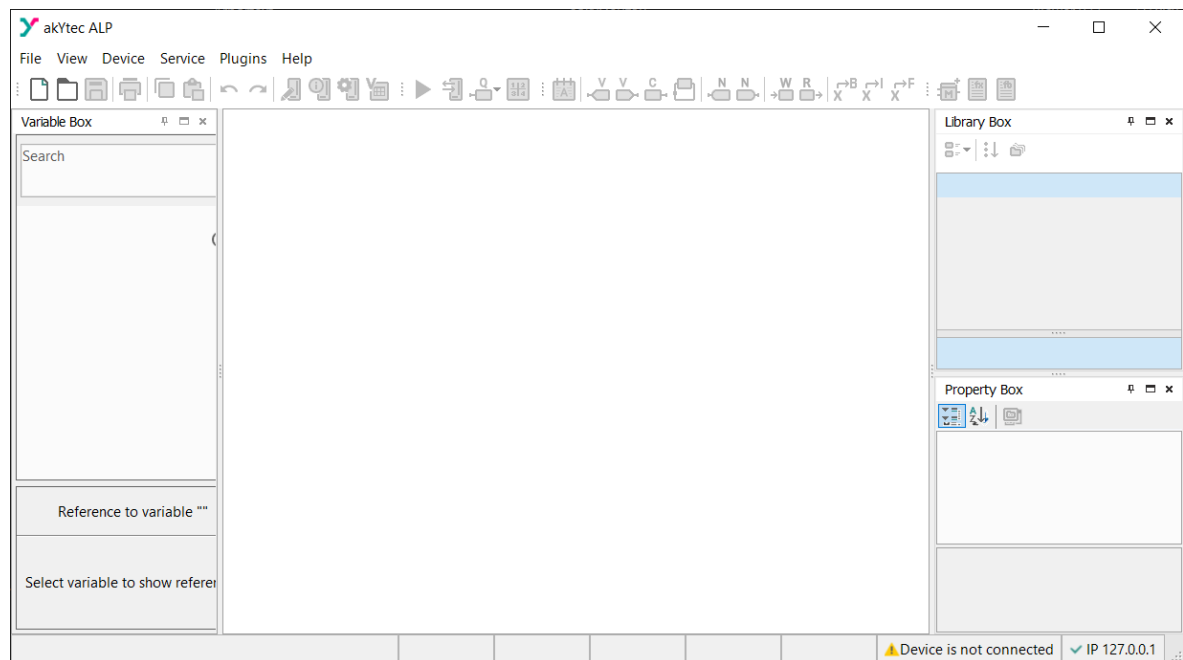


Fig. 2.6 Installation completion

9. Click the **Finish** button.

3 User interface



1. **Title bar** — shows the name of the software and the path to the open project file.
2. **Menu bar** — contains following groups: File, View, Device, Service, Plugins and Help.
3. **Tool bars** — Standard, Service and Insert: quick access to the essential functions of ALP.
4. **Library box** — the panel shows all the functions, function blocks, and macros that can be used in the project (drag-and-drop using).
5. **Property Box** — panel for viewing and modifying properties of the selected project element
6. **Workspace** — graphic area for creating and modifying circuit programs, ST programs, display elements or display forms.
7. **Status bar** — shows status and error messages, target device memory usage, status of the connected device and the programming interface.
8. **Display Manager** — a tool to program the display information (available only for the devices with display).
9. **Variable Box** — this panel shows all project variables with their parameters and references (drag-and-drop using).
10. **Component manager** — special tool in a separate window to access Online Database and to add the elements from the Online Database to the offline library or to the project library. Internet connection is needed.

3.1 Main menu

File

New project	Open a new project. The current project will be closed
Change target device	Change the target device in the project
Open project	Open a previously saved project
Save active workspace	Save the active workspace
Save project	Save the current project
Save project as...	Make a copy of the project in a different folder or under a different name

Create key file...	Create a file with a key to protect the project from unauthorized access
Project information	View and modify the <u>information about the project</u>
New macro	Open a new macro in the separate <u>workspace</u>
Import	Import a macro, an ST function or an ST function block from a file into the project library
Export	Save the current macro, ST function or ST function block as file
Component manager	Open the <u>Component manager</u> interface
Print	Open the dialog to set the print options and print the active workspace
Recent projects	List of recently opened projects
Exit	Close ALP

View

Undo	Undo the last action
Redo	Redo the last undone action
Status indicators	Add / remove the in indicators to / from the <u>status bar</u>
Library Box	Show / hide <u>Library Box</u>
Property Box	Show / hide <u>Property Box</u>
Variable Box	Show / hide <u>Variable Box</u>
Display manager	Show / hide <u>Display Manager</u>
Reset layout	Restore the default panel layout

Device

Transfer application to device	Upload the current project to the device memory
Firmware update	Update the firmware of the connected device
Device information	<u>Information</u> about the software, the target device and the connected device
Variable table	The <u>editable table</u> of the project variables with their parameters
Calibration	Start analog I/O <u>calibration</u>
Device configuration	Open <u>Device configuration</u> interface
Port configuration	Settings of the <u>programming interface</u>

Service

Arrange elements	Function blocks of the same type are automatically renumbered in the workspace from top to bottom and from right to left
Simulation mode	Start / stop <u>simulation</u> mode

Language	Select the interface language
OFFLINE mode	Activate OFFLINE mode

Help

Automatic update check	If activated, the update check is performed on program startup
Check for updates...	Check for software updates
Help	Open Help window
Version history	Running list of the changes have been made in all software versions
About Software	Information about the current software version

3.2 Toolbars





Standard



	New project	Open a new project. The currently opened project will be closed
	Open project	Open a previously saved project
	Save project	Save the current project
	Print	Set the print options for the active workspace
	Copy	Copy the selected element
	Paste	Paste the copied element
	Undo	Undo the last action
	Redo	Redo the last undone action
	Transfer application to device	Upload the current project to the device memory
	Device information	Information about the software, the target device and the connected device
	Device configuration	Device configuration
	Variable table	Table of project variables
















Service



	Simulation	Start / stop simulation
	Online debugging	Start / stop online debugging
	Execution order	Change the execution order for the outputs or delay lines on a circuit program or in a macro
	Arrange elements	Function blocks of the same type are automatically renumbered in the workspace from top to bottom and from right to left

Insert



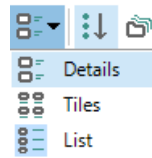
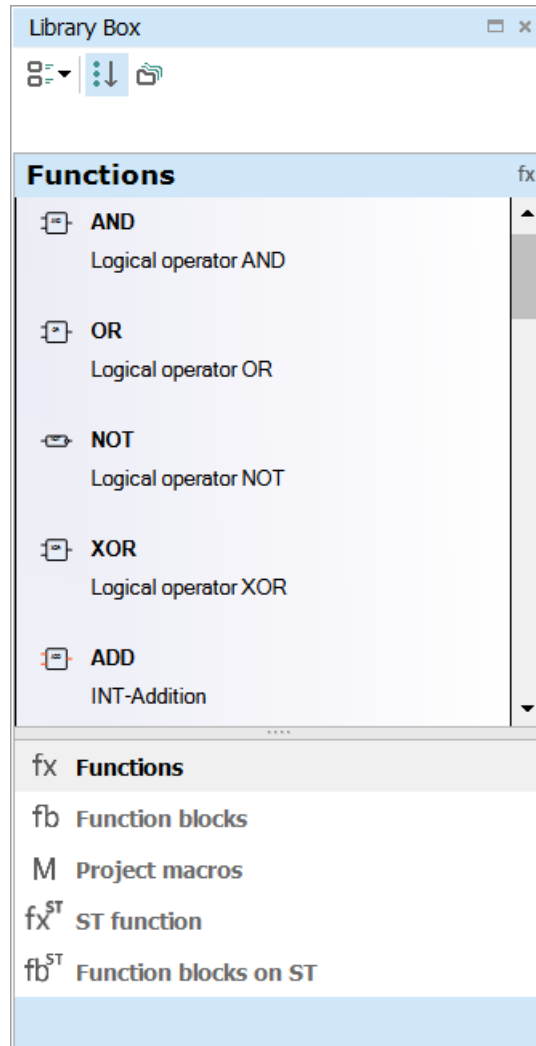
	Text field	Text field to comment the program
	Variable output block	Block for writing a variable into the program
	Variable input block	Block for reading a variable from the program
	Constant block	Constant value
	Delay line	Feedback with one-cycle delay
	Network variable output block	Block for writing a variable via network
	Network variable input block	Block for reading a variable via network
	WriteToFB block	Connects the output value of the block to the selected parameter of the selected function block. Allows you to edit the parameter
	ReadFromFB block	Connects the output value of the block to the selected parameter of the selected function block. Allows you to read the parameter
	Conversion to BOOL	Conversion of any values to a BOOL value
	Conversion to INT	Conversion of any values to an INT value
	Conversion to REAL	Conversion of any values to a REAL value
	New macro	New user macro
	New ST function	New user function in ST language
	New ST function block	New user function block in ST language

3.3 Library Box


The **Library Box** panel is a summary of program blocks available in the project. The panel consists of libraries:

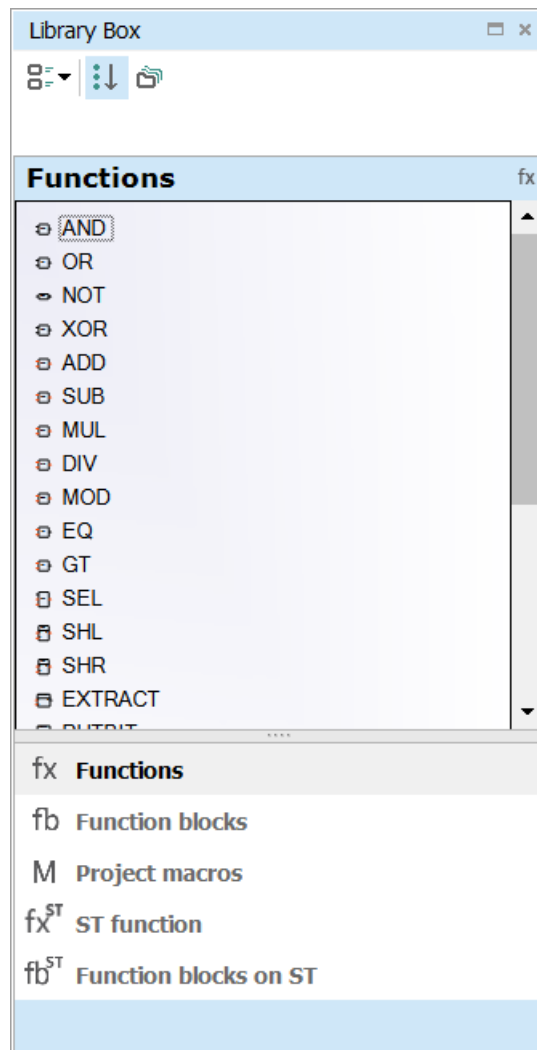
- Functions
- Function blocks
- Project macros
- ST functions
- ST function blocks


Select an item on the lower toolbar of the panel to show the respective content.
The standard position of the panel is the upper right window corner (can be changed).

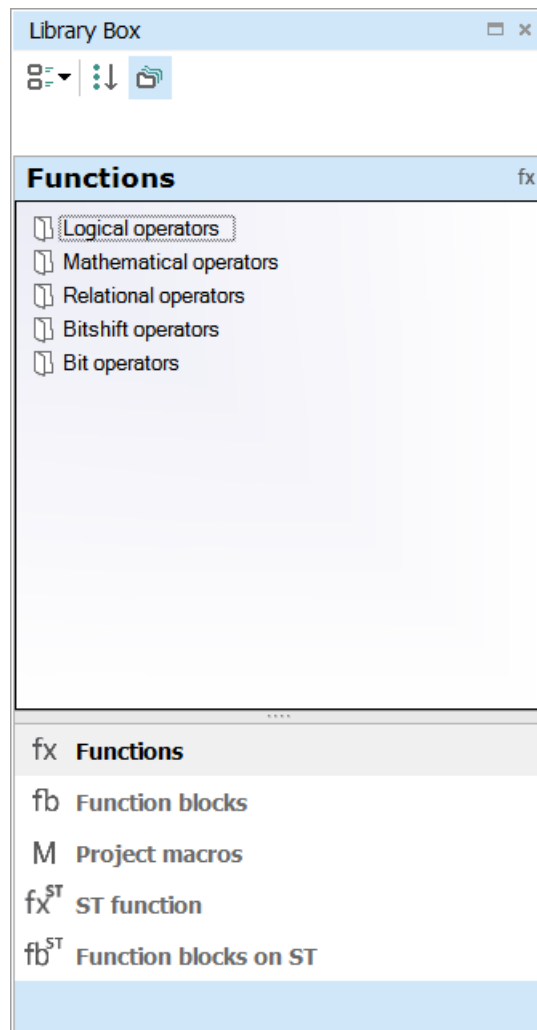


The panel view can be changed using the icons on the upper toolbar.

- Click the icon  **Show all** to show all the blocks of the selected library.



- Click the icon  **Show grouped** to show the blocks of the selected library group. Double-click the folder to open it.




For description of the library groups and individual blocks see section [Library](#).

3.4 Property box

The panel Property Box is used to view and modify the parameters of the program elements. Select the element to view its properties. If no item is selected, the panel displays the properties of the workspace.

The standard position of the panel is the lower right window corner (can be changed).

The parameters are shown grouped by categories by default.

To show them in alphabetical order, click the icon .



To show them grouped, click the icon .

Select the parameter input field to edit its value.

3.5 Variable Box

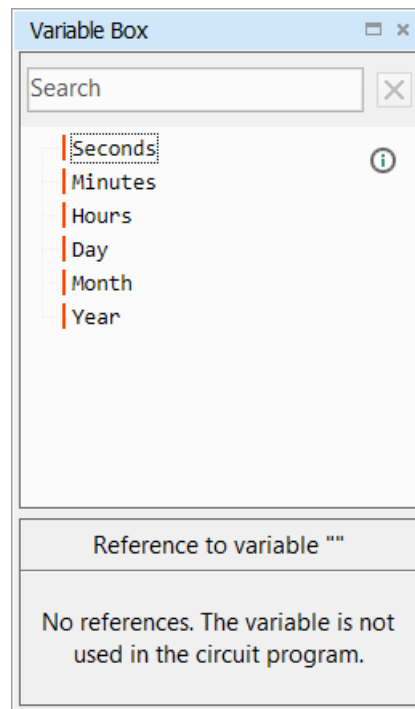
The panel **Variable Box** shows the project variables from the variable table.

The standard position of the panel is the upper left window corner (can be changed).

The variable properties are available in a tooltip text.

Variable block in the workspace

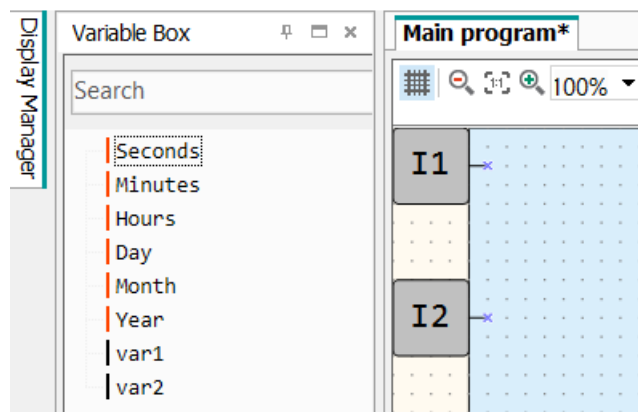
Drag-and-drop a variable to place it in the circuit program as an input block.



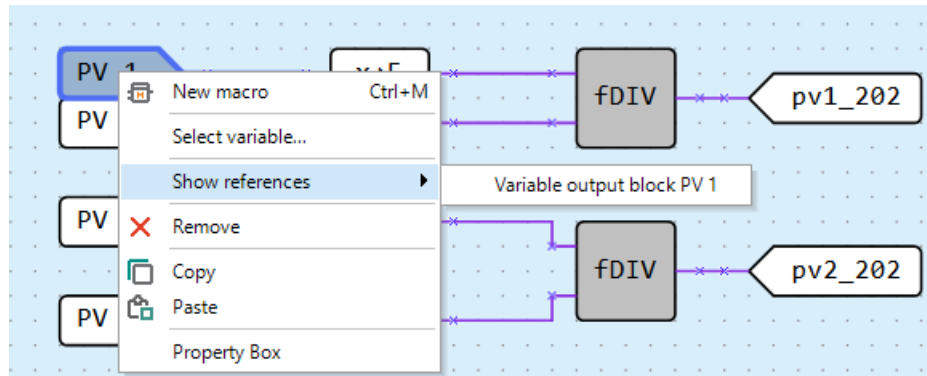
To use a variable as an output block, hold the Shift key pressed as you drag-and-drop the variable. If a variable is drag-and-dropped onto a connection pin of a block, the created variable block is connected to this connection pin.

References

The variable references are shown as links in the lower panel part. If you click on the link, the block to which the variable is referred is highlighted in the workspace.



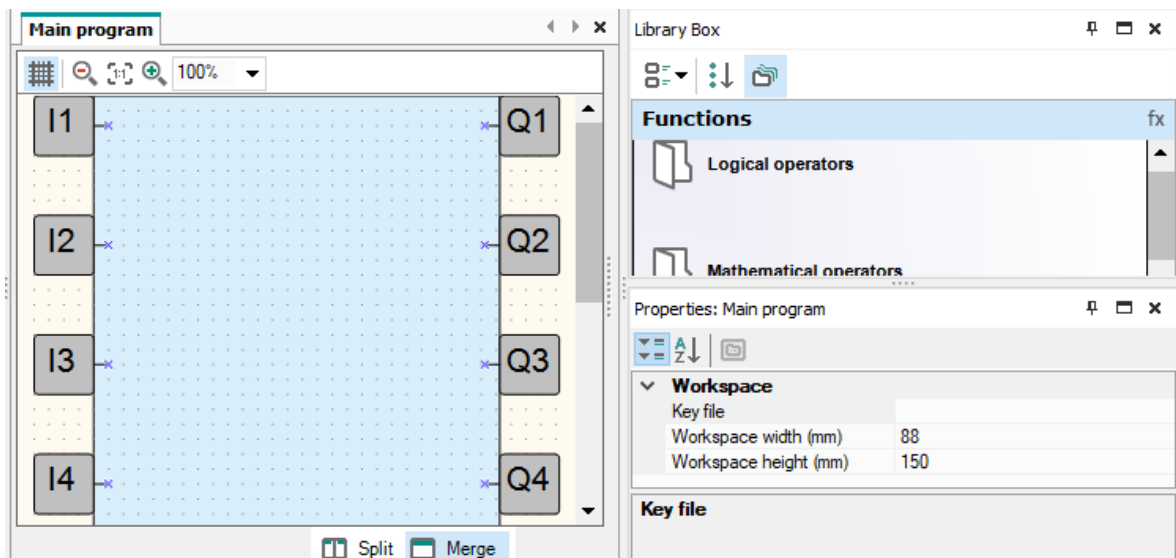
If a variable is used at more than one place in the project, all the references can be viewed with the item **Show references** in the variable block context menu. Click on the link to view the reference.



3.6 Workspace

When a project is open, the workspace with the tab **Main program** is shown in the middle of the window.

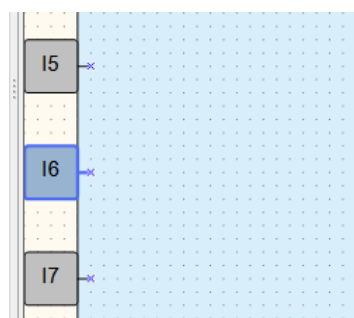
If your device supports ST functions, the **Function Editor** tab appears.







Circuit program is built by placing program blocks and connecting lines between them in the workspace. The size of the workspace can be changed in Property Box. The inputs (left) and outputs (right) are signed as follows:

- **Ix** – digital inputs
- **AIx** – analog inputs
- **Qx** – relay outputs
- **AOx** – analog outputs
- **Fx** – LED indicators





The numbers (x) correspond to the ordinal numbers of physical I/O points of the target device. I/O points can be moved up and down along the workspace by drag-and-drop.



Workspace toolbar

	Show / Hide grid	Show / hide vertical and horizontal rulers and a grid in the workspace. If the grid is visible, the blocks and connecting lines are snapped to the grid
	Zoom –	Decrease the workspace by 10 %
	Original size	Return to the original size (100 %)
	Zoom +	Increase the workspace by 10 %

You can set the required scale using the drop-down menu to the right of the buttons described above.

The icons  **Merge** and  **Split** are located on a toolbar below the workspace. Use the icon  **Split** to show the same circuit program in two workspaces. It can be useful if the program is too large and you want to view two different parts of the program at the same time. Use the icon  **Merge** to return to one workspace.

3.7 Status bar

Status and error messages are displayed in the left field of the status bar.

View

The right field of the Status bar has different status indicators that show information about the memory usage of the target device (resource indicators), status of the connected device and the programming interface. Contents of the status bar depend on the type of the target device.

FB: 0%	Var: 0%	EEPROM: 13%	ROM: 1%	RAM: 6%	✓ PR200-24.2(4)	✓ COM10	...
--------	---------	-------------	---------	---------	-----------------	---------	-----

Resource indicators show the used resource in percent of the total available amount. Move the mouse cursor over the indicator to see the absolute amount of the resource.

Indicators

If the device is connected, Status bar contains the following information:

- **FB** — number of the used and available function blocks.
- **Var** — number of the used and available variables.
- **Stack** — percentage of the available and used stack memory. The stack is used for intermediate calculations in the program.
- **Sys EEPROM** — percentage of used and available system non-volatile memory. The indicator is full if the project uses network variables, if variables are linked to visualization/device parameters, or if the project uses expansion modules.
- **EEPROM** — percentage of used and available non-volatile memory. The indicator is full if the project uses standard non-volatile variables.
- **ROM** — used and available ROM memory.
- **Sys RAM** — used and available system RAM memory. This indicator appears when the RAM is over 80 % full.

**NOTE**

ALP software automatically calculates the available resources of the device and shows a warning if the critical value is reached.

- **RAM** — used and available RAM memory.
- **System Visualization ROM** — the available visualization ROM of the device as a percentage of the total volume (only for devices with a graphic color LCD).

- **User Visualization ROM** — the available visualization ROM for bitmap images (only for devices with a graphic color LCD).
- **Visualization RAM** — the available visualization RAM of the device as a percentage of the total volume. The calculation is performed for the most heavily loaded screen (only for devices with a graphic color LCD).
- **Device** — device model, if connected

**NOTE**

Click the *Device* indicator to switch to **OFFLINE** mode. In this mode the connection to device is interrupted. The next click disables the **OFFLINE** mode.

- **COMx** — the selected COM port number (programming interface)

**NOTE**

Click on the indicator to open the window **Port settings**.

**NOTICE**

If any of the indicators overflow, writing a program to the device is disabled.

You can customize the number of indicators on the Status bar in the **View** menu.

OFFLINE mode

OFFLINE mode interrupts the connection between ALP and the device.

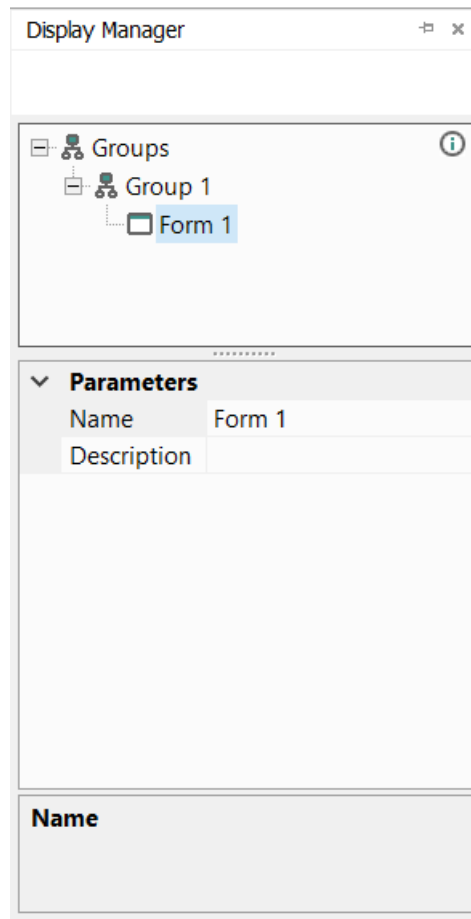
OFFLINE mode can be activated / deactivated using the menu item **Service > OFFLINE mode** or by clicking the status indicator **Device**. The next click disables the OFFLINE mode.

For more details see section [Upload project to device](#).

3.8 Display manager


If the target device has a display, you can program the displayed information using one or more display forms. For further details about display programming see section [Display programming](#)

The programming is carried out using the programming tool **Display Manager**. The tab Display Manager is located in the upper left corner of the window. Click the tab to open the panel. The panel contains a toolbar, a hierarchical structure (tree) of display forms and the parameters of the selected object.



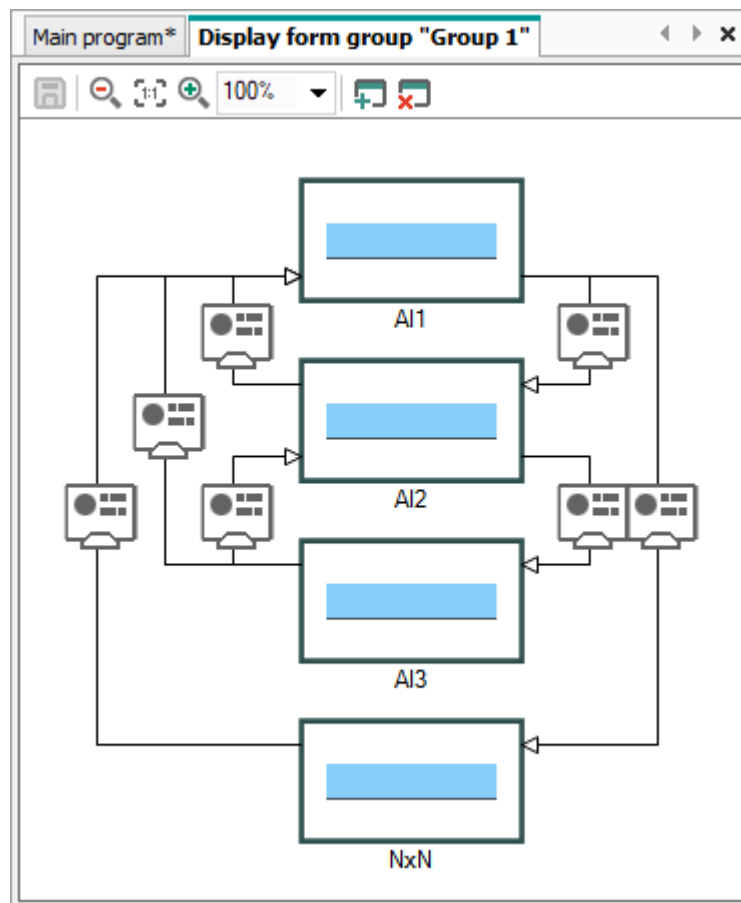
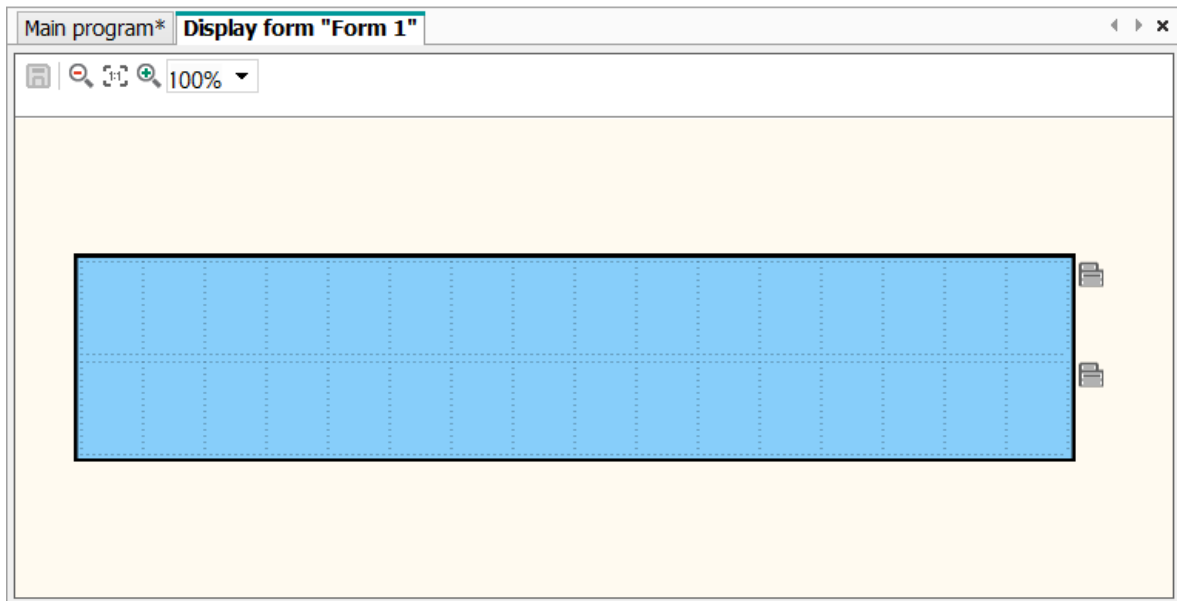
The parameters of the selected display form are shown in the lower part of the panel. To program the selected form, open it in a separate workspace **Display Editor**, using the context menu or double-click the form in the group.

	Add display form
	Delete display form
	Edit display form





The workspace shows the selected display form with the icons  to the right of it, which are used to change the number of displayed rows. The rows displayed first are bold outlined.

Working with screens is discussed in detail in the [section 4.4.1](#).

If there are more than one display forms, you should specify “jumps” between them so that the operator can switch between forms to see the required information. It can be done in a separate workspace **Structure Editor**, which represents the graphical structure of display forms with “jumps” and their conditions. To open it, use the command **Edit group** in the group context menu.



Workspace Structure Editor has the following functions (buttons at the top):

	Save workspace
	Zoom out by 10 %
	Original size
	Zoom in by 10 %

The scale can be changed with the drop-down menu to the right of the buttons.

4 General information

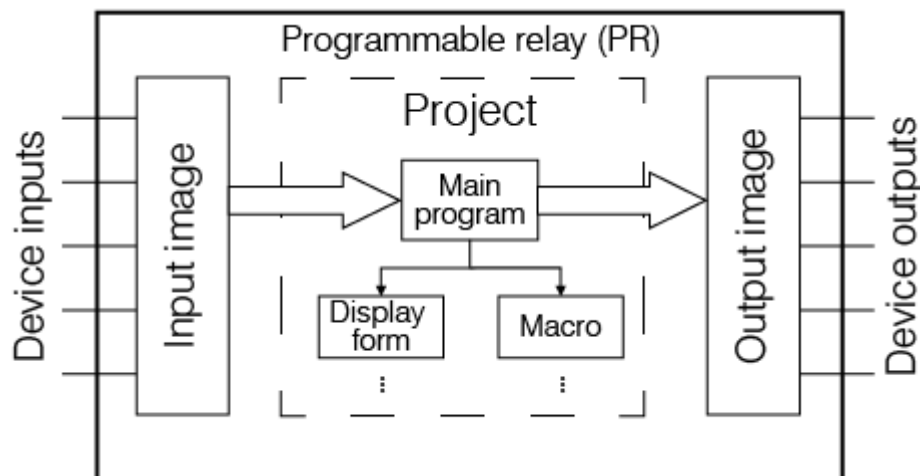
This section describes the basic principles of operating a programmable relay (further referred to as "device") and programming this device:

- Program execution
- Project creation
- Program creation
- Display programming
- Simulation
- Connection to device
- Upload project to device
- Online debugging
- Project information
- Component manager
- Macro development
- ST function
- ST function block

4.1 Program execution

The selected target device determines the number of available inputs and outputs and the availability of a real-time clock.

General structure of a programmable relay:



Programmable relay is a programmable logical controller (PLC) with a cyclically executed project:

Step 1 – Device saves the status of physical inputs to the input memory cells (Input Image Table).

Step 2 – Device reads out the input memory cells and the main program is executed from its first command to the last one.

Step 3 – Device saves the results to the output memory cells (Output Image Table) and applies them to the outputs. When the last step is completed, the project runs again from Step 1.

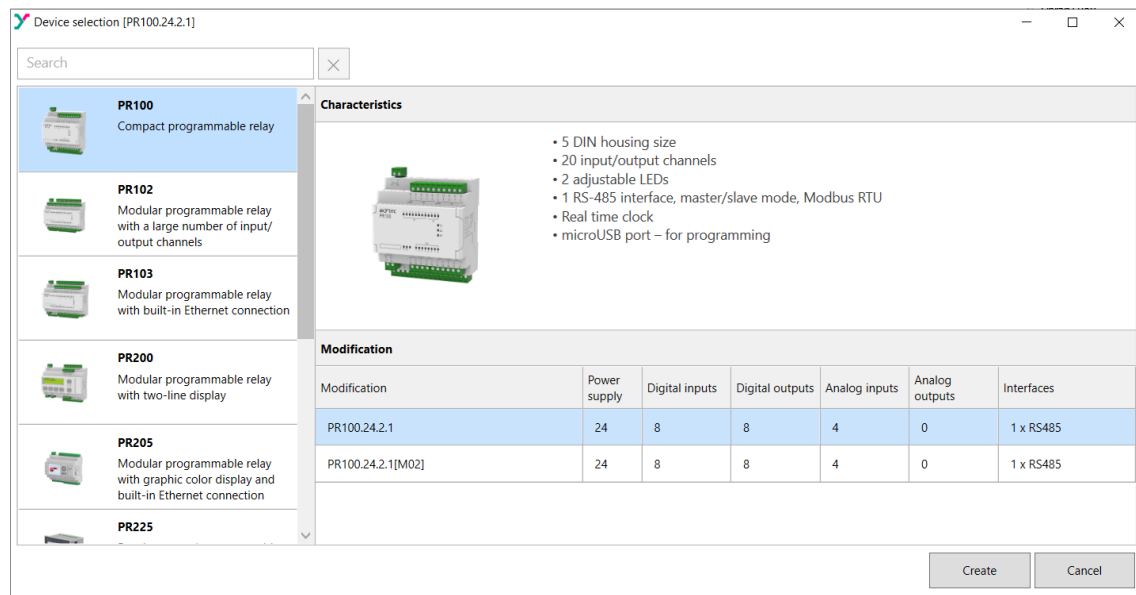
When the last step is completed, the program runs again from the first step.

4.2 Project creation

Project creation

To create a new project:

1. Click icon  in the taskbar or select **File** → **New project...** in the main menu.
2. Select the target device in the dialog window **Device selection** and confirm it with **OK**.



The new project appearance:

- **Workspace** – empty circuit program
- **Status bar** – information about available resources
- **Library Box** – available program blocks
- **Property Box** – workspace properties



NOTE

If the device is already connected to PC, ALP will offer the device model in the selection window.

If the selected device has a display, the **Device Manager** tab appears to the left from the workspace. With this tool you can **program** the displayed information.


You can save the current project or open a saved project using the corresponding buttons on the toolbar or in the main **File** menu.

Circuit program development

Now you can create the **main circuit program** in the workspace using the common program blocks from the toolbar **Insert** and the specific program blocks from **Library Box**. Draw connecting lines between inputs, outputs and blocks to establish logical connections in the program.

Simulation

Program can be simulated offline. Start the **simulation mode** using the menu item **Service**

→ **Simulation** or the toolbar icon , change the state of the inputs and notice the state of the outputs to check how the program works.

Online debugging

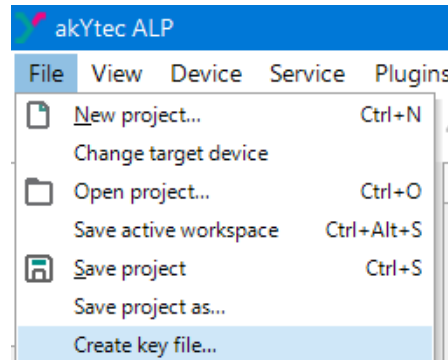
If the device is connected and the program in the device and in ALP is the same, you can use **online debugging** to check the correctness of the program in the device.


4.2.1 Protecting a project with a key

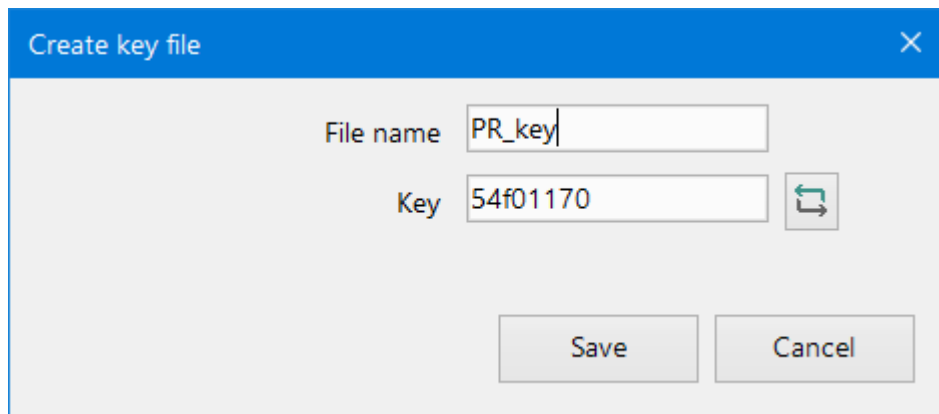
Creating a file with a key

To create a file with a key:

1. Select from the main menu **File** → **Create key file...**



2. In the dialog box that opens, enter file name and generate the key. You can edit the key by clicking the button  or manually.

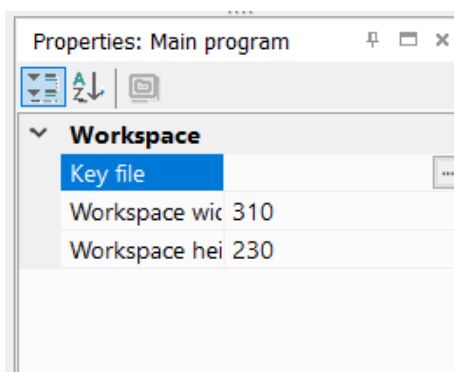


3. Save file. By default ALP saves the key file to folder `C:\Users\User name\documents\ALP\Keys`.

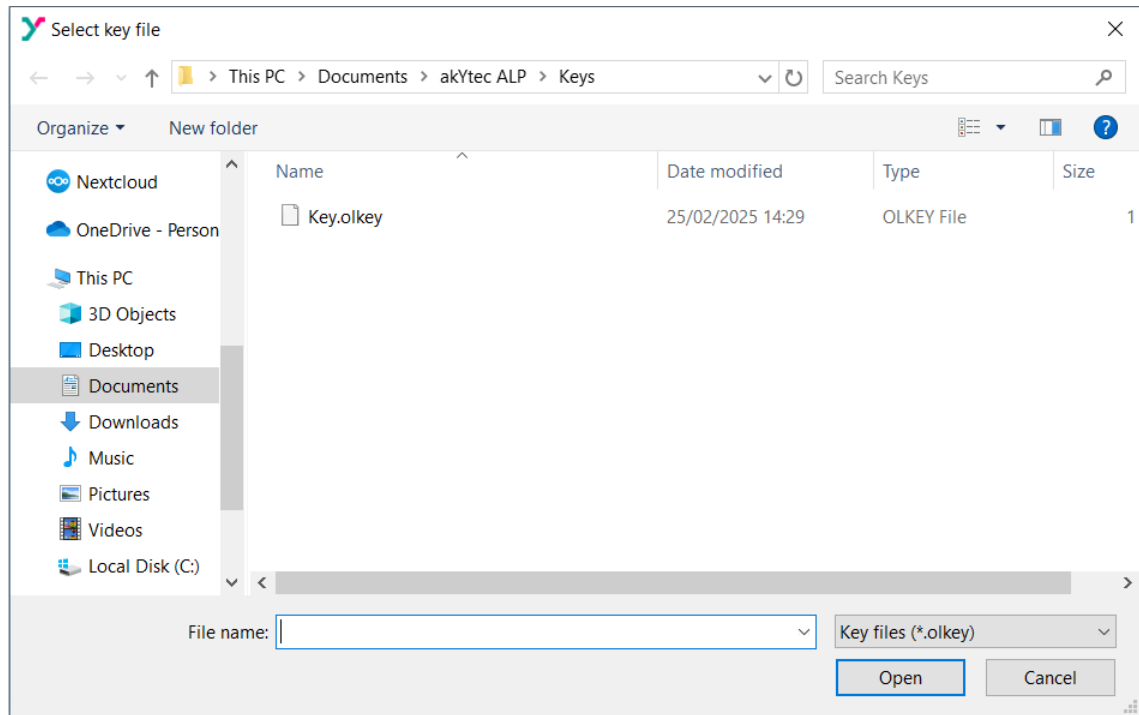
Binding a key file

To bind a key file to a project:

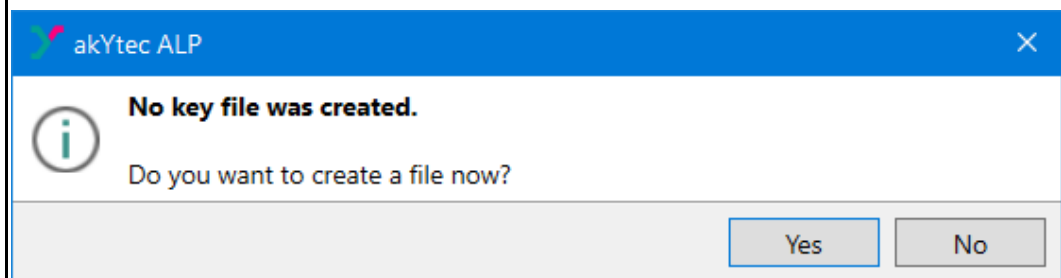
1. Click the icon «...» in the file properties (**Properties: Main program** panel).



2. Select the key file in the new window.

**NOTE**

If there is no key in the folder `C:\Users\User name\documents\ALP\Keys`, ALP will offer to create a new key.



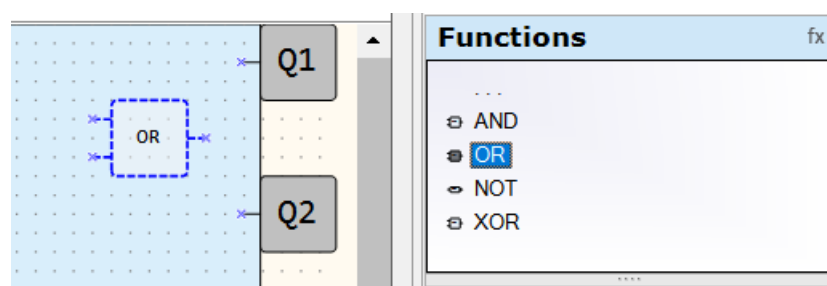
3. Press the **Open** button.

4.3 Program creation

Before creating a program it is recommended to plan it. The plan should describe all possible states of the device during operation in form of a mode diagram, a table of I/O states, an electrical or functional diagram, etc.

When all the operation tasks are defined, you can create the program using the standard block from the toolbar Insert and the specific blocks from the project library. The project library presented in Library Box contains the functions and the function blocks available for the target device, as well as the macros added to the project.

To add a library block into the circuit program, select the block in Library Box and move it to the workspace by drag-and-drop.



To draw connecting lines between inputs and outputs of the device and program blocks, use the left mouse button:

- Click the input pin of the device. The line is attached to it and follows the mouse cursor.
- To change the line direction, click a point in the workspace.
- Pull the line up to the input pin of a block and click on it to finish the line.

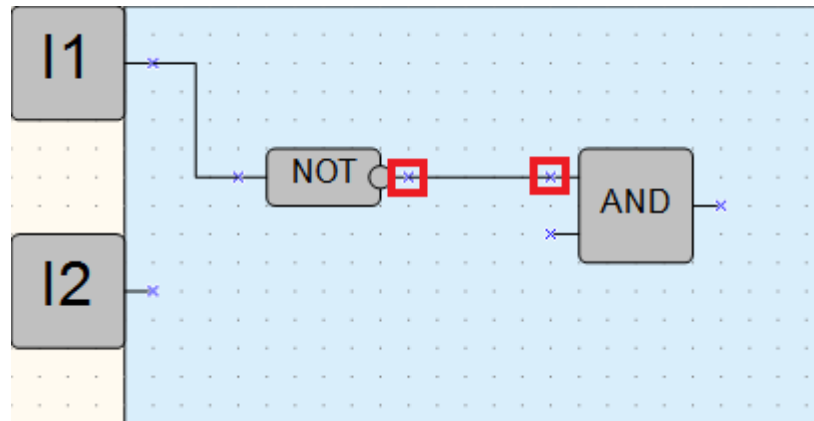
The connecting line can be drawn only between connection pins assigned to the same data type.



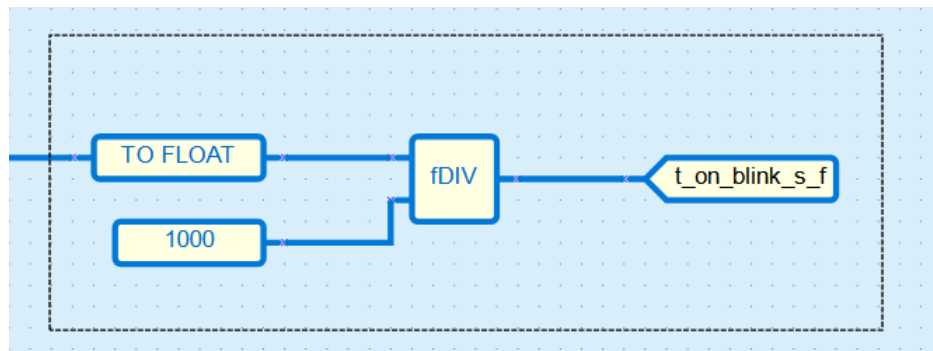
NOTE

If the input and output types are not identical, the line will not be created.

Click the block to select it. To select a group, pull the rectangle around several blocks.

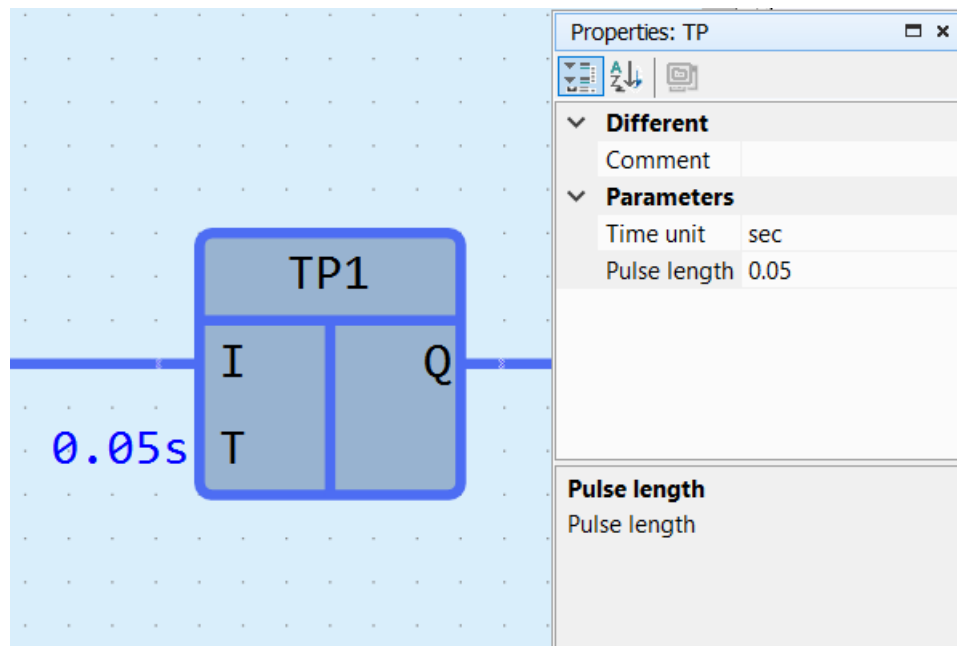


To connect the connection pins assigned to different data types, use conversion blocks.

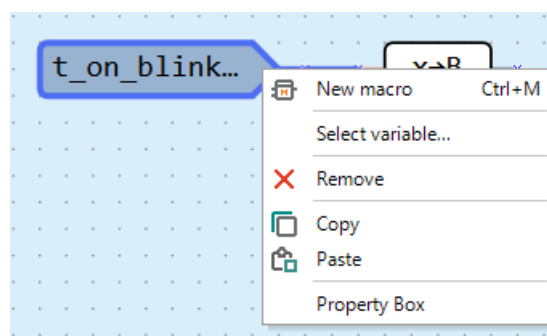


Component settings

You can edit properties of each element in Property Box.



Use block context menu for all manipulations available for the element.




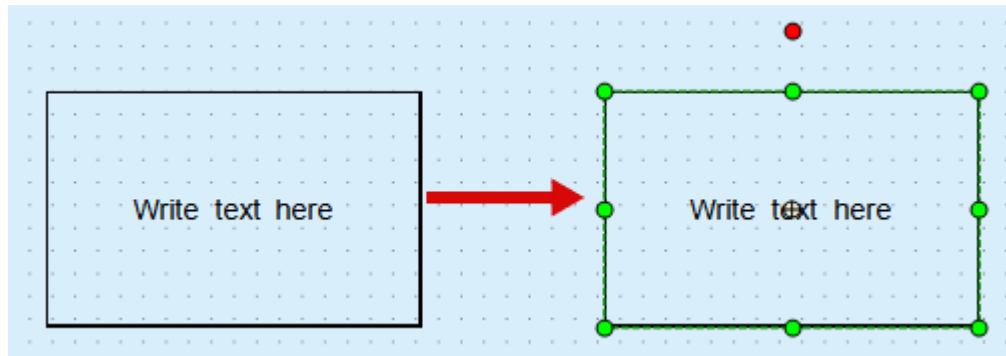
Toolbar **Insert** has following blocks and functions for program development:

<u>Text field</u>	Placing a text comment on the circuit program
<u>Variable block</u>	Placing a variable block for writing or reading program values
<u>Constant block</u>	Placing a block with a fixed numeric value
<u>Delay lines</u>	Creating a delay for one cycle of transferring a value from the component's output to its input
<u>Network variable block</u>	Placing blocks for data exchange between devices connected to a common network
<u>Read / write to FB</u>	Writing/reading the values of individual parameters from the FB to a variable and vice versa
<u>Conversion blocks</u>	Converting values of different types for transmission
<u>Arrange elements</u>	Reassignment of sequence numbers of FB schemes
<u>Execution sequence</u>	Changing the order in which program output values are calculated

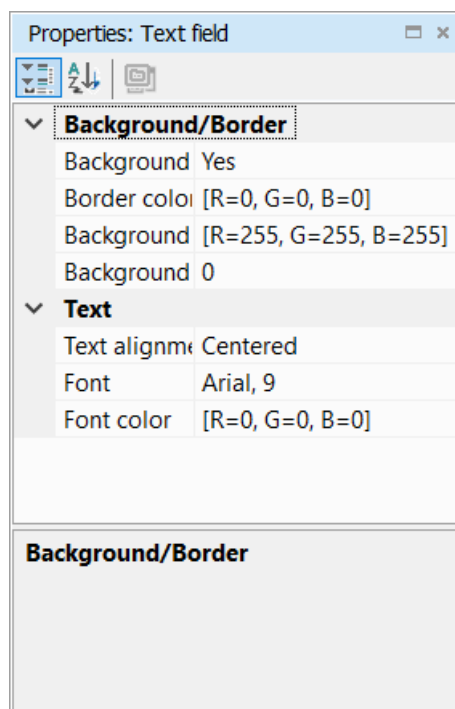
4.3.1 Text field

The text fields are used to explain the program.

To add a text field to the program, click the icon  in the toolbar **Insert**, then click a point in the workspace to place the upper/left corner of the text field and draw a rectangle to set its size.

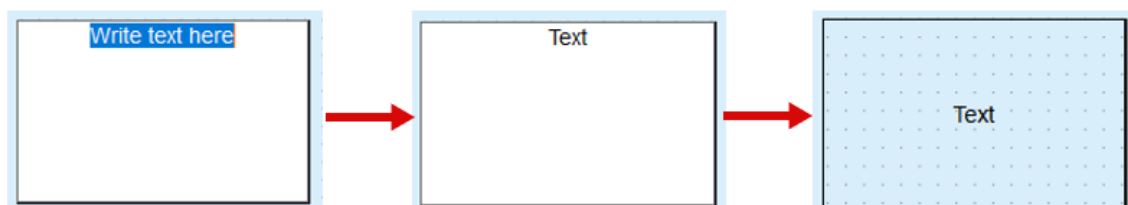


The parameters of the text field can be changed in **Property Box**.



To make the background color of the text block visible, it is recommended to set the **Background transparency** greater than 20 %.


Double-click the text field to write the text.




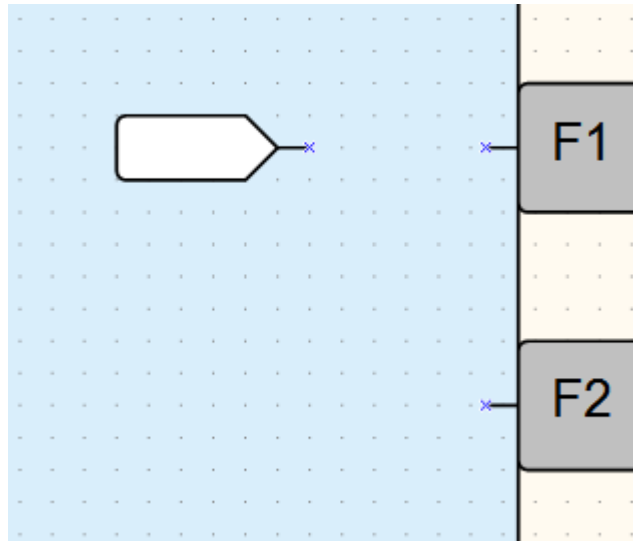
4.3.2 Standard variable block

The variable block enables the use of a variable in a circuit program.

To add a variable block to the program, click on its icon in the insert panel:

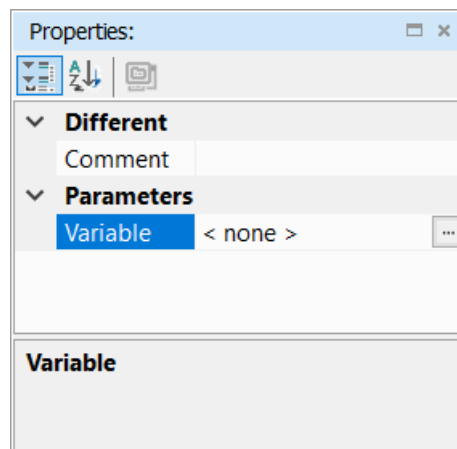
-  **input variable block** — this block writes a variable into the program;

-  **output variable block** — block reads a variable from the program.
Click the point in the workspace to place the variable block.

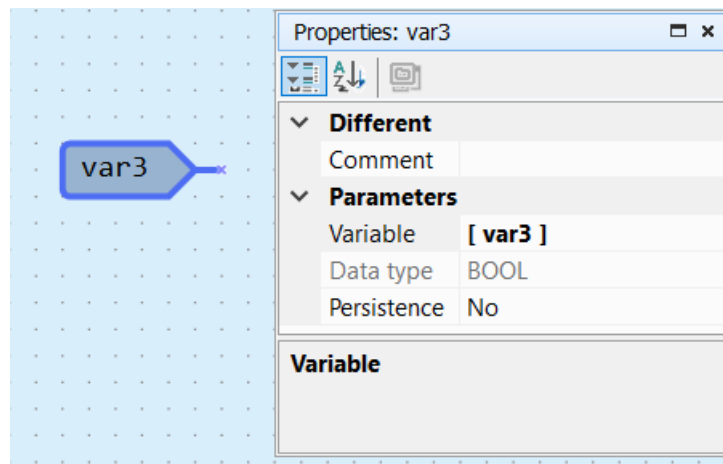


You can also add a variable block to the program from Variable table.
To assign a variable to a variable block:

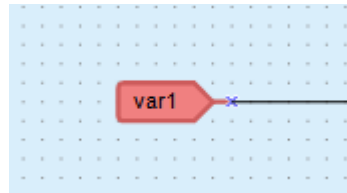
1. Select a variable block.
2. Double click the block or click the icon «...» in the row Variable in Property Box. Variable table opens. Select a variable or create a new one. Only the editable tabs in the table are available for selection. The availability is limited by the block type.



3. Confirm the selection with **OK**. The variable is assigned to the block.




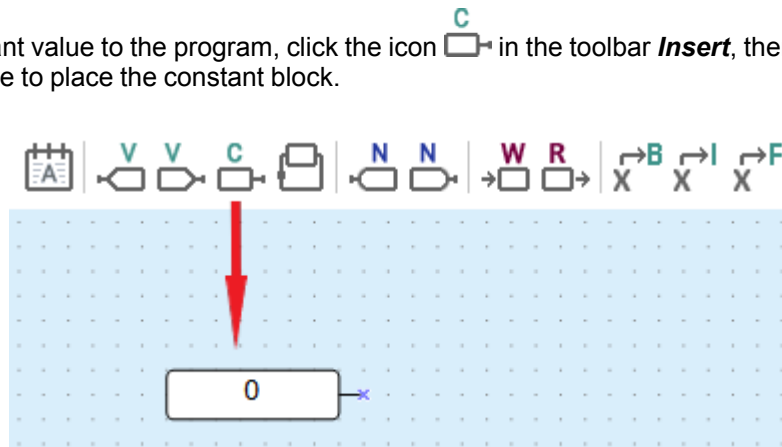
Connect the variable block to the necessary element in the workspace.
If the variable block is highlighted in red, it means that the creation is incorrect or not completed.
Information about the error is shown in the status bar.



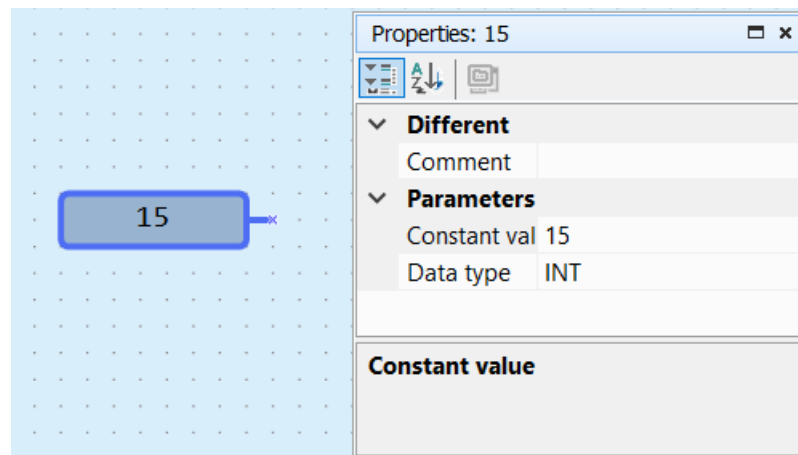
Creating variables in the variable table is the recommended first step in programming.
If the variable is used more than once in a project, all references can be followed with the item **Show references** in the variable block context menu. The function is also available in simulation and online debugging modes.

4.3.3 Constant block

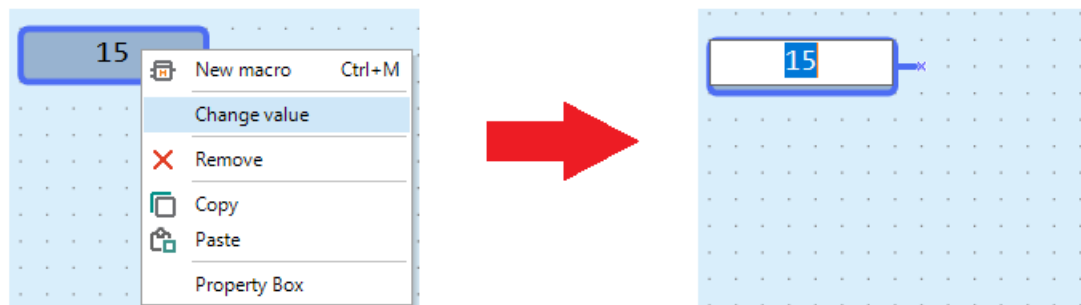
To add a constant value to the program, click the icon  in the toolbar **Insert**, then click the point in the workspace to place the constant block.



Select the data type using the icon «...» in the row **Data type** and enter the value on the row **Constant value** in **Property Box**.



The value of the constant block will not change throughout the program execution. You can edit it by double-clicking on the constant block, in **Property Box** or by selecting **Change value** in the block context menu.




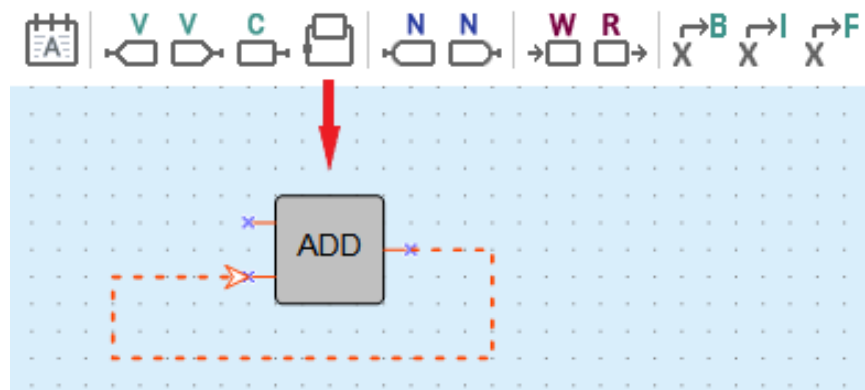
Data types valid values:

BOOL	0 / 1
INT	0 ... 4,294,967,295
REAL	-3.402823e+38 ... 3.402823e+38

4.3.4 Delay line

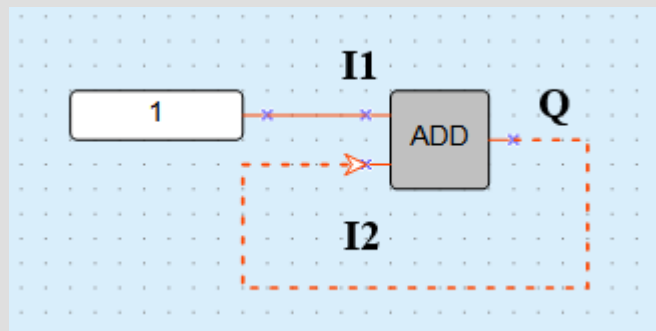
The delay line is used to transfer the value from the block output to the block input, delayed for one cycle. The output and input may belong to different blocks.

Click the icon  in the toolbar **Insert** and draw a line from the output to the input of a function block. The delay line is displayed as a red dashed line with an arrow.



Example:

A constant value 1 is transferred to the input I1 of the addition block ADD (Integer). A value from the block output (Q) calculated in the previous cycle is transferred to the input I1 over delay line.





Cycle signal values:

Cycle	1	2	3	4	5	6	7	8	9	10
I2	0	0	1	1	2	2	3	3	4	4
Q	1	1	2	2	3	3	4	4	5	5

4.3.5 Network variable

The network input and output variable blocks are special type of variable blocks for data exchange between devices connected to a common network.



-  – network output variables are the variables that can be read via the network.
-  – network input variables are the variables that can be written via the network.

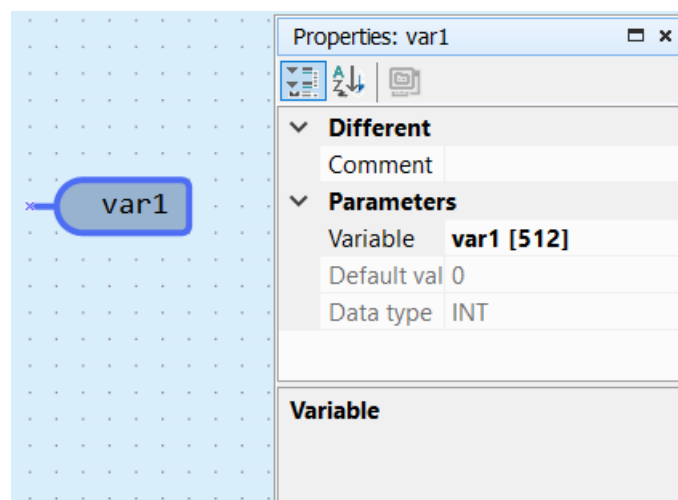


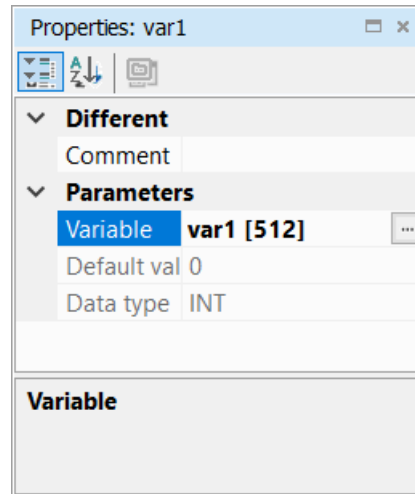
NOTE

A variable cannot be assigned to the block if there are no communication interfaces in the device configuration.

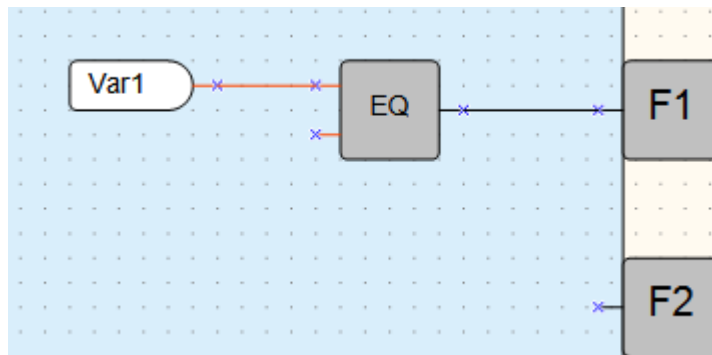
To add a network variable to the program:

1. Click the icon  or  in the toolbar **Insert**.
2. Click the point in the workspace to place the variable block.
3. Double click the block or click the icon «...» in the row Variable in Property Box. Variable table opens. Select a variable or create a new one. Only the editable tabs in the table are available for selection. The availability is limited by the block type.
4. Confirm the selection with **OK**. The variable is assigned to the block.





Connect the network variable block to the necessary element in the workspace.



If the variable block is highlighted in red, it means that the creation is incorrect or not completed.

The information about the error is shown in the status bar.



Creating variables in the variable table is the recommended first step in programming.


If the variable is used more than once in a project, all references can be followed with the item **Show references** in the variable block context menu.

4.3.6 Read from/Write to FB

The ReadFromFB/WriteToFB blocks are used to read or write a parameter value of a function block during the program execution.

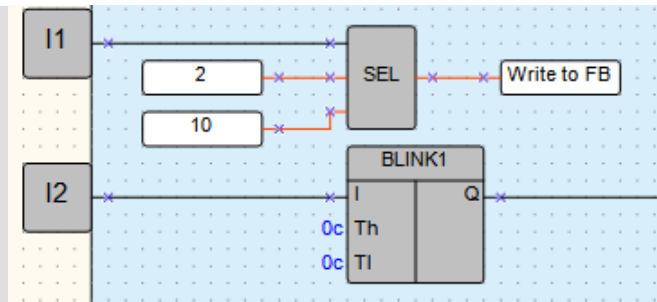
The following blocks can be added to the circuit:

-  — to write the value to a FB;
-  — to read the value from a FB.

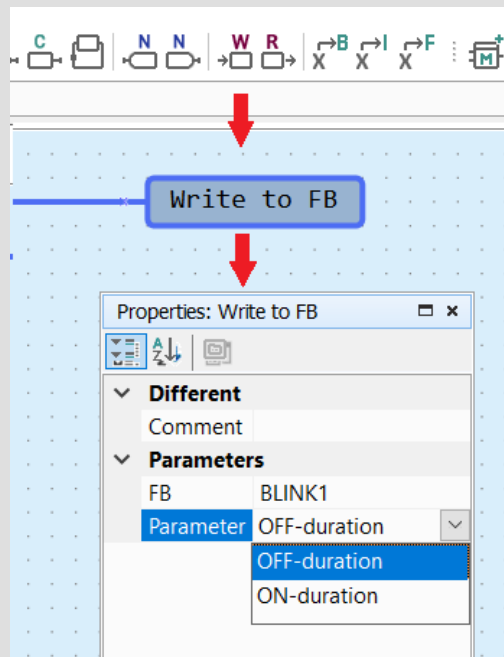
The block **WriteToFB** () is used to change an FB parameter during the process.

Example:

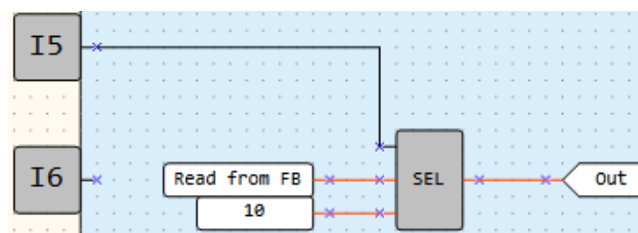
The value of the parameter **ON-duration** of the FB **BLINK1** should be 2 or 10 depending on the value at the input **I1**.



Go to **Property Box**, select the FB **BLINK1** in the row **Function block** and the parameter of the FB in the row **Parameter in FB**.

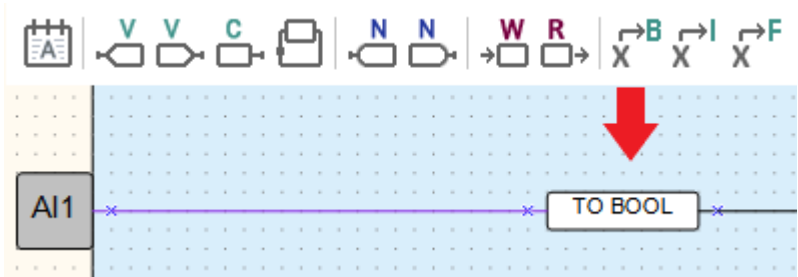


The block **ReadFromFB** (□→) reads the current value of an FB parameter and uses it in the program. Editing is similar to the **WriteToFB** block.



4.3.7 Conversion block

Connecting line can only connect program components of the same type: BOOL→BOOL, INT→INT, or REAL→REAL. You can connect different types of inputs and outputs only with a conversion block. To add the conversion block to the program, click the corresponding icon in the toolbar **Insert**, then click the required place in the workspace.




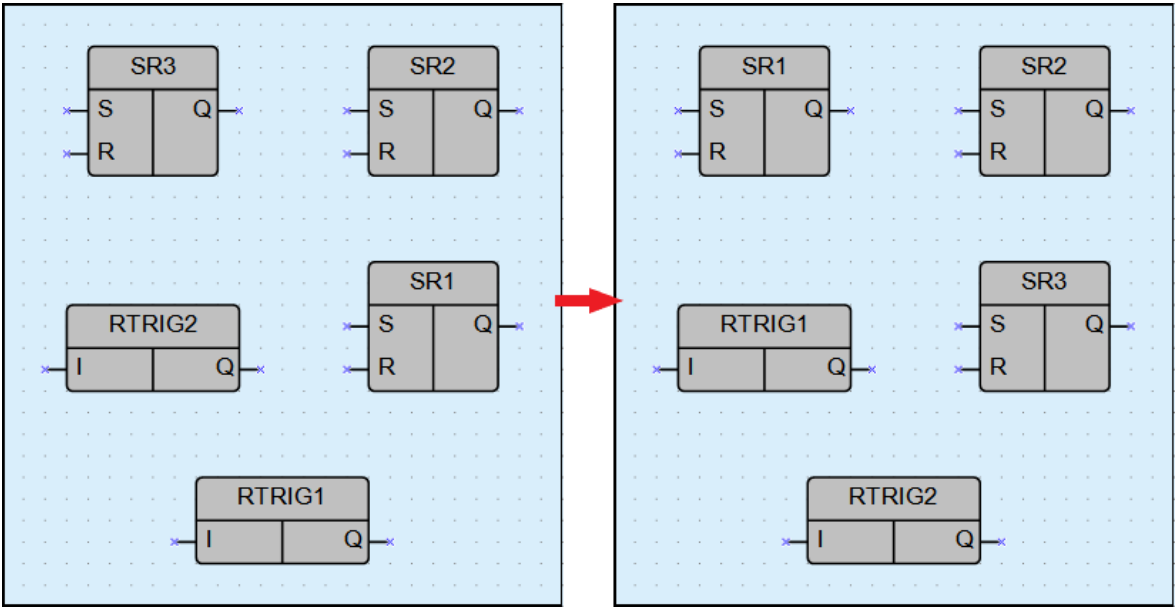
Conversion blocks:

x^B	Conversion to BOOL	Converts INT or REAL to BOOL If the input value > 0, the output = 1 (True)
x^I	Conversion to INT	Converts BOOL or REAL to INT REAL is rounded down to INT, negative value is converted to 0
x^F	Conversion to REAL	Converts BOOL or INT to REAL

4.3.8 Arrange elements


The sequence numbers of the function blocks can be automatically reassigned by clicking the button

 **Arrange elements** in the toolbar **Service**. The blocks of the same type are numbered sequentially from top to bottom and from left to right.



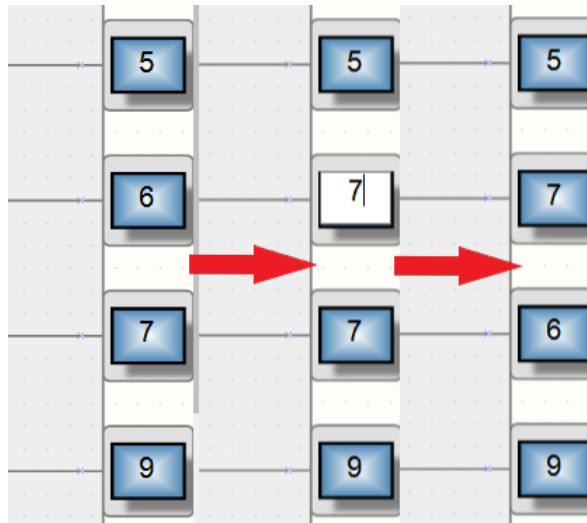
4.3.9 Execution sequence


Calculation of the values for outputs and delay lines is performed in a certain order. To see this order,

click the arrow near the icon  in the toolbar **Service** and select **Delay lines** or **Outputs**.

ALP will switch to the execution order setting mode – the sequence numbers of the execution order will be displayed near the outputs and feedbacks.

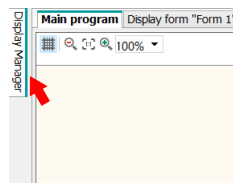
To change the order, double-click an output or a delay line and enter the required number.



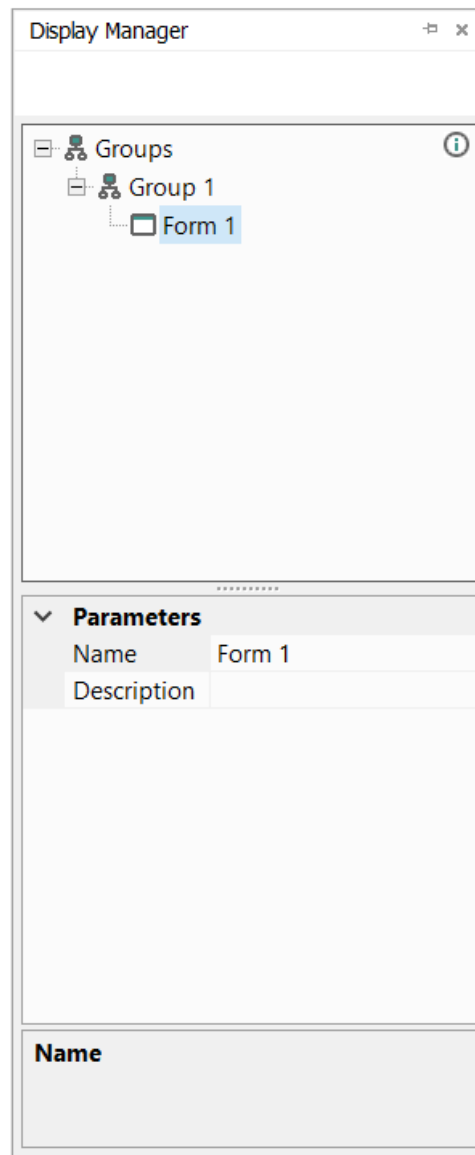
Click the icon  once more to deactivate the edit mode.

4.4 Display programming

To define the displayed information, use the tab Display manager in the upper left corner of the window. Display manager is only for target devices with display available.



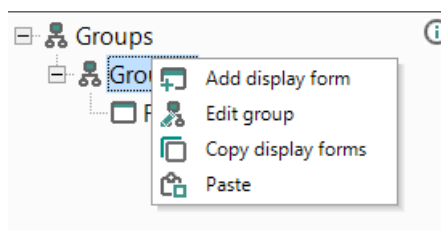
By default, the Display manager shows one form.



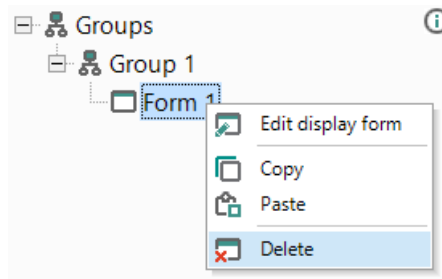
Adding display forms

The display can be programmed using one or more display forms with “jumps” between them. Displayed information can be changed by program events (change of variable) or by the operator (button event).

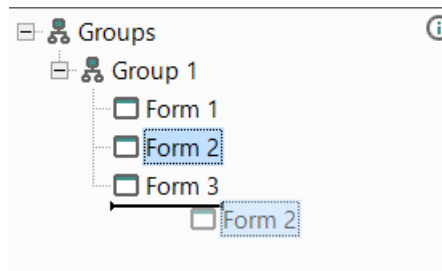
To add a display form, right-click on the **Group 1** element and select **Add display form** in the context menu.



To delete a display form, right-click on the form and select **Delete** in the context menu.



To change the position of the display form, drag it to a new location while holding the **Shift** key.



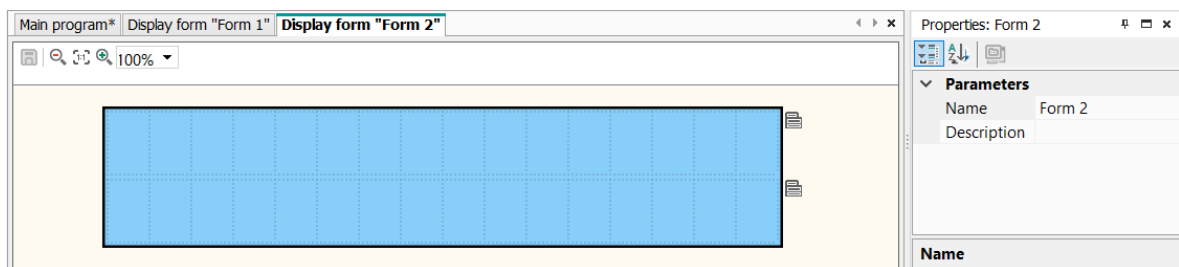
When you drag, the new position will appear as a horizontal marker.

Display form properties

To open the selected form in **Display Editor**, use the command **Edit display form** in the form context menu or double-click the form in the tree.

Display form properties:

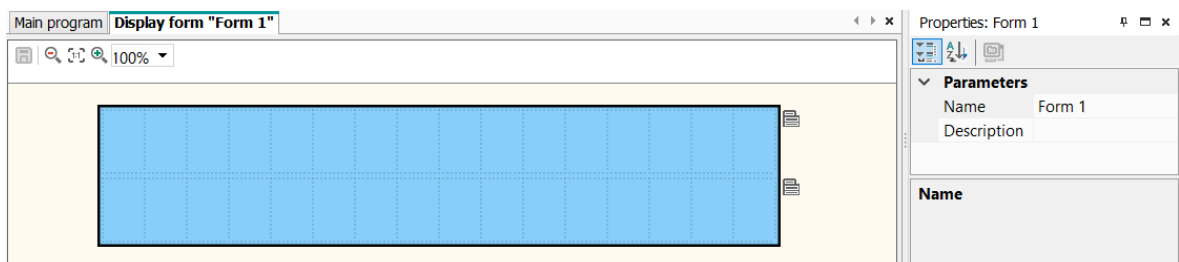
- **Name** – is shown in the display form manager and in the display editor header
- **Description** – text description of the display form




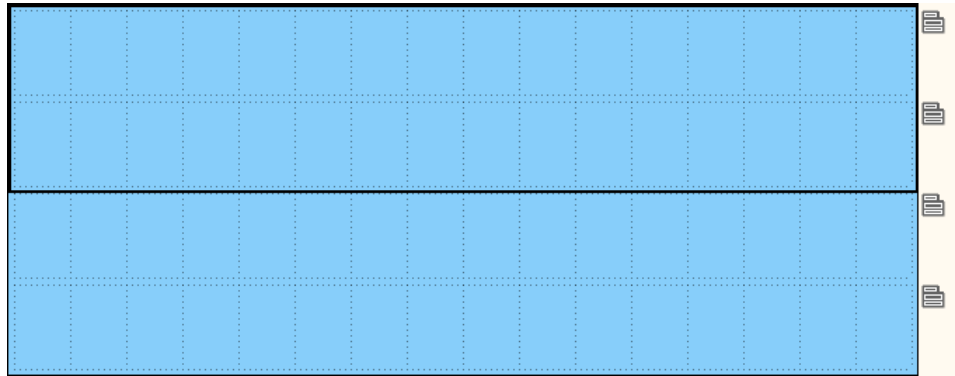
4.4.1 Monochrome text LCD

Display editor

To open the selected form in **Display Editor**, use the command **Edit display form** in the form context menu or double-click the form in the tree.



An icon  on the right edge of each row represents the row context menu, which is used to change the number and the order of the displayed rows.



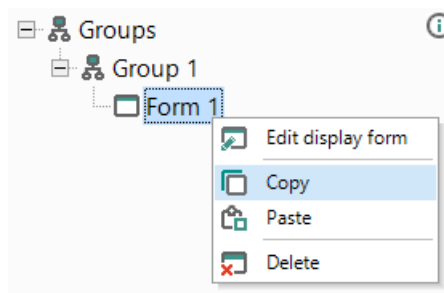
Put the display elements 7.6 from **Library Box** by drag-and-drop onto the form.

**NOTE**

The character set is implemented within the Windows-1251 encoding.

Copy-paste display form

In the Display Manager, you can copy forms for pasting into the current or another project. To copy selected forms, select the **Copy** command in the context menu of a form or group of forms, or press the **Ctrl + C** key combination. Multiple screens can be selected by holding down the **Ctrl** or **Shift** key.



To paste copied forms, select the **Paste** command in the context menu of a form or group of forms, or press the key combination **Ctrl + V**.

All controls and screen properties placed on the form are copied along with the form. The variables associated with the form are also copied, according to the rules described in the section Copy-paste variables.

Jumps between the selected forms are copied as well. If only one of the forms connected by the jump is selected, the jump will be deleted during the insertion.

Jumps

If the display structure consists of more than one form, “jumps” should be defined to enable the navigation between forms.

To create a jump:

1. Right-click on the **Group 1** element in the display manager tree and select **Edit group** in the context menu. The screen group editor tab will open.
2. Select the start form in the form group editor.
3. Click the «...» icon in the row **to display form** in **Property box**. Jump dialog window will open.

In the drop-down menu, select another form to go to.

The 'Jump' dialog box contains two main sections. The first section, 'to display form', has two radio buttons: 'by name' (selected) and 'Variable'. The 'by name' option has a dropdown menu showing 'AI2'. The second section, 'Jump condition', has two radio buttons: 'Device event' (selected) and 'Change by value'. The 'Device event' option has a dropdown menu showing 'DOWN-key down'. An 'OK' button is located at the bottom right of the dialog.

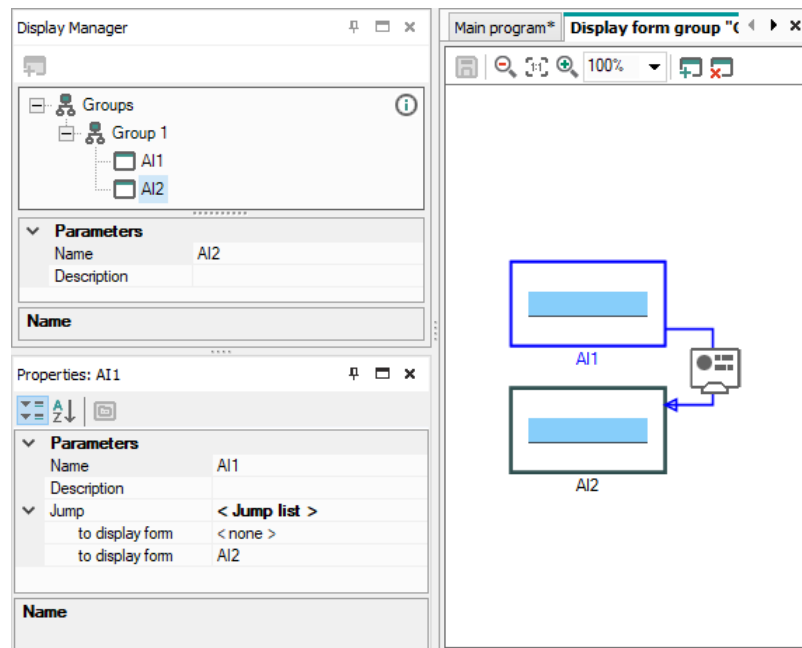
4. Select the event in the section **Jump condition**, as device event or change of a variable by value.

A button event can be selected as **Device event**.

A BOOL variable can be selected for **Change by value** event.

The first screenshot shows the 'Jump condition' section with 'Device event' selected. The dropdown menu is open, displaying a list of events: 'DOWN-key down', 'DOWN-key down', 'DOWN-key up', 'DOWN-key hold', 'OK-key down', 'OK-key up', 'OK-key hold', 'ESC-key down', and 'ESC-key up'. The second screenshot shows the 'Jump condition' section with 'Change by value' selected. The dropdown menu shows 'Unconditional jump', and the variable field contains '[sd1]'.

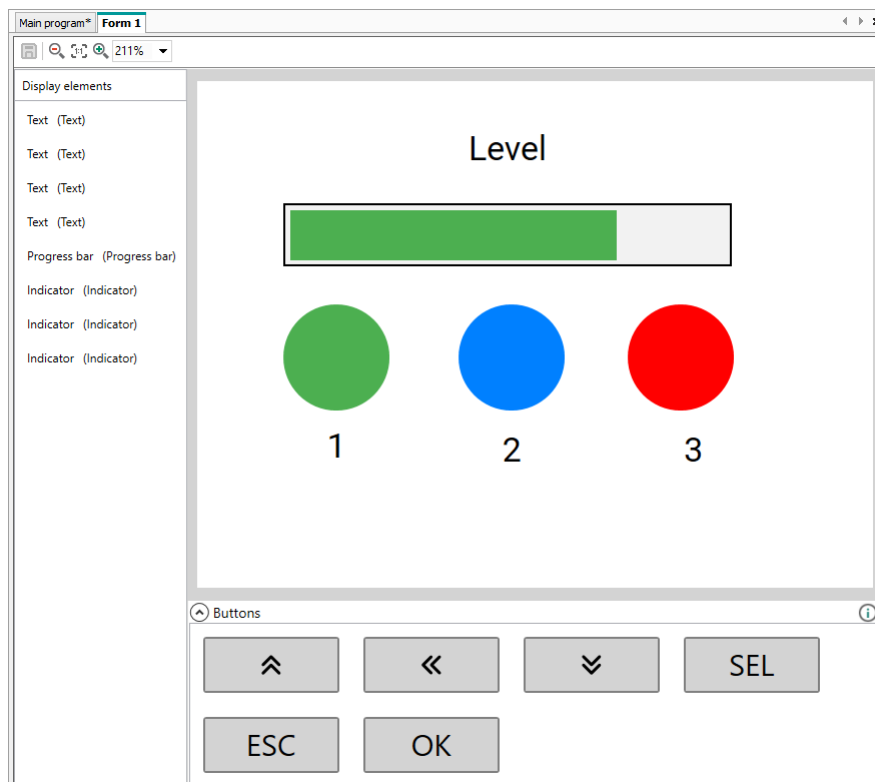
5. Confirm with **OK**. The created jump is shown in the structure.



The jump between two forms can occur by several events and the graphical structure can reach a very high complexity.

4.4.2 Graphic color LCD

For devices with graphic color LCD, the display output is programmed using graphic elements.



Screen Editor Areas:

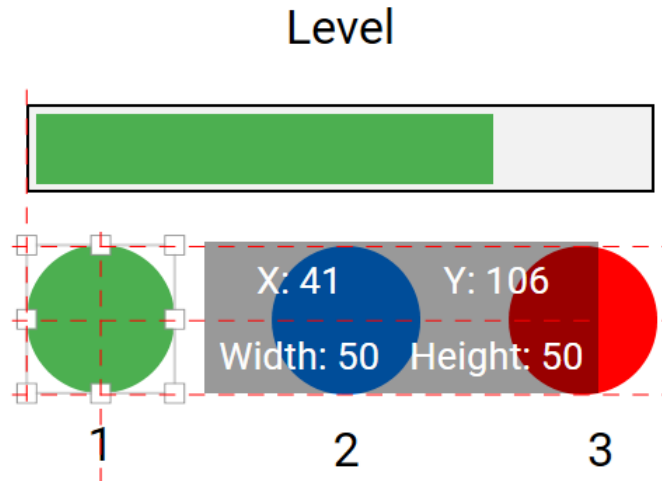
- **Screen elements panel** is used to display the list of components located in the screen field. In the screen elements panel, you can also change the order of graphic elements and assign names to them;
- **Screen field** is used for arrangement of graphic elements;

- **Buttons** are used to program the device buttons.

Graphic elements to be placed into the screen field can be selected from the **Component Library**. Element descriptions are given in *Basic Elements*.

A screen simulation window is available in *simulator mode*.

Screen field



When graphic elements are moved into the field, tooltips appear with coordinates and dimensions of the element and guide lines relative to other elements.

The following context menu actions are available for graphic elements and graphic element groups:

- Copy;
- Insert;
- Delete;
- Move to the foreground and background;
- Align.

Double-click on the graphical elements **Text** and **Dynamic Text** to activate the text input mode or on other elements to open the variable binding window.

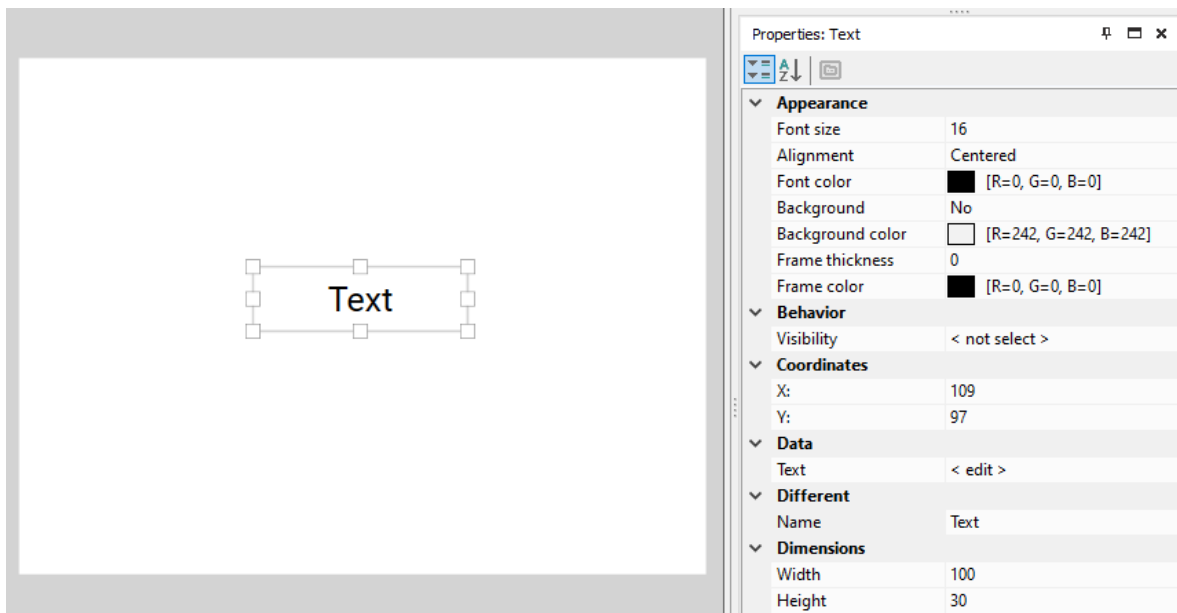
Graphical elements can be moved using the arrows on the keyboard, and other actions can be performed using *keyboard shortcuts*.

Properties of graphic elements

To display the properties of a graphic element in the Properties panel, click on the desired element in the list of elements to the left of the field or in the screen field.

Properties common to all graphic elements:

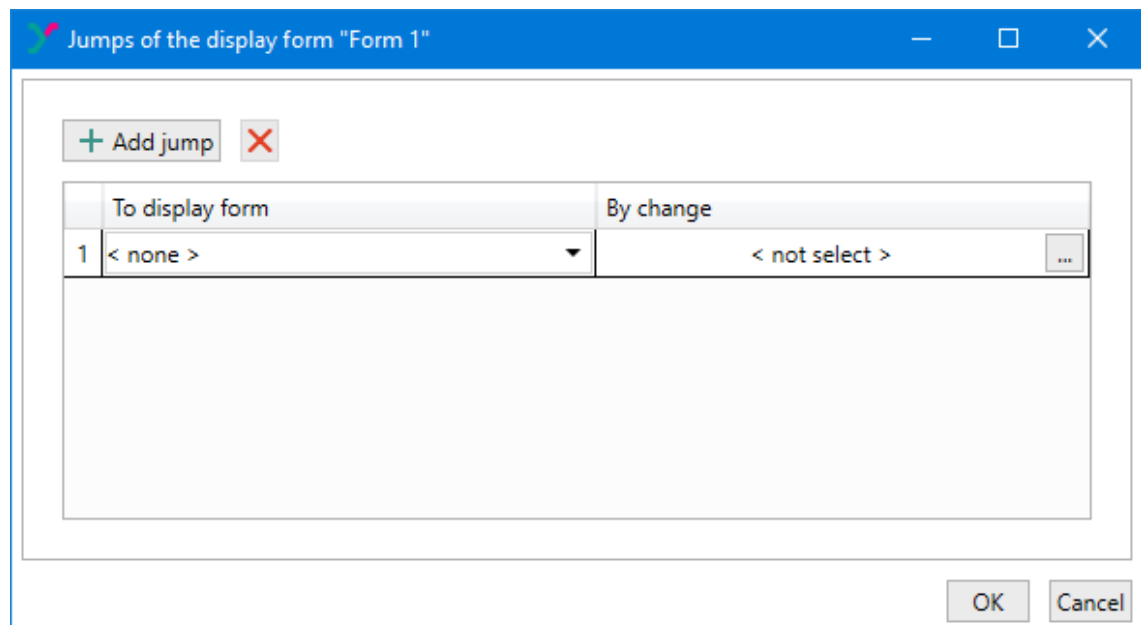
- **Element name**;
- **Size** - width and height in pixels;
- **Coordinates** - on X and Y axes, the reference point is the upper left edge of the element.



Creating transitions between screens

To create a screen transition for a graphical display:

1. Right-click on the screen in the screen manager tree and select **Customize transitions** in the context menu or click on the ... button in the **Transitions** field of the screen properties. The Transition Editor window opens. The selected screen will be the starting point for the transition.



2. From the **Transition to screen** drop-down menu, select another screen to transition to.
3. In the **Change Variable** drop-down menu, specify a variable of Boolean type, which, when its value is changed, triggers the transition. After the transition is executed, the value of the bound variable is changed to "0".



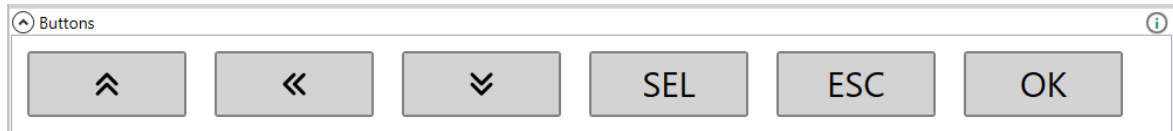
NOTE

To create a button-triggered transition, see below.

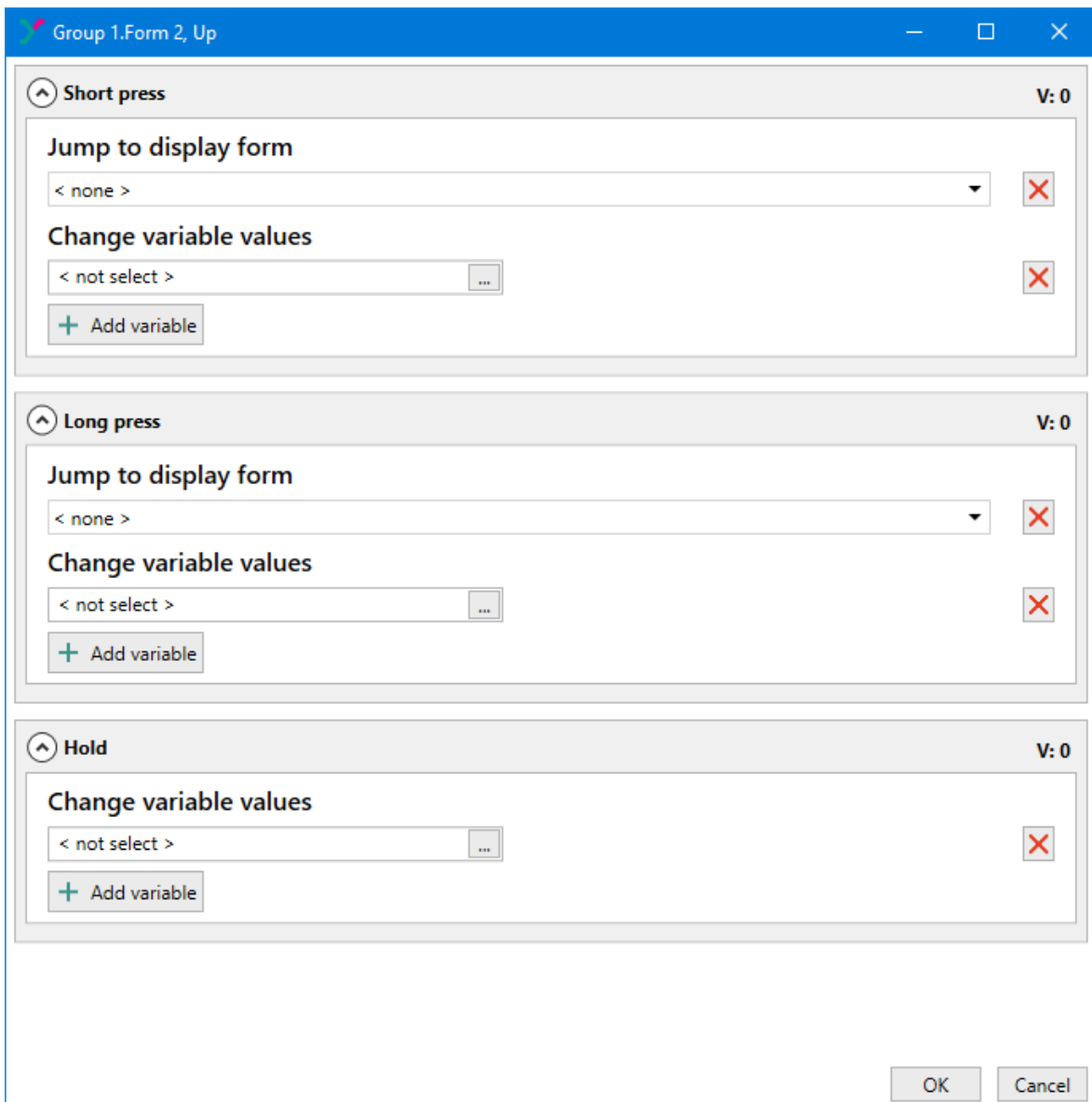
4. To add another transition, click **+ Add Transition**, to delete a selected transition, click **X**.
5. Click **OK**. The created transitions will be displayed in the screen properties. The order in which the transitions are executed corresponds to their order in the screen properties.

▼ Different	
Name	Form 1
▼ Jump	< set >
Group 1.Form	Variable
Description	Form

Buttons



In the bottom part of the visualization editor there is a menu for programming the device buttons. Double-click a button to open the event programming window for the button. The buttons are programmed separately for each screen.





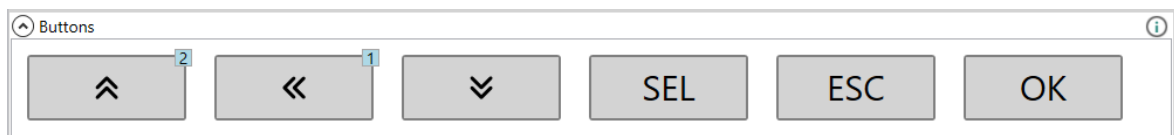
The following events are available for programming the button:

- short press (less than 3 seconds);

- long press (3 seconds or more);
- hold (5 seconds or more).

You can program several handlers for short and long presses (transitions to other screens and changes in the value of Boolean variables). Only a handler for changing the values of Boolean variables can be programmed to hold. Handlers are executed in the program in the order in which they are added.

To add a variable, click  **Add Variable**. To delete a programmed action, press . Programming overrides the default button behaviors. To return to the default behavior of the buttons, delete the event handlers and write the program to the device. If no actions have been set by the user for the **Up** and **Down** buttons, then when they are pressed the device will move to other screens according to the order in the screen manager window. Once programmed, the button menu will indicate the number of events for each button.

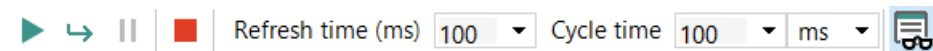







4.5 Simulation

Use the simulation to prove the correctness of the created program. Only the offline simulation is currently possible. The simulation enables to analyze the values of all signals within the circuit program. Change the values at the digital and analog inputs as well as of variables and constants and check the values at the outputs.

To start / stop the simulation mode, click the icon  in the toolbar **Service** → **Simulation**. A new toolbar **Simulation** is displayed.

Simulation toolbar



	Start	Start the permanent simulation
	Single cycle	Step-by-step simulation. Click the icon to execute one program cycle
	Pause	Interrupt the simulation. Click the icon once more time to continue the simulation
	Stop	Stop simulation
	Refresh time	Input field for setting the information refresh period on the scheme in milliseconds
	Cycle time	Input fields for setting the cycle time of program execution in simulation mode. Cycle time units: milliseconds, seconds, minutes, hours
	Watch Window	Open/close the window to watch the variables values at each program step



CAUTION

The parameters **Cycle time** in ALP simulation mode and **Cycle time** in the device are different in spite of the same name.

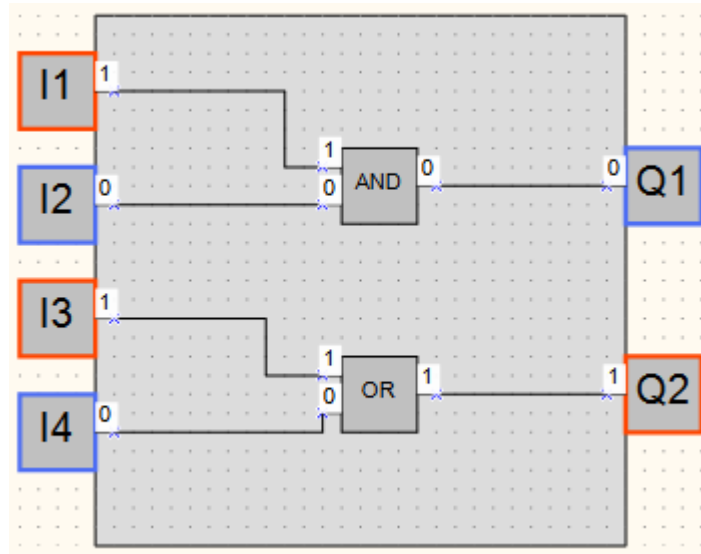
Calendar toolbar

An additional toolbar **Calendar** is displayed in simulation mode if there are FBs of type CLOCK or CLOCK WEEK in the program (available only for devices with real-time clock). It is used for simulation of such blocks.

Date/time 17:31:11 3 Jul 2019

Simulation procedure

1. Run simulation in one of the modes: real time (▶) or step-by-step (↞).
2. Set the input values on program blocks.



3. Select values of parameters **Refresh time**, **Cycle time** and **Cycle unit time** for convenient simulation.
4. Exit simulation mode to correct the program.

In simulation mode you can change the values of the device inputs by clicking on them. In this case a discrete input will change its Boolean value and the color, for analog inputs the value is set in the dialog window with the input field.



NOTE

Macros are excluded from simulation. Simulation for macros should be performed separately in the macro workspace.

Simulation cannot be performed for:

- blocks without connection with device outputs or network variable output blocks
- incorrectly associated variables
- retain variables

You can also specify the variable value directly on the scheme. Double-click on a variable to open a dialog with an input field for a new value. The value of the network variables can be set as well.

Watch window



Click icon on the simulation toolbar to watch the input, output or variable values at every program step.

Watch Window		
Name	Text	Data type

To add an input, an output or a variable to the **Watch list**, click the empty field in the **Name** column and then click the «...» icon appeared to the left.

Watch Window		
Name	Text	Data type
I5	0	BOOL
a2	0	BOOL
b2	0	BOOL

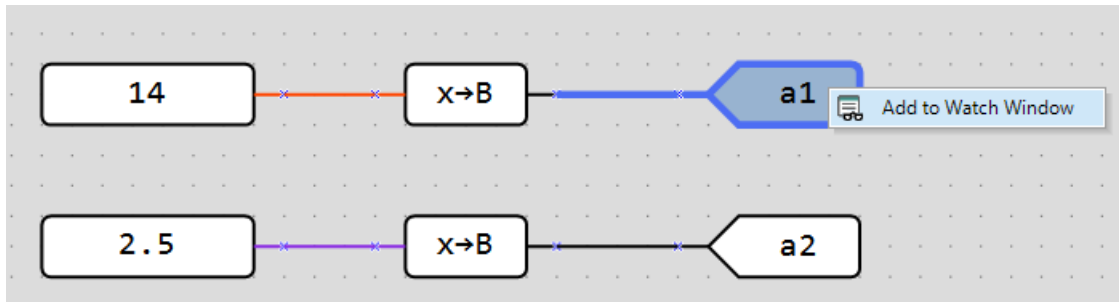
The **Variable table** will open. Here project variables, inputs and outputs can be selected.

Select network variable			
+ 🔍 ✖ 📄		Search	✖
Variable name	Data type	Register address	Comment
v3_net	INT	512	
v4_net	INT	513	
< none >	INT	514	

The selected variables are added to the preview window.

Watch Window		
Name	Text	Data type
a1	0	BOOL

The block context menu can also be used.



The values of the variables, inputs and outputs can be set in the **Value** column during simulation.

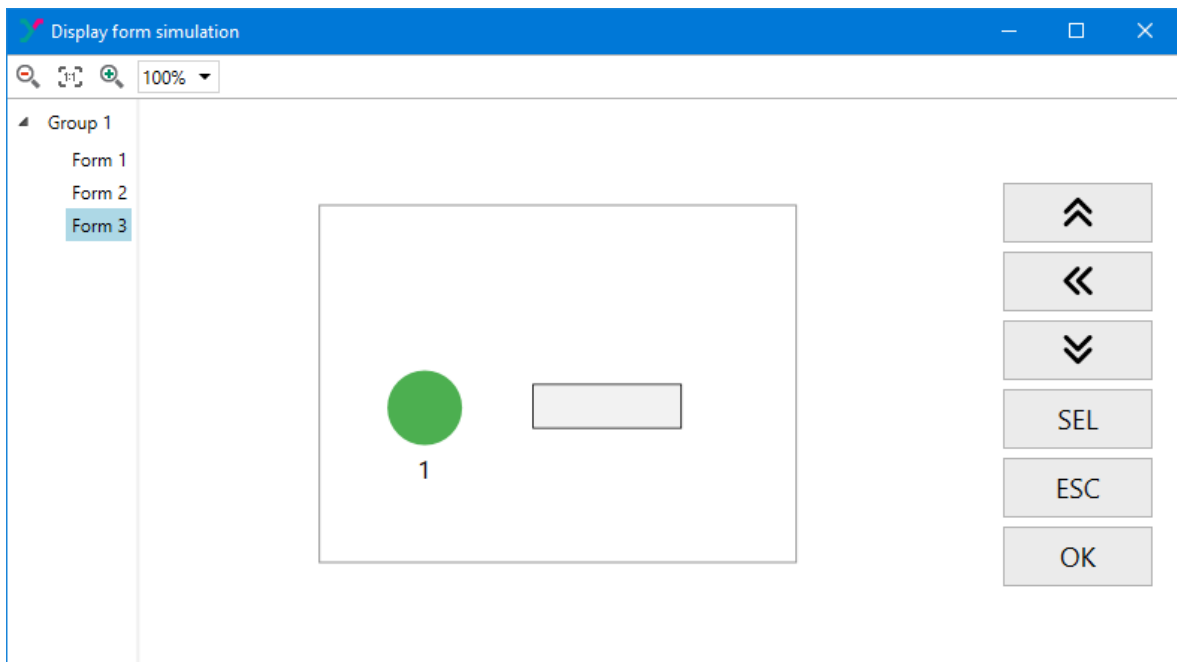
4.5.1 Visualization Simulation

In simulator mode for devices with graphic color LCD, simulation of the device display and buttons is available. After entering the mode, a window displaying the device start screen will open.



NOTE

The Graph visualization element is not supported in simulation mode.



Zooming buttons are located in the upper left corner of the window.

A list of screens available in the project is located in the left part of the window.

The center part of the window displays a simulation of the device screen.

Buttons for simulation of actions with real buttons of the device are located in the right part of the window.

Working with the screen

The following visualization elements can be edited in simulation mode:

- Dynamic Text
- Int/float I/O forms
- Time I/O forms
- IP I/O forms

Working with buttons

Buttons in the right part of the window are intended for simulation of actions with real buttons of the device.


The button actions are set for each screen. Button actions are processed for the screen that is currently displayed on the main area of the window.

Simulation buttons can be controlled with the keys **Q, W, E, A, S, D, Z, X, C**. Pressing buttons on the keyboard is enabled only if the visualization simulation window is in focus.

In a step-by-step simulation, button hold is not enabled.

4.5.2 ST code simulation

Simulation of function or function block code written in the ST language is available in ALP. To use it,

go to the **Function Editor** (or **Function Block Editor**) window and click the  button on the toolbar, or in the main menu **Service** → **Simulator Mode**. The simulation panel will open. In the **Editor** window, the variable values will be displayed in the view boxes next to the variable. If arrays are declared in the program, the word **Modify** is displayed in the view window.

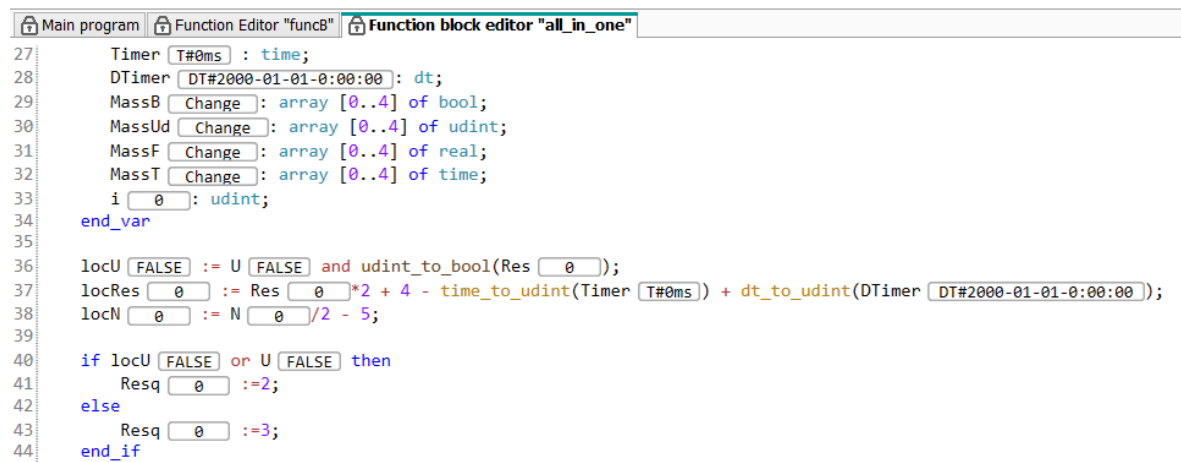


Fig. 4.1 Function block editor window in simulator mode

Color coding of the view windows:

- 13.3 — the current value of the variable;
- 12 — the value of the variable has been changed by the user;
- 13.3 — the value of the variable has been fixed by the user
- Change — the value of one or more elements of the array (not all) has been fixed by the user;
- ??? — error: incorrect variable value (for example, when a non-existent array element is specified using a variable).

If you left-click on the view window to enter the value of an integer variable (uint) or a floating point variable (real) in the simulation mode, a window will appear in which you can enter a new value of the variable and, if necessary, fix the value.

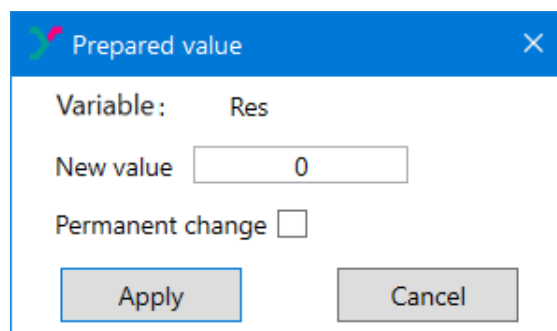


Fig. 4.2 Changing the value of a uint variable

When you left-click on the view window to enter the value of a Boolean variable (bool) in simulation mode, a window will appear where you can select true or false and, if necessary, fix the value.

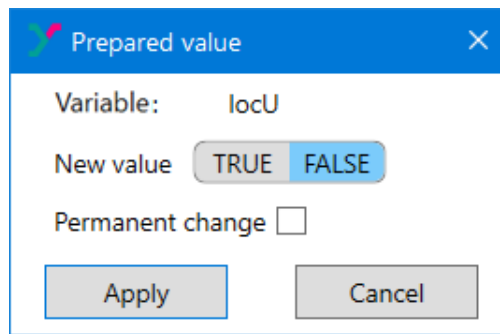


Fig. 4.3 Changing the value of a bool variable

When you left-click the view window to enter the value of a time variable (time or DT) in the simulation mode, a window will appear where you can enter the value in the format T#0d0h0m0s0ms (for time) or DT#2000-01-01-0:00:00 (for dt), and fix the value if necessary.

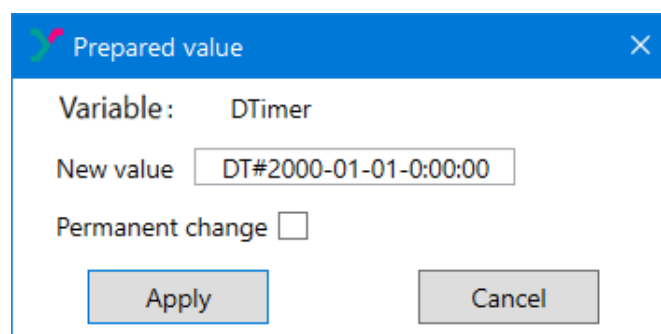


Fig. 4.4 Changing the value of a time variable

**NOTE**

The time is recalculated automatically after the **Accept** button is pressed. For example, the entered value of the variable T#80m will look like T#1h20m when the **Change Value** window is reopened.

If an incorrect value or no value is entered, an error message will appear. The **Accept** button will be disabled.

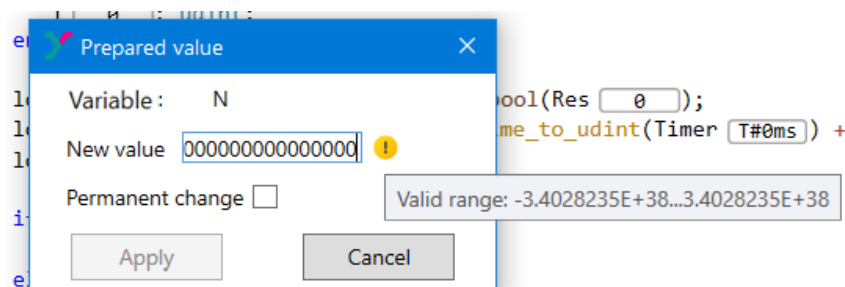
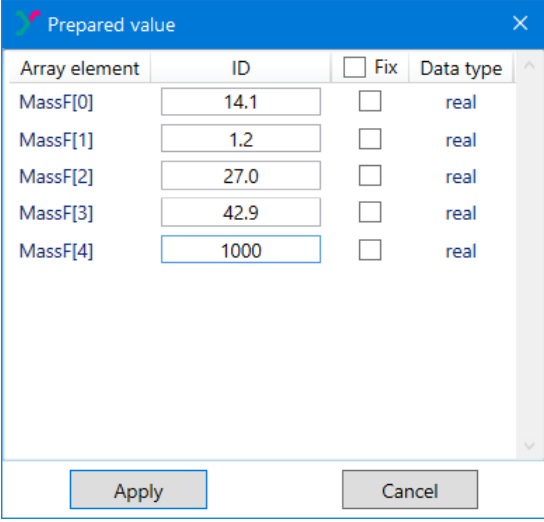


Fig. 4.5 Value entry error

To enter values in the array elements, left-click on the view window labeled **Modify**:

- If the element index is specified by a constant or using a variable, the Change Value for Specific Array Element window will appear;
- If the index of an array element is specified as the result of a mathematical expression or as the result of a function/function block, a window with all elements of the array and variable values available for input will appear:

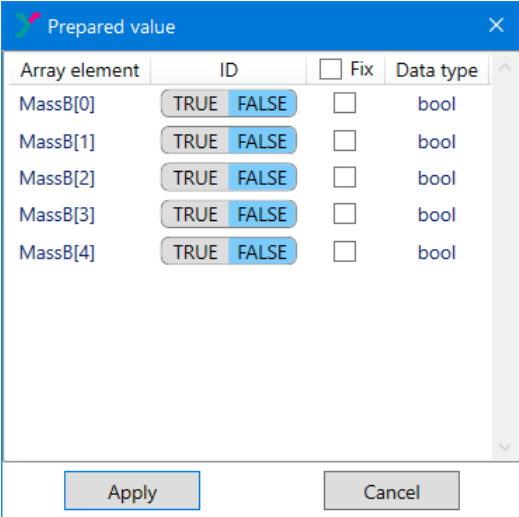


The 'Prepared value' dialog box shows a table with columns: Array element, ID, Fix, and Data type. The data is as follows:

Array element	ID	Fix	Data type
MassF[0]	14.1	<input type="checkbox"/>	real
MassF[1]	1.2	<input type="checkbox"/>	real
MassF[2]	27.0	<input type="checkbox"/>	real
MassF[3]	42.9	<input type="checkbox"/>	real
MassF[4]	1000	<input type="checkbox"/>	real

Buttons: Apply, Cancel

Fig. 4.6 Changing values of array elements with real variables

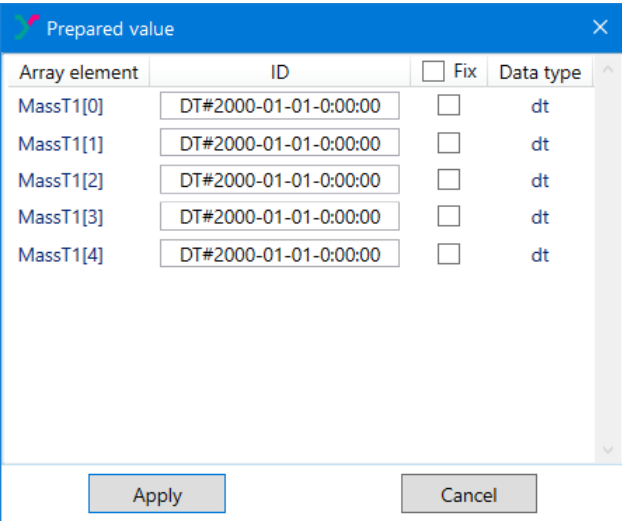


The 'Prepared value' dialog box shows a table with columns: Array element, ID, Fix, and Data type. The data is as follows:

Array element	ID	Fix	Data type
MassB[0]	TRUE FALSE	<input type="checkbox"/>	bool
MassB[1]	TRUE FALSE	<input type="checkbox"/>	bool
MassB[2]	TRUE FALSE	<input type="checkbox"/>	bool
MassB[3]	TRUE FALSE	<input type="checkbox"/>	bool
MassB[4]	TRUE FALSE	<input type="checkbox"/>	bool

Buttons: Apply, Cancel

Fig. 4.7 Changing values of array elements with bool variables



The 'Prepared value' dialog box shows a table with columns: Array element, ID, Fix, and Data type. The data is as follows:

Array element	ID	Fix	Data type
MassT1[0]	DT#2000-01-01-0:00:00	<input type="checkbox"/>	dt
MassT1[1]	DT#2000-01-01-0:00:00	<input type="checkbox"/>	dt
MassT1[2]	DT#2000-01-01-0:00:00	<input type="checkbox"/>	dt
MassT1[3]	DT#2000-01-01-0:00:00	<input type="checkbox"/>	dt
MassT1[4]	DT#2000-01-01-0:00:00	<input type="checkbox"/>	dt

Buttons: Apply, Cancel

Fig. 4.8 Changing values of array elements with dt variables

If an incorrect value or no value is entered, an error message will appear. The **Accept** button will be disabled.

Array element	ID	Fix	Data type
MassUd[0]	140	<input type="checkbox"/>	uint
MassUd[1]	5470	<input type="checkbox"/>	uint
MassUd[2]	560	<input type="checkbox"/>	uint
MassUd[3]	9680	<input type="checkbox"/>	uint
MassUd[4]	36586797800	<input type="checkbox"/>	uint

Valid range: 0...4294967295

Apply Cancel

Fig. 4.9 Value entry error

4.6 Connection to device



CAUTION

The device must be powered off before connecting to PC.

All devices can be connected to PC over USB. If the device has an Ethernet interface, it can be connected over Ethernet. To temporarily interrupt the connection, use **OFFLINE mode**.

Connection over USB

Connection parameters

Connection type
Serial port

Connection parameters

Serial port	COM4
Baud rate	9600
Data bits	8
Parity	none
Stop bits	1
Device address	16

Serial port
Programming port

OK Cancel

Devices can be connected to PC over USB.

1. Connect the device to a USB port of the PC and switch the device on.
2. Start ALP and select the menu item **Device > Port settings**.
3. Select **Serial port** for **Connection type**.

4. Select the serial port in the opened dialog. The number of the emulated COM port can be found in the Windows Device Manager under "Connections (COM and LPT)".
5. Enter the **Device address** (16 by default) and confirm with **OK**. All other parameters are displayed only for your information.

If connection is established successfully, status indicators will show the information about connected device and serial port.

Connection over Ethernet

The screenshot shows a 'Connection parameters' dialog box. It has a title bar with a close button. Inside, there's a 'Connection type' dropdown menu currently showing 'Ethernet / Wi-Fi'. Below this is a section titled 'Connection parameters' with a collapse icon. Under this section, there's a label 'IP address' followed by the value '127.0.0.1'. At the bottom of the dialog, there's a section labeled 'IP address' with the text 'Device IP address for connecting'. The dialog ends with 'OK' and 'Cancel' buttons.

To connect the device to a PC via Ethernet interface or Wi-Fi, consider the following steps:

1. Connect the device to the same local network as the PC.
2. Find out the IP address of the connected device. The default IP address is specified in User Manual for the device.. The current IP address of the device can be read using the software.
3. Select **Ethernet / Wi-Fi** for **Connection type**.
4. Enter **IP address** of the connected device and confirm with **OK**.

If connection is established successfully, status indicators will show the information about connected device and serial port.

4.7 Upload project to device

Upload project to device




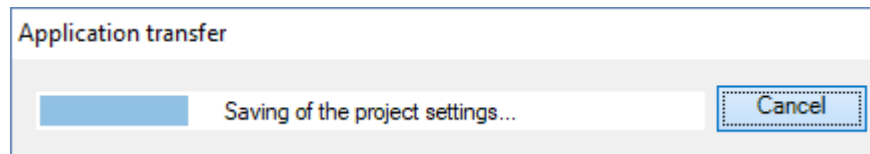
CAUTION

When a new project is uploaded to the device, the program already stored in the device memory (ROM) will be replaced by the new one.

Proceed as follows:

1. Connect the device to PC.
2. Power on the device.
3. Adjust the port settings if necessary.
4. Upload project to the device.

The project can be uploaded to the device using the menu item **File** → **Transfer application to device** or clicking the icon  in the toolbar. When the upload is completed, the device can be powered off and disconnected from the PC.



If the target device does not match the connected device, a warning message will be displayed.

**NOTE**

When the program transfer is completed, the device goes to the operating mode and the program starts automatically.

OFFLINE mode

In the **OFFLINE** mode, the connection between ALP and the device is interrupted.

The mode is helpful when you work with two ALP instances running on PC and trying to communicate with the same device. Both applications will alternately occupy the port and the connection to the device will be constantly interrupted.

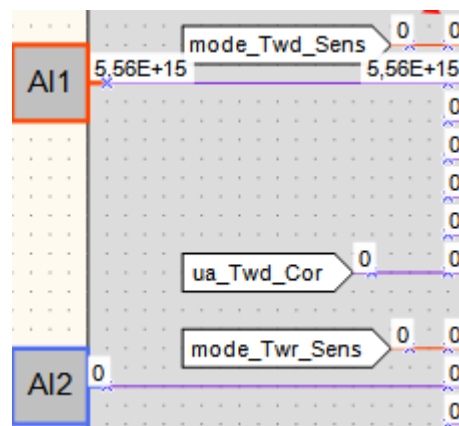
The ALP instance that is not to interact with the device has to be set to OFFLINE mode.

OFFLINE mode can be activated / deactivated using the menu item **Service** → **OFFLINE mode** or by clicking the status indicator 3.7 Device. With the next click OFFLINE mode deactivated.

4.8 Online debugging

To start the online debugging, click the toolbar icon .

In this mode the current values of all program variables including functions, function blocks, macros, inputs and outputs are read out from the connected device and displayed in the workspace. This way you can check the logic of the device program.




The online debugging is possible only if:

- the device is connected to the PC
- the program in the device and the program opened in ALP is the same
- the version of the device firmware is compatible with the current version of ALP

The online debugging is only available for the main program workspace, not within macros.

It is not possible to make changes in the project during online debugging. If you want to modify the

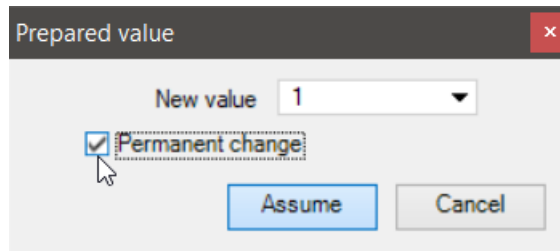
project, exit online debugging by clicking the  icon once more.

**NOTE**

If communication with the device is lost, online debugging is terminated after 10 seconds and the device is switched to operating mode. If the connection is restored within 10 seconds, online debugging continues, but the entered values are reset.

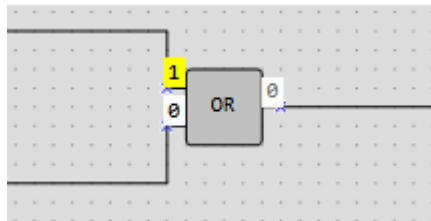
Manual value entry

In the online debugging mode, it is possible to set the input values of functions, function blocks and macros manually by clicking on the displayed value. The new value should be entered in the field **New value** in the opened dialog **Prepared value**. There are two options to change the value: one-off or permanent change.

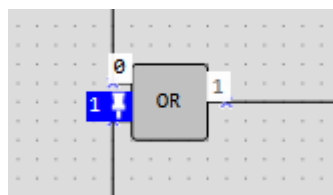


The one-off change is active when the option **Permanent change** is unchecked. This enables to change the block input value for one program cycle. In the subsequent cycle, the signal from one of the device inputs or the output of another program block connected to this input is applied. The option is useful for single pulse simulation.

The new value in the workspace is highlighted in yellow during its validity.



When the option **Permanent change** is checked, the entered value is applied to the input until it is changed or the online debugging is stopped. The permanent value in the workspace is highlighted in blue with a white pin.

**Troubleshooting**

If the connection with the device is lost, the online debugging mode will be reset after 10 seconds, and the device will go into operating mode. If you manage to restore the connection, online debugging will continue, but the recorded values will be reset.

**NOTE**

For each modification of the device there is a limit on the transmitted values in online debugging mode. If the diagram displays empty cells of values, then a limitation is triggered, and you should increase the scale of the diagram so that fewer values fall into the “visible window”. Fixed values remain frozen if they do not fall into the “visible window”, but reduce the limit of transferred values because they occupy memory areas.

4.9 Project information

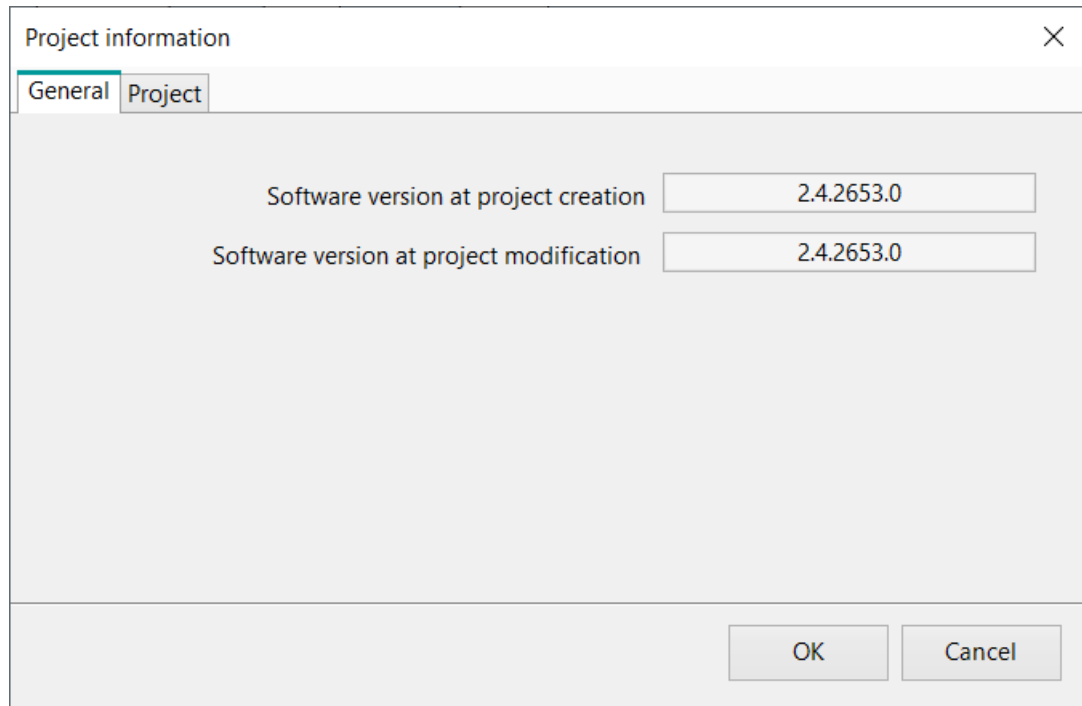
Use the menu item **File** → **Project information** to view and modify the information about the program.

General

The tab **General** contains the information about the software version.

Software version at project creation – the version of the software in which the project has been created.

Software version at project modification – the version of the software in which the project has been modified.



The screenshot shows a dialog box titled "Project information" with a close button (X) in the top right corner. It has two tabs: "General" (selected) and "Project". The "General" tab contains two text input fields. The first field is labeled "Software version at project creation" and contains the text "2.4.2653.0". The second field is labeled "Software version at project modification" and also contains the text "2.4.2653.0". At the bottom right of the dialog box are two buttons: "OK" and "Cancel".

Project

The tab **Project** is not available for each device. In the tab you can specify information about the group, number and version of the program to be displayed in the **Device information** window of the connected device after the project is saved to it.

Project information

General Project

Group PR200

Number 0101

Version 0 . 0 . 1

OK Cancel

- **Group** – project group name
- **Number** – project number within the group
- **Version** – project version

Click **OK** to save the information in the project, or click **Cancel** to discard it.

4.10 Component manager

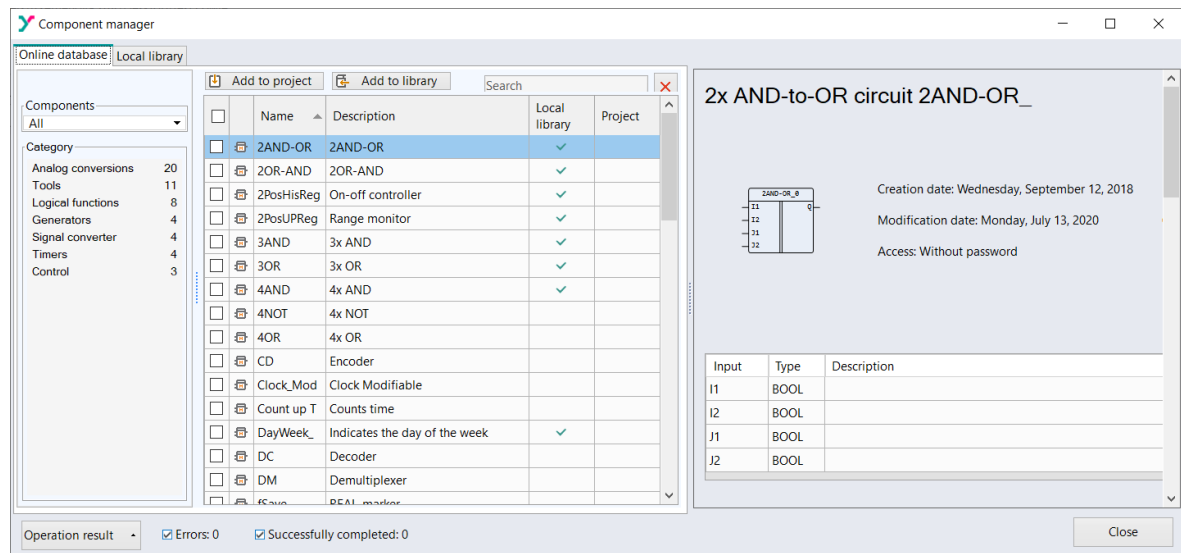
New macros and device templates can be downloaded from akYtec Online Database. Component Manager is the tool for all interactions with this database. The internet access is necessary for this interaction.

Select the menu item **File** → **Component manager** to open it in a separate window.

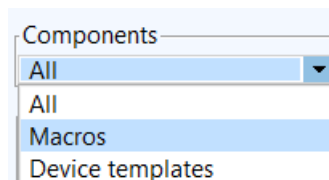
Online Database

- **Add to project** button – the selected blocks (macros or device templates) from Online Database are added to the project library. The blocks are then stored in the project file and can be viewed in Library Box in the **Project Macros** area.
- **Add to library** button – the selected blocks from Online Database are downloaded to the local library and can then be used offline.

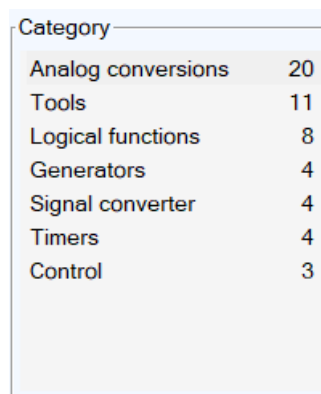
A check mark in the column **Project** or **Library** indicates that the block has been successfully downloaded (added).



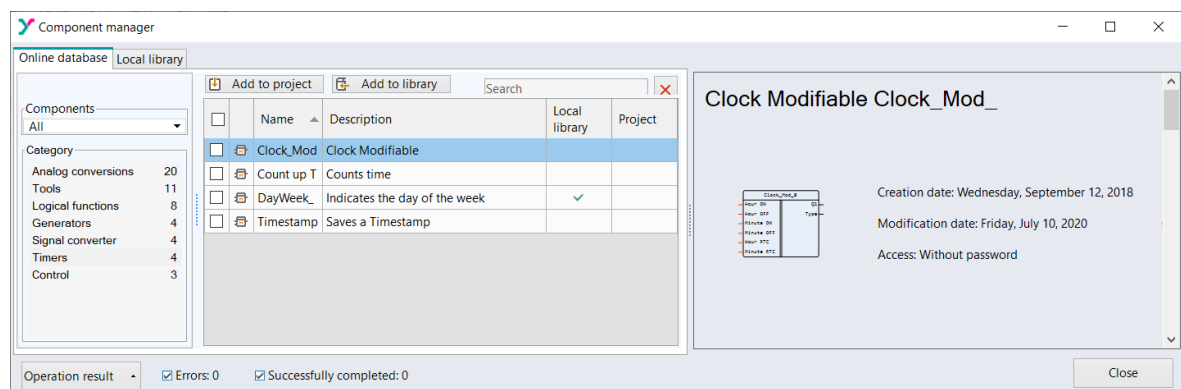
The **Components** drop-down menu allows you to filter the list by type:



Macros are further divided into categories depending on their purpose:






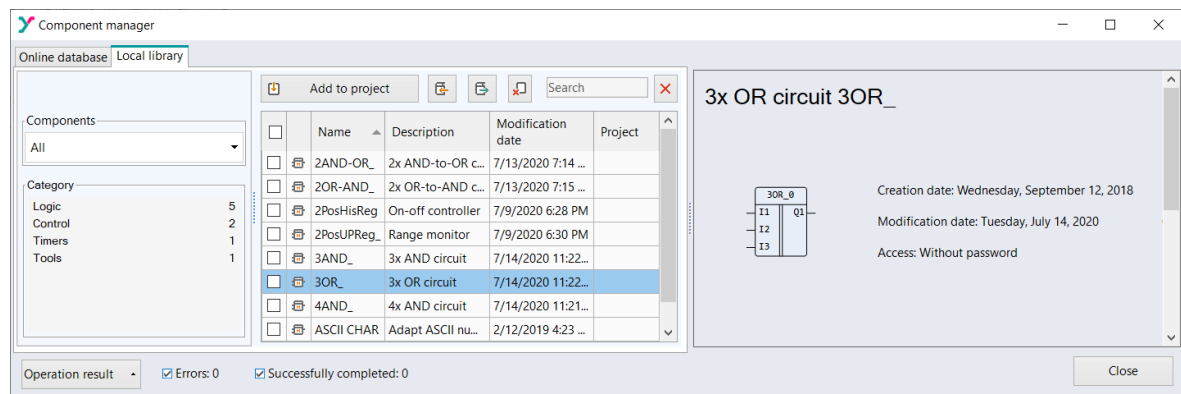
Brief description of the selected block is displayed in the upper right field, and the full description is in the lower right field. The full description is a PDF document. Scroll the document to the end to see the PDF toolbar. Using it, you can download the document or print it.



Click the button **Operation result** at the bottom of the window to view the program messages about the performed operations.

Local library

- **Add to project** button – the selected blocks (macros or device templates) from Online Database are added to the project library. The blocks are saved in the project file and can be viewed in the Library Box in the **Project Macros** area.
-  – the selected blocks are added from a file in the project library
-  – the selected block from the project library is saved as a file under the specified path for further use
-  – the selected blocks are removed from the local library



NOTE

Library files are stored at the local address:

**C: \Users \[username] \ Documents \ akYtec ALP \ Library **

4.11 Macro development

Macro is a user function block opened in a separate workspace. A macro can be created in the project in two ways:

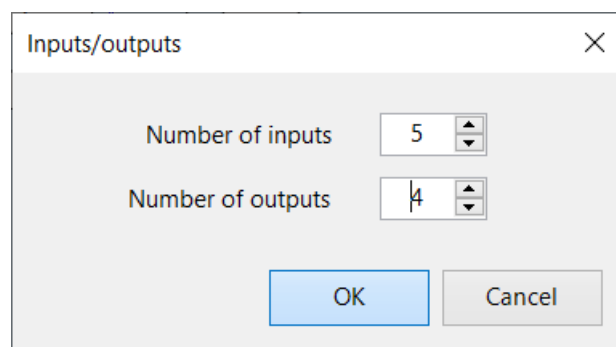
Basic functions with macros:

- using the main menu item **File** → **New macro**
- drawing a rectangle around several blocks in the main workspace to select them and clicking the item **New macro** from the workspace context menu.

New macro using main menu

To create a new macro:

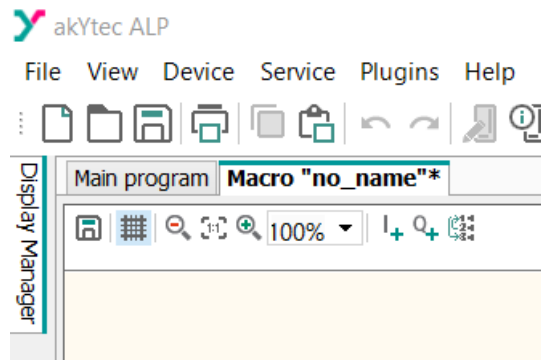
1. Select the item **File** → **New macro** in the main menu. Then specify the number of inputs and outputs in the opened dialog window:



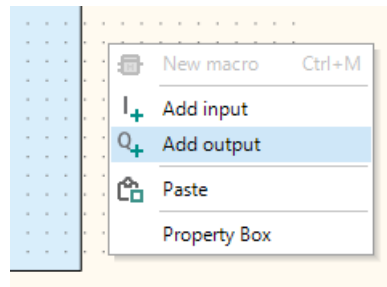
**NOTE**

The number of inputs and outputs can be changed after creating a macro.

2. Develop the macro algorithm in the **Macro Editor** tab, similar to developing the program in the diagram.

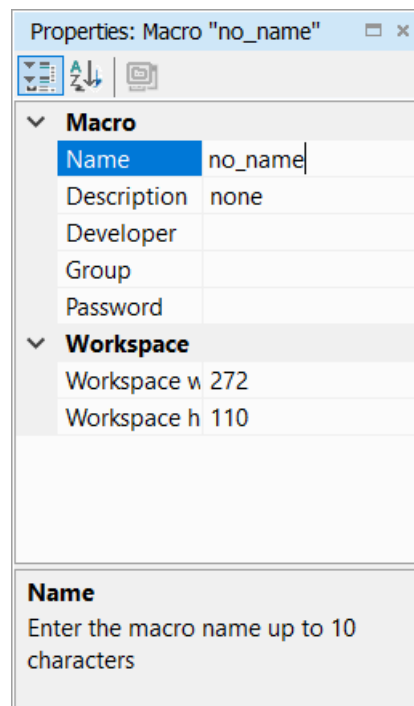


The number of inputs and outputs can be always changed. To add a new input or output, use the items **I+** or **Q+** in the toolbar or in the workspace context menu.



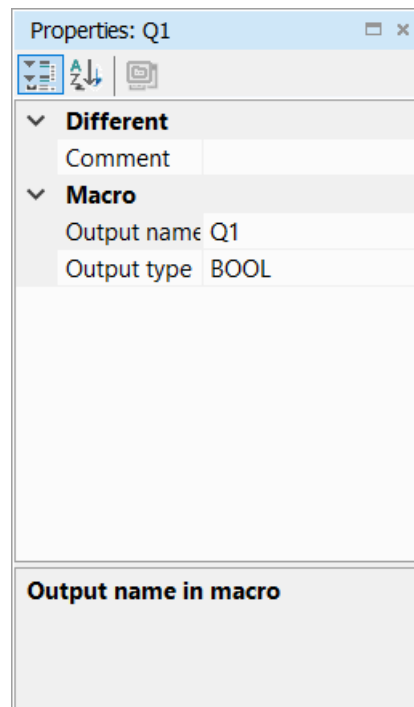
To remove an input or an output, use **Remove** in its context menu.

3. In Property Box, give a name, a description and a group to the macro:

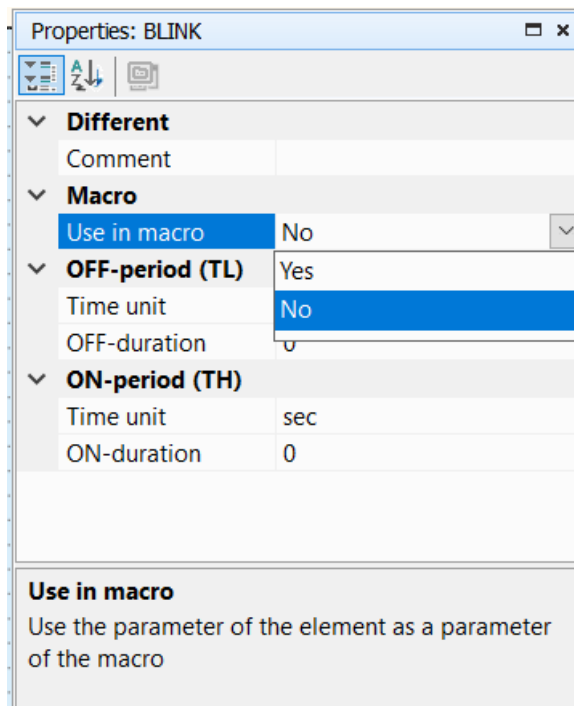


The name is displayed in the workspace tab header and in Library Box.

4. In Property Box, the name and the data type can be changed for each input and output.



5. Next, you can set the visibility of the FB parameters used in the macro in the main program.



If the parameter **Use in macro** is set to **Yes**, the FB parameters become parameters of the macro and a new option **Parameters of macro** is added to the macro in Property Box.

It is a list of FB parameters, where the user can specify the name for each FB parameter in the macro for use in the main program. If you want the parameter names in the macro to be different from those in FB, click on the **<Parameters...>** line to edit the parameter names.

Use in macro	Variable name	Name in macro
Time unit	Time unit	Time unit
ON-duration	ON-duration	ON-duration
OFF-duration	Time unit	Time unit
Time unit	OFF-duration	OFF-duration
ON-duration	OFF-duration	OFF-duration

6. The macro can be simulated in the same way as the main program.
7. Before saving the macro, you can fill in the following fields: **Name**, **Description**, **Developer**, **Group** and **Password**.

Macro	
Name	no_name
Description	none
Developer	
Group	
Password	

Workspace	
Workspace width (mm)	272
Workspace height (mm)	110

Group
Group in the library

It is recommended to select a short and clear name. The text in the parameter **Description** is displayed in Library Box under the macro name and in a tooltip, when the mouse cursor is over the macro in the main workspace.

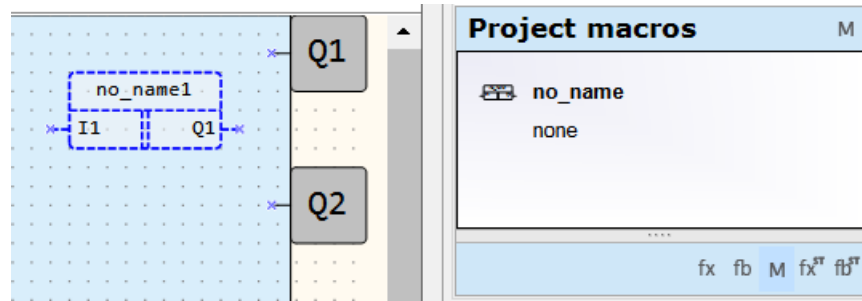
If you set the password for the macro, it will be asked every time the menu item Edit macro is selected. Otherwise, editing of the macro is available to everyone.

The name in the parameter **Group** is used in the project library. If the group name is empty, the macro is assigned to the group **Other** in the library.

The macro can be saved by selecting **File** → **Save macro as...** or by clicking the icon  in the Macro Editor toolbar.

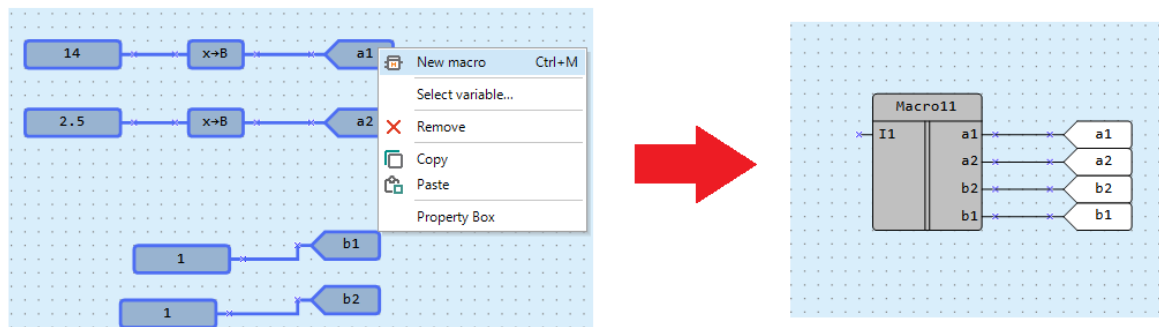
Saved macro is available only for an open project, to use the macro in other project it must be exported in a file and then imported in an other project.

8. Select section **Project macros** in the **Library Box** and drag it to the workspace.



New macro using context menu

You can create a macro by drawing a selection rectangle in the workspace and using the item **New macro** in the workspace context menu. All selected blocks will be moved into the new macro block that will replace the selected blocks in the main workspace. All external connecting lines will be retained.



There are some specific aspects of creating macros using context menu:

1. The number of inputs and outputs of the macro is equal to the number of connected input and output connections in the selected area. In case that blocks without connections are selected, the macro with one input and one output will be created.
2. If a standard variable block is selected, the variable will be copied under the same name into the macro.



NOTE

The variables in the macro and in the main program are different in spite of the same name, there is no conflict between them.

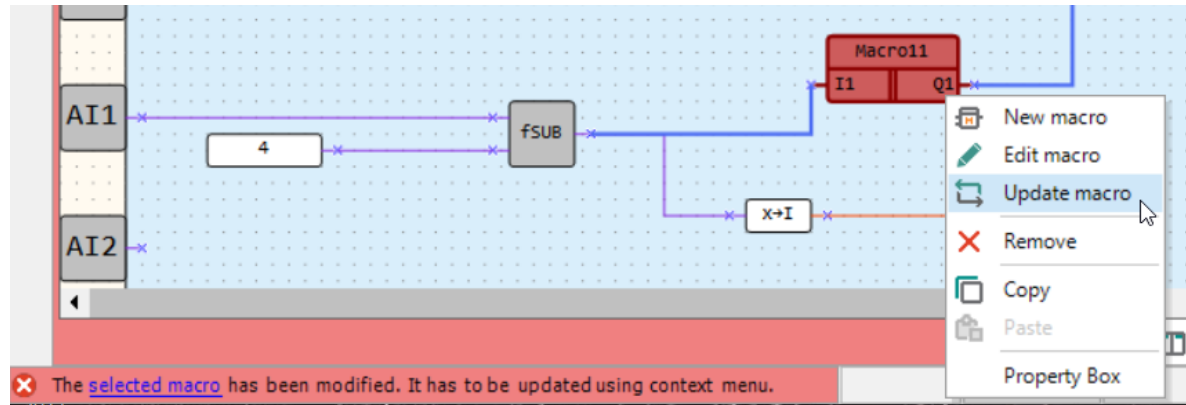
1. If all blocks of a variable are selected and it has no other references in the program, the variable will be moved into the macro.
2. If the selected variable is used (has blocks or other references) outside the selected area, it will be copied under the same name into the macro and the original will remain in the workspace.
3. If only one block of the input or output variable is selected, the variable will be copied under the same name into the macro and the original will remain in the workspace.
4. If the macro is created using the context menu, the following blocks will not be included in it:
 - device inputs and outputs
 - service variables
 - network variables
 - PID controller

In case the above-mentioned blocks are selected, they will remain in the main workspace and will be connected to the corresponding I/O points of the macro.

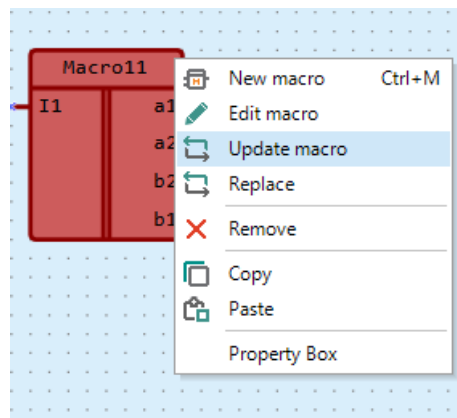
5. If any **WriteToFB** / **ReadFromFB** blocks (see [sect. 4.3.6](#)) are assigned to the selected FB, they will be included to the macro, even if they are not selected. If the read/write blocks are selected but not assigned to FB, they will not be included in the macro.

Update macro

If the macro used in the main program has been modified, (name, type, number of I/O points, elements or the parameter **Use in macro** of any FB), it will be highlighted in red in the main program and the user will be prompted to update the macro. The macro is considered to be modified when the changes made in Macro Editor are saved.



To update the macro, use its context menu.



Once the macro has been updated in the main program, the next modified macro will be prompted to update.

Update rules:

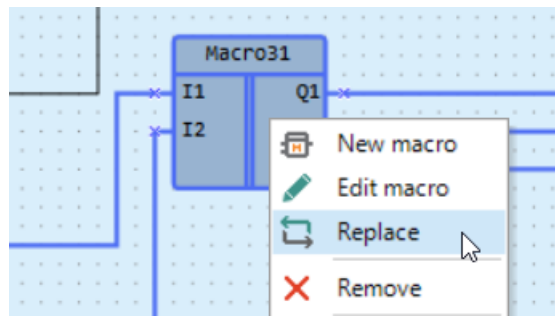
- If the type or name of the macro I/O point with the attached connection is changed, the connection will be disconnected after the update.
- If I/O points are added to the macro, the existing I/O points will not be disconnected after the update.
- Macro I/O points are identified by name and type. If you change the name or type of an I/O point with an external connection and create a new I/O point with the same name and type, the connection will be automatically linked to the new I/O point after the macro update.

Replace macro

If a macro should be replaced with another one, it can be done manually: delete the macro, add a new macro and restore the connecting lines.

It is more efficient to replace the macro using the context menu command **Replace**. The lines connecting I/O points to macro will be retained if the names and the data types of the old and the new I/O points are the same.

If the name or data type of an I/O point does not match, the connecting line will be cut and should be repaired manually.



FB in macro

If an FB is used in the macro, the user can define whether the FB parameters are available (visible) in the main program as the parameters of the macro.

If the parameter **Use in macro** is set to **No**, the FB parameters are visible and can be used only within the macro.

If the parameter **Use in macro** is set to **Yes**, the FB parameters became parameters of the macro and a new option **Parameters of macro** is added to the macro in Property Box.


It is a list of FB parameters, where the user can specify the name for each FB parameter in the macro for use in the main program. If you want the parameter names in the macro to be different from those in FB, click on the **<Parameters...>** line to edit the parameter names.

Changing I/O points order

The I/O points of the macro are placed on the sides of the macro in the order in which they were added, from top to bottom. This order can be changed.

This can be useful if you want to place logically related I/O points nearby, or if you want to insert an empty macro into the program and determine the position of its I/O points later, after developing its algorithm.


Proceed as follows:

1. Open the macro in the editor, drag and drop the I/O points into the desired order.
2. Click on the toolbar icon  **Synchronize I/O order** to synchronize the positions of the I/O points, and then save the macro.



NOTE

The synchronization does not work if the macro I/O points are not connected to other program blocks.

3. Go to the main program. The changed macro is highlighted red and it is offered to update it using its context menu.
4. After the update, the order of the macro I/O points in the main program will be the same as in the macro editor. The connecting lines of the macro will be retained.
5. If the synchronization switch  is not activated, the macro I/O points will be displayed in the main program in the default order. This can be useful if you need to add an I/O point to the macro, but don't want to entangle the existing lines.

Export macro

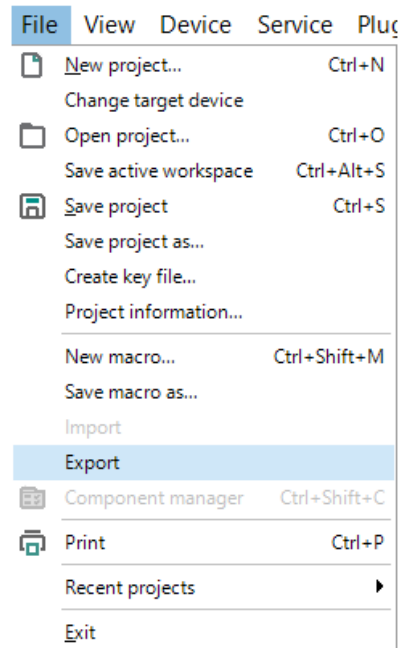
Exporting a macro to a file is only possible when the macro editor window is open. To export a macro, select **File** → **Export** in the main menu.

To export a macro:

1. Open the macro in the editor.

If you need to edit the macro before saving, you should drag it onto the project canvas and select **Edit** and make changes in the macro context menu.

2. Select the main menu item **File** → **Export**.



3. In the window that opens, select a location and save the macro file with the extension **.tpl* or **.tple*. After saving, a message indicating that the macro was exported successfully will be displayed.

Import macro

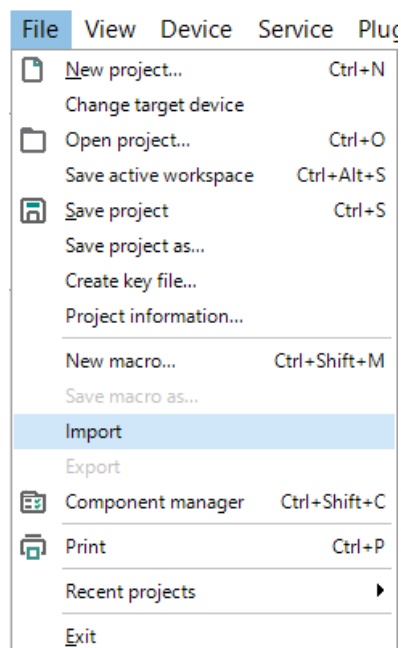
If you need to use a macro created in another project to create a program, you can import the required macro into the project.

To import a macro use the main menu item **File** → **Import**.





NOTE

The **Import** item is active only when the focus is on the workspace.




In the window that opens, select the desired file and click the **OK** button. The macro will be added to the **Library Box** in the **Project macros** section, and can now be used in the project.

Copy macro

Macros can be copied from project to project for reuse and reduced development time. To copy a macro, select the macro block in the source project and click the  on the toolbar or select the **Copy** command in the block context menu. The macro is inserted into another project by clicking the  button on the toolbar or by selecting the **Insert** in the canvas context menu. You can also use keyboard shortcuts to copy and paste, see [Keyboard shortcuts](#). Once inserted, the macro will be available in the **Project macros** section of the **Library Box**.

4.12 Using ST function


Creation of user functions in [ST language 12](#) is available for devices on the new hardware platform. If the project is created for such a device, the toolbar icon  **New ST function** is active.

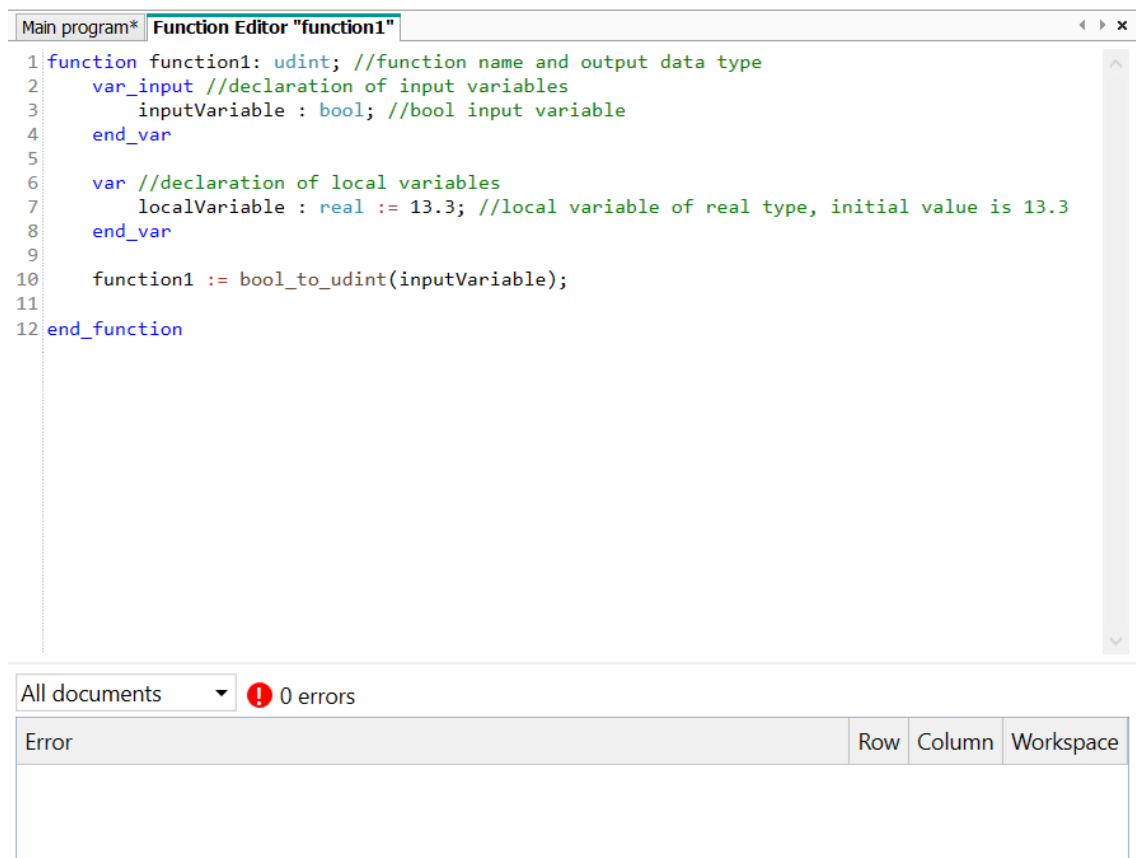


NOTICE

ST functions reserve space in ROM memory after they are added to the project library, regardless of whether they are used in the project or not.

Creation of ST function

1. Click the toolbar icon  **New ST function**. Function editor with an ST function template opens in a new workspace.



```

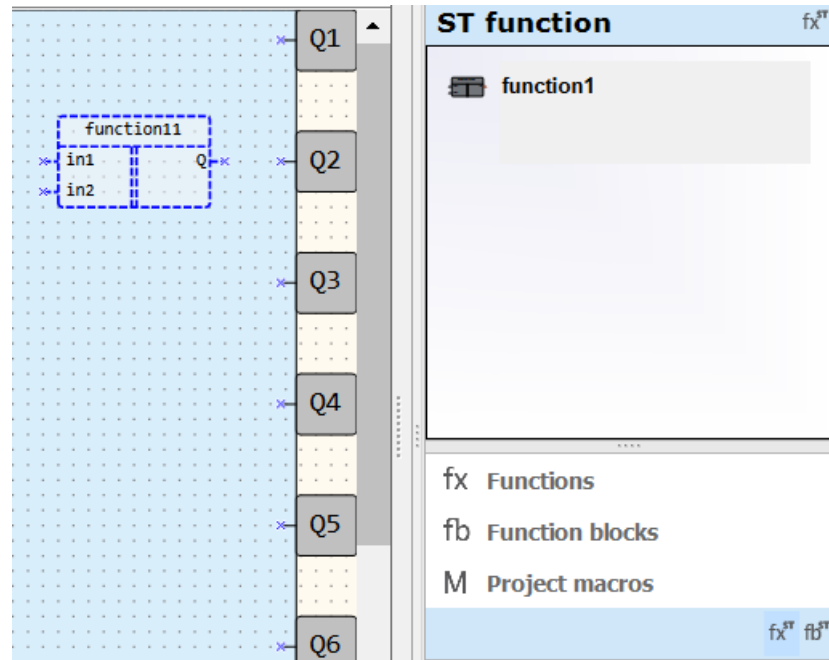
1 function function1: uint; //function name and output data type
2   var_input //declaration of input variables
3     inputVariable : bool; //bool input variable
4   end_var
5
6   var //declaration of local variables
7     localVariable : real := 13.3; //local variable of real type, initial value is 13.3
8   end_var
9
10  function1 := bool_to_uint(inputVariable);
11
12 end_function
  
```

All documents 0 errors

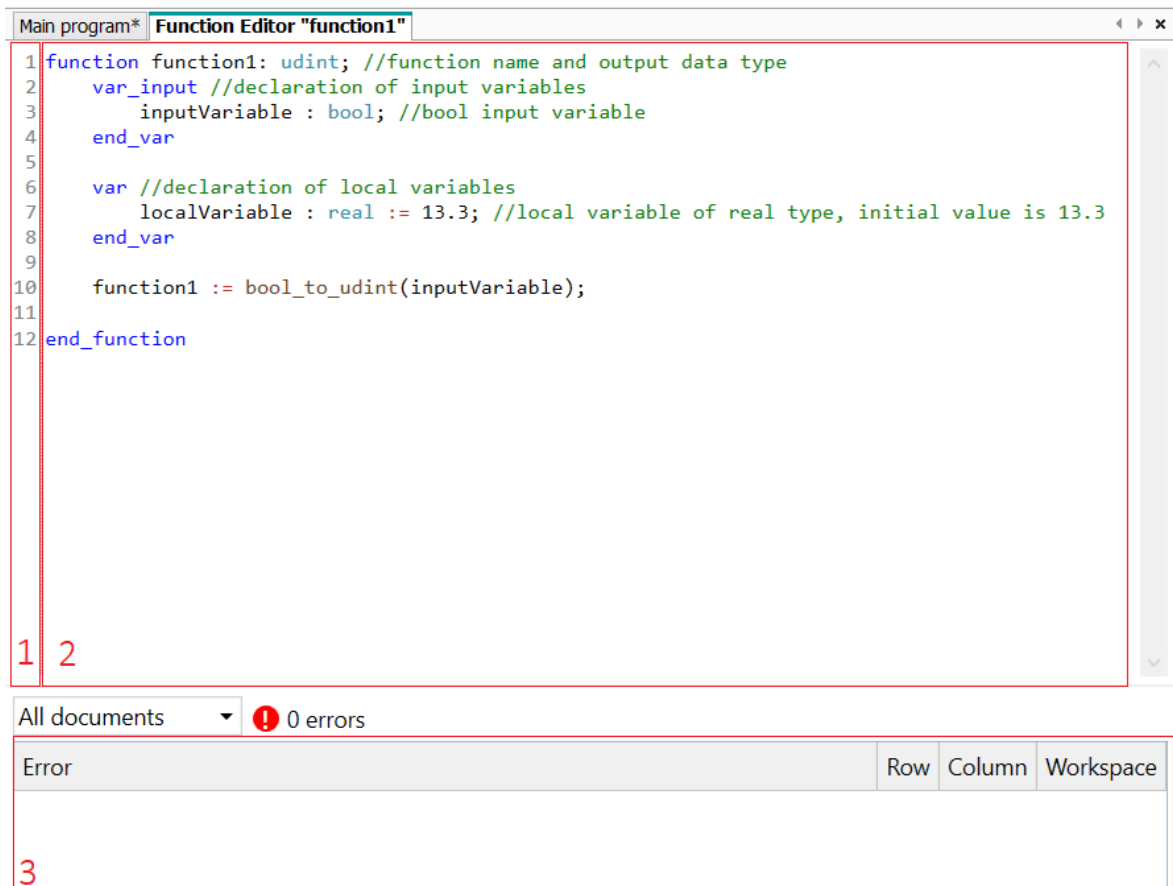
Error	Row	Column	Workspace
-------	-----	--------	-----------

2. Specify the function name and the output data type in the first line.
3. Specify all required inputs variables in the input variable declaration block **var_input**.
4. Specify all required local variables in the local variable declaration block **var**.

5. Develop a function algorithm in accordance with the ST syntax rules.
6. Switch to the **Main program** tab or close the **Function editor** tab. The function will be saved automatically.
7. Select the section **ST functions** in **Library Box** and drag the saved function onto the project workspace.



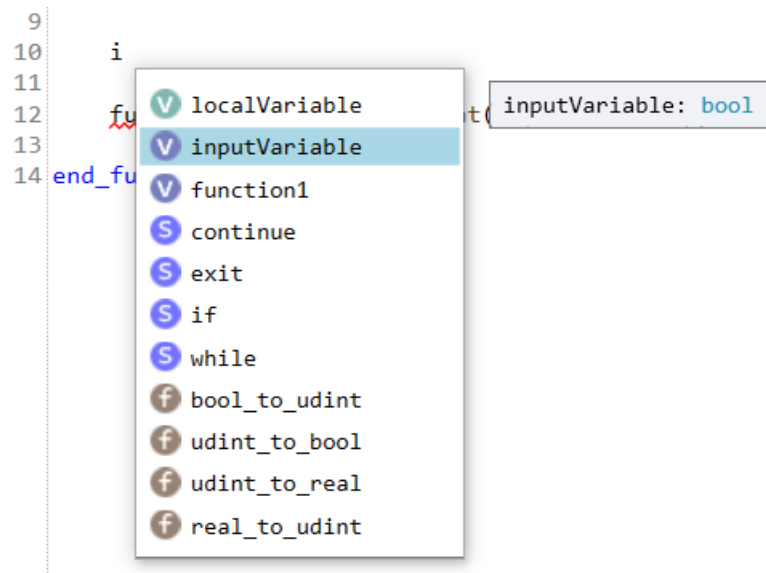
Function editor interface



1. **Line numbers** — sequential line numbers in the program code.
2. **Code editor** — code editing area with automatic syntax highlighting.
3. **Error panel** — error display area.

Snippet management

Snippet management is a text editor feature that allows easy insertion of content from a catalogue of repeatedly used text. If you enter the first character in the editor, a context menu opens with focus on the first line. Use the cursor keys to select a snippet. To insert the selected snippet into the code, press **Enter** or **Tab** or double-click on the list item.



Snippet groups:

- V — local variables
- S — statements (**while**, **for** etc.)
- K — keywords (**true**, **false**)
- F — built-in functions
- T — other functions

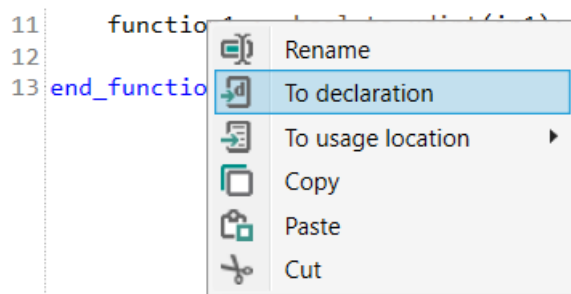
Within groups, snippets are arranged alphabetically.

Jump to declaration or usage location

For convenient work with the code, a search is implemented to find the places in the program code where a function or variable is declared or used.

To jump to declaration:

1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select **To declaration** in the context menu.



To jump to usage location:

1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select **To usage location** in the context menu. A list of places in the code opens the selected function or variable is used.

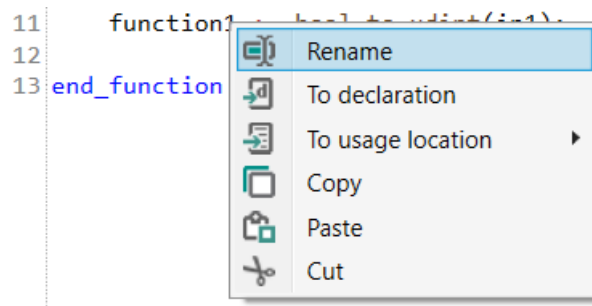


4. Left-click on the selected usage location. The cursor will move to the line where the function or variable is used.

Rename variable or function

Centralized change of the name of a variable or function throughout the code is available. Proceed as follows:

1. Place the cursor on the name of a function or variable in the program code.
2. Right-click on the name.
3. Select **Rename** in the context menu. There is a list of places in the code, where the selected function or variable is used.



4. Enter a new name in one of the green marked locations and click on another place in the code. Now the name is changed in the whole program.

Error panel

All errors occurred during the code writing are listed in **Error panel**. Left-click on a row in the list to jump to the error in the code.

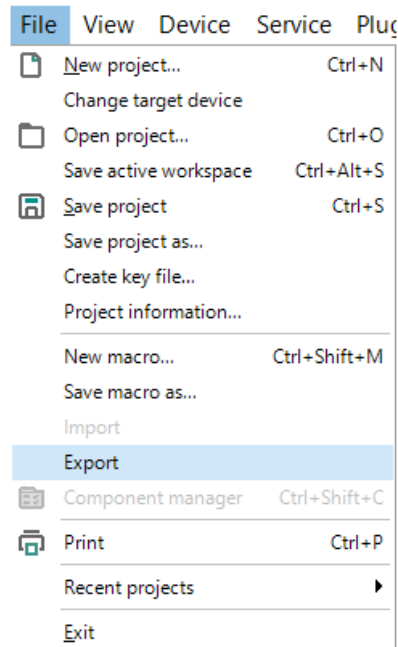
All documents 2 out of 2 errors			
Error	Row	Column	Workspace
Variable in not declared	11	32	function1
Invalid arguments for function bool_to_udint	11	18	function1

Export ST function

Exporting a function to a file is only possible when the function editor tab is open. To export a function, select **File** → **Export** in the main menu.

To export function proceed as follows:

1. Open the function in the editor.
2. Select **File** → **Export** from the main menu.



3. In the window that opens, select a location and save the function file with the extension **.fst*.
Once saved, a message indicating that the function was exported successfully will be displayed.

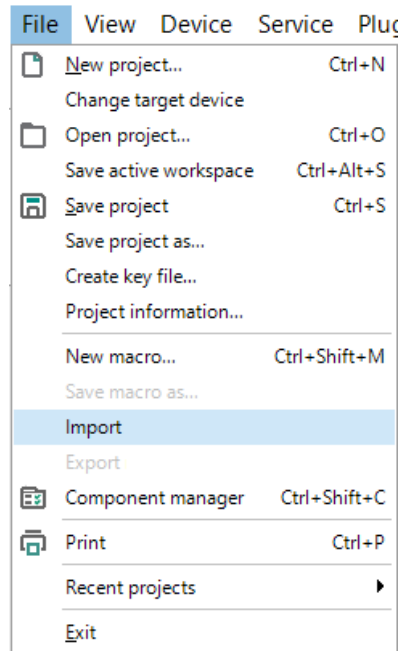
Import ST function

If you need to use a function created in another project to create a program, you can import it into the project.

To import a function block, select **File** → **Import** in the main menu.


NOTE

The **Import** item is active only when the focus is on the project workspace.



In the window that opens, select the desired file and click the **OK** button. The function will be added to the **Library Box** in the **ST functions** section, now it can be used in the project.

4.13 ST function blocks

Creation of user ST function blocks is available for devices on the new hardware platform. If the project is created for such a device, the toolbar icon  **New ST function block** is active.

Creating an ST function block

To create an ST function block:

1. Click the toolbar icon  **New ST function block**. Function block editor with an ST function block template opens in a new workspace.

The screenshot shows the 'Function block editor' window for 'functionblock1'. The code is written in Structured Text (ST) and defines a counter function block. It includes input variable declarations (U, Res, N), output variable declarations (Q), and local variable declarations (CounterValue, RTrig). The logic uses if-then-else statements to increment the counter value based on the input U and the reset input Res.

```

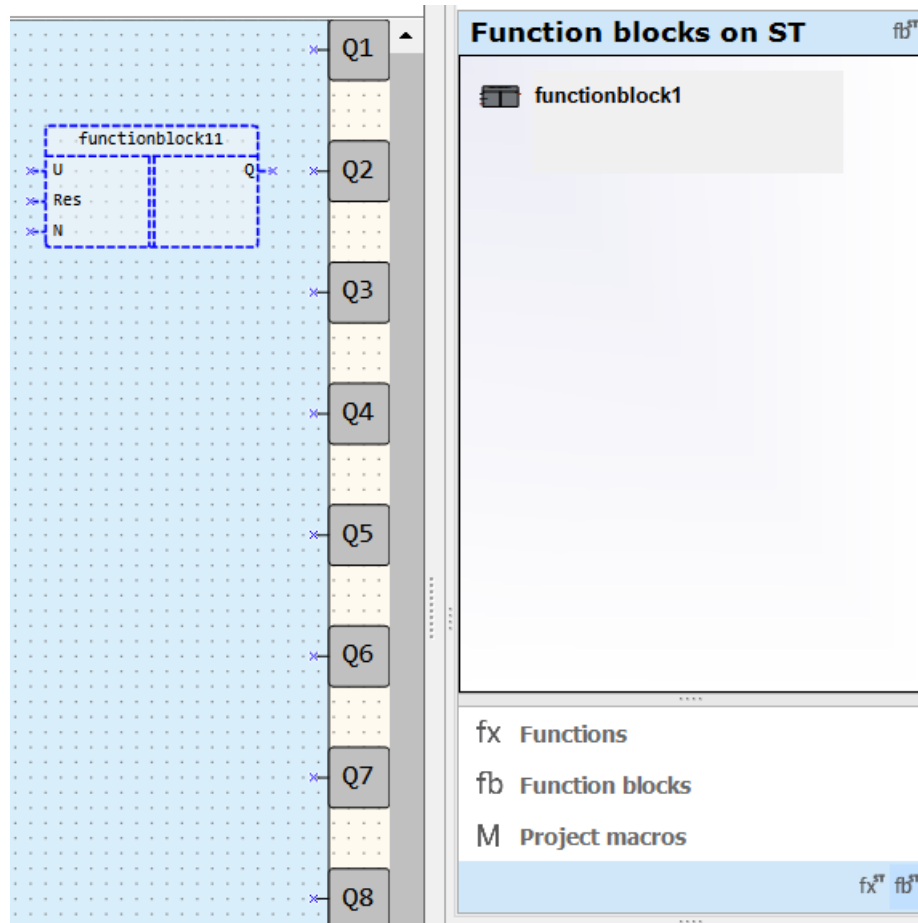
1 function_block functionblock1 // function block name
2
3 // An example of a function block on ST that is a counter and is used for direct counting.
4 // The "up count" operation is performed on the rising edge of the pulse at the "U" input,
5 // which increases the value of the output signal "Q".
6 // When a logical "1" is received at the input Res, the output of the counter "Q" is set to the value of the input "N".
7
8 var_input //declaration of input variables
9   U : bool; //boolean input variable
10  Res : bool; //input variable with data type bool
11  N : uint; //input variable for counter value after reset
12 end_var
13
14 var_output //declaration of output variables
15   Q : uint; //uint output variable
16 end_var
17
18 var //declaration of local variables
19   CounterValue : uint; //variable for current counter value
20   RTrig : bool; //variable for determining the rising edge at input "U"
21 end_var
22
23 if Res then
24   CounterValue := N;
25 end_if
26 if U and not RTrig and not Res then
27   CounterValue := (CounterValue + 1);
28   RTrig := U;
29 end_if
30 if not U and RTrig then
31   RTrig := false;
32 end_if
33 Q := CounterValue;
34
35 end_function_block

```

Below the code editor, there is a status bar showing 'Current document' and '0 out of 2 errors'. An error list table is also visible with columns for 'Error', 'Row', 'Column', and 'Workspace'.

Error	Row	Column	Workspace

2. Specify the function block name and the output data type in the first line.
3. Specify all required inputs variables in the input variable declaration block **var_input**.
4. Specify all required local variables in the local variable declaration block **var**.
5. Develop a function block algorithm in accordance with the ST syntax rules.
6. Switch to the **Main program** tab or close the **Function block editor** tab. The function block will be saved automatically.
7. Select the section **ST function blocks** in **Library Box** and drag the saved block onto the project workspace.



Function block editor interface

Main program* **Function block editor "functionblock1"**

```

1 function_block functionblock1 // function block name
2
3 // An example of a function block on ST that is a counter and is used for direct counting.
4 // The "up count" operation is performed on the rising edge of the pulse at the "U" input,
5 // which increases the value of the output signal "Q".
6 // When a logical "1" is received at the input Res, the output of the counter "Q" is set to the value of the input "N".
7
8 var_input //declaration of input variables
9   U : bool; //boolean input variable
10   Res : bool; //input variable with data type bool
11   N : uint; //input variable for counter value after reset
12 end_var
13
14 var_output //declaration of output variables
15   Q : uint; //uint output variable
16 end_var
17
18 var //declaration of local variables
19   CounterValue : uint; //variable for current counter value
20   RTrig : bool; //variable for determining the rising edge at input "U"
21 end_var
22
23 if Res then
24   CounterValue := N;
25 end_if
26 if U and not RTrig and not Res then
27   CounterValue := (CounterValue + 1);
28   RTrig := U;
29 end_if
30 if not U and RTrig then
31   RTrig := false;
32 end_if
33 Q := CounterValue;
34
35 end_function_block

```

1 2

Current document 0 out of 2 errors

Error	Row	Column	Workspace
3			

1. **Line numbers** — sequential line numbers in the program code.
2. **Code editor** — code editing area with automatic syntax highlighting.
3. **Error panel** — error display area.



NOTE

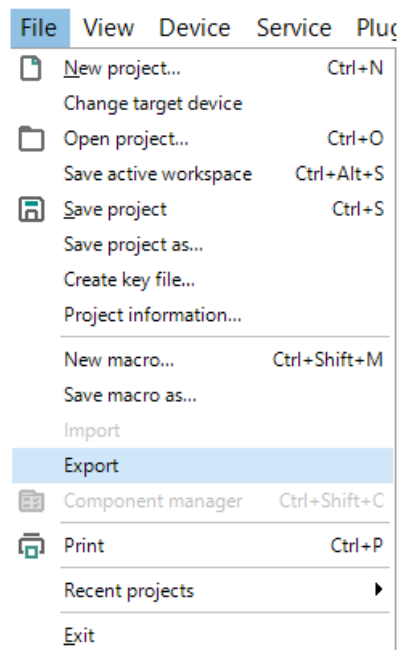
Similar to the function editor, the function block editor supports the functions snippets, tracking the location of declaration and use, renaming and error tracking.

Export ST function block

Exporting a function block to a file is only possible when the function block editor tab is open. To export a function, select **File** → **Export** in the main menu.

To export function block proceed as follows:

1. Open the function block in the editor.
2. Select **File** → **Export** from the main menu.



3. In the window that opens, select a location and save the function file with the extension *.fst. Once saved, a message indicating that the function was exported successfully will be displayed.

Import ST function block

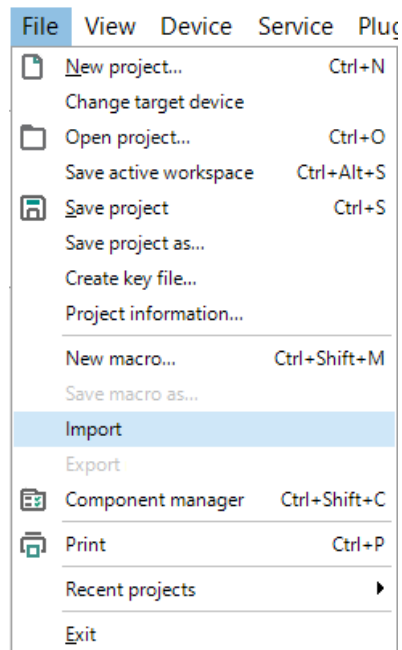
If you need to use a function block created in another project to create a program, you can import it into the project.

To import a function block, select **File** → **Import** in the main menu.



NOTE

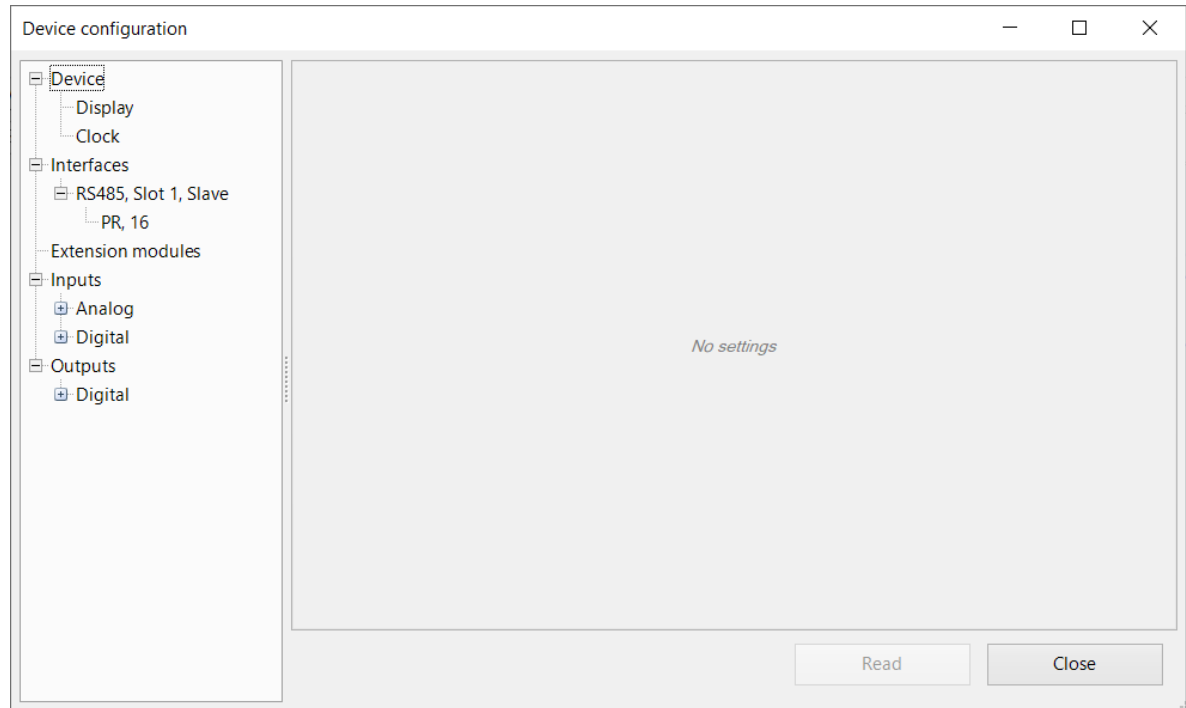
The **Import** item is active only when the focus is on the project workspace.



In the window that opens, select the desired file and click the **OK** button. The function block will be added to the **Library Box** in the **ST function blocks** section, now it can be used in the project.

5 Device configuration

The configuration of the device is a part of a project and can be set using the menu item **Device** → **Configuration**. The dialog window **Device configuration** consists of two parts. The configurable parameters of the device are presented in the parameter tree in the left part of the window. The content of a group is presented in the right part.



The content of the parameter tree depends on the target device and may include the following groups:

- Display
- Clock
- Interfaces
- Extension modules
- Inputs and outputs

All the settings are saved in the project, except the clock settings. The configuration is also possible without connecting the device.

5.1 Display

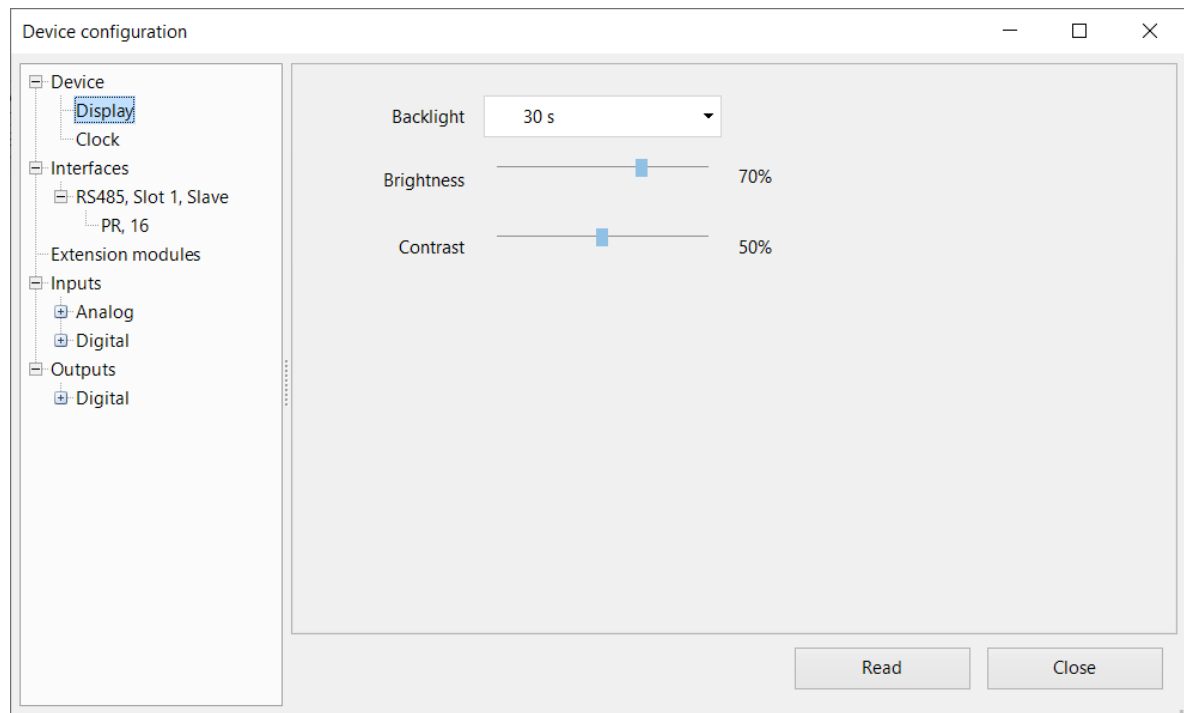
If the target device has a display, the following parameters can be set:

Backlight – the duration of the backlight since the last user activity

Brightness – 0...100%

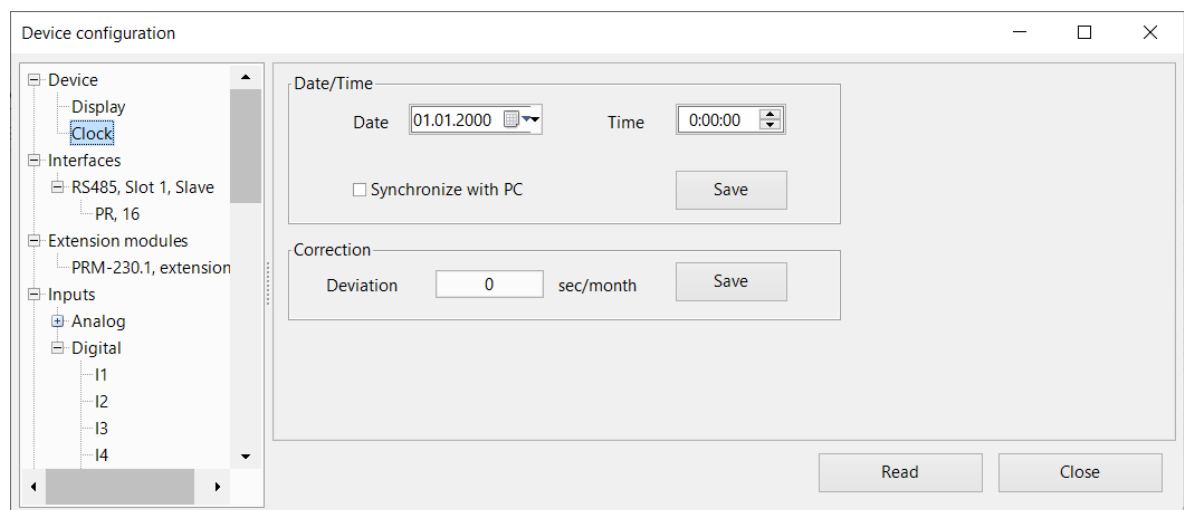
Contrast – 0...100%

The button **Read** can be used to read out the current display settings from the connected device.



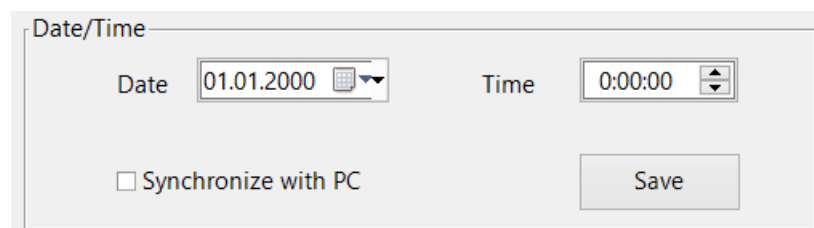
5.2 Clock

If the target device has a built-in real-time clock, the date and time can be set in the **Clock** group.



Date and time

To synchronize the device clock with the PC clock, check the checkbox **Synchronize with PC**. In this case the fields **Date** and **Time** become inactive. To set the device clock to the new values, click the button **Save** in the section **Date/Time**.



Correction

Specify the clock error in seconds per month in the field **Deviation** to set the clock correction. Enter a negative value if the device clock is too fast.

To save the clock correction in the device, click the button **Save** in the section **Correction**. The button **Read** can be used to read the current time settings from the connected device.

Clock configuration for the new hardware devices

The clock settings window for devices on the new hardware platform has a different interface and does not have time correction (their hardware provides greater accuracy). Setting the time zone is required to display local time correctly, since the device stores the time value as Greenwich Mean Time (GMT). Enabling the **Set computer time zone** option synchronizes the real world clock device time with PC clock.

5.3 Data exchange

- Interfaces
- Modbus

5.3.1 Interfaces

If the target device has a serial network interface RS485, its parameters can be set in the group **Interfaces**.

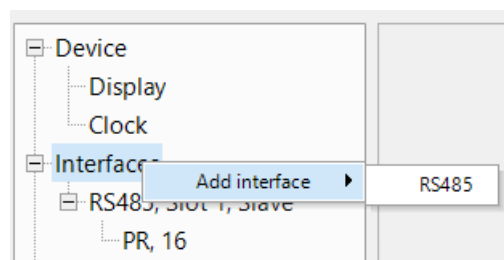
By default, there is one interface configured as a slave and assigned to the hardware slot 1 with the following settings: master device with the name PR and the network address 16.

If the number of interfaces on the target device can be changed, interfaces can be added or deleted in the configuration, but their number cannot exceed the number of the existing slots.

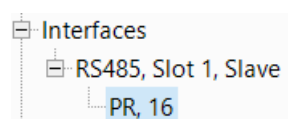
If an interface is configured as a master, slaves can be added to the configuration or removed, but their number may not exceed 16.

Add interface

If the device has a slot, for which no interface is configured, an appropriate interface can be added using the item **Add Interface** in the context menu.

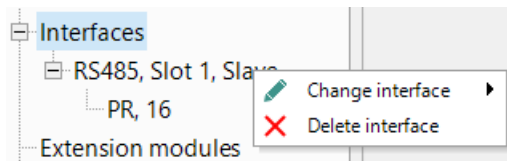


An interface of the selected type with default settings is added.

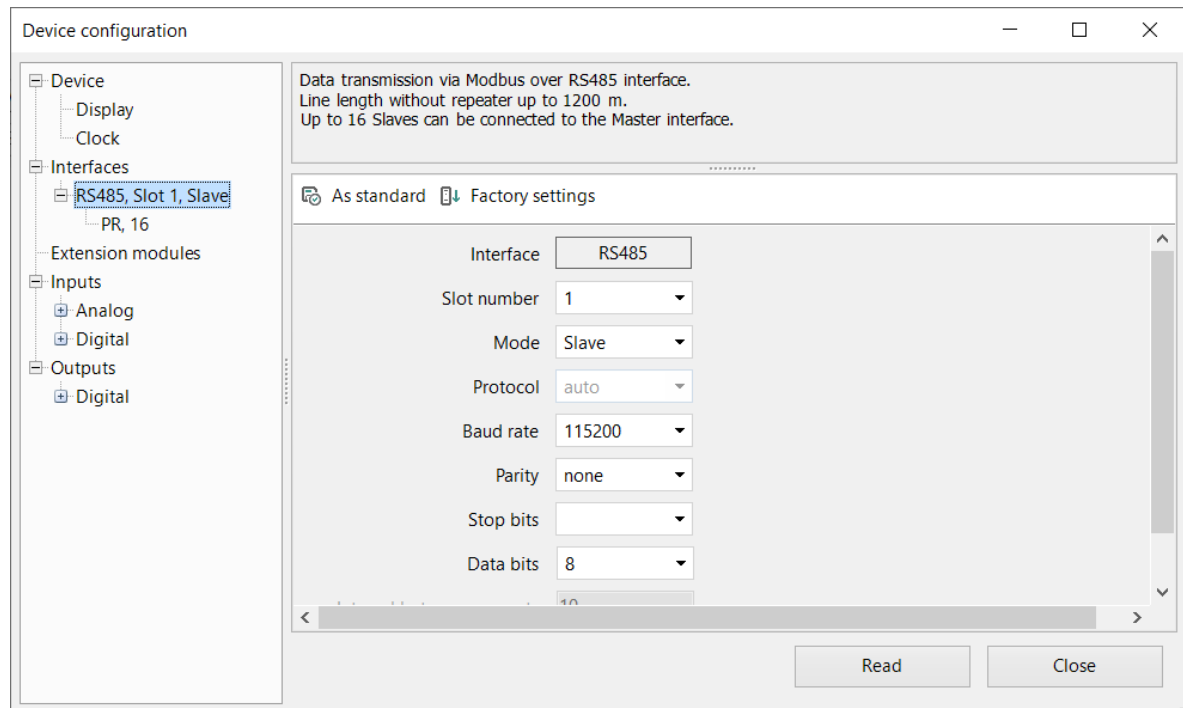


Replace/remove interface

Depending on device, the interface can be replaced by another type of interface or removed using the context menu.





5.3.1.1 RS485 interface configuration



The type of the interface (RS485), the number of the assigned slot and the mode (master / slave) are displayed in the tree.

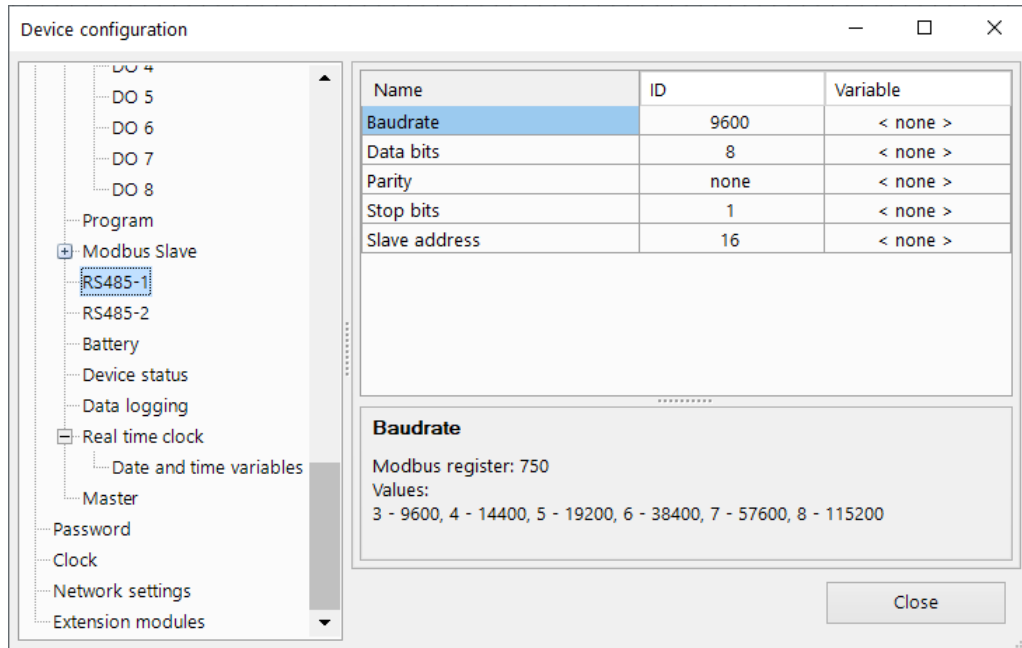
To establish the connection over the interface, it has to be configured. The parameters of the interface are displayed in the right part of the window. The default value depends on the target device. The parameters **Protocol** and **Interval between requests** are only available in the master mode. In the slave mode they are inactive and grayed out.

The icon  **As standard** is used to save the settings as default values for future projects.

The icon  **Factory settings** is used for applying the unchangeable factory settings. The button **Read** is used to read out the current settings from the connected device. Use the button **Close** to save the settings in the project and close the dialog.

Interface parameters for second generation devices

For second generation devices, the interface parameters are located in the **RS-485 port settings** section in the settings tree. The right side of the device settings window displays interface parameters. The settings window looks like the figure below.



5.3.1.2 Ethernet interface configuration

Ethernet settings are available only for second generation devices in the **Ethernet Settings** menu in the **Network settings** section of the settings tree.

The settings window displays current network parameters of the device, and also sets new ones. After saving the settings with the new IP address, the device should be rebooted.



NOTE

After setting a new IP address, the device will lose connection with the PC. For the new connection you need to specify a new IP address (see [Connection to device](#)).

5.3.2 Modbus

- [Modbus working](#);
- [Master mode](#);
- [Slave mode](#).

5.3.2.1 Modbus working

ALP can be used to program devices that support Modbus-RTU or Modbus-ASCII (master / slave) protocols.

In order to organize data exchange in the network over the RS485 interface, a master device is required. There can be only one master in the network.

Cycle time

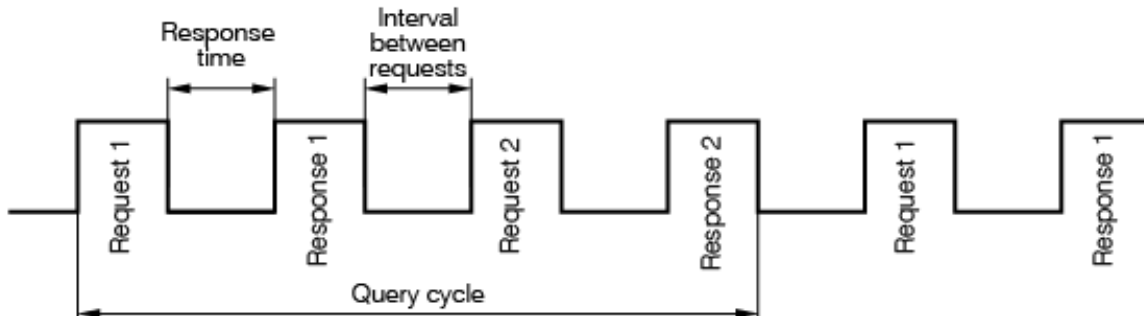
The program execution time (cycle time) is automatically adjusted (auto-tuned) depending on the program complexity. The auto-tuning affects data exchange over Modbus, since the program execution has a higher priority than request processing. If the program is large, it can take up all the CPU time and Modbus data exchange will not be performed correctly.

To avoid this problem, the lower limit for the volume of the Modbus data exchange is reserved: 50 requests per second. Thus, at least 50 requests per second can be executed even if the user program is large, and even more if the program is small and the processor capacity is sufficient. If there is not enough time to poll all devices, the number of requests should be optimized in the user program.

The **Query cycle** setting depends on the number of polled variables and the polling frequency in the program. It is recommended to set **Query cycle** to 1 s. In this case, the device will be able to request up to 50 variables.

Query time

The query time is the actual time it takes the device to run all requests in a queue. If the queue is short, the device will perform all the request-response cycles and wait for the specified **Query cycle** to expire. If the queue is long and the query takes longer than the specified **Query cycle**, the device will poll all slaves in the shortest possible time.

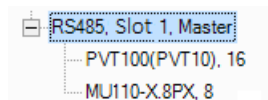


To minimize the request time, the following is recommended:

- If one or several slaves are not connected or temporarily unavailable, consider to block the polling in a program or to minimize the **Timeout** parameter for these devices.
- Consider the number of slaves and the total number of requests when setting the **Query cycle** parameter. If the processing time of all requests (query time) takes longer than **Query cycle**, the parameter will be ignored.

Polling of multiple devices in the network

Slaves are polled according to the generated queue from the smallest to the largest address. In the following example, the slave with the address 8 is polled first, while the one with address 32 is polled last.



Query cycle can be set for each slave individually.

Standard Modbus Error Codes

- **01** — the received function code could not be processed;
- **02** — the data address specified in the request is not available;
- **03** — the value in the request data field is not a valid value.

Functions and data areas

When requested, the Master accesses one of the Slave's **memory areas** using a **function**. A **memory area** is characterized by the type of values stored in it (bits/registers) and the type of access (read-only/read-and-write).

Table 5.1 Modbus protocol data areas

Data area	Designation	Data type	Access type
Coils	0x	BOOL	Read/write
Discrete Inputs	1x	BOOL	Only read
Input Registers	3x	INT	Only read
Holding Registers	4x	INT	Read/write

Each memory area consists of a certain number of cells (depending on the specific device). Each cell has a unique address. For configurable devices, the manufacturer provides a **register map**, which contains information on the correspondence between the device parameters and their addresses. For programmable devices, the user creates such a map independently using the programming environment. There are devices that combine both of the above cases - their register map has a fixed part and a part that the user can supplement in accordance with their task. In some devices, memory areas overlap (for example, **0x** and **4x**) - i.e. the user will be able to access the same registers with different functions.

Function defines the operation (read/write) and the memory area with which this operation will be performed.

Table 5.2 Basic functions of the Modbus protocol

Function code	Function name	Command to execute
1 (0x01)	Read Coil Status	Reading values from multiple flag registers
2 (0x02)	Read Discrete Inputs	Reading values from multiple discrete inputs
3 (0x03)	Read Holding Registers	Reading values from multiple storage registers
4 (0x04)	Read Input Registers	Reading values from multiple input registers
5 (0x05)	Force Single Coil	Writing a value to one flag register
6 (0x06)	Preset Single Register	Writing a value to a single storage register
15 (0x0F)	Force Multiple Coils	Writing values to multiple flag registers
16 (0x10)	Preset Multiple Registers	Writing values to multiple storage registers

In different documents, identical designations may have different meanings depending on the context. For example, the prefix **0x** is often used to indicate the hexadecimal number system, so in one case **0x30** may mean "the 30th bit of the **coils** memory area", and in another - "address 30 in the hexadecimal (HEX) number system" (and this address can refer to any memory area).

Slave polling can be **single** or **group**. During single **polling**, the Master reads each Slave parameter with a separate command.

With group **polling**, the Master reads several parameters at once with one command, whose addresses in the register map are located strictly sequentially and have no gaps. Group polling allows you to reduce network traffic and the time spent polling the device, but in some cases its use is impossible (or possible with restrictions) due to the individual characteristics of the device.

Register order and byte order

The order of registers/bytes is important when reading REAL format system variables (analog inputs/outputs) of the device in Slave mode.

The ALP variables have the following features when working via Modbus:

- INT (uint) - occupies one register, the register/bit order setting does not affect this type;
- BOOL - occupies one bit, you can specify the register bit number;
- REAL - takes up two registers, the order of bytes and registers is important.

5.3.2.2 Master mode

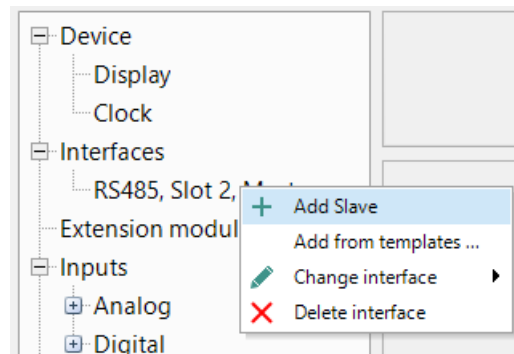
Each interface can control up to 16 slaves. Each slave supports up to 256 variables. The addresses and names of the variables need only be unique if they belong to the same slave.



NOTE

For the interface in Master mode, group polling of slave devices is supported only for second generation devices.

In the master mode, all slaves connected to the interface are sequentially requested. Select the mode **Master** in the parameter list, set other connection parameters and add the required number of slaves using the item **Add slave** in the interface context menu.



The added slave device is displayed with its name and address in the tree below the interface. Select a slave to configure it in the right part of the window. To delete the slave, use the context menu or the icon **Remove Slave**.

Name	Slave		Address	16	
Query cycle (ms)	100		Retries, max.	3	
Time-out (ms)	100				
Status variable	< none > ...		Start query	< none > ...	
<input type="checkbox"/> Change register order			<input checked="" type="checkbox"/> Change byte order		
REAL	2	1	4	3	
<div style="background-color: #4f81bd; height: 15px; width: 100%;"></div>					
Comment					

- **Name** – the name of the slave displayed in the tree
- **Address** – the network address of the slave
- **Query cycle (ms)** – the time interval between queries. A query comprises the number of requests according to the number of variables listed for the slave. The valid range is 0...65535 ms.
- **Timeout (ms)** – the time that request can take before the attempt is considered as failed. The valid range is 0...65535 ms.
- **Retries, max.** – the number of the failed request attempts before query is stopped and the status of the device changes. The valid range is 0...255.
- **Burst request** – group request of consecutive registers to increase the data throughput
- **Status variable** – select a BOOL variable using the icon «...» to record the device status:
 - 1 – the device functions properly
 - 0 – the device is not connected.
- **Start query** – select a BOOL variable using the icon «...» to control the query:
 - 0 – query disabled
 - 1 – query enabled.
- **Change register order** – determines the register order in two-register variables
- **Change byte order** – determines the byte order in the register


**NOTE**



This parameter affects INT variables if the number of registers selected in the poll is 2.

**NOTE**See more in sect. Register order and byte order.


- **Comment** – description text


The list of the variables to be requested from this slave is in the lower part of the window. Each variable created in this list can be found in the variable table under the tab **Network, Slot X** with a separate list of variables for each slave device.

Add a variable by clicking the icon , and set its properties.

+  			
Variable name	Type	Register address	Comment
Var1	BOOL	0	
Var2	BOOL	0	

- **Name** – the name of the variable
- **Type** – the data type of the variable: BOOL, INT or REAL
- **Register** – the register address
- **Bit** – the number of the bit of the register (0...15) (only for BOOL variables)
- **Read function / Write function** – selection of the read / write function or disable reading / writing.
- **Number of registers** – the number of registers occupied by the variable (only for INT variables)
- **Start reading** – assign the BOOL variable for forced reading of the requested variable
- **Start writing** – assign the BOOL variable for forced writing of the requested variable
- **Status variable** – assign the INT variable to record the error code
- **Comment** – description text

To create several variables with the same settings, select a variable and click the icon  **Duplicate**.

 Duplicate variable

Parameters

Name

Var1

Start number

1

Quantity

1

Address step

1

OK

Cancel

- **Name** – the name of the duplicated variable
- **Start number** – the initial number to add to the name of the duplicated variable
- **Quantity** – the quantity of the duplicated variables
- **Address step** – the address increment

Click **OK** to add the duplicated variables to the list of variables. The variables will be stored in adjacent register cells with consecutive addresses.

To remove the variable from the list, use the icon  **Delete**.

Master mode for the second generation devices

To configure parameters for polling connected devices, select the **Modbus Master** node in the device parameter tree.

Modbus Master parameters:

- **Interval between requests, ms** - the time period after which the survey is repeated. The valid range is from 1 to 10,000 ms.

**NOTE**

The maximum number of devices on one interface is 32.

To change the parameters of the device being polled, click on its name in the settings tree. The right part of the window will display the available parameters: in the upper part – device parameters, in the lower part – network variables of the device.

Parameters of the device being polled:

- **Name** — device name to be displayed in the settings tree
- **Interface** — interface through which the device being polled is connected. The list of available parameters depends on the selected interface.
- **Address** — device network address
- **Number of re-request** — number of unsuccessful polling attempts. Valid range is from 0 to 3
- **Response timeout, ms** — the time after which a polling attempt is considered unsuccessful. Valid range is from 10 to 10,000 ms
- **Group queries** - query more than one variable;

**NOTE**

Group polling allows you to read several parameters at once with one command. The parameter addresses in the register map must be located in strict sequence and without gaps, the function code and the polling period must match. If one of the conditions is violated, the next group poll is automatically generated.

- **Number of registers in the request** — from 2 to 16;
- **Byte order** — determines the order of bytes in the packet
- **Comment** — text description of the device

Specific parameters of the polled device connected via the Ethernet interface:

- **IP address** — unique network address of the device, valid range from 0.0.0.0 to 255.255.255.255
- **Port** — port number, valid range from 0 to 65535.

Properties of network variables of the polled device:

- **Name** — name to display in variable table
- **Type** — type of the variable: boolean, integer or floating point
- **Register** — the value of the register accessed by the device is displayed in the table
- **Bit (Boolean variables only)** — bit number to read
- **Number of registers (integer variables only)** — number of registers occupied by a variable: 1 or 2
- **Comment** — text description of the variable to be displayed in variable table
- **Function** — disable or select the write/read function.

**NOTE**

Creating variables with the same names is not allowed.

The list of parameters to configure depends on the choice of the write/read function.

Reading function parameters:

- **Reading period** is a time interval between requests;
- **Read command** is a Boolean type variable, changing which causes the parameter to be read.

Writing function parameters:

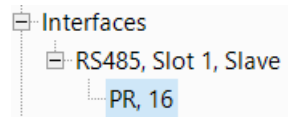
- **Writing period** is a time interval between rewrite operations;
- **Write command** is a Boolean type variable, the change of which causes the parameter to be written.

- **Write on change** – if this function is activated, then the master writes the variable to the slave device when its value changes.

5.3.2.3 Slave mode

An RS485 interface added to the tree item **Interfaces** has the default mode Slave and the default master with the name PR and the address 16 added below. Select the interface to set the connection parameters.

To configure the data transfer parameters, click on the device name (PR, 16 by default) in the tree.



Select the master in the tree to set the parameters for data exchange.

Name	PR		Address	16	
	<input type="checkbox"/> Change register order		<input checked="" type="checkbox"/> Change byte order		
REAL	2	1	4	3	
Comment					

The common parameters for data exchange can be set in the upper window part.

- **Name** – name of the master displayed in the tree
- **Address** – network address of the master
- **Change register order** – changes register order in two-register variables
- **Change byte order** – changes byte order in the register
- **Comment** – description text




In Slave mode, the High Register Forward and High Byte Forward settings do not affect the operation of the device. Data is transmitted in the following format:

- High register forward —no;
- High byte forward — yes.


For a device in Slave mode, polling of up to 64 registers is allowed during group polling.

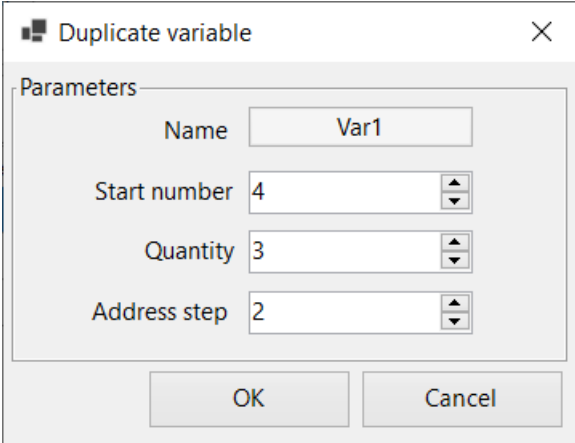
The list of the variables to be requested by the master is in the lower part of the window. Each variable created in this list can be found in the variable table under the tab **Network, Slot X**.

Add a variable by clicking the icon  **New variable** and set its properties.

+   			
Variable name	Type	Register address	Comment
Var1	INT	512	
Var2	INT	513	
Var3	INT	514	

- **Name** – the name of the variable
- **Type** – the data type of the variable: BOOL, INT or REAL
- **Register** – the register address. The range of the available addresses is specified in the device user guide.
- **Comment** – description text

To create several variables with the same settings, select a variable and click the icon  **Duplicate**.




The 'Duplicate variable' dialog box contains the following fields:

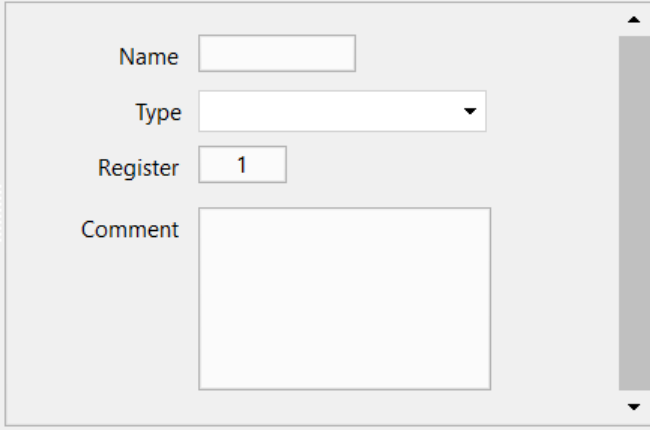
- Name:** Text box containing 'Var1'.
- Start number:** Spin box with value 4.
- Quantity:** Spin box with value 3.
- Address step:** Spin box with value 2.
- Buttons:** 'OK' and 'Cancel' buttons at the bottom.

- **Name** – name of the duplicated variable
- **Start number** – initial number added to the name of the duplicated variable
- **Quantity** – quantity of the duplicated variables
- **Address step** – address increment

Click **OK** to add duplicated variables to the list of variables. The variables will be stored in adjacent register cells with consecutive addresses.

To remove the variable from the list, use the icon  **Delete**.

Device variable properties



The 'Device variable properties' dialog box contains the following fields:

- Name:** Text box.
- Type:** Dropdown menu.
- Register:** Text box containing '1'.
- Comment:** Large text area.

The device variable properties are configured to the right of the table:

- **Name** – Name of the device variable (set by the user)
- **Type** – type of the device variable: integer or floating-point



NOTE

Boolean variables can be read into an integer variable. The state of a discrete input can be extracted using the EXTRACT block or a suitable macro from the Component manager.

- **Register** – register address of the device variable (set by the user). The range of available addresses is specified in the device's *user manual*.
- **Comment** – a textual description of the variable's value in the variable table.

Slave mode for the second generation devices

Second-generation devices operate in Slave mode by default. Device variables are configured through the variable table on the **Network, Slave** tab.

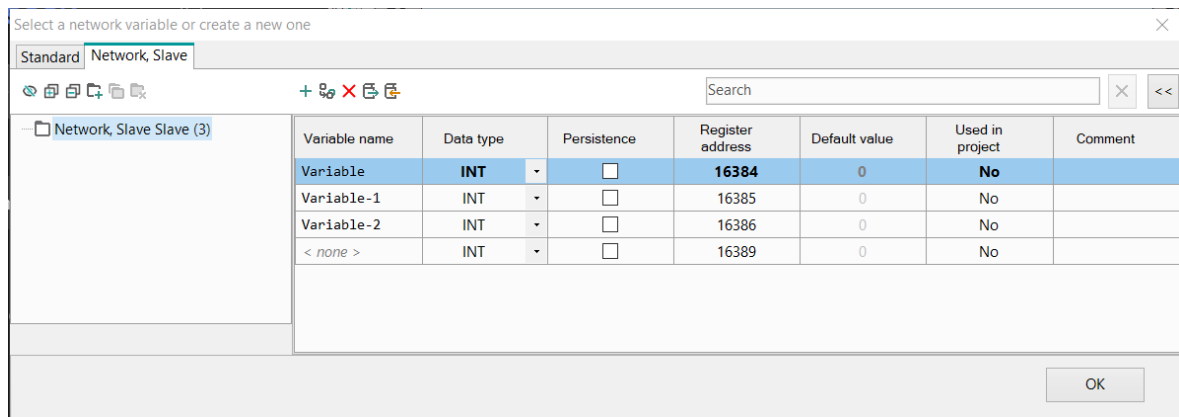


Fig. 5.1 Variable table

- **Variable name** – name of the device variable (set by the user)
- **Data type** – integer or floating-point

**NOTE**

Boolean variables can be read into an integer variable. The state of a discrete input can be extracted using the EXTRACT block or a suitable macro from the Component Manager.

- **Persistence** – when the checkbox is enabled, the variable is stored in the non-volatile system EEPROM area. Information about variable persistence is saved with the project.
- **Register address** – address of network variables in the range from 16,384 to 20,479
- **Default value** – variable value that will be initialized when the program starts
- **Used in project** – indicates if the variable is linked to blocks in the program. If the variable is bound, the value is **Yes**.
- **Comment** – a textual description of the variable for display in the variable table

After connecting the device to akYtec ToolPro or akYtec Cloud, the network variables will be displayed in the parameter tree.



To configure the display, select a variable in the table and click the **Parameter Settings** button in the upper right corner of the window. The **Parameter Configuration** panel will appear.

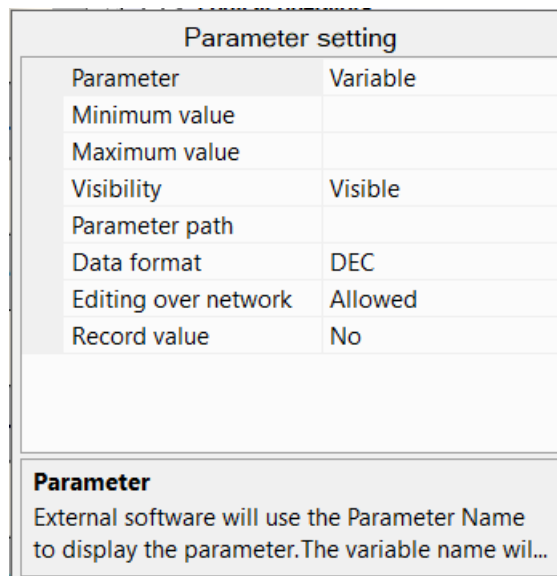


Fig. 5.2 Parameter setting

- **Parameter** – parameter name that will be displayed in the parameter tree when working with akYtec ToolPro and akYtec Cloud.

- **Minimum value** – minimum parameter value for display in akYtec ToolPro and akYtec Cloud, does not affect program execution logic.
- **Maximum value** – maximum parameter value for display in akYtec ToolPro and akYtec Cloud, does not affect program execution logic;
- **Visibility** – parameter visibility when working with akYtec ToolPro and akYtec Cloud: **Visible** – parameter is displayed in the parameter tree, **Hidden** – the parameter is not displayed in the parameter tree;
- **Parameter path** – parameter will be displayed when the device works with external software. Directory nesting is added using a backslash – \.
- **Data format** – format in which the parameter value will be displayed; relevant for integer variables; available formats: **DEC** (decimal), **BIN** (binary, displayed in akYtec ToolPro as a bit mask), **hex** (hexadecimal), and **enum** (enumeration, for uint16 only).
- **Editing over network** – ability to edit the parameter value from akYtec ToolPro and akYtec Cloud.
- **Record value** – whether parameter value needs to be archived: **Yes** – the parameter value is recorded in the device archive, **No** – the parameter value is not recorded in the device archive.

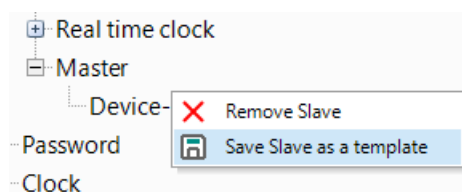
5.3.2.4 Network device templates

Network Device Templates

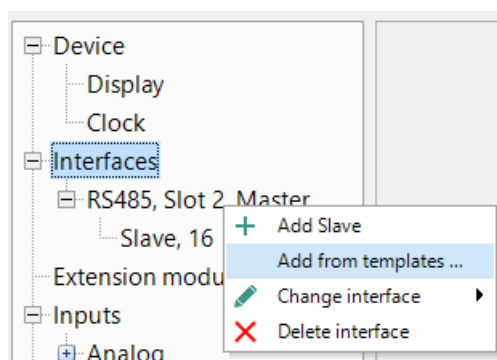
For interfaces operating in Master mode, the creation and use of network device templates is available. The parameters and variables of a configured device can be saved as a template file.

Creating a Template

To create a template, select **Save Slave as a template** in the device's context menu or in the upper part of the device configuration window.



In the opened file explorer window, select the location and enter the file name. The file will be saved with the *.dvtpe extension. The saved template can be used for other interfaces and projects. To use a saved template, right-click on the interface name in the configuration tree and select **Add from templates...** In the opened file explorer window, select the file containing the template.



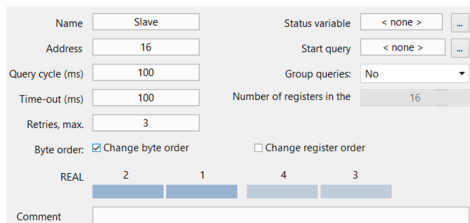
The device from the template will be added to the interface along with all its parameters.

For a number of akYtec devices, there are ready-made templates that can be downloaded from the online database using the Component Manager.

When using network device templates created in projects for first-generation devices in projects with second-generation devices:

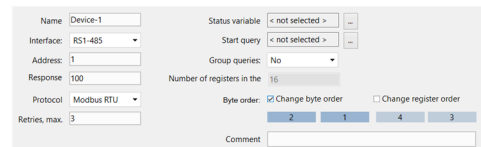
- When **group polling** is selected, the number of registers in the request will either retain its value (if the entered value is from 2 to 16), or will be set to 16 if the value stored in the template is greater than 16:

Slave parameters
for first-generation devices



Form for Slave parameters for first-generation devices. Fields include: Name (Slave), Address (16), Query cycle (ms) (100), Time-out (ms) (100), Retries, max. (3), Status variable (< none >), Start query (< none >), Group queries (No), Number of registers in the (16), Byte order (Change byte order checked), and Comment.

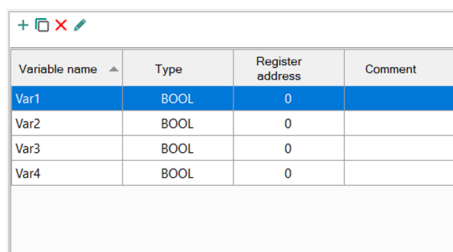
Slave parameters
for second-generation devices



Form for Slave parameters for second-generation devices. Fields include: Name (Device-1), Interface (RS1-485), Address (1), Response (100), Protocol (Modbus RTU), Retries, max. (3), Status variable (= not selected >), Start query (= not selected >), Group queries (No), Number of registers in the (16), Byte order (Change byte order checked), and Comment.

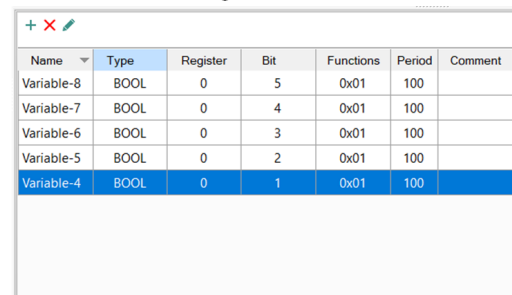
- A variable with read and write function is split into two variables: a read function `_read` and a write function `_write`:

Variable table of Slave
for first-generation devices



Variable name	Type	Register address	Comment
Var1	BOOL	0	
Var2	BOOL	0	
Var3	BOOL	0	
Var4	BOOL	0	


Variable table of Slave
for second-generation devices

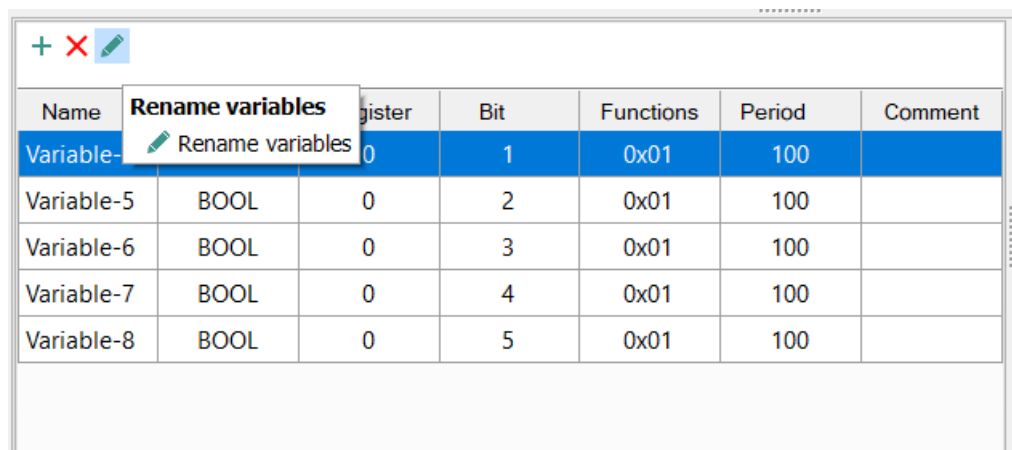


Name	Type	Register	Bit	Functions	Period	Comment
Variable-8	BOOL	0	5	0x01	100	
Variable-7	BOOL	0	4	0x01	100	
Variable-6	BOOL	0	3	0x01	100	
Variable-5	BOOL	0	2	0x01	100	
Variable-4	BOOL	0	1	0x01	100	

When using network device templates created in projects for second-generation devices in projects with first-generation devices, the polling period parameter will revert to its default value (100 ms).

Group variable renaming

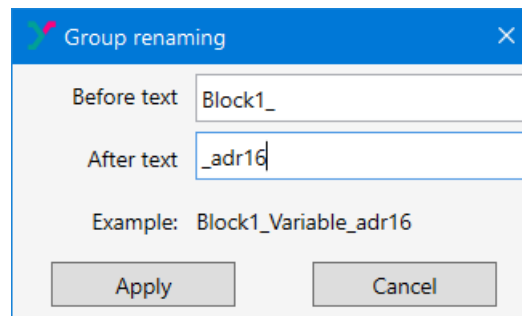
To batch rename variables in the variable table of the Slave device in the device configuration window, click the  button.



Name	Type	Register	Bit	Functions	Period	Comment
Variable-4	BOOL	0	1	0x01	100	
Variable-5	BOOL	0	2	0x01	100	
Variable-6	BOOL	0	3	0x01	100	
Variable-7	BOOL	0	4	0x01	100	
Variable-8	BOOL	0	5	0x01	100	

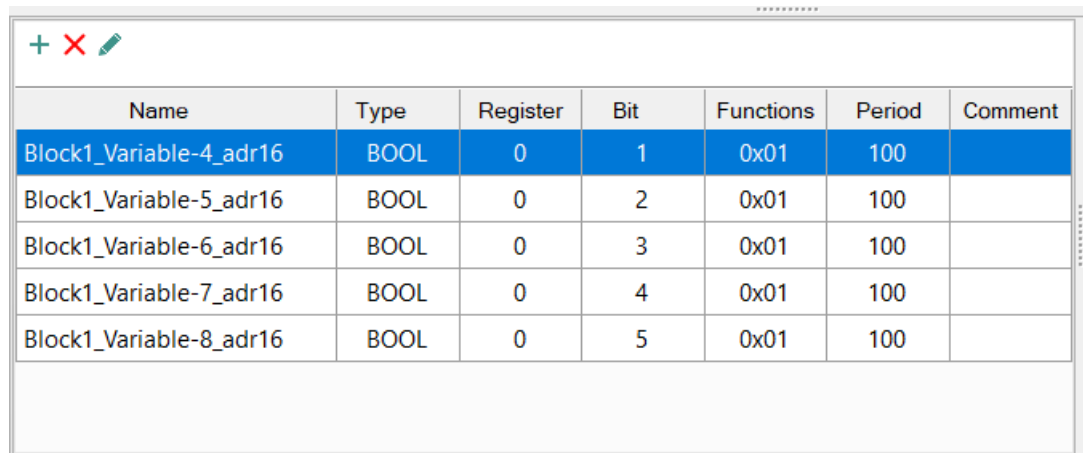
A tooltip 'Rename variables' with a pencil icon is shown over the first row.

In the window that appears, enter the text to display before and after the variable name, and click the **Apply** button.



A dialog box titled "Group renaming" with a close button (X) in the top right corner. It contains two text input fields: "Before text" with the value "Block1_" and "After text" with the value "_adr16". Below these fields is an "Example:" label followed by the text "Block1_Variable_adr16". At the bottom are two buttons: "Apply" and "Cancel".

As a result, all variables will be renamed according to the entered data:



A table with 7 columns: Name, Type, Register, Bit, Functions, Period, and Comment. The table contains 5 rows of data. The first row is highlighted in blue. Above the table are three icons: a plus sign, a red X, and a pencil. To the right of the table is a vertical scrollbar.

Name	Type	Register	Bit	Functions	Period	Comment
Block1_Variable-4_adr16	BOOL	0	1	0x01	100	
Block1_Variable-5_adr16	BOOL	0	2	0x01	100	
Block1_Variable-6_adr16	BOOL	0	3	0x01	100	
Block1_Variable-7_adr16	BOOL	0	4	0x01	100	
Block1_Variable-8_adr16	BOOL	0	5	0x01	100	

When replicating a variable that was renamed using batch renaming, the "Text after" will not change; the initial and subsequent numbers will be added to the name of the replicated variable.

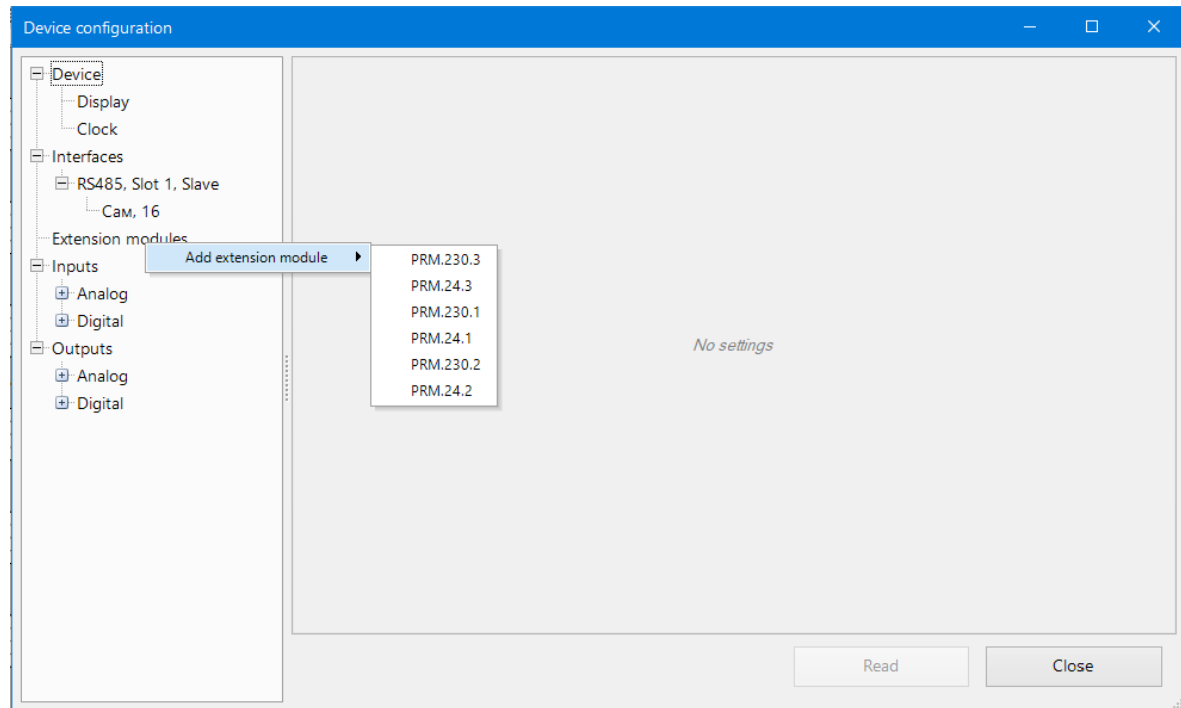
The batch rename settings are saved in the template and are available for editing when the template is subsequently used.

Renaming each variable individually is available in the [variable table](#) and in the [program code](#).

5.4 Extension modules

Up to two I/O extension modules of type PRM can be connected to base device. For further information about extension modules refer to the PRM user guide.

To use module I/O points in the circuit program, add the module to the group **Extension modules** using its context menu.



The additional I/O points of the added modules can be configured in branches **Inputs** and **Outputs** respectively. They are displayed in the tree as **Ix(y)** and **Qx(y)** respectively, where **x** is the ordinal number of the I/O point on the module and **y** is the ordinal number of the module counting from the base device.

Before uploading the project to the base device, all modules must be connected via the internal bus to base device and powered on. The module firmware is synchronized with the current version of ALP when uploading a project.

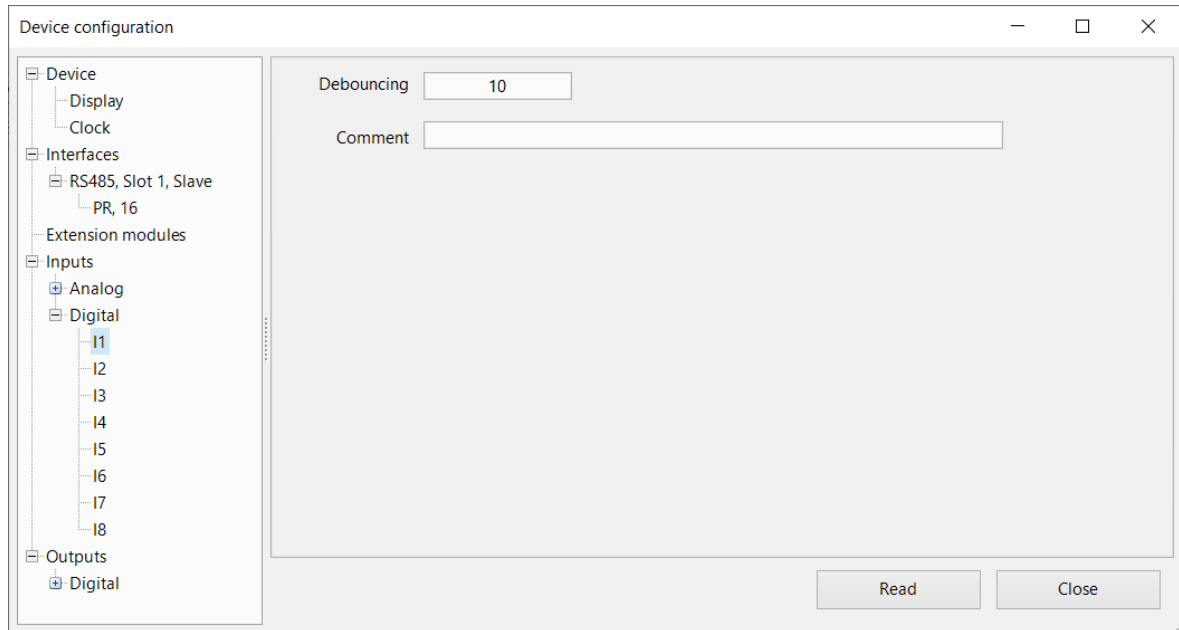
5.5 Inputs and outputs

Inputs

The content of the branch **Inputs** depends on the resources of the target device. It can be analog and/or digital inputs.

The parameter **Comment** is common for all types of inputs. The text in this field is displayed in a tooltip, when the mouse cursor is over the input in the workspace. The text can be entered in Property Box too.

For further details about the configuration of the inputs, refer to the device user guide.



Other input parameters depend on the types of the input and the device.

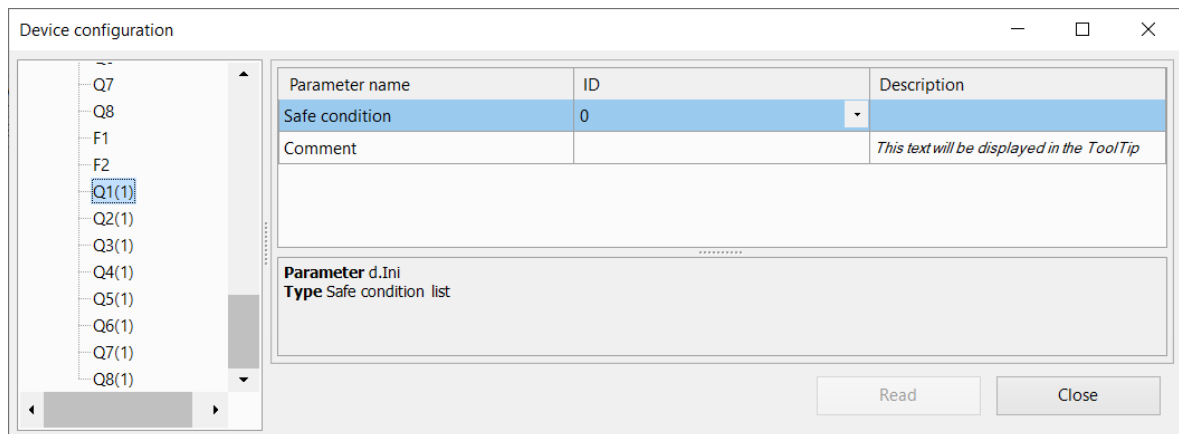
Outputs

The content of the branch **Outputs** depends on the resources of the target device. It can be analog and/or digital outputs.

The parameter **Comment** is common for all types of outputs. The text in this field is displayed in a tooltip, when the mouse cursor is over the output in the workspace. It can be entered in Property Box too.

For further details about the configuration of the outputs, refer to the device user guide.

The digital outputs of the extension module have an additional parameter **Safe condition**. The parameter specifies the output state in case the connection between the module and the base device is lost.



Settings for devices on the new hardware platform

The window for setting up inputs and outputs for devices on the new platform has a different interface, and the parameters on the right side of the window are presented in a table.

For devices on the new platform, the menu for setting the safe state of the outputs is located in the branch of the added expansion module.

5.6 Password

For devices on the new platform, you can set a password to protect the device.

The password is set in the **Password** section in the settings tree only for the device connected to PC.

Create password

If a password is not set in the device, then password creation will be active in the settings window. Enter your password and confirm to continue.

Changing and resetting password

If the device already has set a password, you can change or reset it.

To change the password, enter current password and new password in the **Change password** columns.

To reset your password, enter your current password in the **Reset password** column.

If you have lost your password, see the device user guide in order to reset it.

For a password-protected device, a password is required when transferring a program, see [Upload project to device](#).

5.7 Connecting to akYtec Cloud

akytec Cloud connectivity is available for devices on the new platform (i.e. PR103).



NOTE

For a list of devices on the new platform, see [About](#).



NOTE

To connect to akYtec Cloud, the device must be password protected. Without the password, the device will not connect to akytec Cloud. For setting the password, see [Password](#).

Before setting up connection to akYtec Cloud, you should set up authorization of the device in akYtec Cloud, for details see [akYtec Cloud User Manual](#) and device *User Manual*. During operation, the device must be connected to the Internet for data transfer.

In the **akYtec Cloud Connection Settings** section of the Settings tree, you can set the connection and akYtec Cloud connection status.

Name	ID	Variable
IP address	10.2.11.122	< none >
Subnet mask	255.255.0.0	< none >
Gateway	10.2.1.1	< none >
New IP address	10.2.11.122	< none >
New subnet mask	255.255.0.0	< none >
New gateway	10.2.1.1	< none >
DHCP	Service button	< none >
DNS server 1	0.0.0.0	< none >
DNS server 2	0.0.0.0	< none >
Connection status	no connection	< none >
Cloud connection	Off	< none >

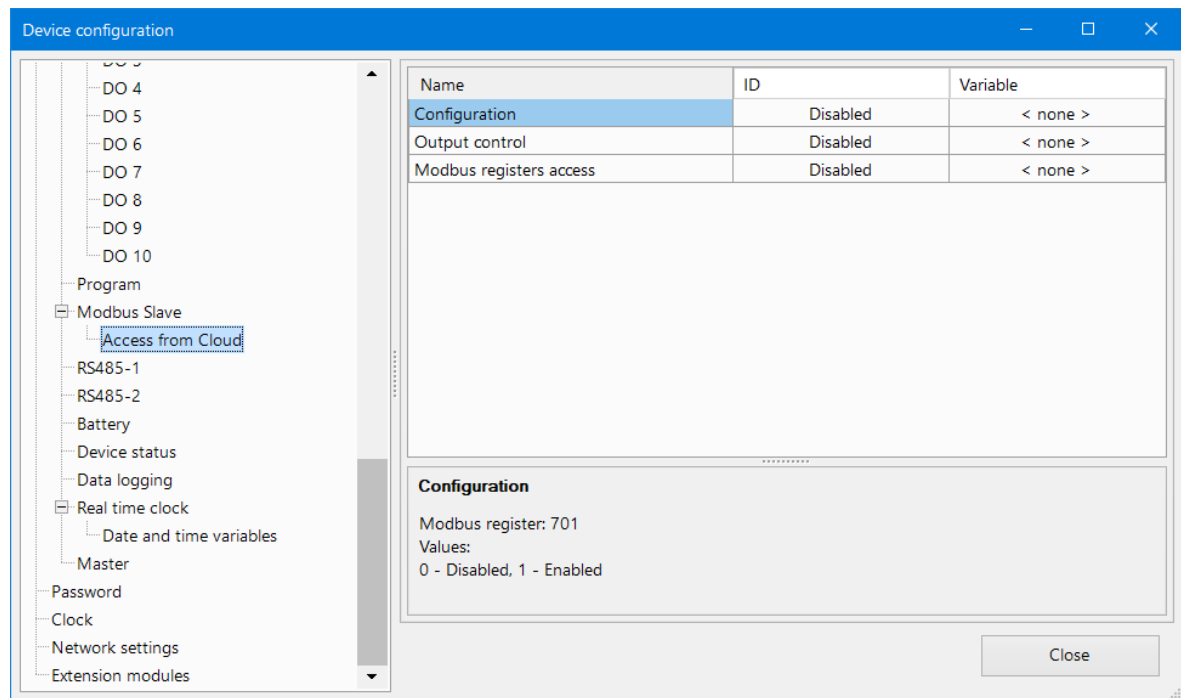
Cloud connection

Modbus register: 35
Values:
0 - Off, 1 - On

When saving the settings, the device will be rebooted.

Read Save Close

In the **Access from Cloud** section of the configuration tree, you can set the akYtec Cloud access levels to the device.



For an example of working with akYtec Cloud, see [section Task 3. Direct connection of PR103 to akYtec Cloud.](#)

6 Variables

Variables are used to write and read values in the project diagram and also for the purpose of programming **screens**.

To use a variable in a project, you must first create it in the variable table.

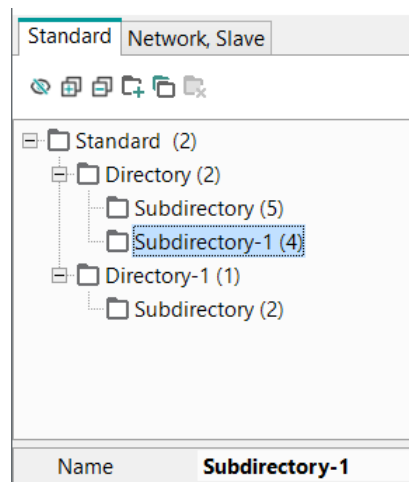
To see all project variables, click the icon  in the toolbar or use the menu item **Device** → **Variable table**.







The variables are divided into three groups, each of which has a separate tab in the table:

- **Standard**
- Service
- Network

Variable Directory Tree

Allows you to create directory sections for variable placement.



	Enable/disable the display of nested variables
	Expand nested sections
	Collapse nested sections
	Create a variable catalog section. Creating a section is also available through the catalog's context menu.
	Duplicate catalog — the selected folder will be duplicated in the parent catalog with all nested sections and added variables. Copying a section is also available through the catalog's context menu. Select Copy , then select the catalog where you want to add the copied section, right-click and select Paste NOTE The target catalog for adding information must be one level above the copied section.
	Delete catalog. Deleting a catalog section is also available through the catalog's context menu. NOTE A catalog section containing variables cannot be deleted.

To move directory sections, press and hold the **Shift** key, and use the mouse to move the section to a new location.

**NOTE**

Moving sections is only available within the parent directory.

To rename a directory section, enter a new name on the **Name** line at the bottom of the **Variable Directory Tree**.

**NOTE**

The directory section name field cannot be empty or duplicate the name of another section within the parent directory. If an incorrect name is entered, an error message box will appear.

To determine the directory section in which a variable resides, highlight the variable in the table and press **Ctrl + H**. The section where the variable is located highlights in blue in the directory tree. The line below shows name of the section, and the section with the variable opens in the variable table. You may also hover your mouse over a variable to summon a pop-up window with the variable's name and location.

Variables

The project variables are displayed in the variable table.

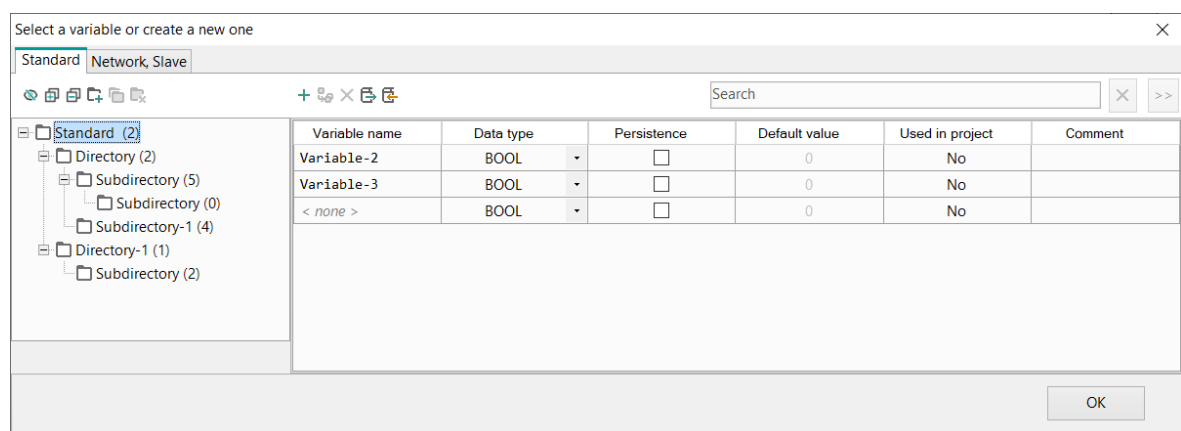



Fig. 6.1 Variable selection and creation window

To create a variable, enter a name and set the [type 6.1](#) or click the  button at the top left of the table. The other parameters are filled in according to the operational needs:

- **Variable name** - name to be displayed on the Variables panel and on the variable block in the project;
- **Variable type** - one of three types can be assigned: Boolean, integer, floating point. See the [Data Types 6.1](#) section for more details;
- **Persistence** – enable the checkbox in the setting field to save the variable value in the device ROM in case of power failure. The storage time of the variable in ROM depends on the device type, see device *operation manual*


**NOTE**

For devices on the new platform, when the Persistence checkbox is enabled in the variable table, the variable value is saved:

- To the system EEPROM for a standard variable bound to an device parameter or visualization element. For the network Slave variable.
- To the EEPROM for a standard variable not bound to the device parameter or to the visualization element.

When you save a project, the information about the non-volatile variables is also saved.

- **Default value** – the value that the variable will keep until a new value is written to it;
- **Used in the project** – indication of binding to blocks in the program. If the variable is bound, the value is **Yes**;
- **Comment** – text description to be displayed in the tooltip on the project diagram when the cursor is placed over the variable block.

To remove a variable from the table, right-click on the variable row and select **Delete Variable** in the context menu or click the  button at the top of the table.

To duplicate a variable, right-click on the variable line and select the **Duplicate a variable** from the

context menu or click the  button at the top of the table.

The variable table supports searching and filtering by name. The **Space** symbol acts as a *logical OR* and allows you to filter variables by several criteria.

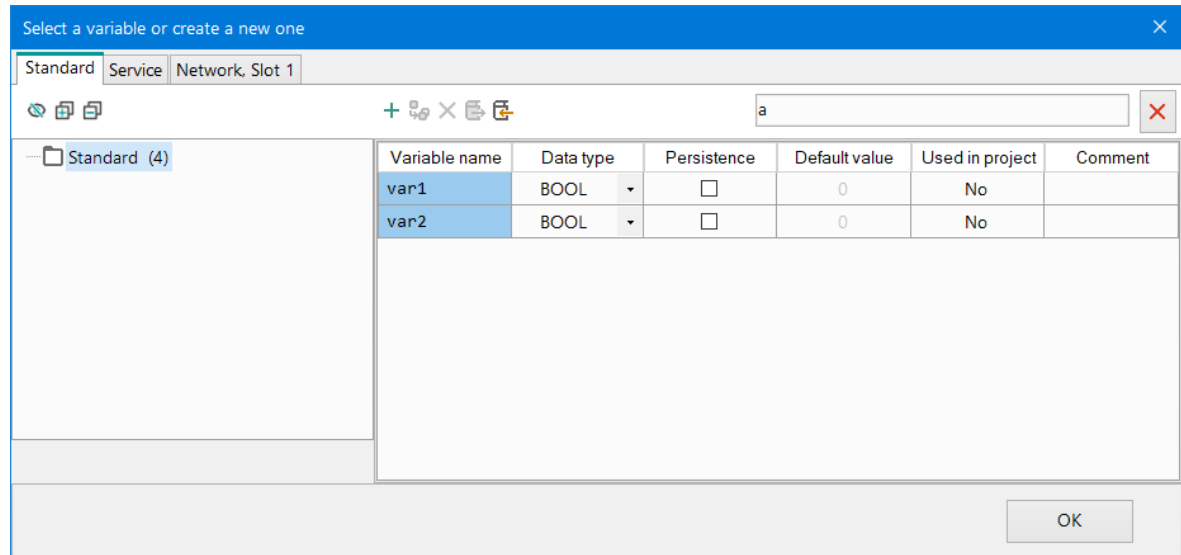



Fig. 6.2 Search for variables

If the variable entered in the search line does not exist, you can create it by clicking the  button at the top of the table. The variable is automatically assigned the name entered in the search box.

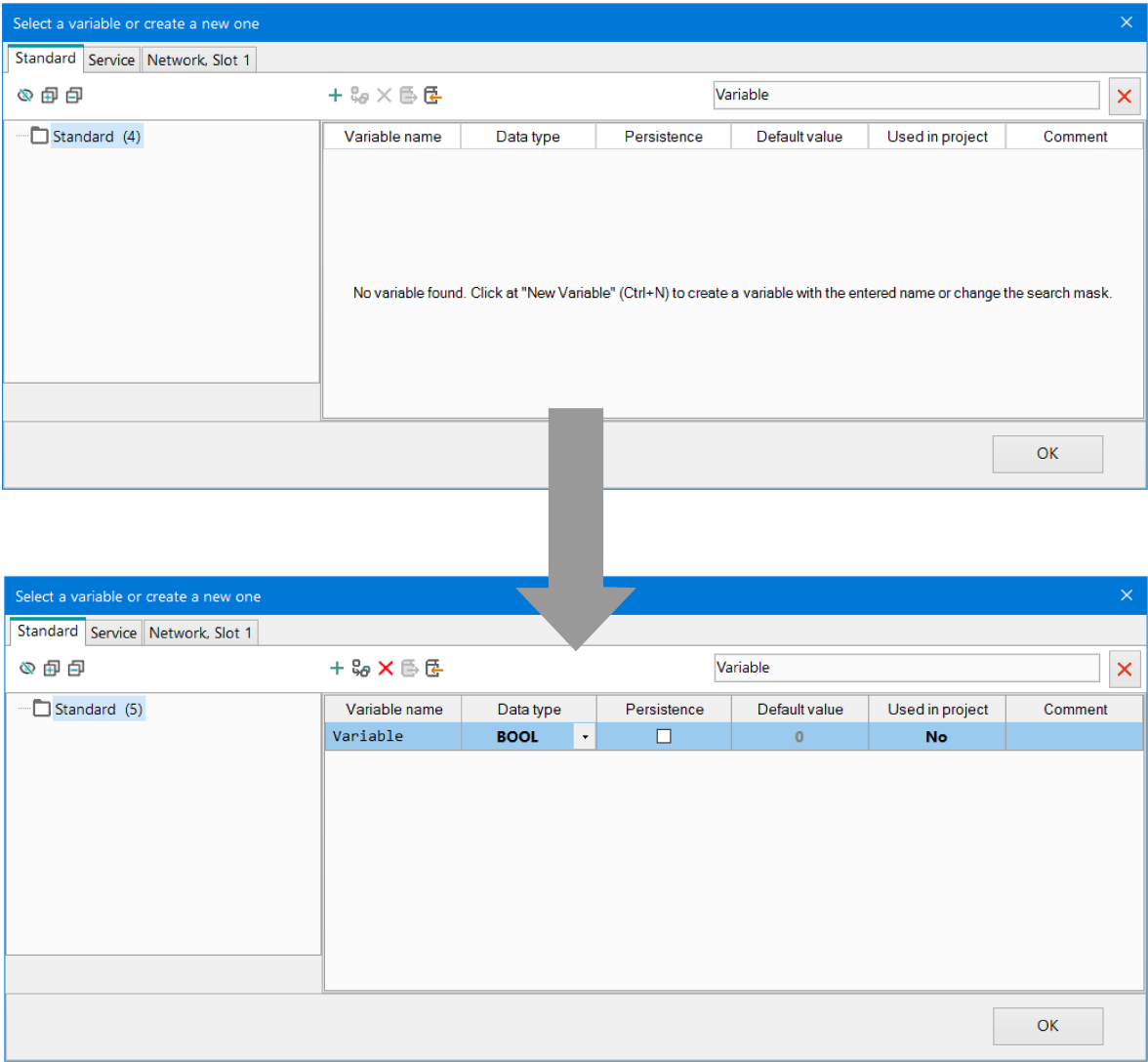


Fig. 6.3 Creating a variable

The variable table allows sorting by value. To sort values, right-click the table column name and select the sorting order (ascending or descending) in the context menu.

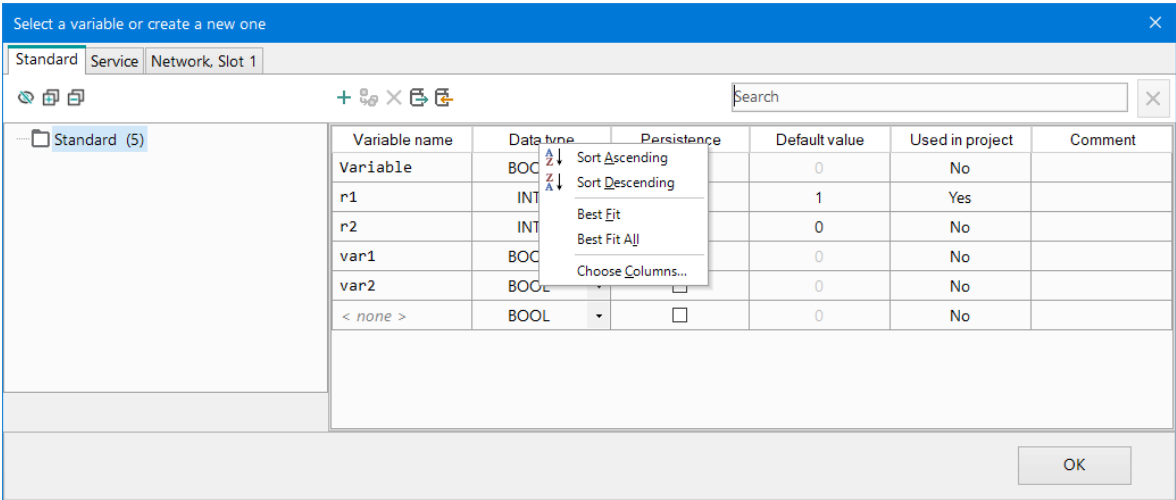
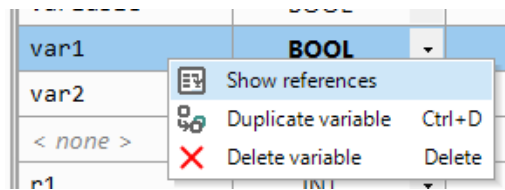


Fig. 6.4 Sorting variables

To find the areas where the variable is used in the project, right-click on the variable row in the table and select **Show References**.



The new window shows information about the selected variable.

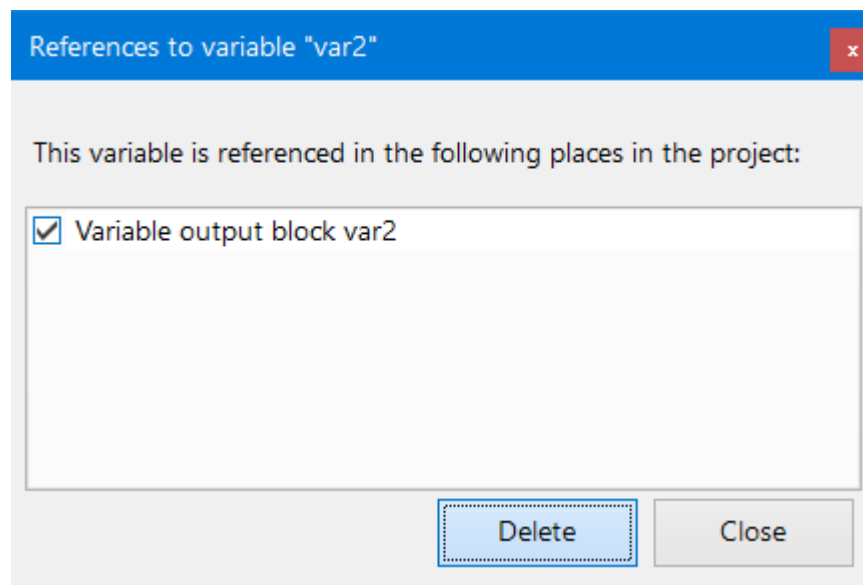


Fig. 6.5 References to a variable

To delete a bind to a variable block in the project, check the necessary variables and press the **Delete** button. The variable will remain in the table, but will not be used in the project.

Exporting variables to a file

The variable tab can be exported as a table in **.csv** format. Proceed as follows:

1. Click **Export variables tab to CSV file** in the upper left part of the table.
2. In the window that opens, specify the location for uploading the file.
3. Press the **Save** button.



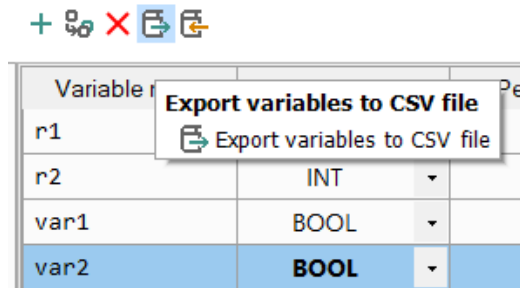
NOTE

The file name is formed depending on the exported tab according to the template **ProjectName_Tab_Variables..**




NOTE

For devices on the new platform, the Slave tab of network variables is exported along with the variables panel.



Importing variables from a file

To import a variable table from a file in **.csv** format, click the  **Import Variables from CSV File** button at the top of the table. Select the file from which to import the data and click **Open**. Directories, if present, will also be imported from the file with the nesting of variables preserved. If the file contains incorrect data, for example, if a column for the name, variable type, or register is missing, an error message will be displayed. After correcting the error, perform the import again. If variable names and/or registers in the project match those in the imported file, an information window will appear with a message about all name and/or register conflicts and an option to continue or cancel the import.

6.1 Data types

The variables of the following types can be used in a program:

- Boolean (**BOOL**)
- Integer (**INT**)
- Real (**REAL**).

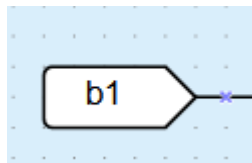


NOTE

Different devices can have restrictions related to support of certain types of variables.

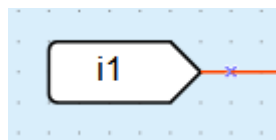
BOOL

A variable of this type has only two possible values: 1 (**True**) or 0 (**False**).
The connecting lines between the BOOL variables in the circuit program are gray.



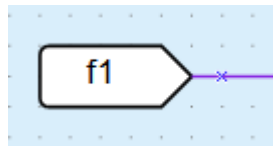
INT

A variable of this type is an unsigned integer in the range of 0...4,294,967,295 (4 Byte).
The connecting lines between the INT variables in the circuit program are red.



REAL

A variable of this type has a value in the range of -3.402823e+38...3.402823e+38. It is represented by a floating-point number of single-precision (4 Byte).
The connecting lines between the REAL variables in the circuit program are violet.

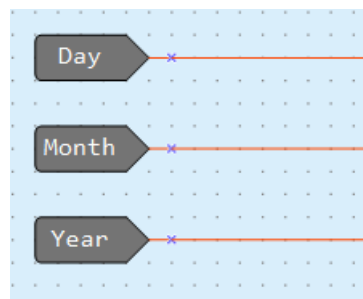


6.2 Service variables

Service variables are associated with the device settings and can differ, depending on the device. Service variables are related to hardware features, such as the real-time clock, interface card in the slot, etc., and cannot be deleted. Access rights to service variables may be limited. The service variables are listed in the variable table under the tab **Service**.

Select variable		Standard Service Network, Slot 1
Name	Data type	
<input type="checkbox"/> Real-time clock		
Seconds	INT	
Minutes	INT	
Hours	INT	
Day	INT	
Month	INT	
Year	INT	
		OK

The blocks of service variables are shown in the circuit program with a gray background.



6.3 Network variables

Each interface slot has a separate tab in the table.

If the interface is configured as a master, there are separate tabs for each slave device within the slot tab. The Slave tab contains the variables to be requested for this slave device.

Select a network variable or create a new one

+

Search

Slave, 16 Slave, 16

Variable name	Data type	Read function	Write function	Register address	Bit number	Comment
Var2	BOOL	0x01	0x05	0	1	
Var1	BOOL	0x01	0x05	0	0	
Var3	BOOL	0x01	0x05	0	2	
< none >	BOOL	0x01	0x05	0	3	

OK

Network variables and their references are deleted in the same way as standard variables. For more details about network variables for master interface see [Master mode 5.3.2.2](#) section. If the interface is configured as a Slave, all network variables to be requested by the master are shown in one list.

Select a network variable or create a new one

+

Search

Variable name	Data type	Register address	Comment
Variable-2	INT	514	
Variable-1	INT	513	
Variable	INT	512	
< none >	INT	515	

OK

6.4 Binding variables to parameters



NOTE

Variable binding to parameters is only available for **second-generation devices**. For a list of second-generation devices, see the [About](#) section.

In second-generation devices, you can bind variables to device parameters and use them in the program. Variables bound to parameters are available for reading and modification.

To bind a variable to a device parameter:

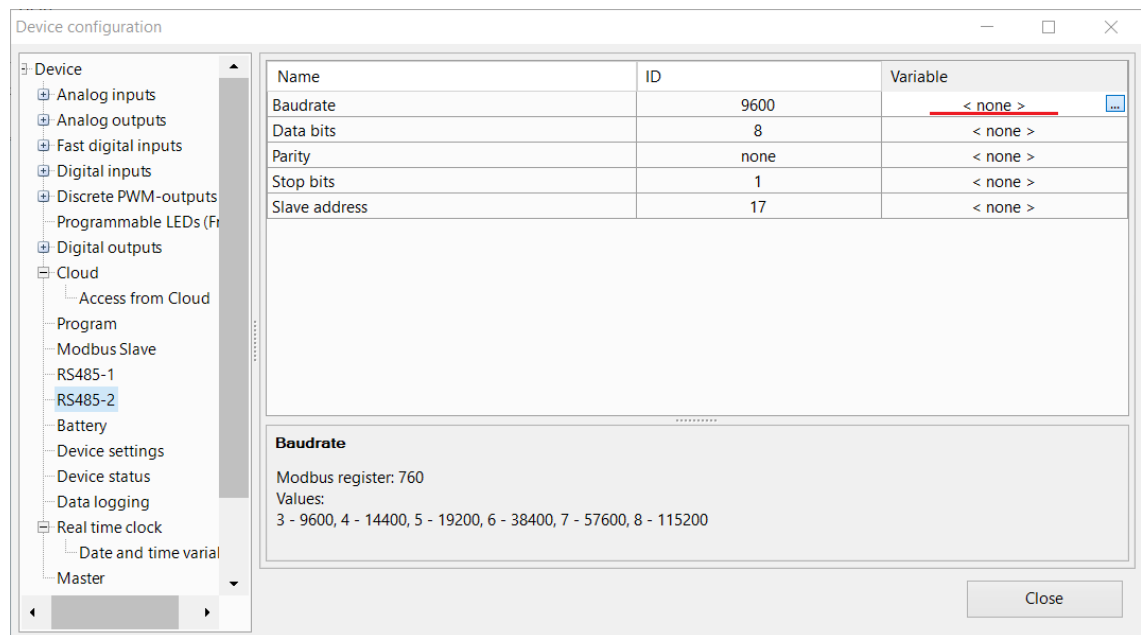
1. Open **Device Settings**.



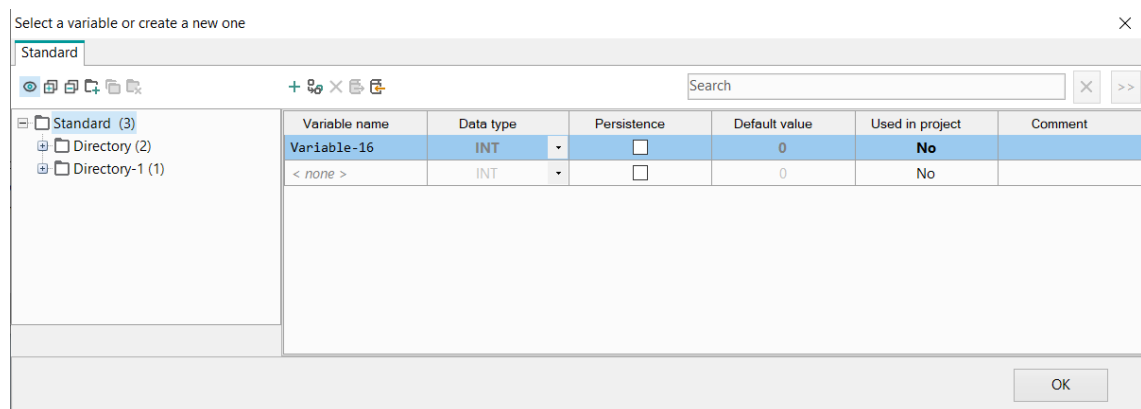
NOTE

Variable binding is not available for all parameters.

2. Select the parameter for binding and click the "..." button in the **Variable** column of the settings window.



3. In the Variables window that opens, select or create a variable for binding. Variables highlighted in gray are read-only.



The name of the bound variable will be displayed in the "Variable" column next to the parameter.



After binding to parameters, variables can be used in the program. This allows parameter values to be read and modified within the program.



NOTE

If a variable bound to a parameter is used for writing via a block, and the parameter value has not changed, the value written through the block will be stored in the variable for one cycle. In the next cycle, the variable value will be read from the parameter.

6.5 Copy-paste variable block

Variables can be copied from project to project for reuse and to reduce development time. To copy a variable, select the variable block in the source project and click the  button on the toolbar or select the **Copy** command from the block's context menu. The variable is pasted into another project by clicking the  button on the toolbar or by selecting the **Paste** command from the canvas context menu.

Variables

When pasting a copied variable block into a project, all variable properties are checked. If the project contains a variable with completely matching properties, it is duplicated into the block. Variables with differing properties are automatically created in the project. If variable names match but other

properties differ, a new variable is created, but the name conflict is displayed in the variable table and requires changing the name of one of the variables.

**NOTE**

Variables of floating-point type cannot be pasted into a project for a device that does not support the floating-point data type.

**NOTE**

Non-volatile variables cannot be pasted into a project for a device that does not support them.

Service variables

Service variables cannot be copied into a project for a device that does not support real-time clocks.

Network variables

Only network variables in Slave mode for an identical slot number can be copied to another project. Network variables in Master mode must be created manually. When pasting a copied network variable block into a project, all variable properties are checked. If the project contains a variable with completely matching properties, it is duplicated into the block. Variables with differing properties are automatically created in the project. If variable names or registers match but other properties differ, a new variable is created, but a register or name conflict occurs. The conflict can be resolved in the variable table by assigning a different name or registers.

7 Library

If a project is open, the panel **Library Box** contains the following libraries:

- Functions
- Function blocks
- Project macros

Select an icon in the lower part of the panel to show the respective content.

Project macros library comprises the macros that have been created, imported or included to the project from Online Database.

View options can be changed using the icons located in the panel toolbar.

7.1 Functions

The library contains the following function groups:

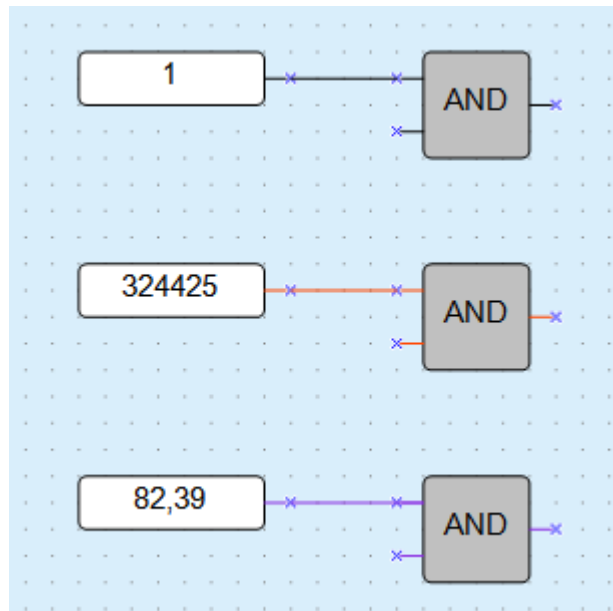
- Logical operators
- Mathematical operators
- Relational operators
- Bitshift operators
- Bit operators

7.1.1 Logical operators

- Conjunction (AND)
- Disjunction (OR)
- Negation (NOT)
- Exclusive OR (XOR)

The logical operators can operate with BOOL or INT variables.

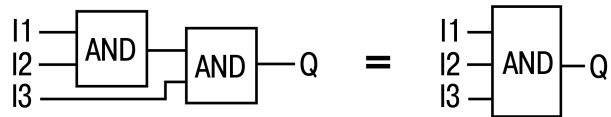
If the input values are INT, the operation is performed bitwise and the output is also an INT.



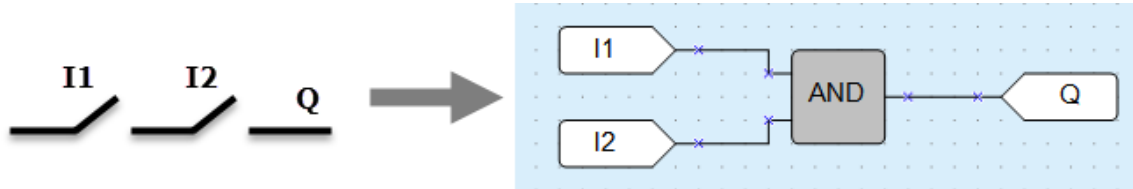
For **AND** and **OR** operators, it should be taken into account that unconnected block inputs will have the following states:

- for **AND** – TRUE
- for **OR** – FALSE

In this case, the blocks act as a signal repeater. To increase the number of inputs for logical operators, their cascade connection is used:



7.1.1.1 Conjunction (AND)



The output **Q** is **True** if both inputs are **True**. The function **AND** represents a serial connection in an electrical circuit.

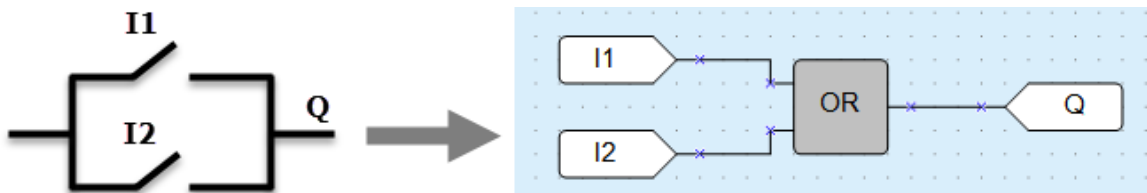
Truth table:

I1	I2	Q
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operation example with integer inputs:

AND	0011 (decimal 3)
	0101 (decimal 5)
	0001 (decimal 1)

7.1.1.2 Disjunction (OR)



The output **Q** is **True** if at least one of the inputs is **True**. The function **OR** represents a parallel connection in an electrical circuit.

Truth table:

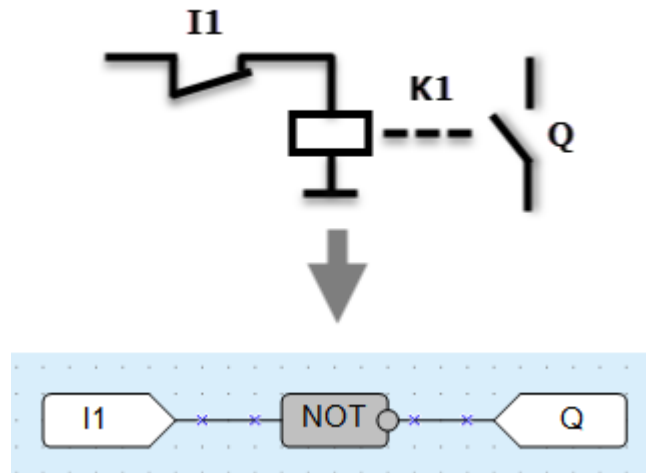
I1	I2	Q
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operation example with integer inputs:

OR	0011
	0101
	0111

7.1.1.3 Negation (NOT)

The function **NOT** inverts the signal. The output **Q** is **True** if the input is **False** and vice versa.



Truth table:

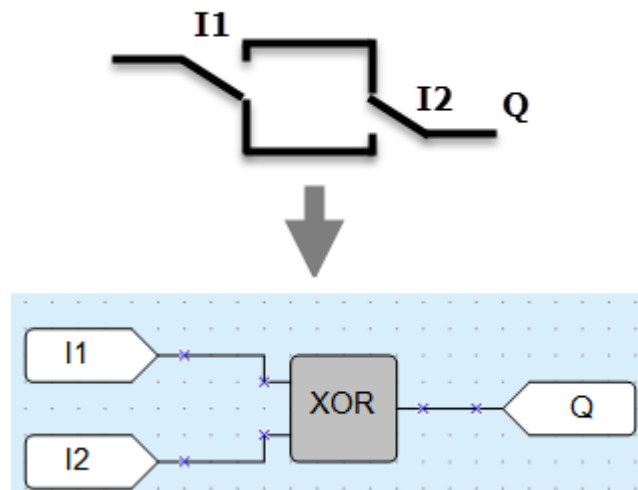
I1	Q
0	1
1	0

Bitwise operation example with integer inputs:

NOT	01
	10

The bitwise NOT, or complement, is a unary operation that performs logical negation on each bit, forming the ones' complement of the given binary value.

7.1.1.4 Exclusive OR (XOR)



The output **Q** is **True** if only one of the inputs is **True**.

Truth table:

I1	I2	Q
0	0	0
0	1	1

I1	I2	Q
1	0	1
1	1	0

Bitwise operation example with integer inputs:

XOR	0011
	0101
	0110

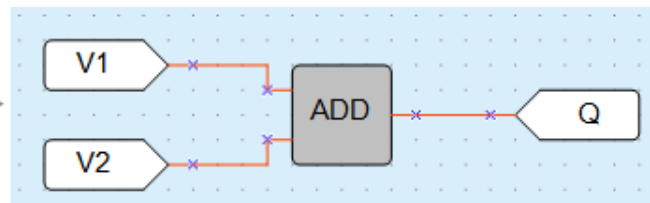
7.1.2 Mathematical operators

There are different operators for different data types:

Operator	INT	REAL
Addition	<u>ADD</u>	<u>fADD</u>
Subtraction	<u>SUB</u>	<u>fSUB</u>
Multiplication	<u>MUL</u>	<u>fMUL</u>
Division	<u>DIV</u>	<u>fDIV</u>
Modulo operation	<u>MOD</u>	–
Power function	–	<u>fPOW</u>
Absolute value	–	<u>fABS</u>

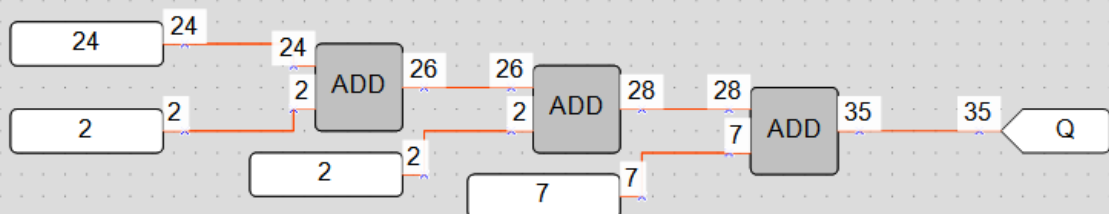
7.1.2.1 Addition (ADD, fADD)

$$V1 + V2 = Q$$



The function ADD operates with INT variables, while the function fADD operates with REAL variables. The output value Q is the sum of the input values.

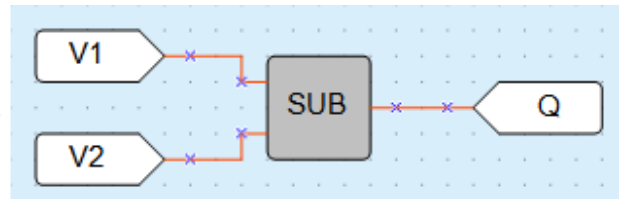
Example:



The output value may not exceed 4294967295 (32 bits). Otherwise the extra bits will be truncated.

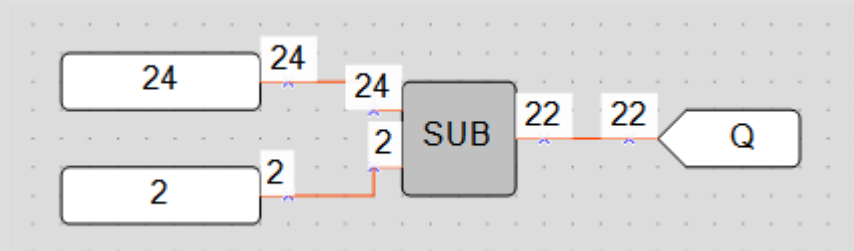
7.1.2.2 Subtraction (SUB, fSUB)

$$V1 - V2 = Q$$



The function **SUB** operates with INT variables, while the function **fSUB** operates with REAL variables.

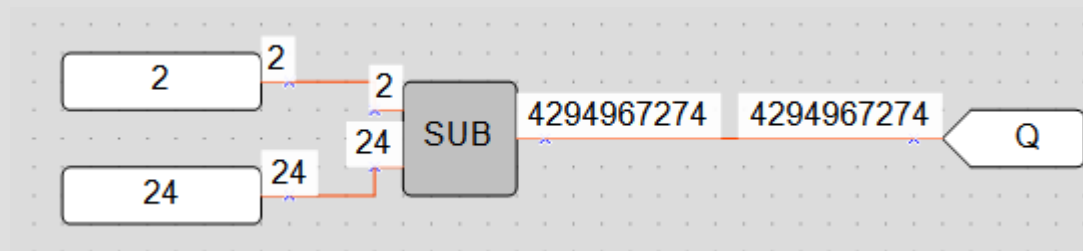
The output value **Q** is the result of subtraction of the value **I2** from the value **I1**.

Example:

If the value **I1** is less than the value **I2**, the output is calculated as follows:

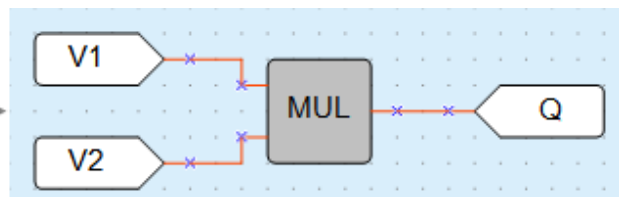
$$Q = I1 + 0x100000000 - I2$$

$$0x100000000 = 4294967296$$



7.1.2.3 Multiplication (MUL, fMUL)

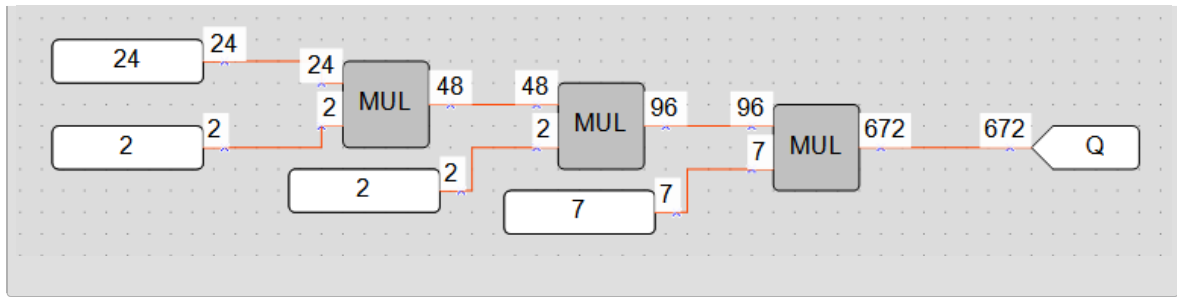
$$V1 \times V2 = Q$$



The function **MUL** operates with INT variables, while the function **fMUL** operates with REAL variables.

The output value **Q** is the product of the input values.

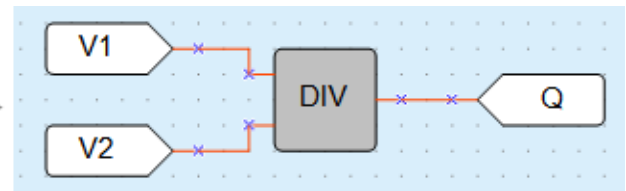
Example:



The output value may not exceed 4294967295 (32 bits). If it does happen, the extra bits will be truncated.

7.1.2.4 Division (DIV, fDIV)

$$V1 \div V2 = Q$$

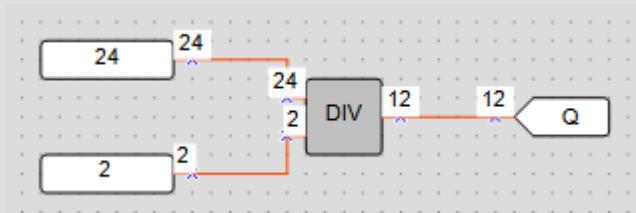


The function **DIV** operates with INT variables, the function **fDIV** operates with REAL variables. The output value **Q** is the quotient of the input values, where the value **I1** is the dividend and the value **I2** is the divisor.

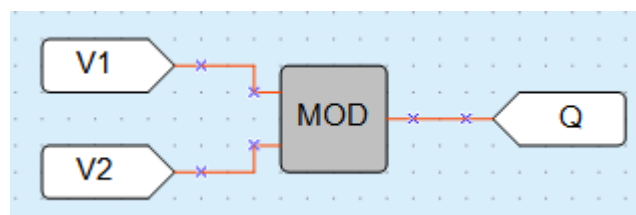
If the quotient is not an INT, it is rounded down to an INT.

In case of division by 0 the output value is 0xFFFFFFFF.

Example:

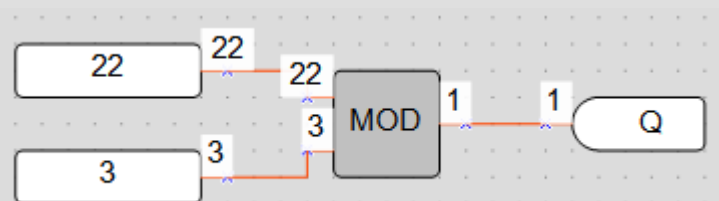


7.1.2.5 Modulo operator (MOD)



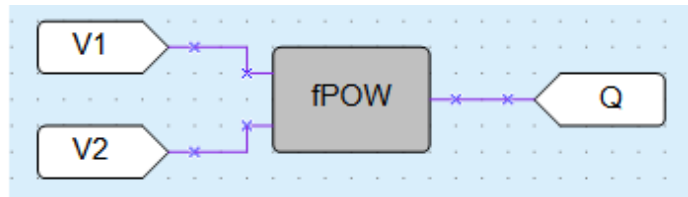
The function **MOD** operates with INT variables. The output **Q** is a remainder of the division of input values.

Example:



7.1.2.6 REAL-Power function (fPOW)

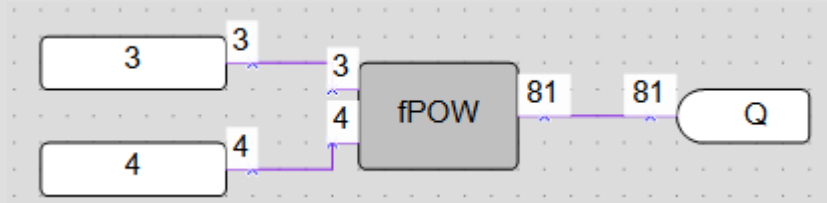
$$V1^{(V2)} = Q \rightarrow$$



The function **fPOW** operates with REAL variables.

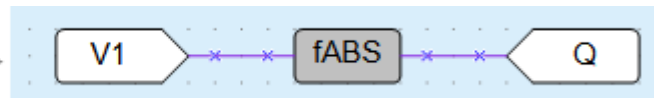
The output value **Q** is the value **I1** raised to the power of the value **I2**.

Example:



7.1.2.7 REAL-Absolute function (fABS)

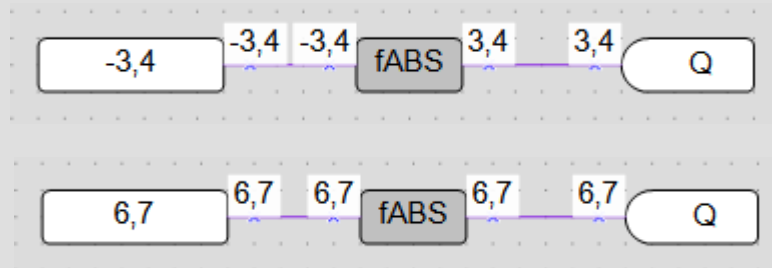
$$|V| = Q \rightarrow$$



The function **fABS** operates with REAL variables.

The output value **Q** is an absolute value of the input value.

Example:

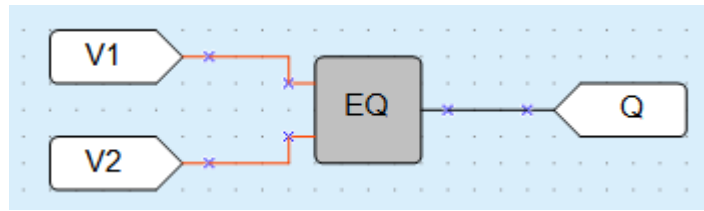


7.1.3 Relational operators

The relational operators are functions that test or define some kind of relation between two or more values.

- Equal (EQ)
- Greater than (GT, fGT)
- Binary selection (SEL)

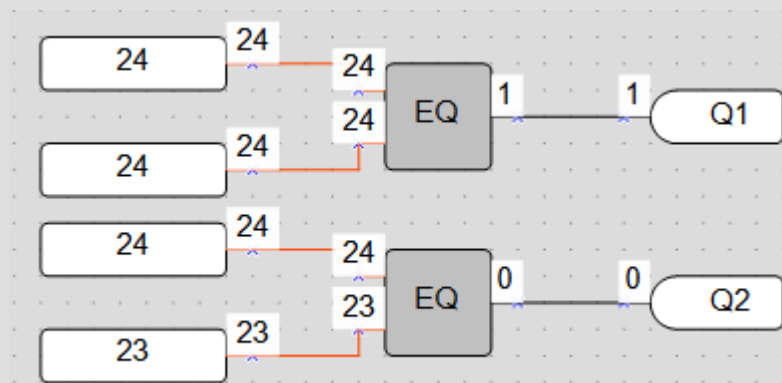
7.1.3.1 Equal (EQ)



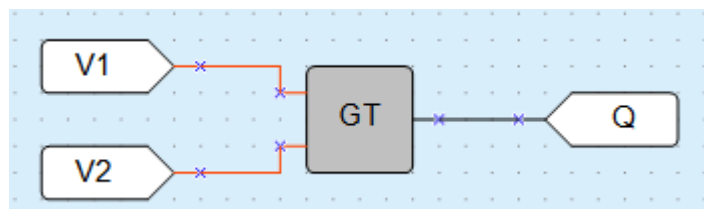
The function **EQ** operates with INT variables.
The output value **Q** is **True** if values **I1** and **I2** are equal.

- $V1 = V2 \rightarrow Q = 1$;
- $V1 > V2 \rightarrow Q = 0$;
- $V1 < V2 \rightarrow Q = 0$.

Example:



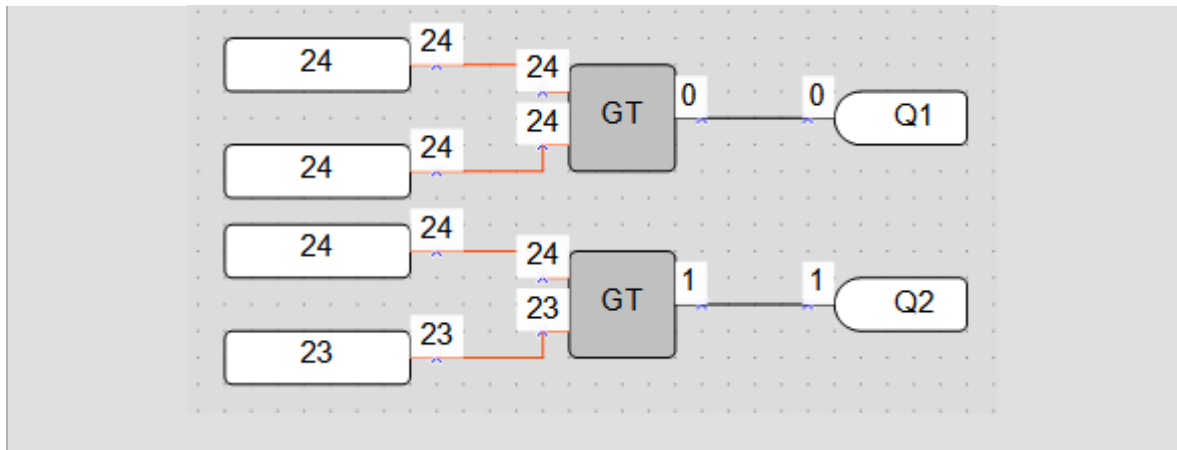
7.1.3.2 Greater than (GT, fGT)



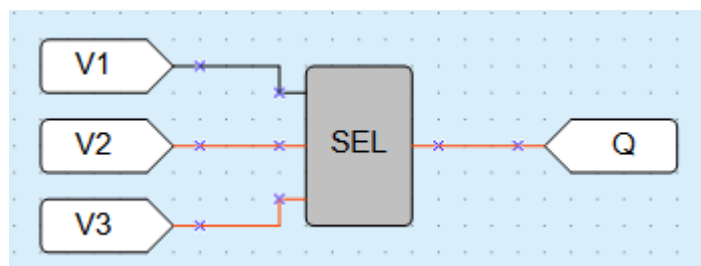
The function **GT** operates with INT variables, while the function **fGT** operates with REAL variables.
The output value **Q** is **True** if the value **I1** is greater than the value **I2**.

- $V1 = V2 \rightarrow Q = 0$;
- $V1 > V2 \rightarrow Q = 1$;
- $V1 < V2 \rightarrow Q = 0$.

Example:



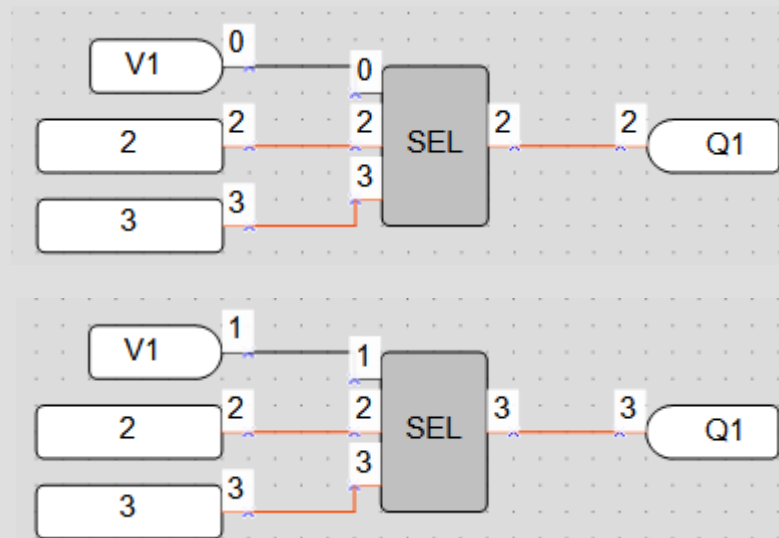
7.1.3.3 Binary selection (SEL, fSEL)



The function **SEL** operates with INT variables, the function **fSEL** operates with REAL variables. If the value **I1** is **False**, the output value **Q** is set to the value **I2**, else to the value **I3**.

- V1 = 0 → Q = V2;
- V1 = 1 → Q = V3.

Example:



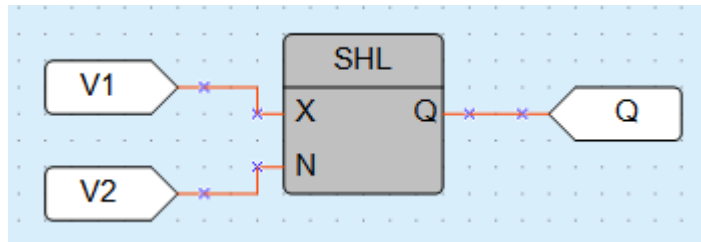
7.1.4 Bitshift operators

The bitshift operators treat a variable as a series of bits that can be moved (shifted) to the left or right.

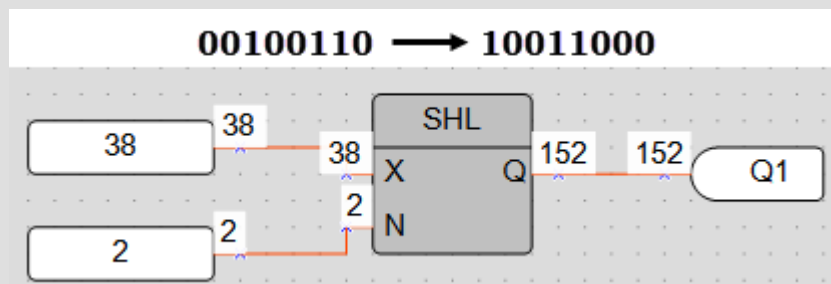
- Shift register left (SHL)
- Shift register right (SHR)

7.1.4.1 Shift register left (SHL)

The function **SHL** operates with INT variables. It is used to shift all bits of the operand **X** to the left by the **N** number of bits; vacated bits are zero-filled. The result is set to the output **Q**.

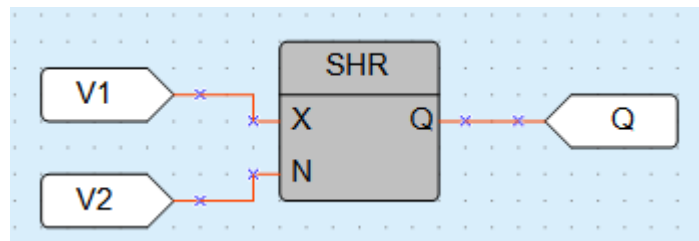
**Example:**

Left shift of the number 38 (decimal) = 00100110 (binary) by 2 bits

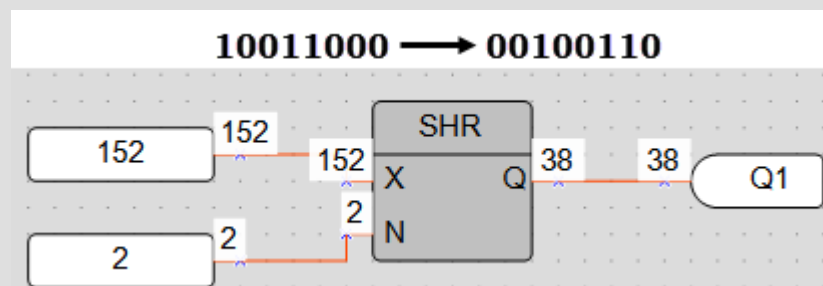


7.1.4.2 Shift register right (SHR)

The function **SHR** operates with INT variables. It is used to shift all bits of the operand **X** to the right by the **N** number of bits; vacated bits are zero-filled. The result is set to the output **Q**.

**Example:**

Right shift of the number 152 (decimal) = 10011000 (binary) by 2 bits

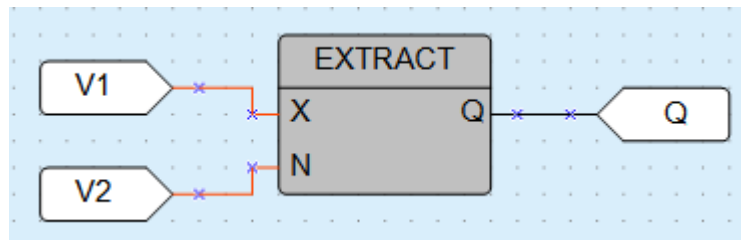


7.1.5 Bit operators

The bit operator treats a value as a series of bits to perform operations on one or more individual bits of an operand.

- Read single bit (EXTRACT)
- Set single bit (PUTBIT)
- Decoder (DC32)
- Encoder (CD32)

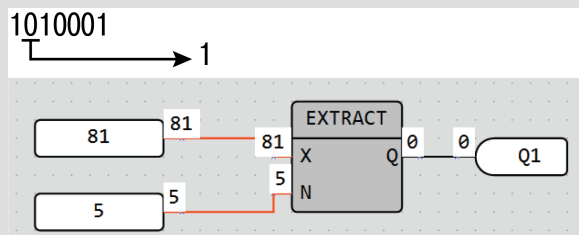
7.1.5.1 Read single bit(EXTRACT)



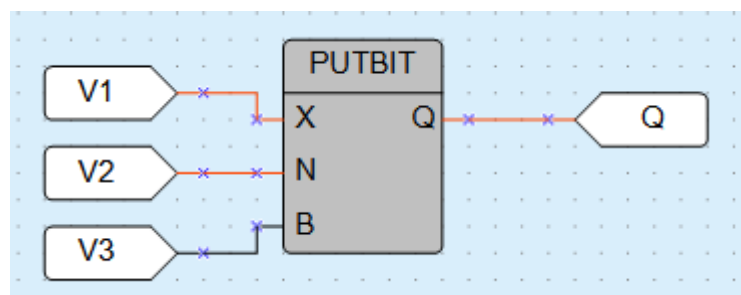
The output value **Q** (BOOL) of the function **EXTRACT** is the value of bit **N** (INT) in the operand **X** (INT). The bit numbering is zero-based.

Example:

Reading of the 5th bit from the number 81 (decimal) = 1010001 (binary):



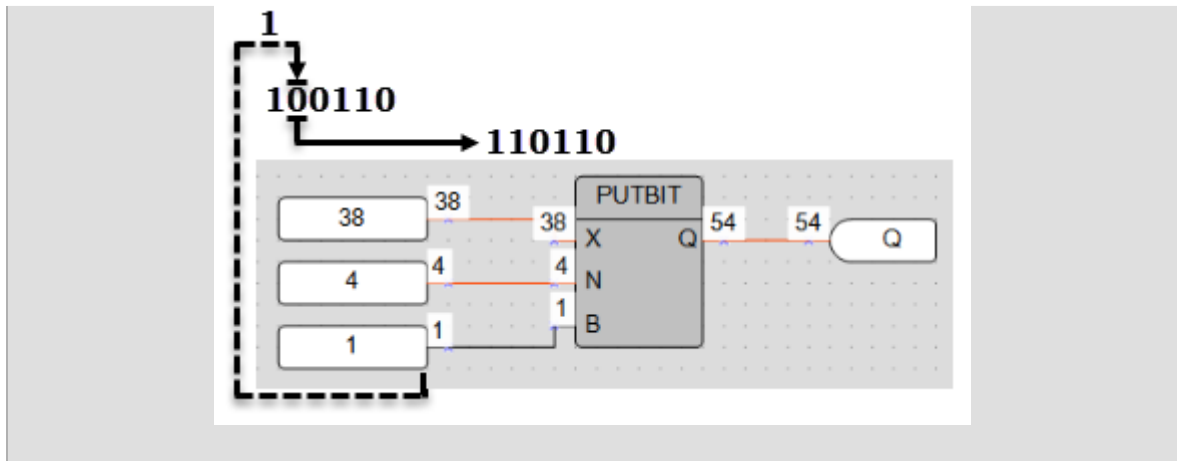
7.1.5.2 Set single bit (PUTBIT)



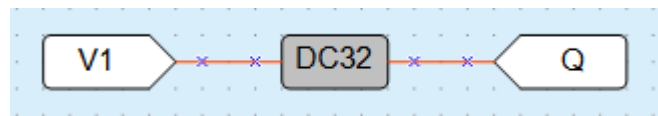
This output value **Q** (INT) is the value of the operand **X** (INT) where the bit **N** (INT) is set to the value at the input **B** (BOOL). The bit numbering is zero-based.

Example:

Setting of the 4th bit to 1 in the number 38 (decimal) = 100110 (binary):



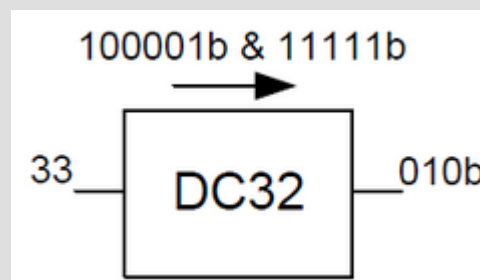
7.1.5.3 Decoder (DC32)



The decoder converts a binary code at the input to a position code at the output. Decoding is carried out bitwise by the logical operation **AND** with the operand 0x1F (11111b). Truth table:

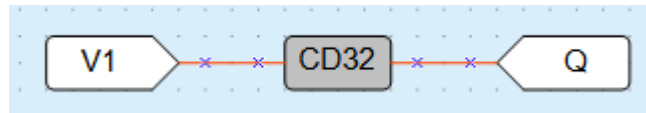
Binary code							Position code									
5	4	3	2	1	32	31	...	6	5	4	3	2	1			
0	0	0	0	0	0	0		0	0	0	0	0	1			
0	0	0	0	1	0	0		0	0	0	0	1	0			
0	0	0	1	0	0	0		0	0	0	1	0	0			
0	0	0	1	1	0	0		0	0	1	0	0	0			
0	0	1	0	0	0	0		0	1	0	0	0	0			
...												
1	1	1	0	1	0	0		0	0	0	0	0	0			
1	1	1	1	0	0	1		0	0	0	0	0	0			
1	1	1	1	1	1	0		0	0	0	0	0	0			

Example:



7.1.5.4 Encoder

Encoder (CD32) is used to perform the operation of converting the positional code from the input into binary code at the output.



If there is more than one “1” bits in the position code, the encoder operates only with the most significant “1” bit.

Truth table:

Binary code					Position code										
5	4	3	2	1	32	31	...	6	5	4	3	2	1		
0	0	0	0	0	0	0		0	0	0	0	1			
0	0	0	0	1	0	0		0	0	0	1	0			
0	0	0	1	0	0	0		0	0	0	1	0			
0	0	0	1	1	0	0		0	0	1	0	0			
0	0	1	0	0	0	0		0	1	0	0	0			
...											
1	1	1	0	1	0	0		0	0	0	0	0	0		
1	1	1	1	0	0	1		0	0	0	0	0	0		
1	1	1	1	1	1	0		0	0	0	0	0	0		

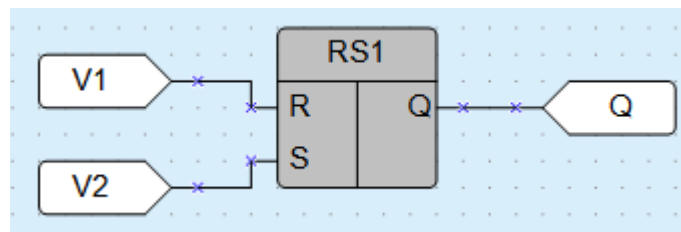
7.2 Function blocks

- Triggers
- Timers
- Generators
- Counters
- Controllers

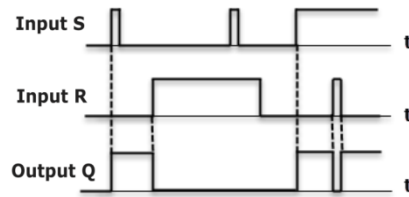
7.2.1 Triggers

- RS trigger reset dominant (RS)
- SR trigger set dominant (SR)
- Rising edge (RTRIG)
- Falling edge (FTRIG)
- D-trigger (DTRIG)

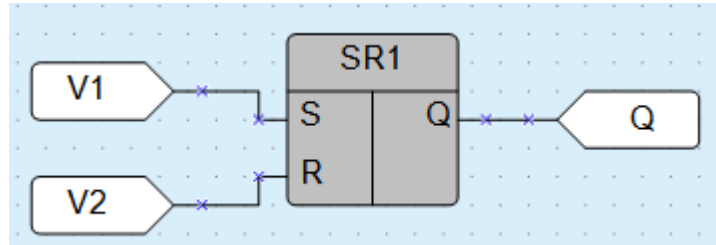
7.2.1.1 RS trigger reset dominant



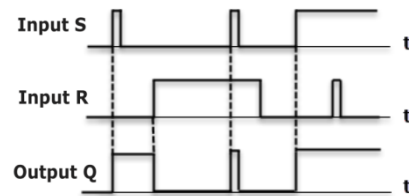
The output **Q** is **True** with a rising edge at the input **S** (Set) and **False** with a rising edge at the input **R** (Reset). The input **R** has higher priority.



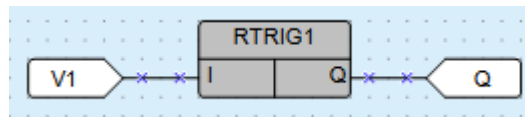
7.2.1.2 SR trigger set dominant



The output **Q** is **True** with a rising edge at the input **S** (Set) and **False** with a rising edge at the input **R** (Reset). The input **S** has higher priority.

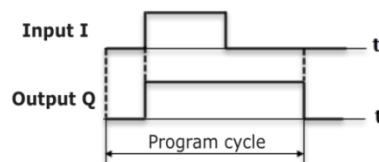


7.2.1.3 Rising edge (RTRIG)

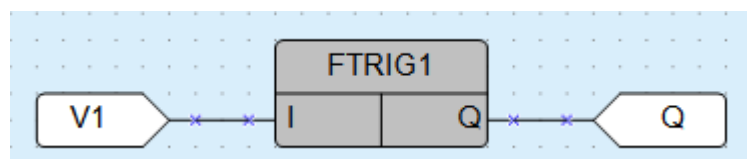


Detector for a rising edge appears.

The output **Q** remains **False** until a rising edge at the input **I**. As soon as the input **I** becomes **True**, the output becomes **True** and remains **True** for one program cycle.

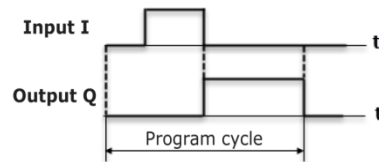


7.2.1.4 Falling edge (FTRIG)

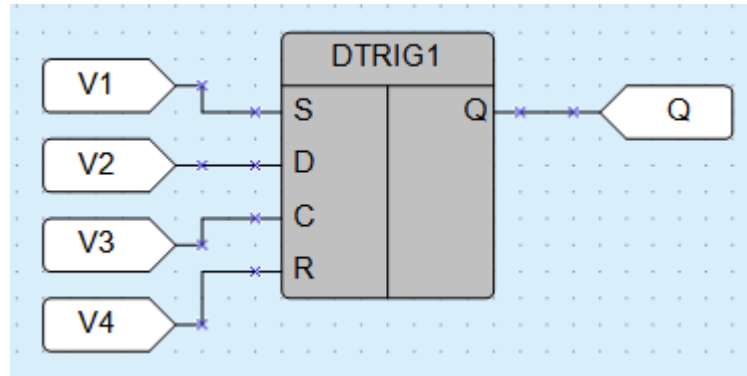


Detector for a falling edge.

The output **Q** remains **False** until a falling edge at the input **I**. As soon as the input **I** becomes **False**, the output becomes **True** and remains **True** for one program cycle.

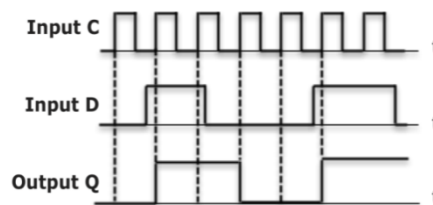


7.2.1.5 D-trigger (DTRIG)



D-trigger generates a pulse at the output **Q** with the pulse duration specified at the input **D** and synchronized with the clock pulse at the input **C**.

If the input **D** is **True**, the output **Q** becomes **True** with a rising edge of the clock pulse at the input **C**. If the input **D** is **False**, the output **Q** becomes **False** with a rising edge of the clock pulse at the input **C**.

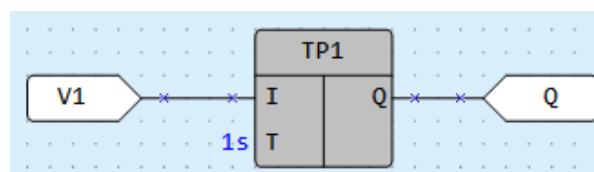


The output **Q** can be forced to set to **True** with a rising edge at the input **S** (Set) and forced to reset to **False** with a rising edge at the input **R** (Reset), regardless of the states of the inputs **C** and **D**. The input **R** has higher priority.

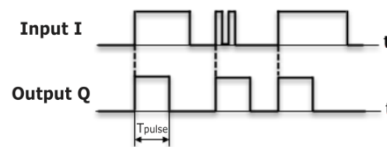
7.2.2 Timers

- Pulse (TP)
- ON-delay timer (TON)
- OFF-delay timer (TOF)
- Timer (CLOCK)
- Weekly timer (CLOCKWEEK)

7.2.2.1 Pulse (TP)

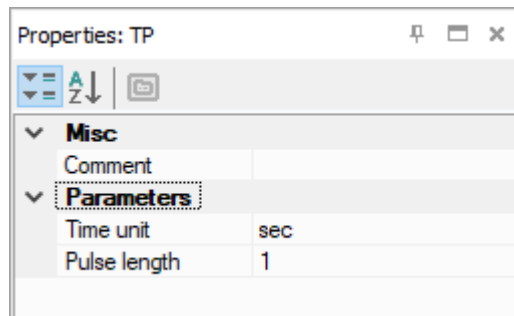


The block **TP** is used to generate one output pulse with the specified pulse duration.



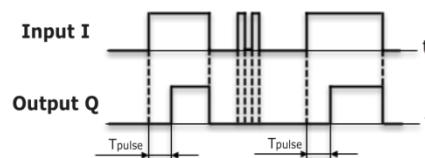
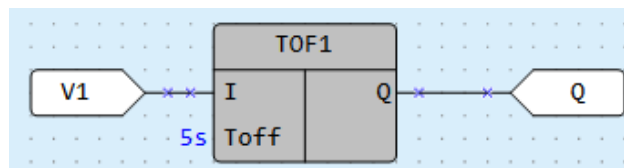
The output **Q** becomes **True** with a rising edge at the input **I** for the time specified at the input **T**. During this time, the output **Q** remains **True** regardless of the signal change at the input **I**. The output **Q** is reset to **False** with the end of pulse.

The pulse duration and the time unit can be set in Property Box.

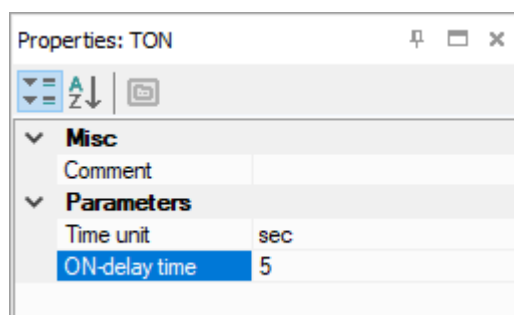


Time range: 0...4147200000 ms or 48 days.

7.2.2.2 ON-delay timer (TON)

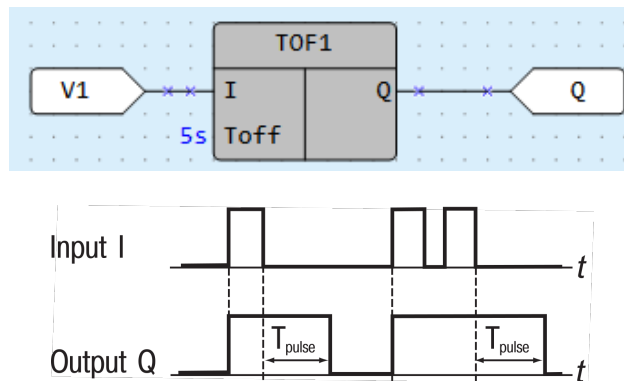


The output **Q = False** if the input **I = False**. The delay time specified at the input **TON** starts with a rising edge at the input **I**. When the time **TON** is elapsed, the output **Q** becomes **True** and remains **False** until a falling edge appears at the input **I**. Input changes shorter than **TON** are ignored. The delay time and the time unit can be set in Property Box.

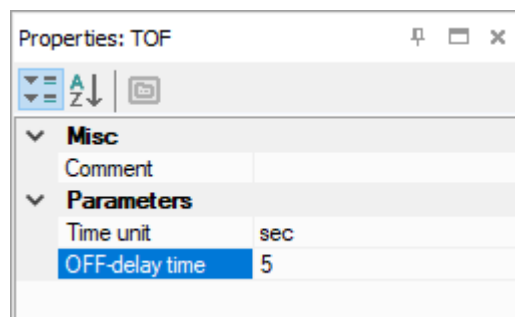


Time range: 0...4147200000 ms or 48 days.

7.2.2.3 OFF-delay timer (TOF)

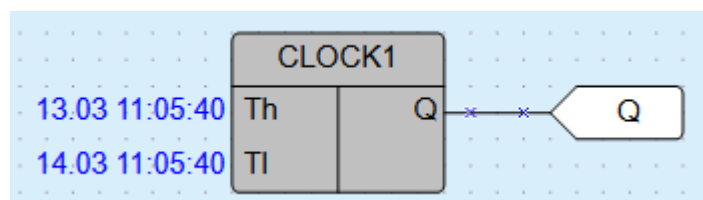


The output **Q = False** if the input **I = False**. The delay time specified at the input **TOFF** starts with a falling edge at the input **I**. When the time **TOFF** is elapsed, the output **Q** becomes **False** and remains **False** until a rising edge appears at the input **I**. Input changes shorter than **TOFF** are ignored. The delay time and the time unit can be set in Property Box.

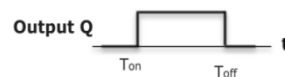


Time range: 0...4147200000 ms or 48 days.

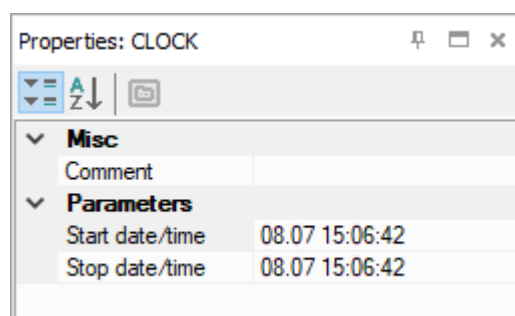
7.2.2.4 Timer (CLOCK)



The block **CLOCK** is an interval timer controlled by a real-time clock.

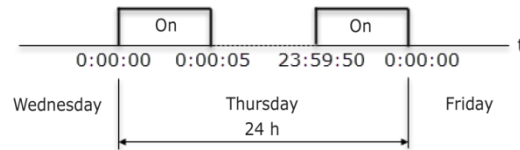


The times **TH** and **TL** can be set in Property Box.

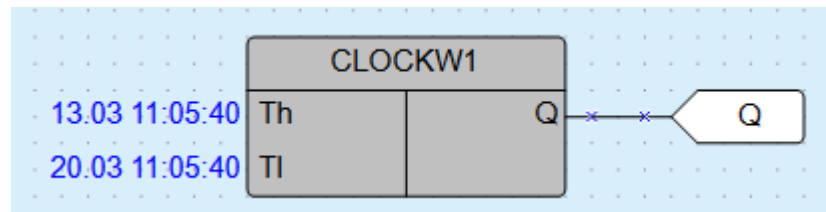


Time range: from 0.00 seconds to 24 hours.

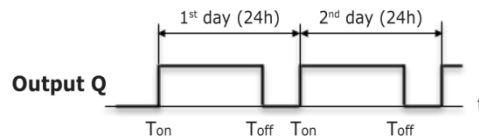
If **TH** < **TL**, the state of the output **Q** is as follows:



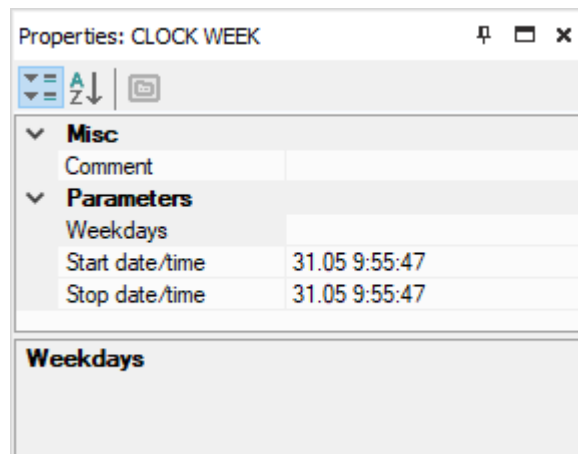
7.2.2.5 Weekly timer (CLOCKWEEK)



The block **CLOCKWEEK** is an interval timer with the parameter **Weekdays** controlled by a real-time clock.



The times **TH** and **TL** can be set in Property Box.

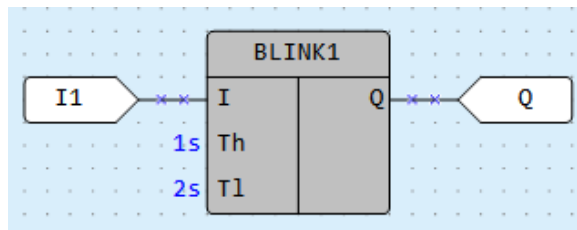


Time range: from 0.00 seconds to 24 hours.

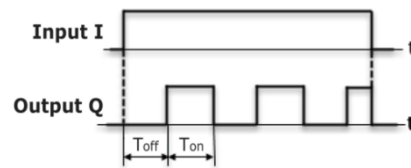
7.2.3 Generators

- Pulse generator (BLINK)

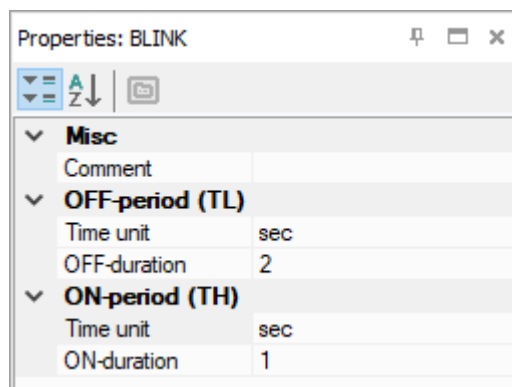
7.2.3.1 Pulse generator (BLINK)



If the input **I** becomes **True**, the block **BLINK** generates a square wave on the output **Q** with a period of **TH** + **TL**, starting with an interval of the duration of **TL**, followed by a pulse of the duration of **TH**. It continues that way until the input **I** is **False**.



The times **TH** and **TL** and the time units can be set in Property Box.

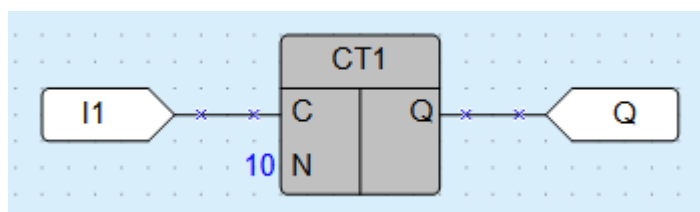


Time range: 0...4233600000 milliseconds or 49 days.

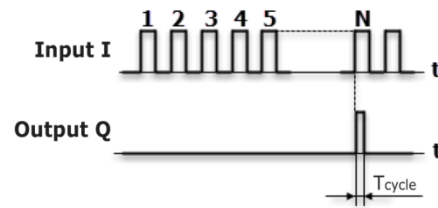
7.2.4 Counters

- Threshold counter with self-reset (CT)
- Universal counter (CTN)
- Threshold counter (CTU)

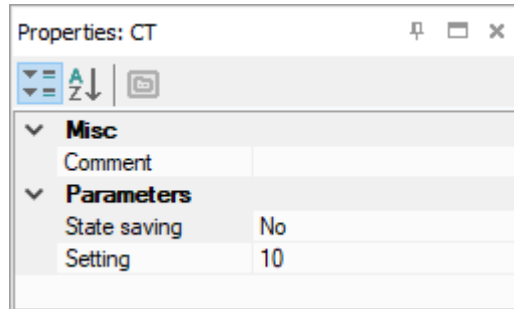
7.2.4.1 Threshold counter with self-reset (CT)



The output **Q** is of type **BOOL**. If the number of pulses counted on the input **C** exceeds the threshold (**Setting**) specified at the input **N**, the output **Q** becomes **True** and remains for one program cycle. The operation of the counter is explained in the diagram below.

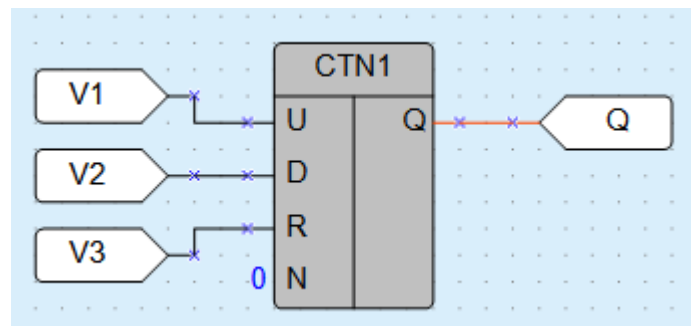


The parameters **Setting** and **State saving** can be set in Property Box.

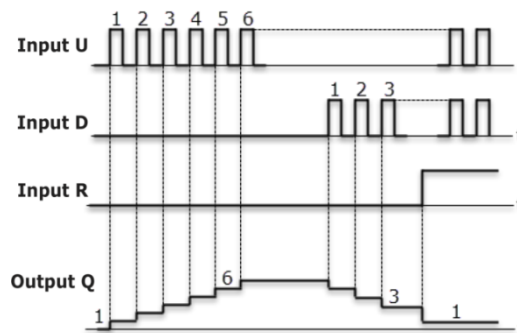


Threshold range: 0...65535.

7.2.4.2 Universal counter (CTN)

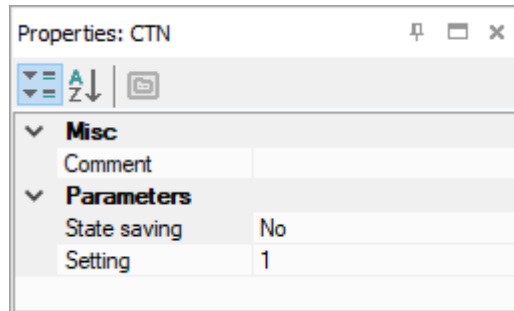


The output **Q** is of type INT. A rising edge at the input **U** increases the value at the output **Q** by 1. A rising edge at the input **D** decreases the value at the output **Q** by 1. If the input **R = True**, the output **Q** gains the value **Setting** from the input **N**.



The input **U** has higher priority than the input **D**.

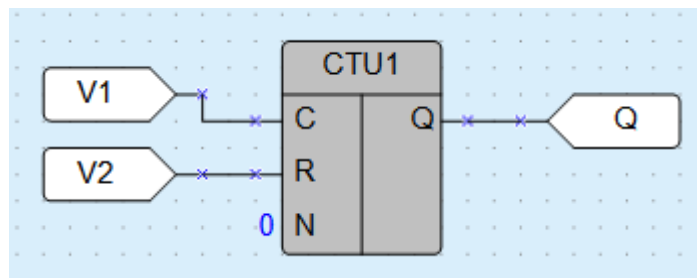
The parameters **Setting** and **State saving** can be set in Property Box.



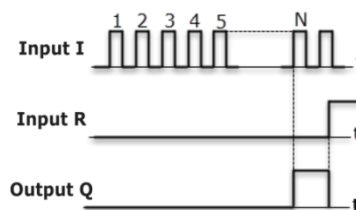
Setting range: 0...65535.

If **State saving** = **Yes**, the state of the counter is permanently stored in the non-volatile memory.

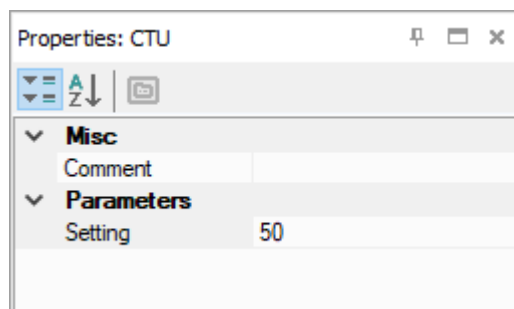
7.2.4.3 Threshold counter (CTU)



The output **Q** is of type Boolean. If the number of pulses counted on the input **C** exceeds the threshold (**Setting**) specified at the input **N**, the output **Q** becomes **True** and remains **True** until a rising edge at the input **R**. The input **R** has higher priority than the input **C**. The operation of the counter is explained in the diagram below.



The parameter **Setting** can be set in Property Box.

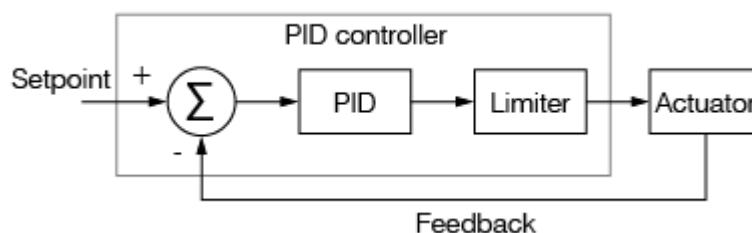
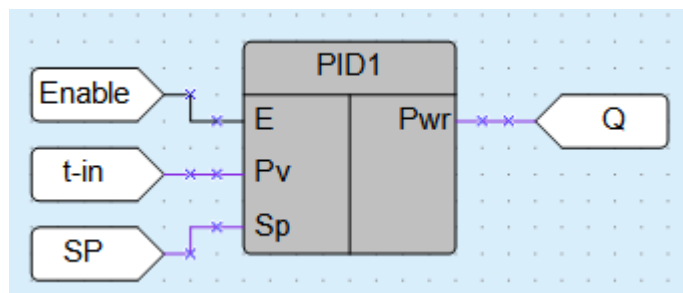


Threshold range: 0...65535.

7.2.5 Analog

- PID-controller (PID) for first generation devices
- PID-controller (PID_REG) for second generation devices

7.2.5.1 PID-controller (PID) for first generation devices



The function block **PID** is used to implement of the proportional-integral-derivative control.

Table 7.1 PID block inputs/outputs

Name	Type	I/O	Description	Values
E	BOOL	I	Enable control (0 = Off, 1 = On). If disabled, the parameter Pwr takes the value of the parameter Output safe state .	0 – Off 1 – On
Pv	REAL	I	Process value	
Sp	REAL	I	Setpoint	
Pwr	REAL	O	Output power, %	0...100

Table 7.2 PID block parameters

Name	Type	Description	Values	Access		
				Property Box	Write ToFB	Read From-FB
Control mode	BOOL	0 – Heating 1 – Cooling	0/1	X	X	
Output safe state	REAL	Output value when control is disabled, %	0...100	X	X	
Kp	REAL	Proportional gain, multiplication factor for proportional control	0...100	X	X	
Ti (s)	REAL	Integral time, time constant for integral control in seconds	–3,402823E+38... 3,402823E+38	X	X	
Td (s)	REAL	Derivative time, time constant for derivative control in seconds	–3,402823E+38... 3,402823E+38	X	X	
Output max.	REAL	Output upper limit, % (default 80 %)	0...100	X	X	

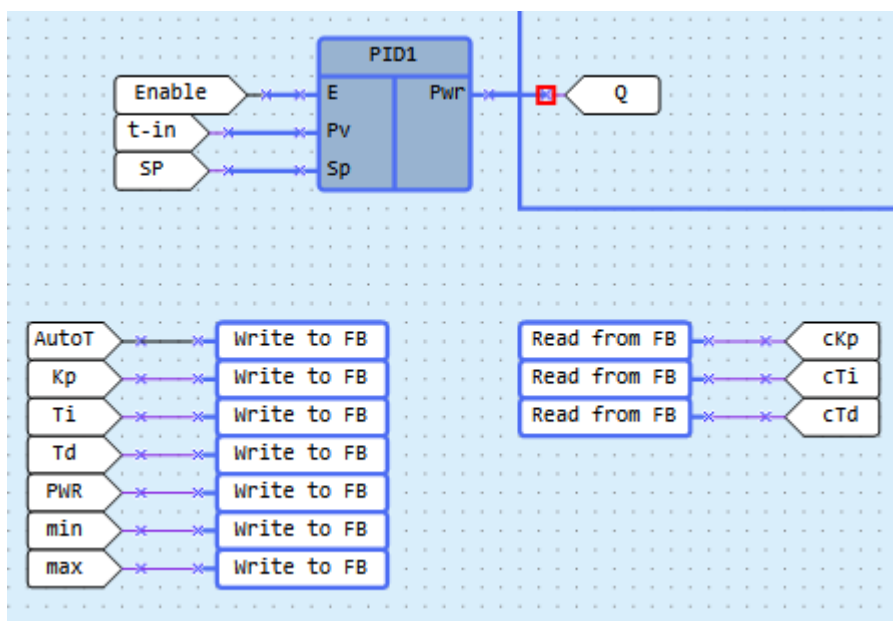
Name	Type	Description	Values	Access		
				Property Box	Write ToFB	Read From-FB
Output min.	REAL	Output lower limit, % (default 20 %)	0...100	X	X	
Start AT	BOOL	0 – stop auto-tuning 1 – start auto-tuning	0/1		X	
AT completed	BOOL	Flag: 0 – auto-tuning stopped 1 – auto-tuning started	0/1			X
Kp calculated	REAL	Calculated proportional gain	–3,402823E+38... 3,402823E+38			X
Ti calculated	REAL	Calculated integral time	–3,402823E+38... 3,402823E+38			X
Td calculated	REAL	Calculated derivative time	–3,402823E+38... 3,402823E+38			X

Tuning of a control loop is the adjustment of its control parameters (**Kp**, **Ti**, **Td**) to the optimal values for the desired control response.

Auto-tuning

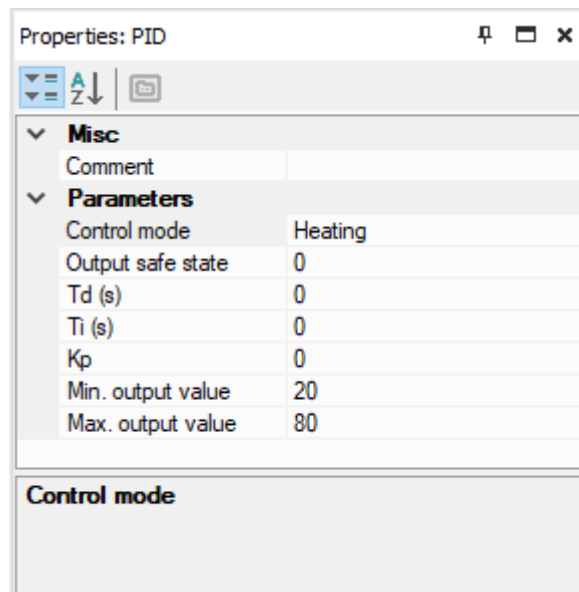
Programmable loop tuning can be performed using the blocks WriteToFB ^W and ReadFromFB ^R.

To write the parameters, use the block WriteToFB or Property Box.
To read the parameters, use the block ReadFromFB.



To use auto-tuning, add the block WriteToFB to the circuit program and set the reference to the parameter **Start AT** of the PID block.

To start the auto-tuning, enable control (**E = 1**) and set the parameter **Start AT = 1**.



Upon completion of the auto-tuning, the new values of the parameters **Kp**, **Ti** and **Td** are calculated and the flag **AT completed** becomes 1.

If **Start AT = 0**, the flag **AT completed = 0** as well.

If you set **Start AT = 0** before the completion of auto-tuning, the auto-tuning is stopped, the flag **AT completed** becomes 0 and no new coefficients are calculated.

During the auto-tuning, a test signal limited by parameters **Output max.** and **Output min.** is applied to the output **Pwr**.

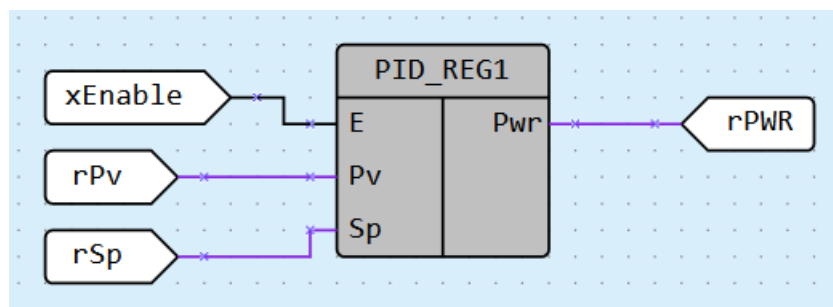
**NOTE**

*If the maximum gain is not sufficient to reach the setpoint, the auto-tuning cannot be completed and will continue until it is stopped with **Start AT = 0**.*

7.2.5.2 PID-controller (PID_REG) for second generation devices

**NOTE**

The PID REG is not available for PR103 devices.



The PID controller is used to implement the proportional-integral-derivative (PID) control law. The controller's output power is calculated using the formula:

$$MV(t) = \frac{1}{X_p} \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right)$$

, where

X_p – proportional band (measurement units)

T_i – integral time constant [s]

T_d – derivative time constant [s]

$e(t)$ – error (deviation)

$MV(t)$ – PID output

Block Inputs*Table 7.3 Block Inputs*

Name	Type	Description	Values
E	BOOL	Controller enable. When disabled, the output is set to the value specified in the 'Output Power in Off State' property	0 – Off; 1 – On
Pv	REAL	Current value of the controlled variable	–9999...9999*
Sp	REAL	Setpoint value of the controlled variable	

*If a value outside this range is applied to the input, the regulator's Pwr output will be reset to the value specified in the 'Output Power in Off State' parameter.

Block Outputs*Table 7.4 Block Outputs*

Name	Type	Description	Values
Pwr	REAL	Controller output power	

Inputs Available via Write to FB*Table 7.5 Inputs Available via Write to FB*

Name	Type	Description	Values
Control mode	BOOL	Controller operating modes: – 'Heater' mode: used for controlling actuators that increase the controlled variable; – 'Cooler' mode: used for controlling actuators that decrease the controlled variable	0 – Heater; 1 – Cooler
Start AT	BOOL	Controller auto-tuning control	0 – Auto-tuning off 1 – Auto-tuning started
Proportional band	REAL	PID controller proportional band, measurement units	0..9999
Integral time constant	REAL	Integral time constant, sec	0..4000
Derivative time constant	REAL	Derivative time constant, sec	0..4000
Output power in OFF state	REAL	Pwr output state when input E is 0	–9999..9999
Output max.	REAL	Upper limit of output power	–9999..9999
Output min.	REAL	Lower limit of output power	–9999..9999
Dead zone	REAL	Dead band around setpoint/2 where controller output remains unchanged, measurement units	

Name	Type	Description	Values
Output type	BOOL	Controller output signal selection	0 – Power 1 – Discrete CRV
Full travel length	REAL	Time for valve stem to move from fully closed to fully open position, sec	5..999**
Minimum pulse time	REAL	Minimum control pulse duration for the valve, sec	0.001..100
Backlash compensation time	REAL	Valve backlash sampling time, sec	0..10
Setpoint change rate	REAL	Setpoint rate of change, measurement units/sec	0 – Function disabled 0...9999
Manual mode of operation of the regulator	BOOL	Flag for switching controller to manual mode	0 – Automatic mode 1 – Manual mode
Manual output control "More"	BOOL	In manual mode: command to send "Increase" signal to block output	
Manual output control "Less"	BOOL	In manual mode: command to send "Decrease" signal to block output	
Initial value of controlled variable for PID-autotuning (Pv_0)	REAL	Controlled variable value that will be at input Pv when zero power is applied (object at rest). This parameter is required for correct auto-tuning	-9999..9999

**To apply a new full stroke time for the valve, restart the block by toggling the signal at input E.

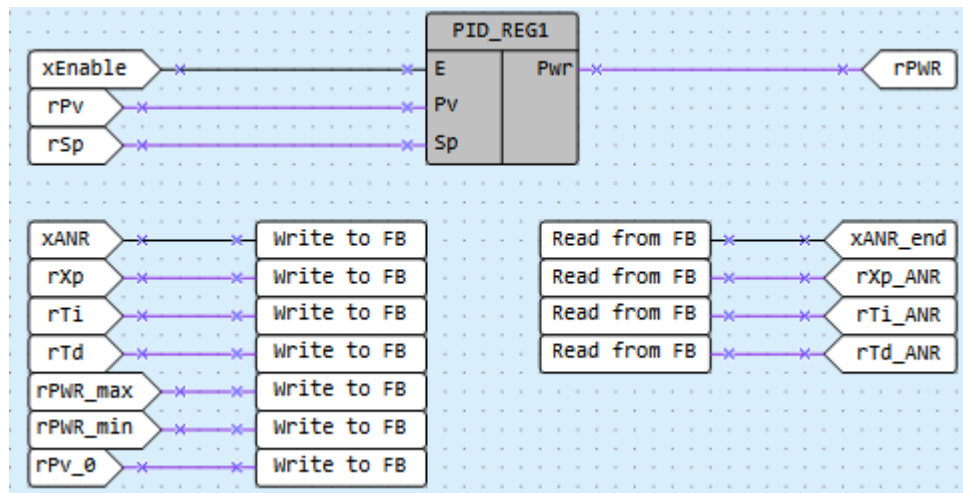
Outputs Available via Read from FB

Table 7.6 Outputs Available via Read from FB

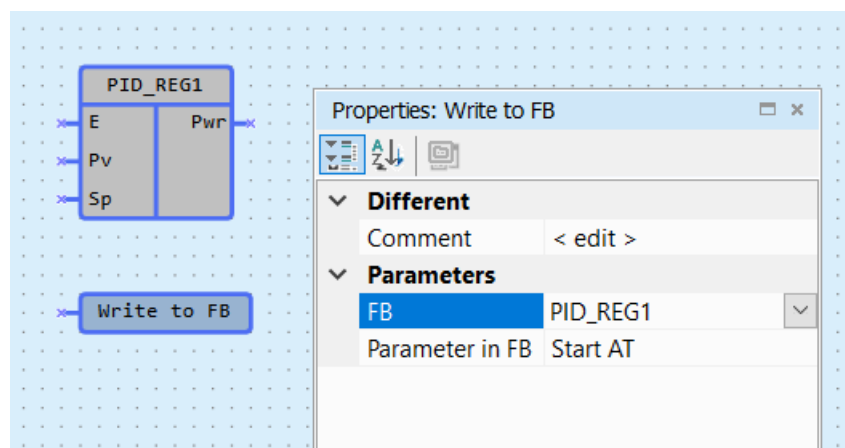
Name	Type	Description	Values
AT completed	BOOL	Auto-tuning completion flag	0 – AT not completed 1 – AT completed
Calculated proportional band	REAL	Proportional band calculated after auto-tuning, measurement units	0..9999
Calculated integral time constant	REAL	Integral time constant calculated after auto-tuning, sec	0..4000
Calculated differential time constant	REAL	Derivative time constant calculated after auto-tuning, sec	0..4000
"More"	BOOL	Valve open signal	
"Less"	BOOL	Valve close signal	

Auto-Tuning

PID controller auto-tuning is performed using Read from and Write to FB blocks.



To start auto-tuning, add a **Write to FB** block and bind it to the PID controller's **Start Auto-Tuning** variable.



Values for other FB parameters can be set using the **Write to FB** block, as shown in the figure above, or configured in the properties panel.

Use **Read from FB** blocks to read the values of parameters **Calculated proportional band**, **Calculated integral time constant**, **Calculated differential time constant**, and **AT completed**.

To start auto-tuning, apply a logical "1" to input **E**.

Upon completion of the auto-tuning process, new coefficient values are available for reading:

Calculated proportional band, **Calculated integral time constant**, and **Calculated differential time constant**. The **AT completed** parameter is set to logical one.



CAUTION

The auto-tuning completion flag remains at logical one for one cycle only.

If the **Start AT** input is reset to logical zero before tuning completion, the process stops, the completion flag is not set, and new coefficient values are not calculated.

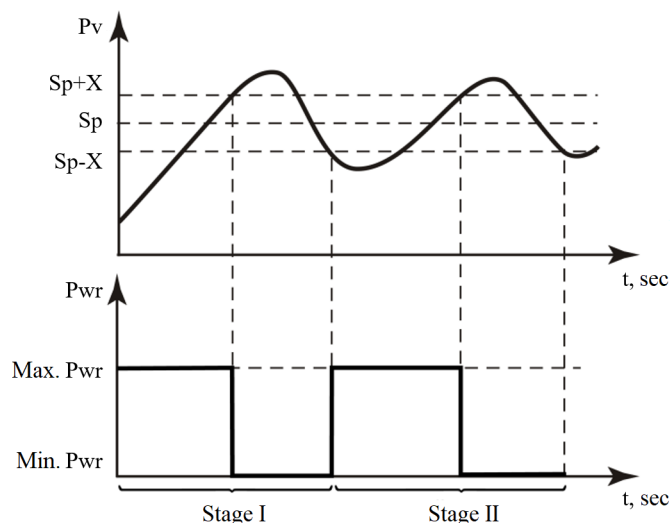


CAUTION

For correct auto-tuning, the **Initial value of controlled variable for PID-autotuning** parameter must be set. It should equal the controlled variable value corresponding to zero actuation (object at rest).

During the tuning process, test power limited by the **Output max.** and **Output min.** parameter values is applied to the PID controller output.

Auto-tuning sequence for 'Heater' mode:



1. When the current value is below the setpoint by more than $X = 0.04 \cdot (Sp - Pv_0)$, maximum power (according to settings) is applied to the block output.
2. Once the current value exceeds the setpoint by more than $X = 0.04 \cdot (Sp - Pv_0)$, minimum power is applied to the block output.
3. Steps 1 and 2 are repeated once more.
4. The calculated PID controller parameters are sent to the corresponding outputs, and the completion flag is set (for one cycle).

If the maximum power setting is insufficient to reach the setpoint, the auto-tuning process will not complete until manually reset.

Working with digital CRV

To work with a discrete control valve, set the value to **"Control valve (digital)"** in the corresponding parameter (Output Type), or, when working with this parameter via **Write to FB**, write the value "1" to it.

For correct operation, the valve parameters must be set: **Full travel time**, **Minimum pulse length**, and **Backlash compensation time**.

Access to control signals (Increase/Decrease) is through the corresponding outputs of the controller's **Read from FB** block.

In this mode, the **Pwr** output displays the virtual valve position (which may differ from the actual position).

When logical "1" is removed from input E, the valve remains in its current position.

For quick setup, we recommend using ready-made macros from the Component Manager:

- For power control — **PID AT_** block;
- Macro with full functionality in power control mode — **PID AT_F_**;
- For discrete CRV control — **PID Valve_** block;
- Macro with full functionality in discrete CRV control mode — **PID Valve_F_**.

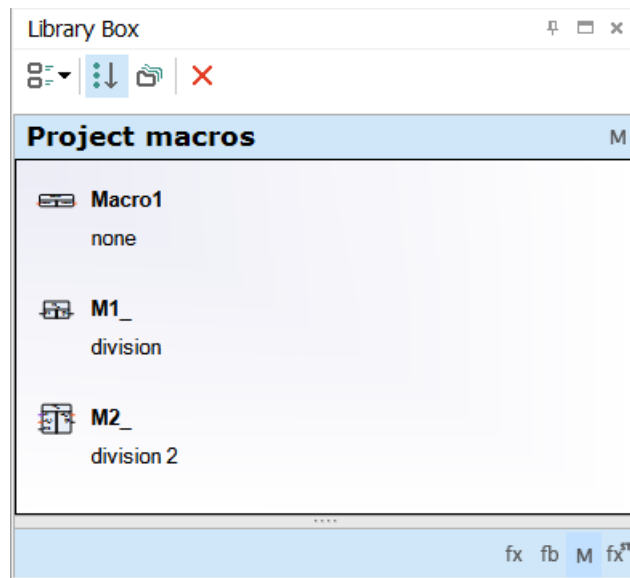
Manual Mode

Operation in power control mode: when switching the controller to manual mode (the **Manual mode of operation of the regulator** input is set to 1), the calculated power value is fixed.

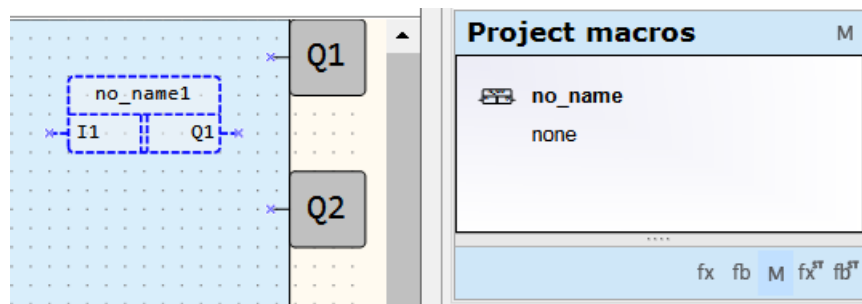
Operation in discrete CRV control mode: when switching the controller to manual mode (the **Manual mode of operation of the regulator** input is set to 1), the calculated valve position is fixed, and the manual valve open/close control inputs become available.

7.3 Project macros

Project Macros section contains macros created by the user or downloaded from Online Database using Component Manager.



To add a macro to a project, drag-and-drop the macro from the Library Box to the workspace.



To open the project macro in the separate workspace for editing, select it in the workspace or in the library and use the item **Edit macro** in the macro context menu.

To remove the macro from the Library Box, select the macro and click the **X** icon in the panel toolbar.

For details about macros creation, development and handling see section Macro development.

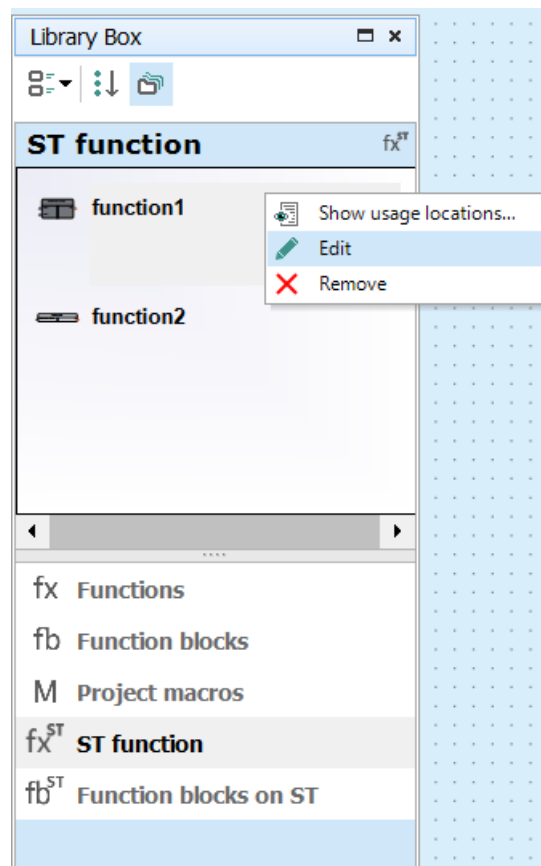
7.4 ST functions



NOTE


User can create functions in ST language for PR100(M02), PR102, PR200, PR103, PR205, PR225 and SMI200 devices.

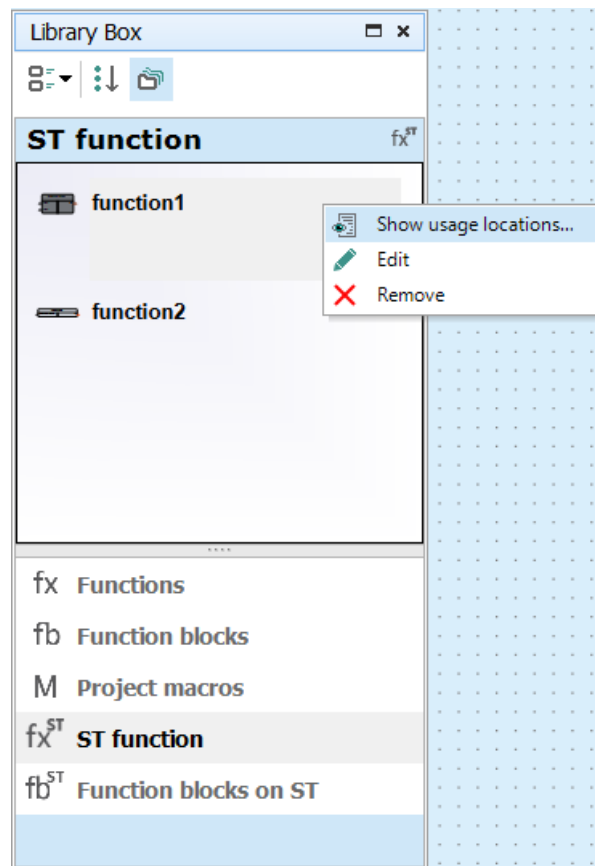
If you have created ST functions in your project, they will be available in the Library Box.



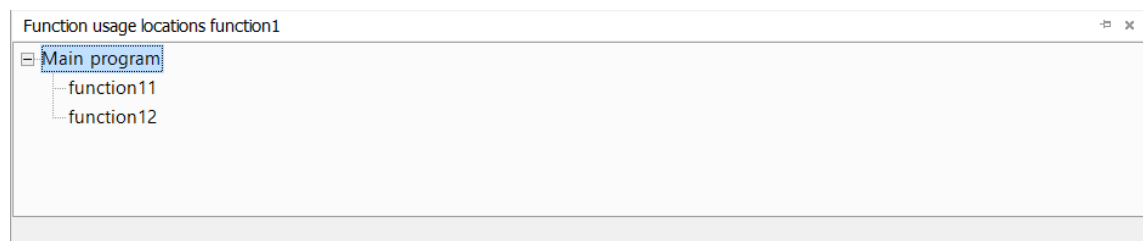
Usage locations


To view all places where the function is used:

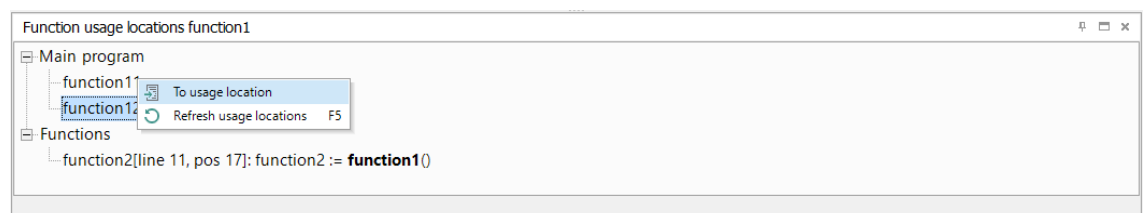
1. Right-click on the function name
2. Select item  **Show usage location....**




The **Function usage locations** panel will open at the bottom of the window, displaying where the function is used in the diagram and in the function editor.



3. Right-click on the line that indicates where the function is used.
4. Select item  **To usage location**.



The focus will shift to where the function is used in the diagram or in the function editor.

NOTE
 Double-click leads to the same result.

If the places where functions are used have changed while working with the program, you should update the **Function usage locations** panel:

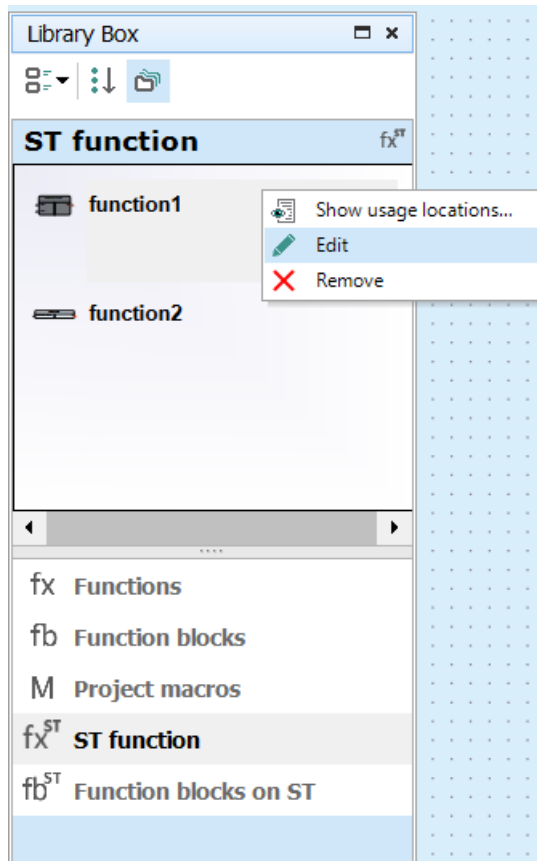
1. Right-click on any line of the panel.

2. Select the item  **Refresh usage locations**.

Go to the function editor

To go to the function editor:

1. Right-click on the function name.
2. Select the item **Edit**.

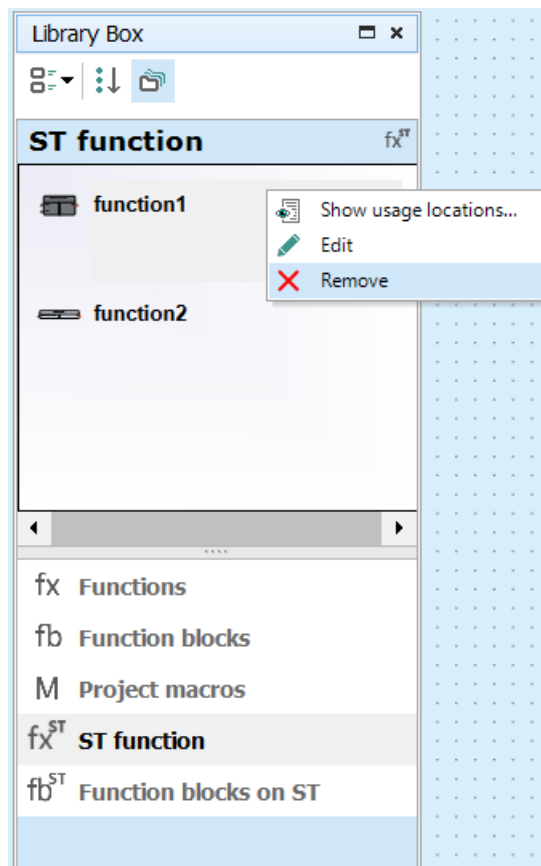


The function editor will open.

Delete function

To remove a function from a project:

1. Right-click on the function name.
2. Select the item **Remove**.

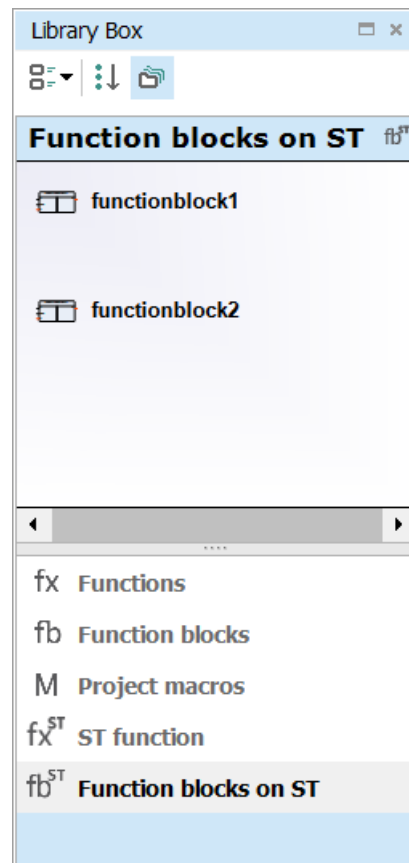
**NOTICE**

If the function is used in the diagram and/or in other functions, deletion may result in compilation errors.

7.5 ST function block**NOTE**


User can create functions in ST language for PR100(M02), PR102, PR200, PR103, PR205, PR225 and SMI200 devices.

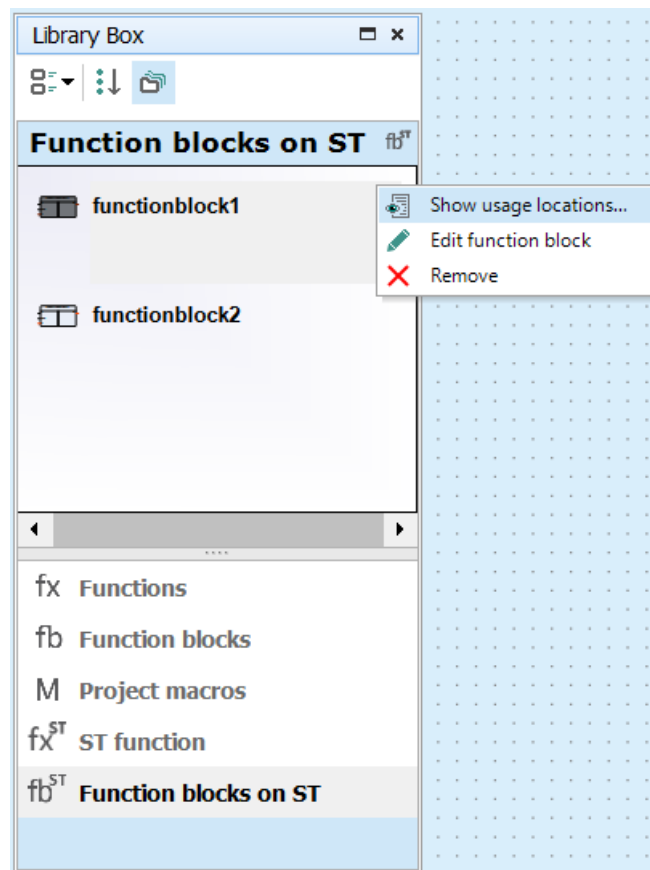
If ST function blocks are created in the project, they will be available in the component library.



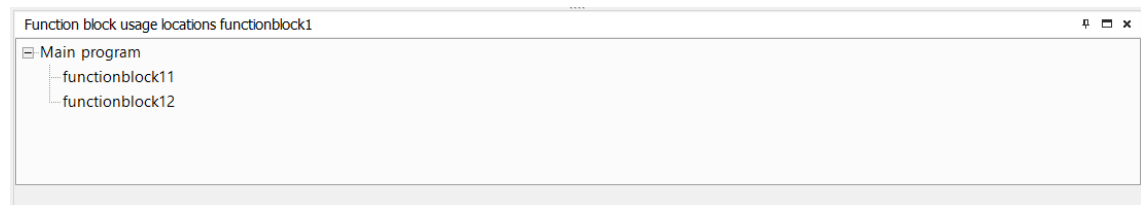
Usage location

To view all places where the function is used:

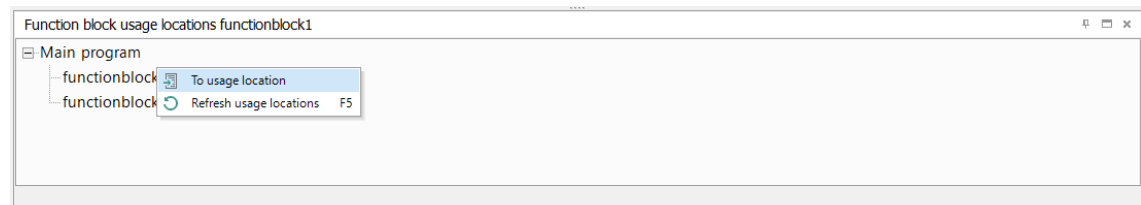
1. Right-click on the name of the function block.
2. Select the item  **Show usage locations....**



The **Function block usage locations** panel will open at the bottom of the window, showing where the function block is used in the main program and in the editor.



3. Right-click on the line with the location where the function block is used.
4. Select the item **To usage location**.



The focus will shift to where the function block is used on the diagram or in the editor.

i **NOTE**
Double-click leads to the same result.

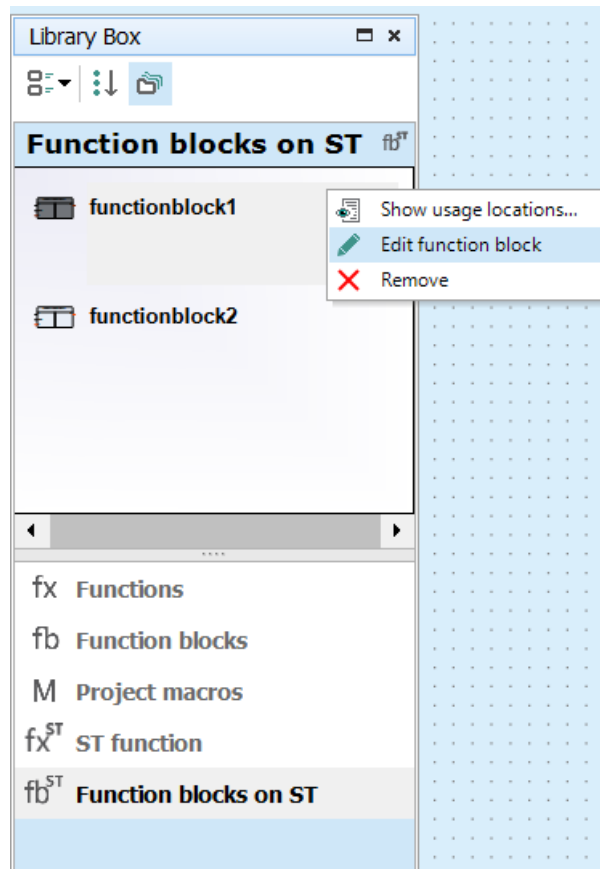
If the places where function blocks are used have changed while working with the program, you should update the **Function block usage location** panel:

1. Right-click on any line of the panel.
2. Select the item **Refresh usage locations**.

Go to the function block editor

To go to the function block editor:

1. Right-click on the function block name.
2. Select the item **Edit**.

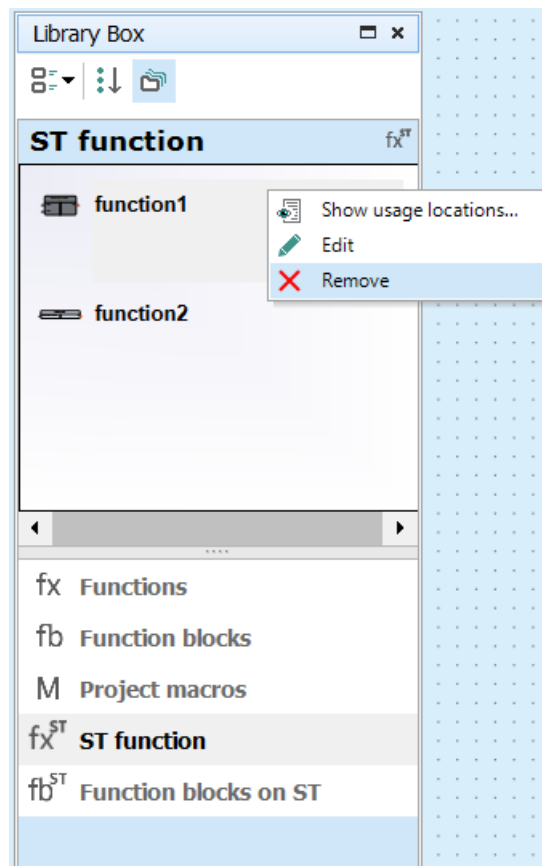


The function block editor 4.13 will open.

Delete function block

To delete a function block:

1. Right-click the function block name.
2. Select the item **Remove**.

**NOTICE**

If the function block is used in the main program and/or in other functions, deletion may result in compilation errors.

7.6 Display elements

If the workspace with a display form is active, only display elements are available in Library Box. With these blocks, the information on the device display can be controlled. The display elements can be placed within the display form by drag-and-drop. The following elements are available:

- Text box
- I/O box (INT/REAL)
- I/O box (BOOL)
- Dynamic box
- Combobox

Use Property Box to customize an element.

Common parameters for all elements:

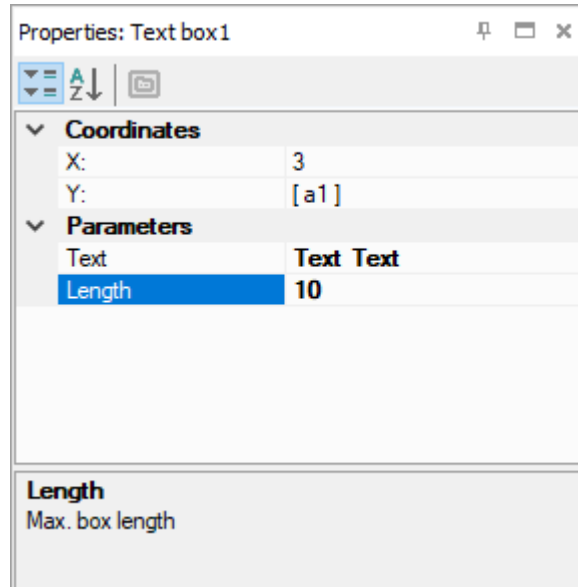
- **Coordinate X** – the position of the first (left) character placeholder of the element from the left edge of the form (from 0 to 15).
- **Coordinate Y** – the position of the first (left) character placeholder of the element from the upper edge of the form, depending on number of the rows in the form.
- There are two ways to determine coordinates: constant (default) or variable. To use a coordinate dependent on a variable, select the coordinate and open the list on the right of the input field.
 - **Constant** – specify the coordinates in Property Box or place the element within the form by drag-and-drop.
 - **Variable** – click **Select** to select an INT variable from the list and confirm with **OK**. The display element will move according to the coordinate value controlled by the variable.
- **Length** – the number of reserved characters. The display element occupies one display row in height, its length can be from 1 to 16 characters.

7.6.1 Text box

Text box is used to display plain text.

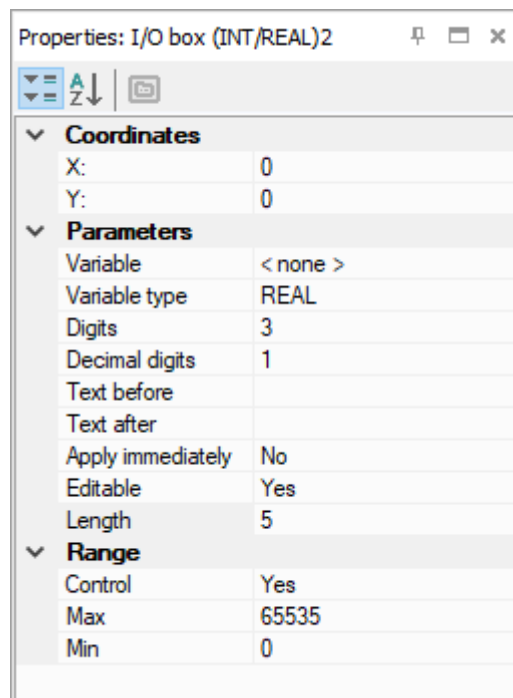
Parameters

Text – text to display. The parameter **Length** specifies the number of the reserved characters.



7.6.2 I/O box (INT/REAL)

I/O box (INT/REAL) is used to display a variable of type INT or REAL. The variable value can be changed with the device function buttons.



Parameters

- **Variable** – the reference to a variable. Use the icon «...» in the input field to select the variable.
- **Data type** – INT or REAL. If the variable has been already selected, its data type will be accepted.

- **Digits** – the total number of displayed digits.
- **Decimal digits** – the number of the characters after the decimal point: 0...6 characters or **Auto** for Auto-precision*.
- **Text before** – the text to the left of the displayed variable.
- **Text after** – the text to the right of the displayed variable.
- **Editable** – if set to **Yes**, the displayed value can be changed using the device function buttons.

**NOTE**

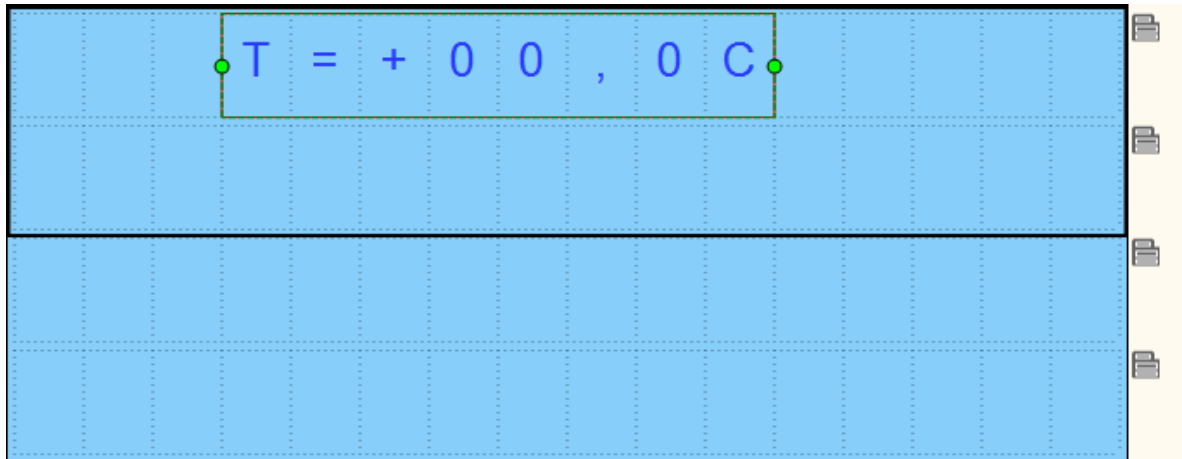
An output variable should be selected. The option has no effect with an input variable.

- **Length** – the total number of reserved characters including both the text before and after.

Range:

The group of parameters is used to limit the input value. If **Editable** = **No**, the parameters of this group have no effect.

- **Limit** – if set to **Yes**, the value entered using the device function buttons is limited by the user parameters **Max** and **Min**, else it is limited only by the available memory area.
- **Max** – the maximum input value.
- **Min** – the minimum input value.

*** Auto-precision**

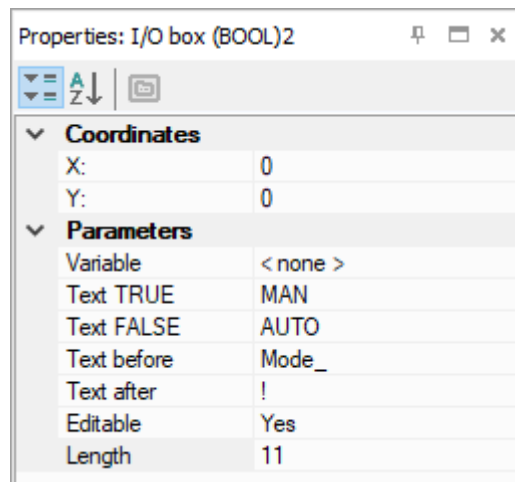
The option enables to display a REAL variable most precisely for the set number of reserved characters (parameter **Digits**). To use the option, select in the workspace an I/O-Box display element with associated variable of REAL type and select **Auto** for the parameter **Decimal digits** in the Property Box.

Example:

To display the variable VAR1, 4 digits with Auto-precision are reserved. The value 1.546745 will be displayed rounded as 1.547. If the value changes to 110.478696, it will be displayed as 110.5.

7.6.3 I/O box (BOOL)

I/O box (BOOL) is used to display a variable of BOOL type. The value of the variable can be changed with the device function buttons.



Parameters

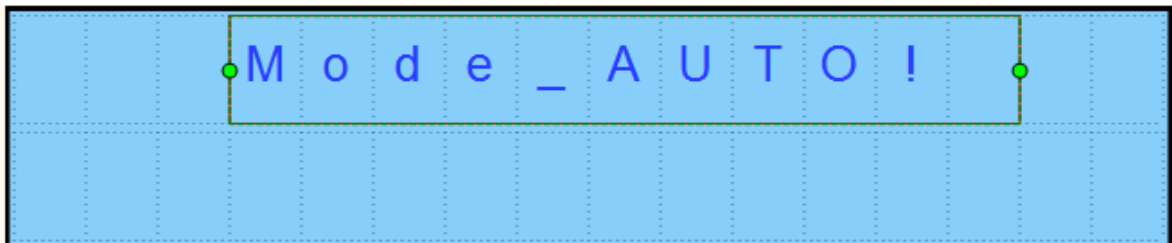
- **Variable** – the reference to a variable. Use the icon «...» in the input field to select the variable.
- **Text TRUE** – the text displayed if the variable is **True**.
- **Text FALSE** – the text displayed if the variable is **False**.
- **Text before** – the text to the left of the displayed variable.
- **Text after** – the text to the right of the displayed variable.
- **Editable** – if set to **Yes**, the displayed value can be changed using the device function buttons.



NOTE

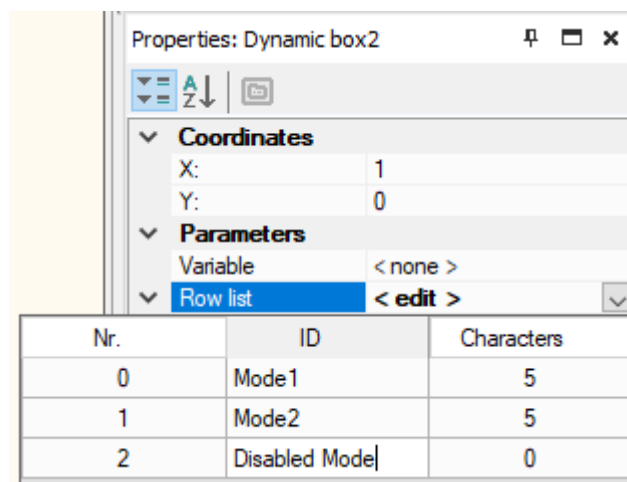
An output variable should be selected. The option has no effect with an input variable.

- **Length** – the total number of reserved characters including both the text before and after.



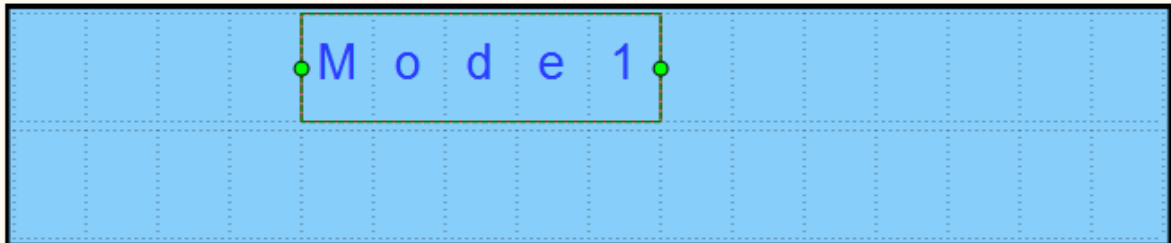
7.6.4 Dynamic box

Dynamic box is an output field. It is used to display one of the text rows from a list depending on a row **ID**. The row **ID** is saved in a referenced variable of INT type.



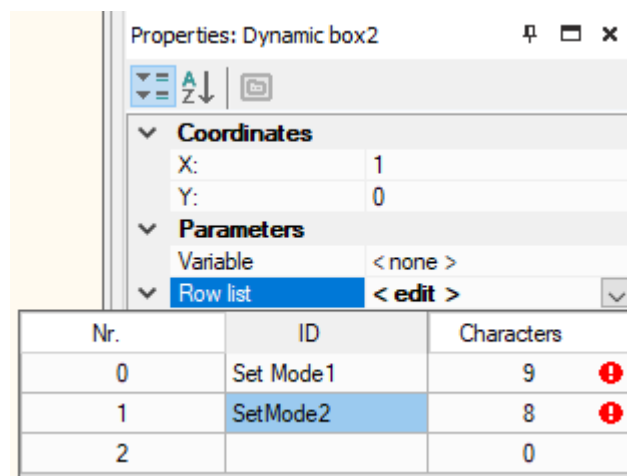
Parameters

- **Variable** — the reference to an integer project variable. To select the variable, click the «...» button and select from the variable table 6;
- **Row list** — the list with text rows. The **Text** from the row is displayed if the value of the referenced variable equals to the row **ID**. The column **Characters** shows the number of characters in the text. If the value of **Length** parameter is exceeded, there will be an exclamation mark near the number.
- **Length** — the number of reserved characters.



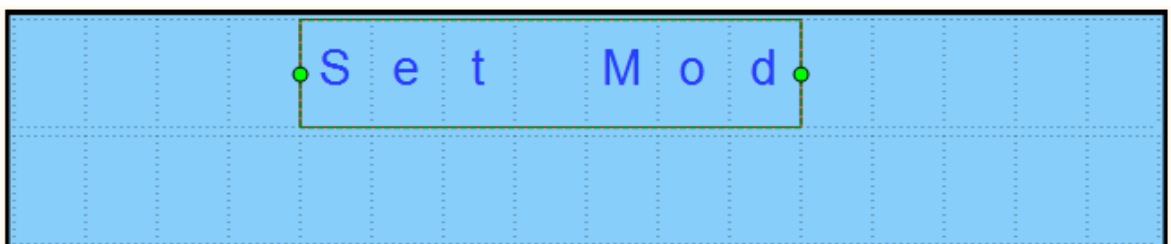
7.6.5 ComboBox

ComboBox is an input / output field. It is used to display one of the text rows from a list depending on a row **ID**. The row **ID** is saved in a referenced variable of INT type. The **ID** can also be selected using the device function buttons.



Parameters

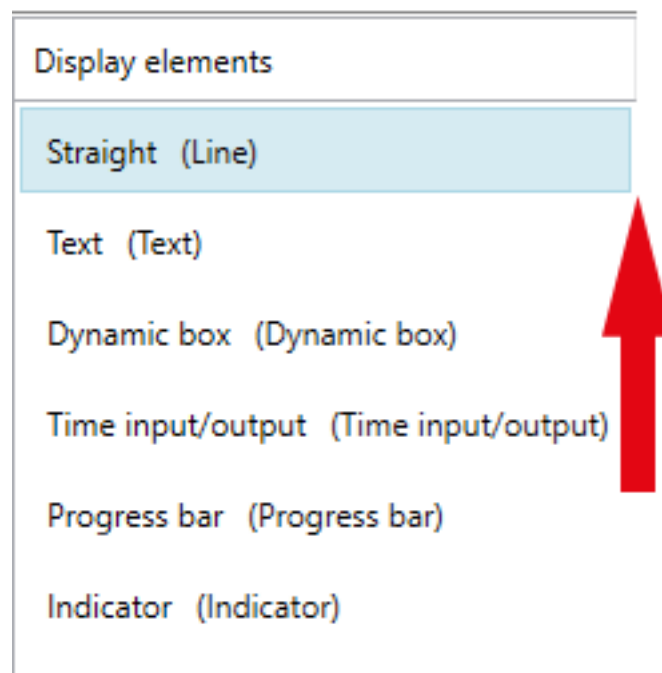
- **Variable** — the reference to a program variable. Use the icon «...» in the input field to select the variable.
- **Row list** — the table with text rows. The **Text** of the selected row is displayed and the row **ID** is saved in the referenced output variable. The column **Characters** shows the number of characters in the text. If the value of **Length** parameter is exceeded, there will be an exclamation mark near the number.
- **Length** — the number of the reserved characters.



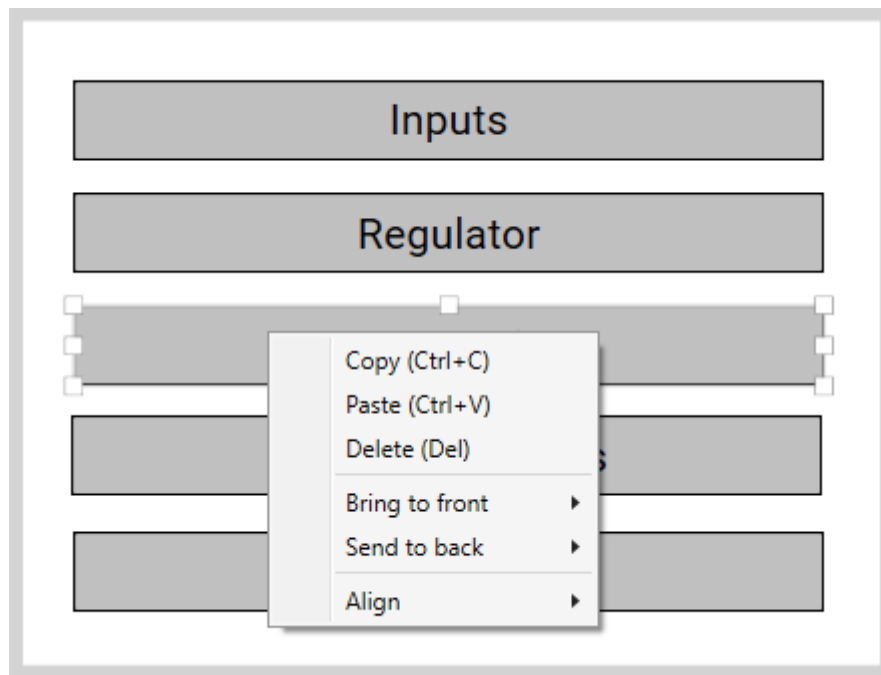
7.7 Basic graphic elements

- Text
- Indicator
- Progress bar
- Dynamic text
- Input/output int/float
- Time input/output
- IP Input/Output
- Line
- Polygon
- Circle
- Button 7.7.11
- Switch 7.7.12
- Switch group 7.7.13
- Image
- Menu
- Graph

These elements are used to design the display of devices with a graphic color LCD. If several editable visualization elements are added to the screen, the order of their selection using the SEL button will be determined by the order in the "Screen Components" list from bottom to top (the lower elements will be highlighted first):



You can change the order of elements by dragging and dropping an element in the list or using the context menu by right-clicking on an element on the canvas:

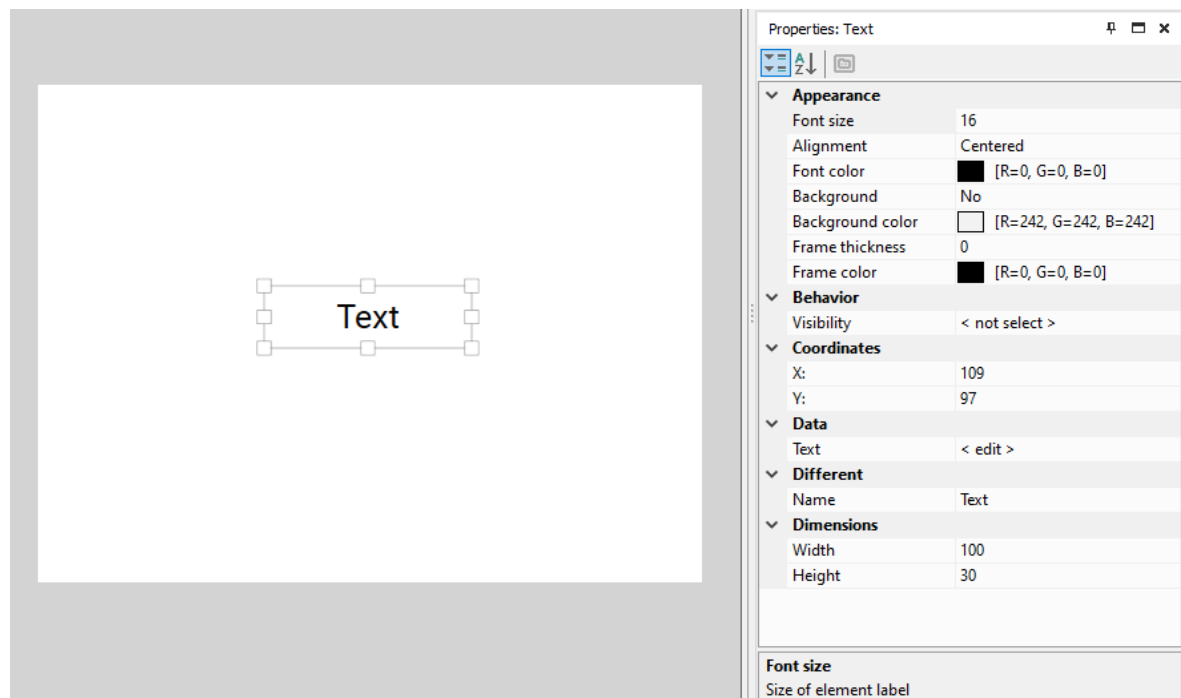


7.7.1 Text

The **Text** element is intended for placing a text block in the visualization screen.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions, enter text and set other parameters on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis – from left to right, the final value is determined by the size of the element and the width of the screen;

- along the Y axis – from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

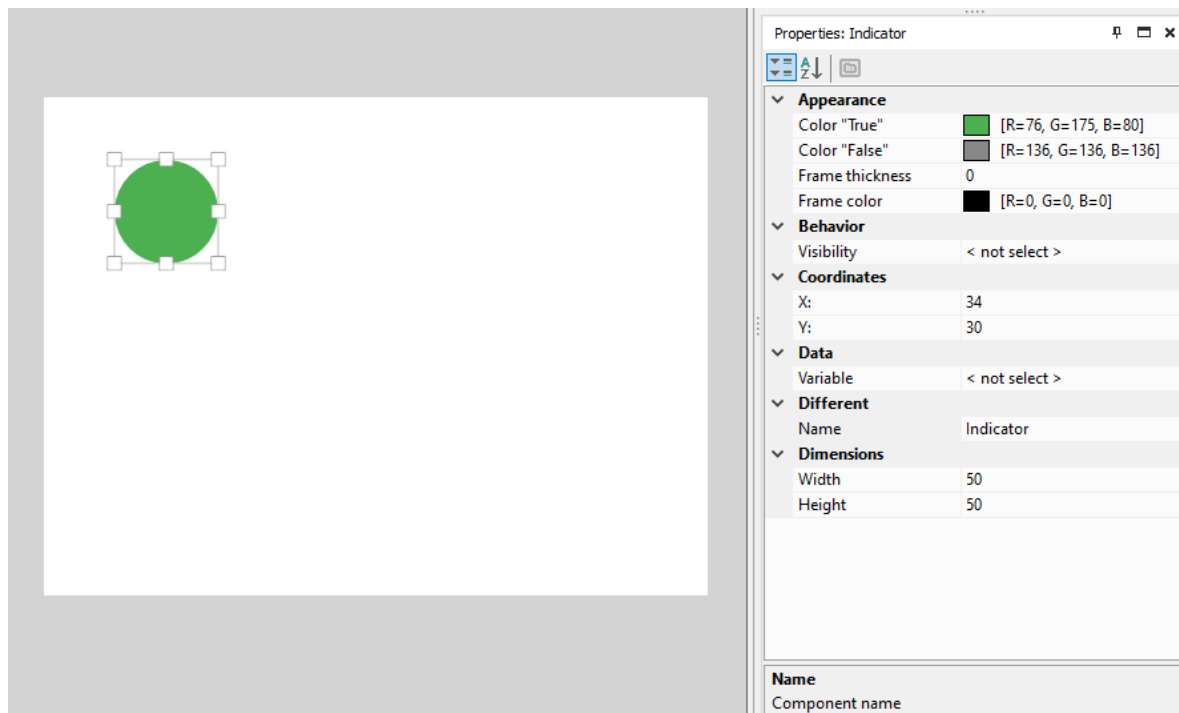
Group	Parameter	Description
Appearance	Font size	Size of text displayed in element: 16, 32, 48
	Alignment	Alignment of the text displayed in the element: left, center, right
	Font color	The color of the text displayed in the element
	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Text	The field contains the text that will be displayed in the element
Different	Name	Displayed in the list of used elements on the left side of the visualization editor

7.7.2 Indicator

The **Indicator** element is designed to display the value of a Boolean variable in the visualization screen. If the variable value is **1**, the indicator takes the color "True". If the variable value is **0**, the indicator takes the color "False".

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

Group	Parameter	Description
Appearance	Color "True"	The color of the element if the value of the bound variable is TRUE
	Color "False"	The color of the element if the value of the bound variable is FALSE
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Variable	A boolean variable is bound to the parameter for display
Different	Name	Displayed in the list of used elements on the left side of the visualization editor



NOTE

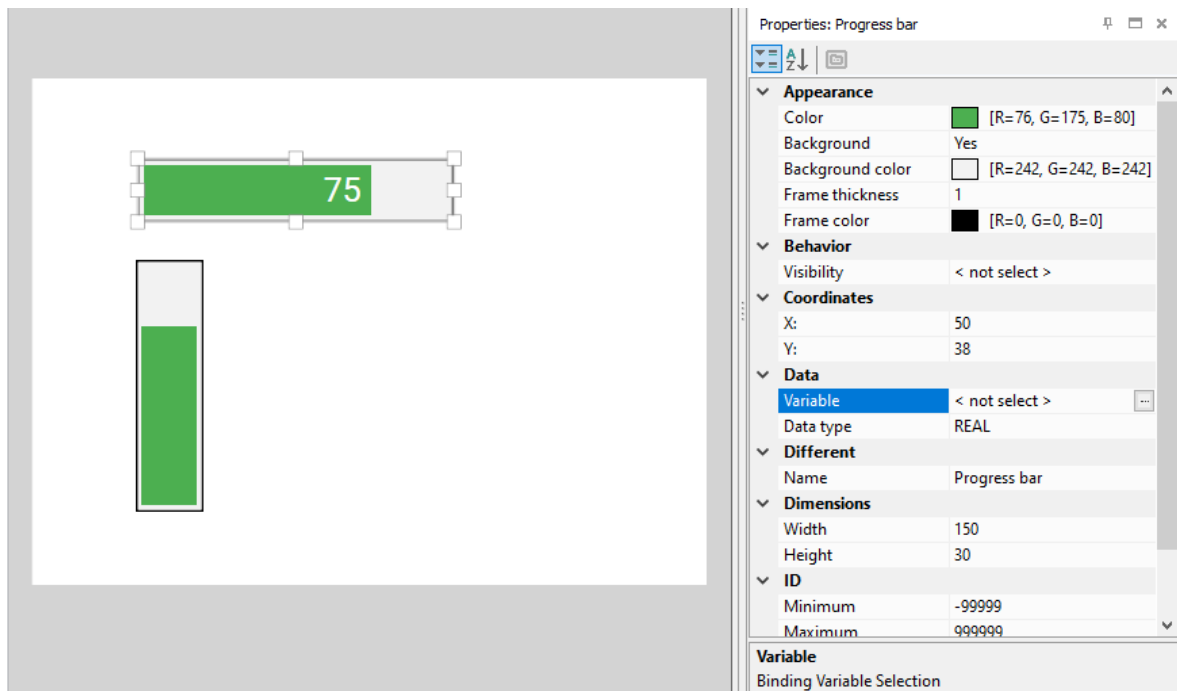
If no variable is bound to the **Variable** parameter, the color of the element will always be **True**.

7.7.3 Progress bar

The Progress bar element is designed to display the value of an integer or floating-point variable as a scale on the visualization screen.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The element's dimensions are defined by the X and Y axes and are set in the properties panel or by changing the element's borders on the screen field. For vertical orientation of the element, the Y value must be greater than the X value.

Parameters

Group	Parameter	Description
Appearance	Color	Element scale color
	Background	Background of the displayed element
	Background color	Background color of the displayed element
	Frame thickness	Element frame thickness
	Frame color	Element border color

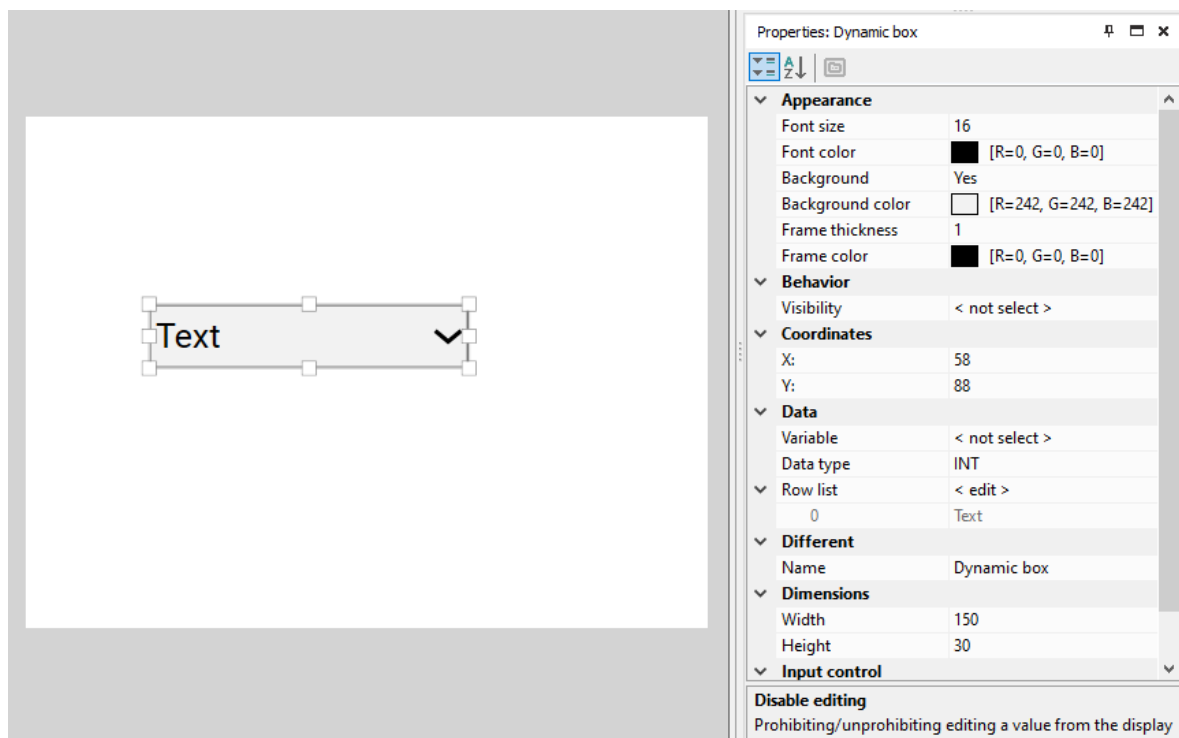
Group	Parameter	Description
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Variable	An integer or floating point variable is bound to the parameter for display.
	Variable type	Type of variable bound to element: floating point, integer
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
ID	Minimum	0 % of element scale
	Maximum	100 % of the measurement scale
	Visibility	Displaying the scale value in an element
	Font size	The font size of the text displayed in the element: 16, 32, 48

7.7.4 Dynamic box

The **Dynamic box** element is designed to select and display the value of an integer variable from a drop-down list on the visualization screen. The integer value can be assigned a text designation, which will be displayed in the drop-down list.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions, and set the parameters and data on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:


- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;

- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

Group	Parameter	Description
Appearance	Font size	The size of the text displayed in the element: 16, 32, 48
	Font color	The color of the text displayed in the element
	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element.
Data	Variable	A variable is bound to the parameter to store values
	Data type	Selecting the data type of the bound variable: INT and BOOL <div>  NOTE If a Boolean variable type is selected, the element's drop-down list can contain only two lines. If an integer variable type is selected, the drop-down list can contain a maximum of 128 rows. </div>
	Row list	In tabular form, the variable values and their corresponding names, which will be displayed in the element's drop-down list
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
Input control	Disable editing	If set to Yes , the value of the bound variable can only be viewed, the input option will not be available. Otherwise, entering a value in the element overwrites the variable value

To add a new line to an element's drop-down list, follow these steps:

1. Double-click the **Dynamic box** element in the screen editor.
2. In the window that opens, right-click on one of the rows in the table.
3. Select **Insert row below**.

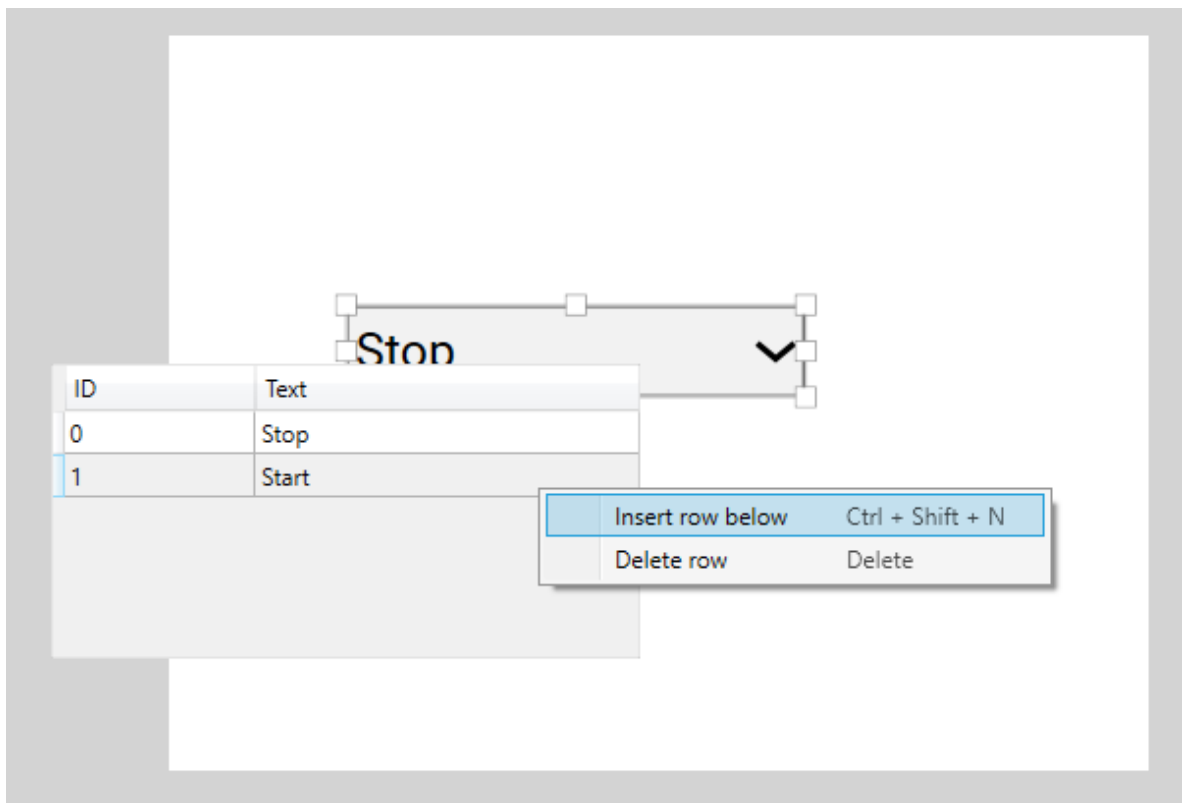
**Example:**

Table 7.7 List of lines

0	Stop
1	Start
2	Pause

7.7.5 REAL/INT input/output

The **REAL/INT input/output** element is intended for inputting and displaying the value of an integer or floating-point variable on the visualization screen.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions, and set the parameters and data on the properties panel.

Coordinates

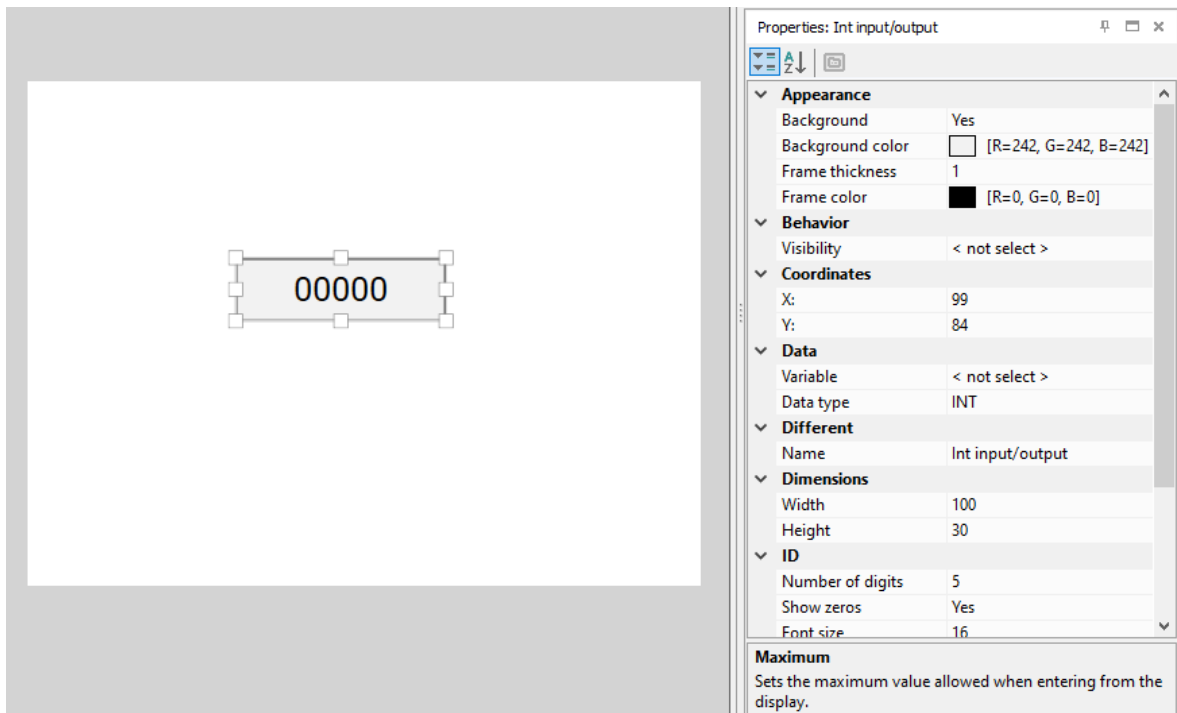
The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters (int)



Group	Parameter	Description
Appearance	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Variable	A variable is bound to the parameter
	Data type	To input/output an integer value, specify the Integer type.
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
ID	Number of digits	The number of digits displayed in the element
	Show zeros	Setting up the display of trailing zeros
	Font size	The size of the text displayed in the element: 16, 32, 48
	Font color	The color of the text displayed in the element
	Alignment	Alignment of the text displayed in an element
Input control	Disable editing	If set to Yes , the value of the bound variable can only be viewed, the input option will not be available. Otherwise, entering a value in the element overwrites the variable value
	Minimum	Minimum value to enter in an element
	Maximum	Maximum value for input in element

**NOTE**

If the output variable is outside the limits specified in the **Minimum** / **Maximum** parameters, then the element on the device will display the current state of the variable and highlight the frame of this element in red.

Parameters (float)

The screenshot shows a visualization editor with a central element displaying the value "-000.00". To the right, the "Properties: Float input/output" panel is open, showing various configuration options:

- Appearance:** Background (Yes), Background color (white, [R=242, G=242, B=242]), Frame thickness (1), Frame color (black, [R=0, G=0, B=0]).
- Behavior:** Visibility (< not select >).
- Coordinates:** X: 99, Y: 84.
- Data:** Variable (< not select >), Data type (REAL).
- Different:** Name (Float input/output).
- Dimensions:** Width (100), Height (30).
- ID:** Number of digits (5), Decimal point position (2), Show zeros (Yes), Font size (16).

At the bottom of the properties panel, the "Data type" section shows "Binding variable type".

Group	Parameter	Description
Appearance	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Date	Variable	A variable is bound to the parameter
	Data type	To input/output an integer value, specify the Integer type.
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
ID	Number of digits	The number of digits displayed in the element
	Decimal point position	The number of digits in the fractional part of the value displayed in the element
	Show zeros	Setting up the display of trailing zeros
	Font size	The size of the text displayed in the element: 16, 32, 48
	Font color	The color of the text displayed in the element
	Alignment	Alignment of the text displayed in an element

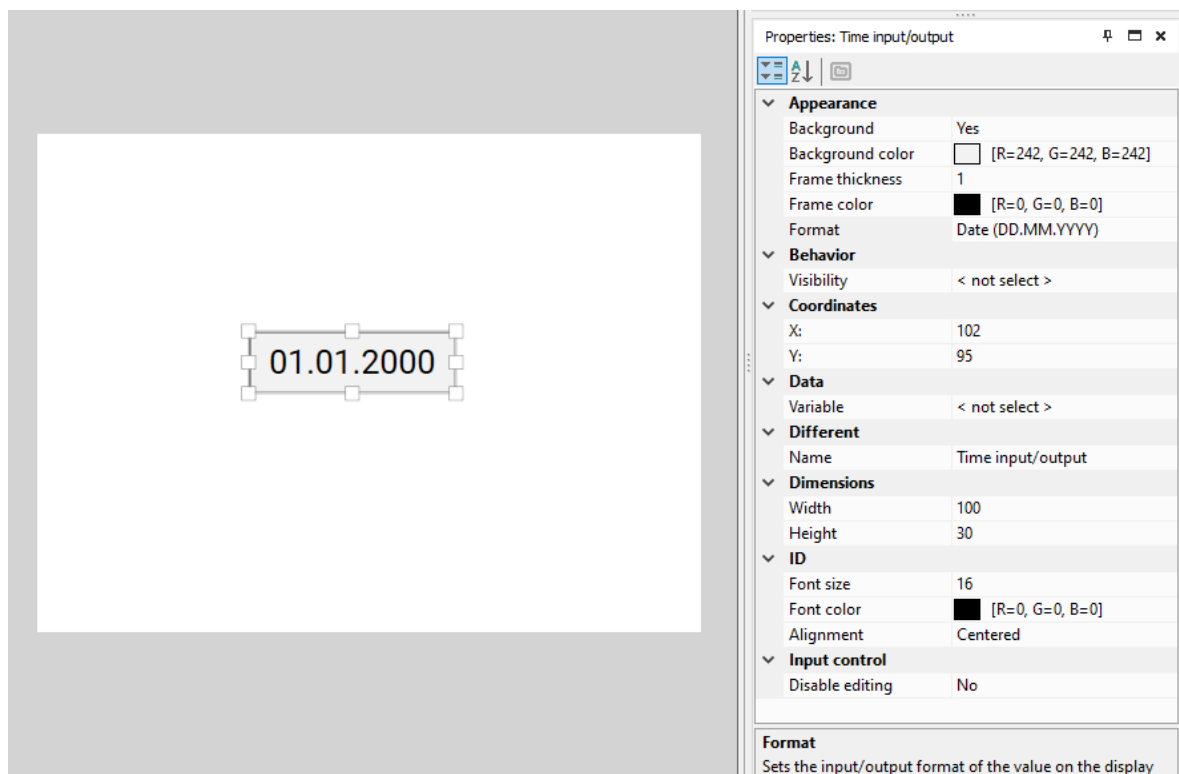
Group	Parameter	Description
Input control	Disable editing	If set to Yes , the value of the bound variable can only be viewed, the input option will not be available. Otherwise, entering a value in the element overwrites the variable value
	Minimum	Minimum value to enter in an element
	Maximun	Maximum value for input in element

7.7.6 Time input/output

The **Time input/output** element is used to input and display the date and time value on the visualization screen. An integer variable is used to store the date and time value, which stores the number of seconds since 00:00:00 January 01, 2000.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions, and set the parameters and data on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

Group	Parameter	Description
Appearance	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
	Format	Select from the drop-down list the format for displaying the date and time. Format notations: – ss – seconds; – mm – minutes; – hh – hours; – DD – day; – MM – month; – YYYY – year.
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Variable	A variable is bound to the parameter
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
ID	Font size	The size of the text displayed in the element: 16, 32, 48
	Font color	The color of the text displayed in the element
	Alignment	Alignment of the text displayed in an element
Input control	Disable editing	If set to Yes , the value of the bound variable can only be viewed, the input option will not be available. Otherwise, entering a value in the element overwrites the variable value

7.7.7 Input/output of IP address

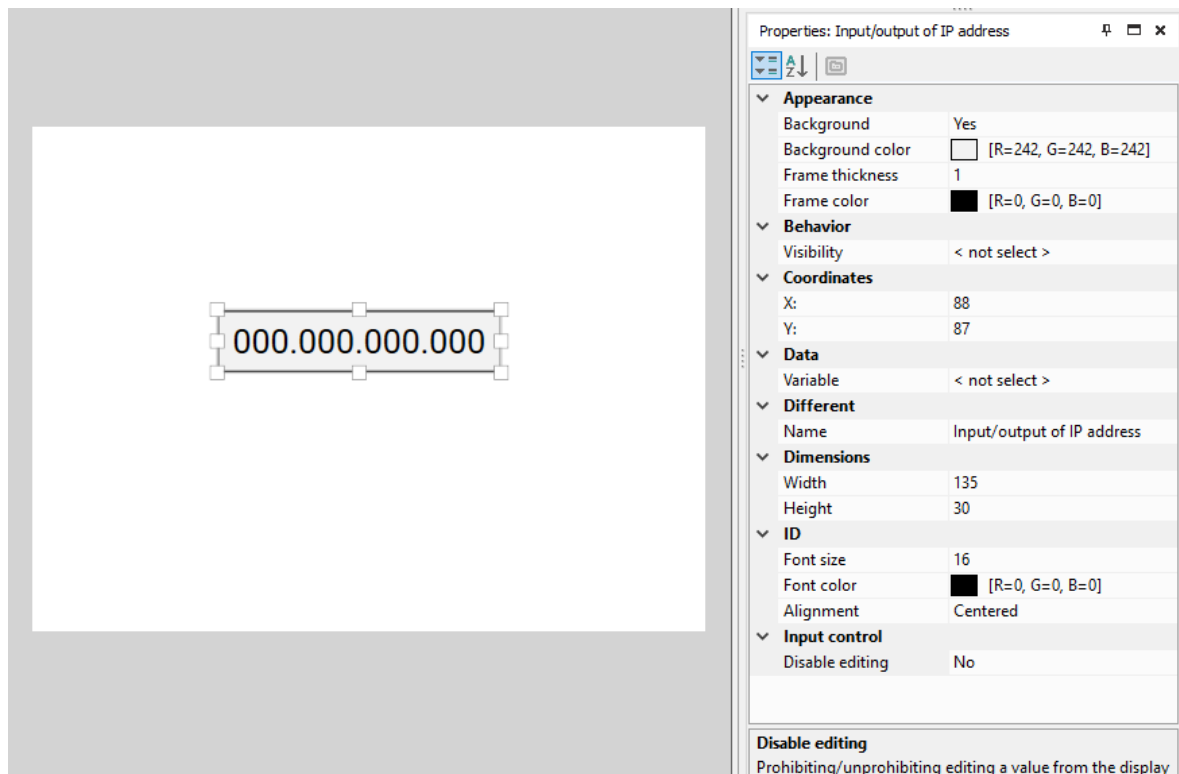
The **Input/output of IP address** element is used to enter and display the IP address (IPv4) value on the visualization screen. An integer variable is used to store the IP address value, the value of which is calculated using the formula:

$$int = o1 * 256^3 + o2 * 256^2 + o3 * 256^1 + o4 * 256^0,$$

where *int* is the value of an integer variable;
o1, *o2*, *o3*, *o4* – IP address octets.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions, and set the parameters and data on the properties panel.



Coordinates

The element's location is determined by the X and Y axes. The coordinates are set in the properties panel or by moving the element across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

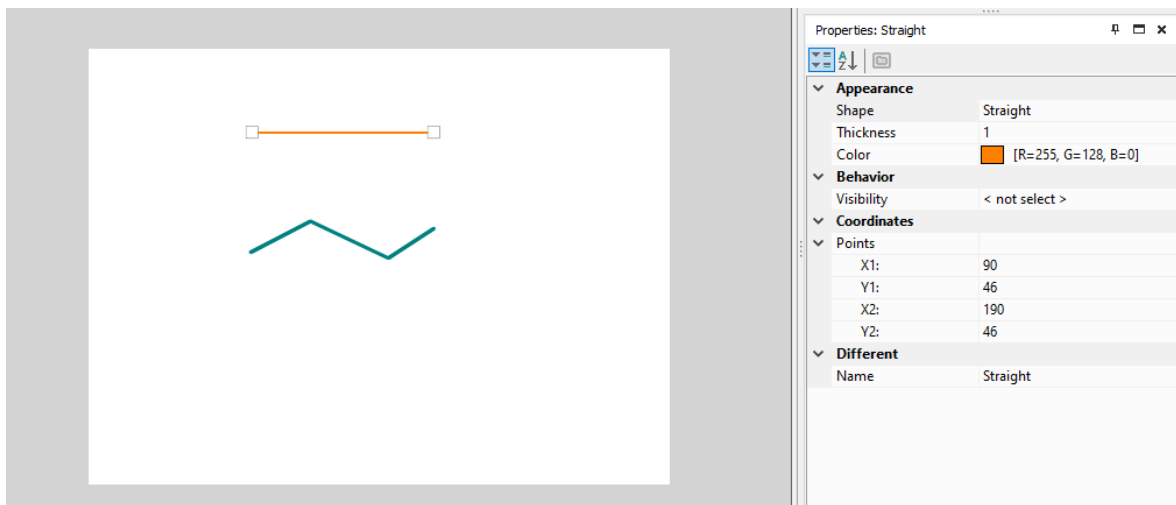
Group	Parameter	Description
Appearance	Background	The background of the text displayed in the element
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element border color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
Data	Variable	A variable is bound to the parameter
ID	Font size	Size of text displayed in element: 16, 32, 48
	Font color	The color of the text displayed in the element

Group	Parameter	Description
	Alignment	Alignment of the text displayed in an element
Input control	Disable editing	If set to Yes , the value of the bound variable can only be viewed, the input option will not be available. Otherwise, entering a value in the element overwrites the variable value

7.7.8 Line

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The location of the straight line is determined by two points, the coordinates of which along the X and Y axes are specified in the properties panel or by moving the points across the screen. The coordinates along both axes start from 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

The location of the broken line is determined by four points.

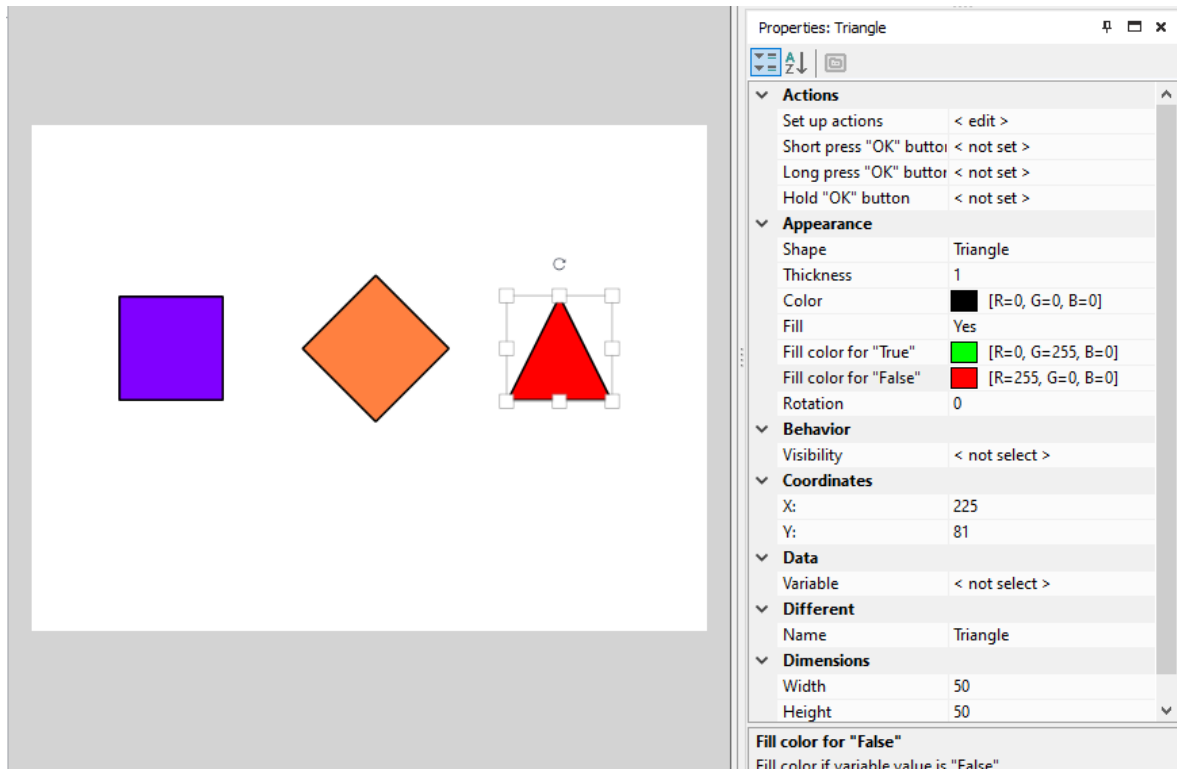
Parameters

Group	Parameter	Description
Appearance	Shape	Display line shape: straight, polyline
	Thickness	Line thickness
	Color	Element color
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Different	Name	Displayed in the list of used elements on the left side of the visualization editor

7.7.9 Polygon

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The polygon's location is determined by the coordinates of its center along the X and Y axes. The coordinates are set in the properties panel or by moving the object across the screen. The coordinates along both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

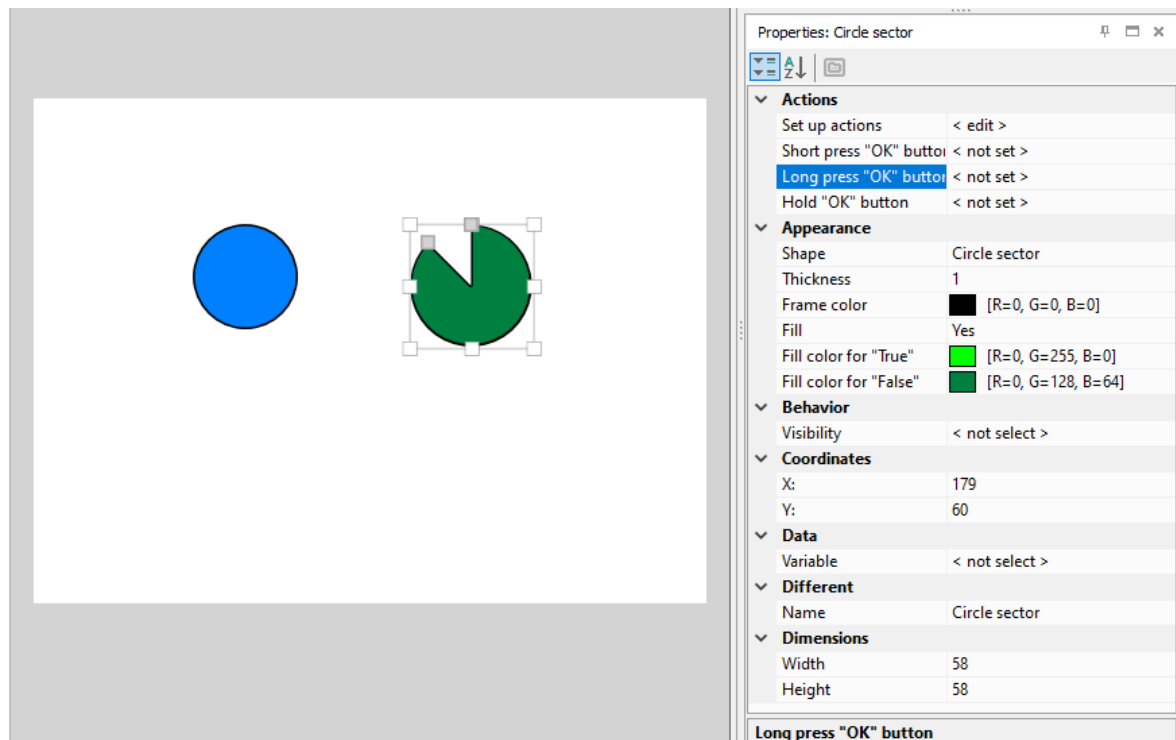
Group	Parameter	Description
Actions	Set up actions	Select an action to navigate between screens
	Short press "OK" button	List of actions performed by short pressing the "OK" button
	Long press "OK" button	List of actions performed by long pressing the "OK" button
	Hold "OK"	List of actions performed when holding the "OK" button
Appearance	Shape	The shape of the displayed polygon: triangle, rectangle, rhombus

Group	Parameter	Description
	Thickness	Polygon outline thickness
	Color	Polygon outline color
	Fill	Presence of fill in the figure
	Fill color for «True»	Fill color of the shape if the bound variable is True
	Fill color for «False»	The fill color of the shape if the bound variable is False
	Rotation	Rotate an element around its center in degrees
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element
Data	Variable	A variable is bound to the parameter
Different	Name	Displayed in the list of used elements on the left side of the visualization editor

7.7.10 Circle

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The location of the circle is determined by the coordinates of its center along the X and Y axes. The coordinates are set in the properties panel or by moving the object across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;

- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

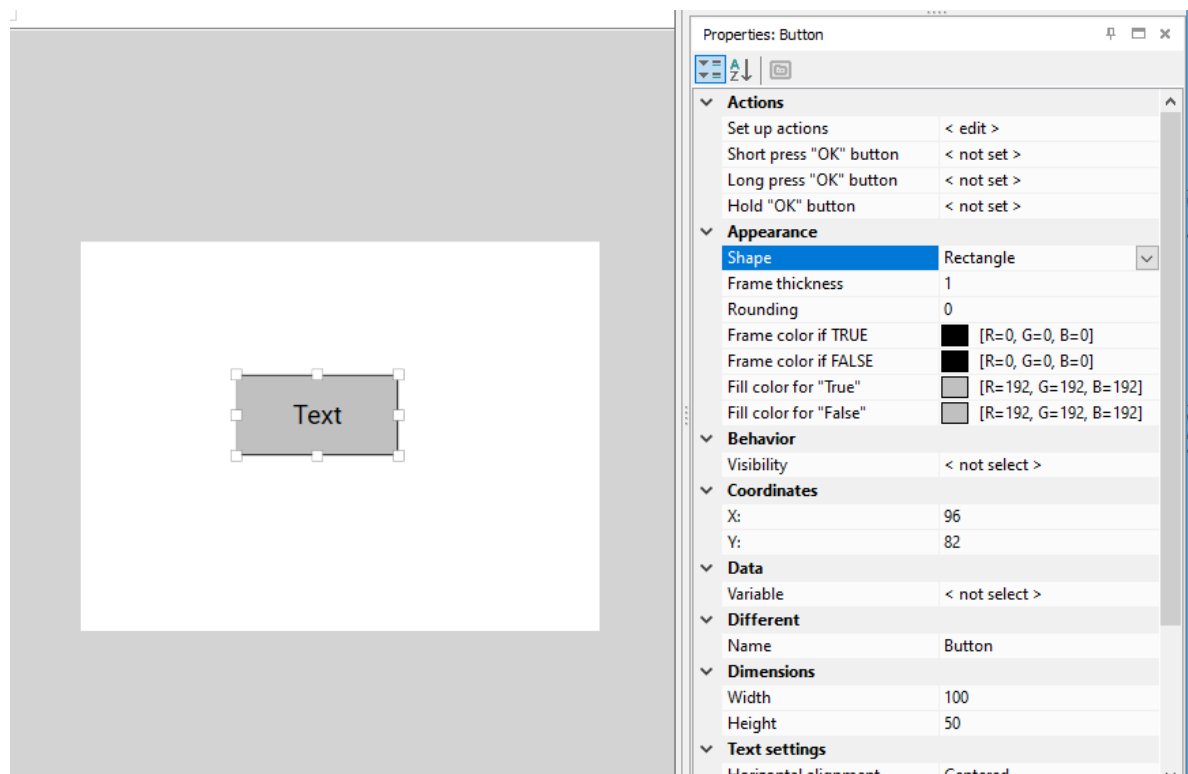
Group	Parameter	Description
Actions	Set up actions	Select an action to navigate between screens
	Short press "OK" button	List of actions performed by short pressing the OK button
	Long press "OK" button	List of actions performed by long pressing the OK button
	Hold "OK" button	List of actions performed when holding the OK button
Appearance	Shape	The shape of the displayed circle: circle, circle sector
	Thickness	Thickness of the outline of the figure
	Frame color	Color of the outline of the figure
	Filling	Presence of fill in the figure
	Fill color for "True"	The fill color of the shape if the bound variable is True
	Fill color for "False"	The fill color of the shape if the bound variable is False
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element.
Data	Variable	A variable is bound to the parameter
Different	Name	Displayed in the list of used elements on the left side of the visualization editor

7.7.11 Button

The **Button** element is designed to switch the state of a Boolean variable.

Setting up properties

For correct display on the device screen, you should specify the location coordinates, element dimensions and set the parameters on the properties panel.



Coordinates

The location of the button is determined by the coordinates of its center along the X and Y axes. The coordinates are set in the properties panel or by moving the object across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Parameters

Group	Parameter	Description
Actions	Set up actions	Select an action to navigate between screens
	Short press "OK" button	List of actions performed by short pressing the "OK" button
	Long press "OK" button	List of actions performed by long pressing the "OK" button
	Hold "OK" button	List of actions performed when holding the "OK" button
Appearance	Shape	The shape of the button to display: rectangle or circle
	Frame thickness	Button outline thickness
	Rounding	<div> <div>i</div> <div> NOTE Available only when the Rectangle shape is selected. </div> </div>

Group	Parameter	Description
		Rounding the corners of the frame
	Frame color if TRUE	The fill color of the frame if the value of the bound variable is "True"
	Frame color if FALSE	The fill color of the frame if the bound variable is "False"
	Fill color for "True"	The fill color of the shape if the value of the bound variable is "True"
	Fill color for "False"	The fill color of the shape if the bound variable is "False"
Behavior	Visibility	A Boolean variable is bound to the parameter, which will determine the visibility of the element.
Data	Variable	A variable is bound to the parameter
Different	Name	Displayed in the list of used elements on the left side of the visualization editor
Text settings	Horizontal alignment	Alignment of the text displayed in the element: left, center, right
	Vertical alignment	Alignment of the text displayed in the element: top, center, bottom
	Font size	Size of text displayed in element: 16, 32, 48
	Text "True"	The text to display when the button is pressed
	Text "False"	The text to display when the button is released
	Text color "True"	The color of the text displayed when the button is pressed
	Text color "False"	The color of the displayed text when the button is released

7.7.12 Switch

The **Switch** element is designed to control the state of a Boolean variable, as well as to display its current state.

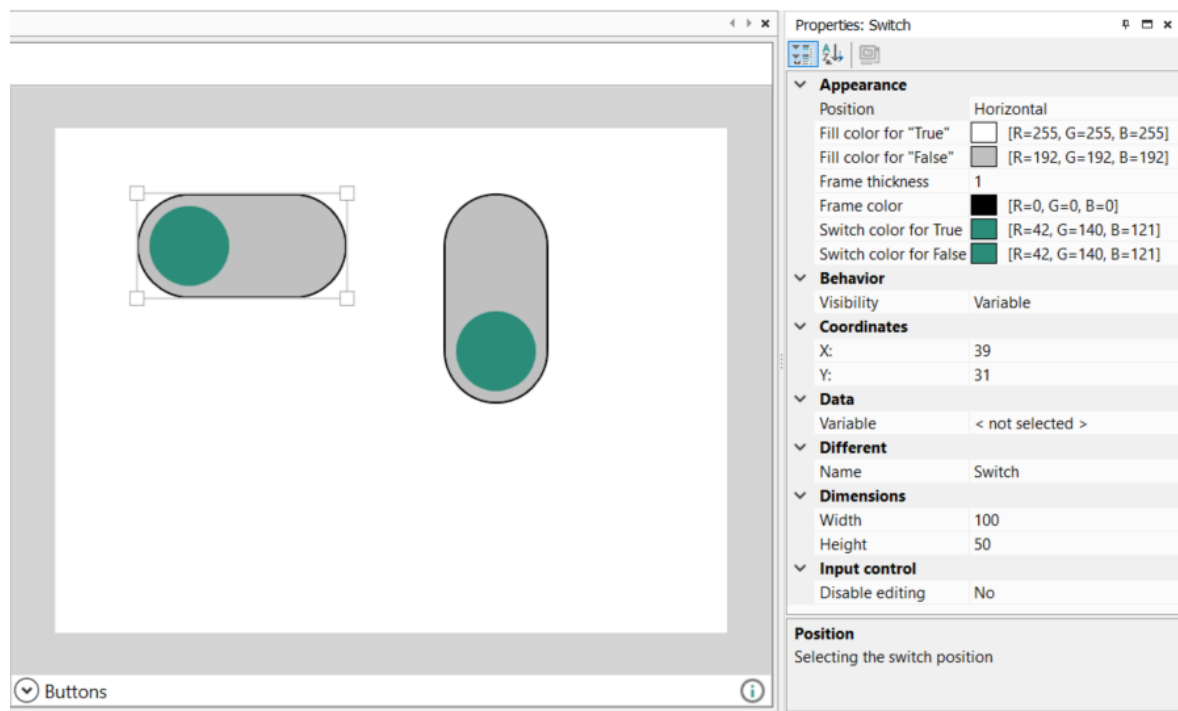


Fig. 7.1 Appearance and Properties of the Switch element

Coordinates

The location of the button is determined by the coordinates of its center along the X and Y axes. The coordinates are set in the properties panel or by moving the object across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;
- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

Properties

To configure the required type of **Switch** element on the device screen, the following properties are available:

Group	Parameter	Description
Appearance	Position	Vertical or horizontal switch placement
	Fill color for "True"	Fill color of the element if the switch is on
	Fill color for "False"	Fill color of the element when the switch is off
	Frame thickness	Element outline thickness
	Frame color	Element outline color
	Switch color for True	The color of the switch when it is on
	Switch color for False	The color of the switch when it is off
Behavior	Visibility	A Boolean variable is bound to the parameter, which controls the element's visibility

Group	Parameter	Description
Data	Variable	A Boolean variable is bound to the parameter
Different	Name	It appears in the list of used elements on the left side of the visualization editor
Input control	Disable editing	Enabling or disabling the ability to change the switch value from the device screen

7.7.13 Switch group

The **Switch group** element is designed to manage parameter lists and display the status of these lists:

- **Radio button** – allows selection of only one parameter from the list.
- **Checkbox** – allows selection of one or multiple parameters from the list.

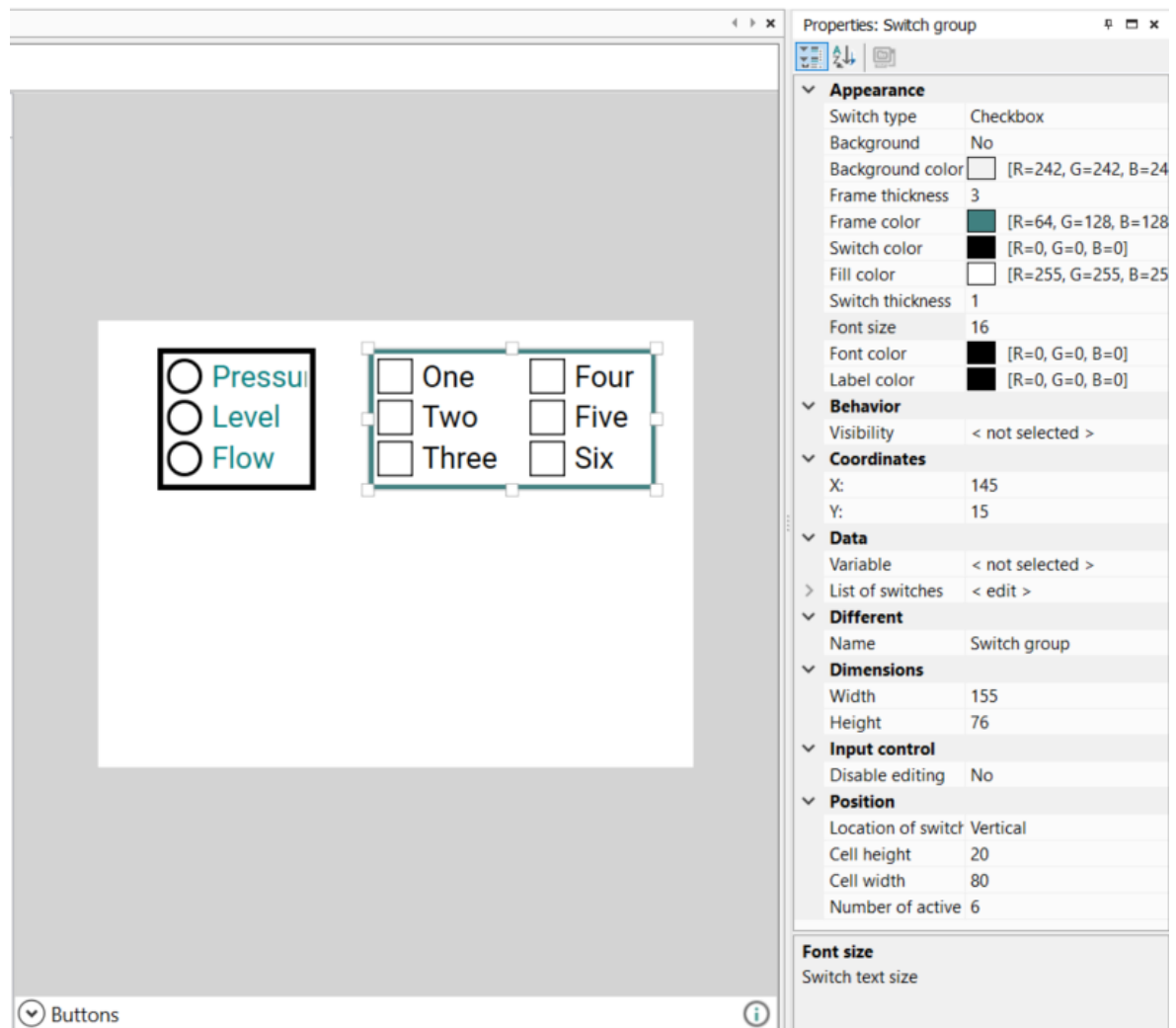


Fig. 7.2 Appearance and Properties of the Switch Group element

Coordinates

The location of the button is determined by the coordinates of its center along the X and Y axes. The coordinates are set in the properties panel or by moving the object across the screen. The coordinates for both axes start at 0:

- along the X axis - from left to right, the final value is determined by the size of the element and the width of the screen;

- along the Y axis - from top to bottom, the final value is determined by the size of the element and the height of the screen.

Dimensions

The dimensions of an element are determined by the X and Y axes and are set in the properties panel or by changing the boundaries of the element on the screen.

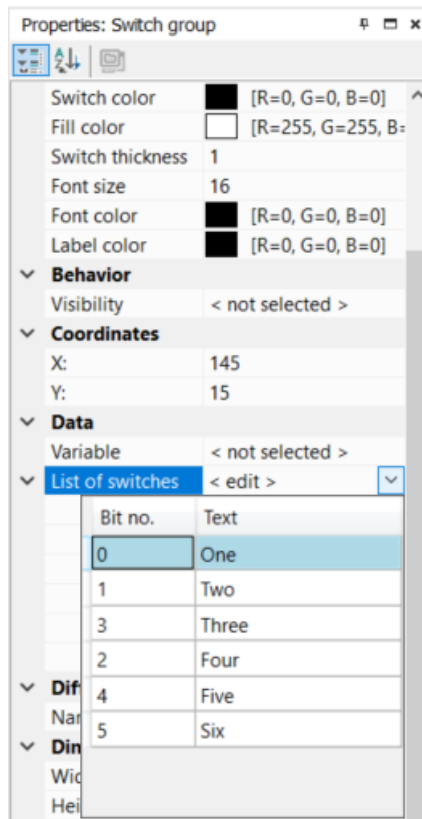
Properties

To configure the desired appearance of the **Switch Group** element on the device screen, the following properties are available:

Group	Parameter	Description
Appearance	Switch type	Select the switch type: radio button or checkbox
	Background	Presence or absence of a background for the text displayed in the element.
	Background color	The background color of the text displayed in the element
	Frame thickness	Element frame thickness
	Frame color	Element frame color
	Switch color	Label field border color
	Fill color	Radio button/checkbox fill color
	Switch thickness	Label field border thickness
	Font size	Size of text displayed in element: 16, 32, 48
	Font color	The color of the text displayed in the element
	Label color	The color of the label being set
Behavior	Visibility	A Boolean variable is bound to the parameter, which controls the element's visibility
Data	Variable	An integer variable is bound to the parameter.
	List of switches (max. 32 lines)	The variable's values and their corresponding text to be displayed in the element.
Different	Name	It appears in the list of used elements on the left side of the visualization editor
Input control	Disable editing	Enabling/disabling the ability to interact with the element from the device screen: <ul style="list-style-type: none"> – Switching between list parameters (for the radio button element) – Enabling/disabling list parameters (for the checkbox element)
Position	Location of switches	Selection of the list layout: horizontal or vertical
	Cell height	The height of the cell for arranging list items
	Cell width	The width of the cell for arranging list items

Group	Parameter	Description
	Number of active checkboxes	<p>NOTE Available only when the switch type "Checkbox" is selected.</p> <p>Restricts the simultaneous selection of multiple parameters from the list. The limit cannot exceed the number of rows added to the Switch List.</p>

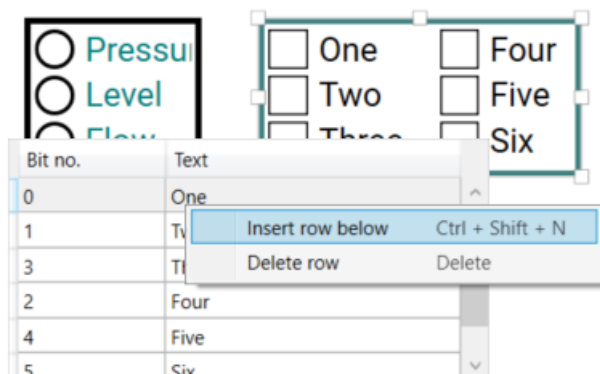
To add rows, click "Edit" and use the context menu of the table that opens:



Specify the variable's value and the text to display in the element.

Adding rows to the parameter list is also available in the editor window:

1. Double-click the element placed on the screen with the left mouse button.
2. To add rows, use the context menu of the table that opens:.



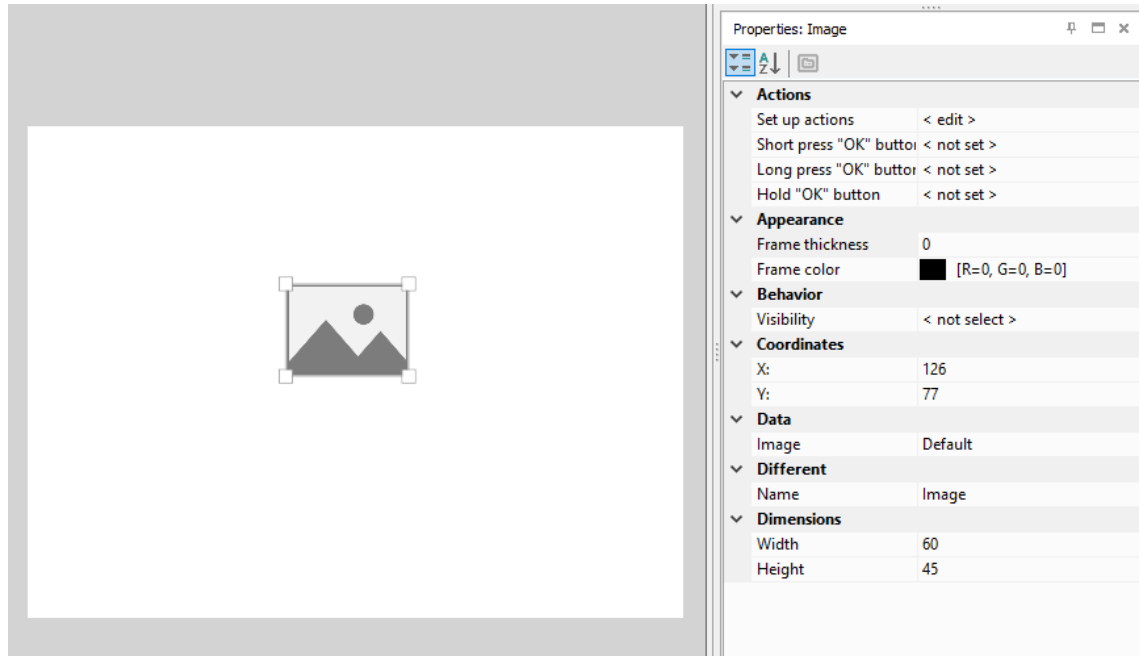
3. Specify the variable's value and the text to display in the element.

7.7.14 Image

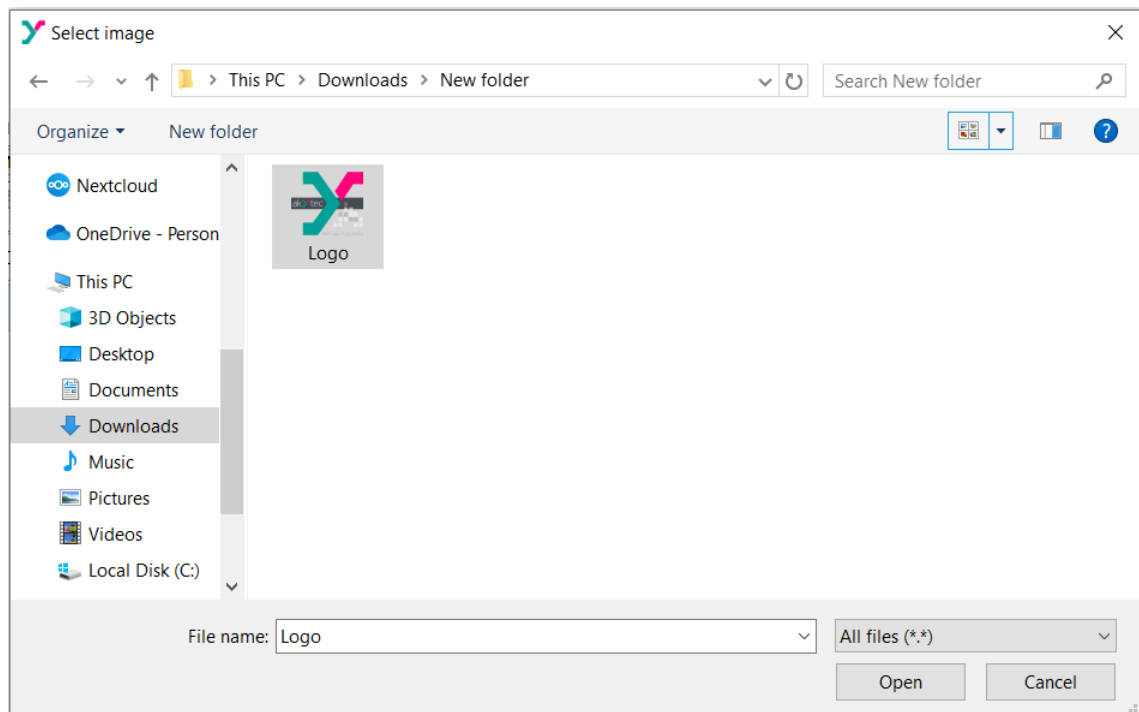
Adding a Custom Image to a Project

To add a custom image:

1. Place an **Image** object on the workspace.



2. Double-click the placed object with the left mouse button. The image selection window will open.

**NOTE**

Supported file types: *.jpg, *.jpeg, *.jpe, *.bmp.

The maximum image size is limited by the device's screen resolution. Images can be scaled down within the ALP editor if necessary.

3. Select an image and click **Open**.

Configuring Properties

To ensure the image displays correctly on the device screen, configure its location, dimensions, and other parameters in the properties panel.

Coordinates

The element's position is determined by the coordinates of its center along the X and Y axes. Coordinates can be set in the properties panel or by dragging the object on the screen. The coordinate origin (0,0) is the top-left corner:

- **X-axis:** Increases from left to right. The maximum value is constrained by the element's size and the screen width.
- **Y-axis:** Increases from top to bottom. The maximum value is constrained by the element's size and the screen height.

Dimensions

The element's size is defined by its width (X-axis) and height (Y-axis). Dimensions can be set numerically in the properties panel or adjusted visually by dragging the element's boundaries on the screen.

Parameters

Group	Parameter	Description
Appearance	Frame thickness	Sets the thickness of the image's border.
	Frame color	Sets the color of the image's border.
Behavior	Visibility	Bind a Boolean variable to control the element's visibility. The element is visible when the variable is TRUE.
Data	Image	Selects the image file to display.
Different	Name	The name displayed for this element in the list of used elements on the left side of the visualization editor.
Actions	Set up actions	Select an action to execute, such as navigating to another screen.
	Short press "OK"	Defines actions triggered by a short press of the OK button.
	Long press "OK"	Defines actions triggered by a long press of the OK button.
	Hold "OK"	Defines actions triggered by holding down the OK button.



NOTE

If the same image file is linked to multiple elements, it only occupies space once in the user visualization ROM. This is reflected in the **User visualization ROM** indicator in the status bar.

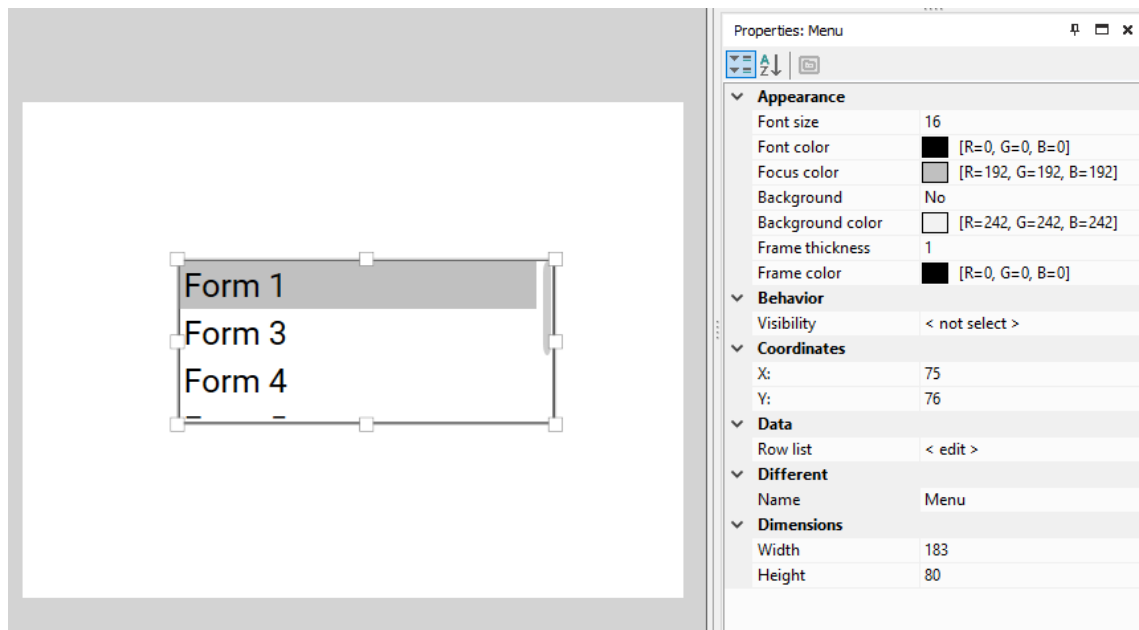
7.7.15 Menu

The **Menu** element is designed for navigation between visualization screens. Each menu item is linked to a specific screen.

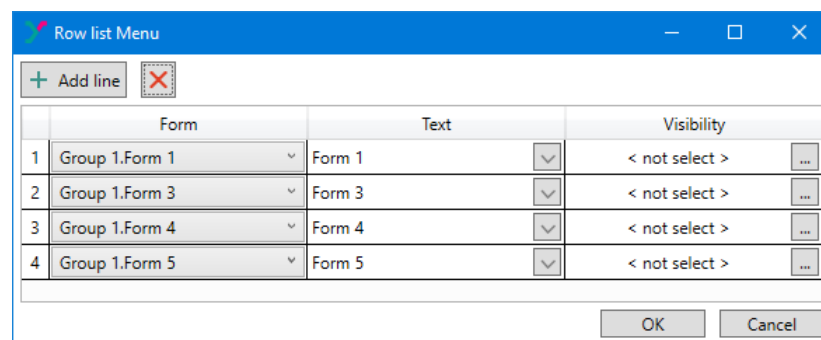
Adding a Menu to a Project

To add a **Menu** object to a visualization screen:

1. Place a **Menu** object on the workspace.



2. Double-click the placed object with the left mouse button (LMB). The line editing window will open.



3. Add the required number of menu items (lines).

NOTE
Maximum: 128 lines.

4. In the **Screen** column, select the screen to navigate to when the menu item is selected.

NOTE
All project screens except the current one are available for selection.

5. In the **Text** column, specify the text to display for the menu item.
6. Set the visibility for each item by binding Boolean variables in the **Visibility** column. The item is visible when the bound variable is TRUE.
7. Click the **OK** button to confirm and close the window.

Configuring Properties

For correct display on the device screen, configure the element's location, dimensions, and other parameters in the properties panel.

Coordinates

The element's position is determined by the coordinates of its center along the X and Y axes. Coordinates can be set in the properties panel or by dragging the object on the screen. The coordinate origin (0,0) is the top-left corner:

- **X-axis:** Increases from left to right. The maximum value is constrained by the element's size and the screen width.
- **Y-axis:** Increases from top to bottom. The maximum value is constrained by the element's size and the screen height.

Dimensions

The element's size is defined by its width (X-axis) and height (Y-axis). Dimensions can be set numerically in the properties panel or adjusted visually by dragging the element's boundaries on the screen.

Parameters

Group	Parameter	Description
Appearance	Font size	Size of the text displayed in the element: 16, 32, 48
	Font color	Color of the text displayed in the element
	Focus color	Background color of the focused menu row
	Text background	Enables or disables a background behind the text
	Background color	Color of the text background
	Frame thickness	Thickness of the element's border
	Frame color	Color of the element's border
Data	Row list	Opens the line editing window to configure menu items
Different	Name	The name displayed for this element in the list of used elements on the left side of the visualization editor.

7.7.16 Chart

The **Chart** element is designed to display the change in variable values in graphical form on the device screen.

- Up to four variables can be displayed simultaneously (data types – int and float).
- The polling time interval for drawing the chart can be configured (the maximum allowed value is 23h59m59s).
- The polling period is calculated automatically, depends on the maximum value of the x-axis, the width of the **Chart** element, and the border thickness, and must not be less than 1 second.
- The number of slices is determined by the following formula:


Number of slices = max. X / t poll, where:

- max. X – maximum value of the x-axis;
- t poll – polling time.



NOTE

Chart plotting is only available for devices with color LCD. Chart operation in simulation mode is not supported.

To add a graph to the screen, select the  **Chart** element in the Library Box and place it in the working area.

The coordinate axes will appear with the scale and grid displayed. The X axis is the time axis, the Y axis is the axis of variable values (uint or float).

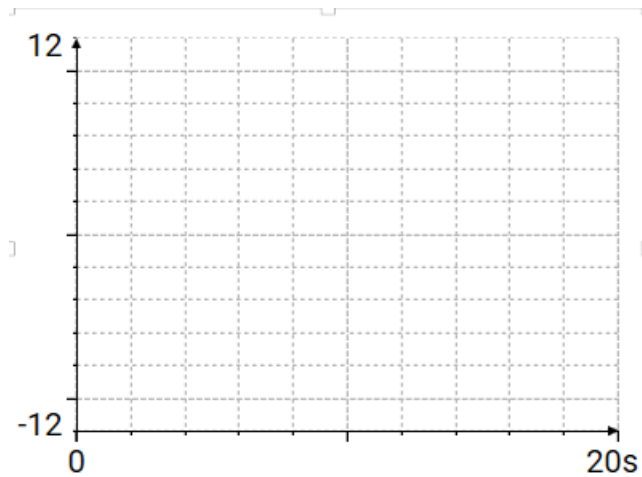


Fig. 7.3 Coordinate axes with default values

To configure parameters and add variables, use the **Property Box**:


Properties: Chart	
<div> <div></div> <div></div> <div></div> </div>	
Appearance	
Grid	Yes
Grid color	<div></div> [R=181, G=181, B=181]
Background	No
Background color	<div></div> [R=242, G=242, B=242]
Frame thickness	0
Frame color	<div></div> [R=0, G=0, B=0]
Axes	< edit >
Behavior	
Visibility	< not selected >
Coordinates	
X:	0
Y:	0
Data	
Lines	< edit >
Work in the backgrou	No
Query cycle	00:00:01
Different	
Name	Chart
Dimensions	
Width	320
Height	240
Axes Customizing the appearance of the graph axes.	

Fig. 7.4 Graph Properties Window

Customize the **Appearance** of the graph:

- presence and color of the grid

- presence and color of background
- presence, thickness and color of the frame
- axes:

Select the Axes line and left-click on <Edit> and then on the button that appears . The Axes Graph window will open:

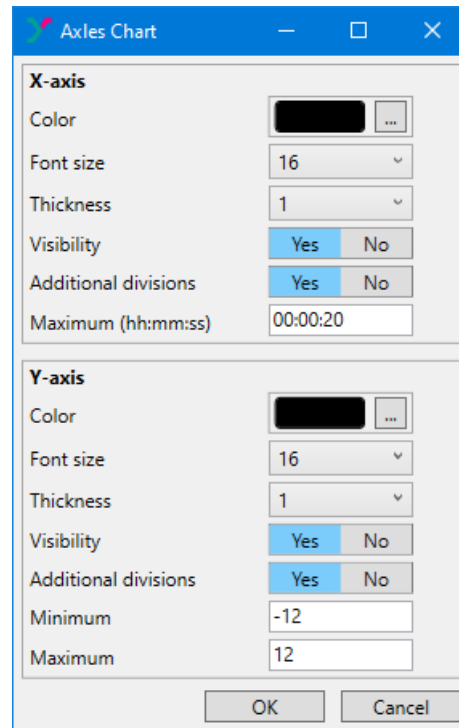


Fig. 7.5 Axis Window

If necessary, adjust the color and thickness of the axes, font size, visibility of the axes and additional tick marks. Enter the desired maximum value of the X-axis in the format hh:mm:ss. The maximum allowed value is 23:59:59. The default value is 20 seconds.



NOTE

Minimum value of X axis = 0 – non-editable parameter.

Enter the minimum and maximum values for the Y-axis. Default value: -12, 12.

If you enter an incorrect value, an error message will appear and the OK button will become inactive:

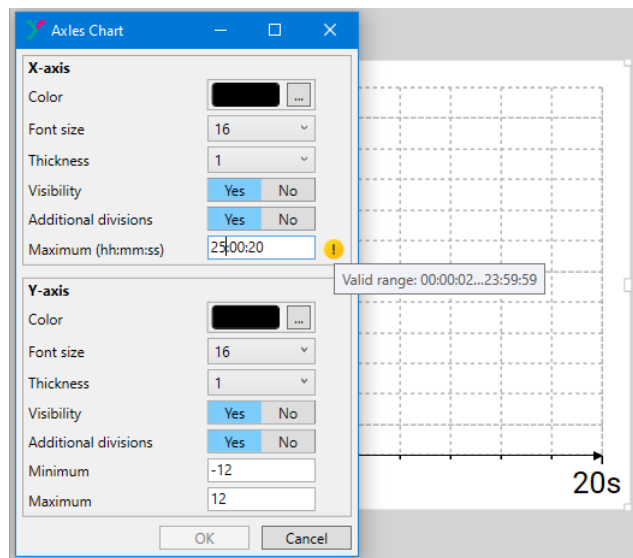



Fig. 7.6 Data entry error

To add variables to the chart, in the **Data** section of the Graph Properties window, select the Lines line and left-click on <Edit> and then on the button that appears , or double-click on the chart field. The List of lines Chart window will open:

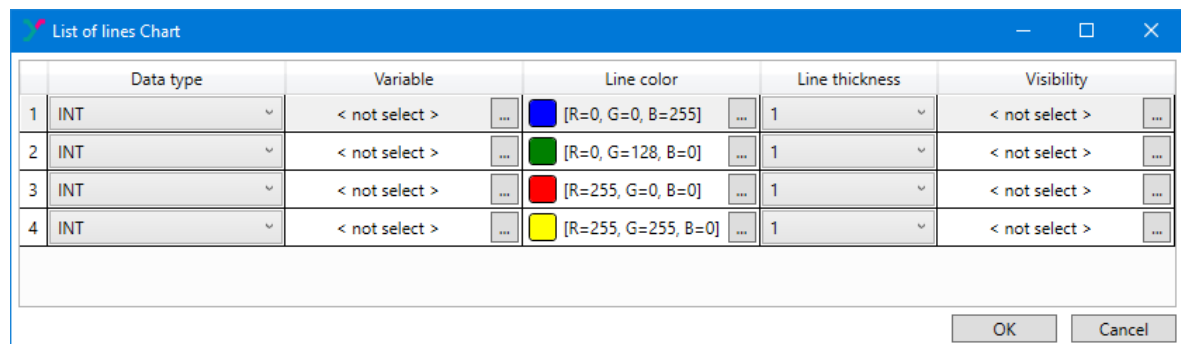


Fig. 7.7 List of lines – Chart window

In the window that opens, select:

- data type – integer / floating point
- variable
- color and thickness of the line
- visibility – it is possible to select a Boolean type variable from the Standard Variable Tables category.

NOTE

Maximum number of variables on a graph - 4.

In order to easily show the number of added variables and the settings for the lines' appearance on the chart, the lines are displayed with an offset along the Y axis relative to each other.

The Run in background (Yes/No) option in the **Data** section affects the saving of data when switching to another screen.

The polling period is a non-editable parameter, it depends on the maximum value of the X-axis, the width of the Chart element and the thickness of the frame.

NOTE

The polling period cannot be less than 1 second.

The position of the chart is determined by the coordinates of the upper left corner along the X and Y axes. The coordinates are set by moving the object across the screen. If you need a precise position

of the graph on the device screen, set the coordinates of the Property Box in the **Coordinates** section.

If necessary, in the **Behavior** section, configure the visibility of the entire graph depending on the Boolean variable.

You can change the name of a chart in the **Different** section of the Property Box.

To scale the chart, move the mouse over the corner or side marker holding down the left mouse button or set the required graph size on the screen. You can also change the size in the **Dimensions** section: the width and height are set in pixels.

After uploading the program the device screen will display a graph with the added variables:

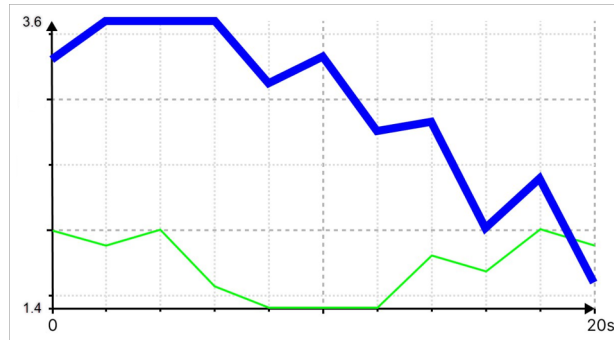


Fig. 7.8 Chart on the device screen



NOTE

If the parameter value goes beyond the minimum or maximum value of the Y axis on the device screen, the line on the chart will continue to move along the corresponding boundary.

If a variable added to the graph is deleted from the project, an information window will appear:

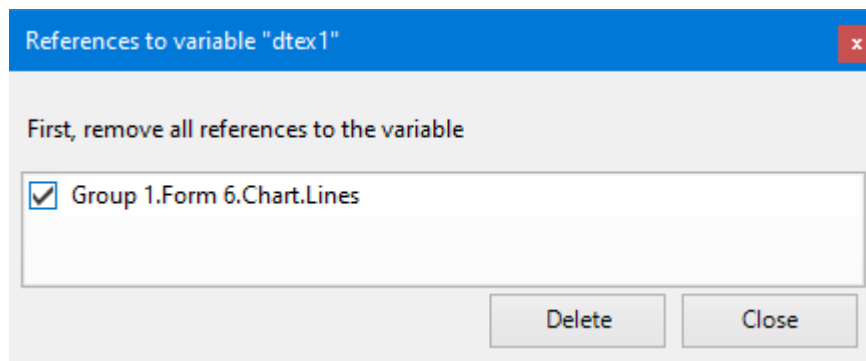


Fig. 7.9 Information window

If you select **Delete** in the Lines Graph window, the variable binding is deleted.


8 Device

This section describes the operating functions and configuration of the device:

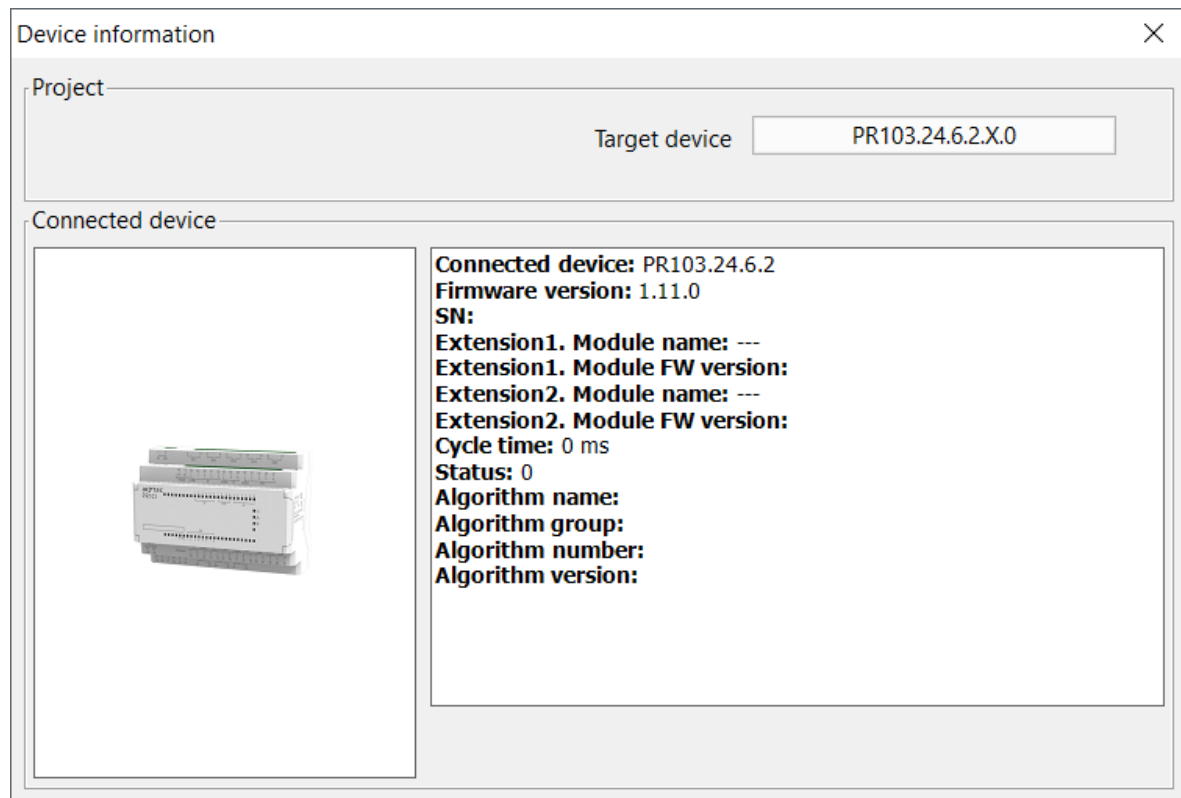
- Device information
- Cycle time
- Firmware update
- Calibration

8.1 Device information

To view information about the software, the target device and the connected device use the menu

item **Device** → **Information...** or the icon  in the toolbar.

A window containing information about the connected device appears:



The window **Device Information** contains the following information:

Target device – the device for which the project was created

Connected device – the information about the device connected to the PC

Alternatively, the type of each output can be manually changed in **Property Box** in accordance with the hardware.

Information about the device on the new hardware platform

For devices on the new platform, the information displayed in the window differs.

Project information:

- **Selected device model** - the model and modification of the device selected when creating the project.

Information about the connected device:

- **Device name** – model and modification of the connected device
- **Firmware version** – firmware version of the connected device
- **S/N** – unique device identifier
- **PRM Slot. Module name** – model of the extension module, connected to the device

- **PRM Slot. Module Firmware Version** – firmware version of the extension module, connected to the device.

8.2 Cycle time

Cycle time is the time it takes to complete the operating cycle of the device, namely:

- polling the state of the physical inputs of the device and copying their values into memory cells
- program processing
- read/write program network variables
- writing the results of the program to the physical outputs of the device

The default cycle time is **1 ms**. The device adjusts the cycle time depending on the complexity of the program.

Conditions for increasing cycle time:

- the complexity of the algorithm increases (a large number of FBs and macros are involved)
- the program uses a large number of network variables
- the project uses a large number of data controls via the device display

The user cannot set the cycle time. If the device is equipped with a display, the current cycle time can be viewed in the system menu of the device. If the device is connected to a PC, the cycle time is shown in the Device Information window.

8.3 Firmware update / repair

If a new ALP version includes a new version of the firmware for the connected device or extension module, a prompt appears to update the firmware before uploading a user program to the device. No internet connection is needed. Click **Yes** to start the update.



NOTE

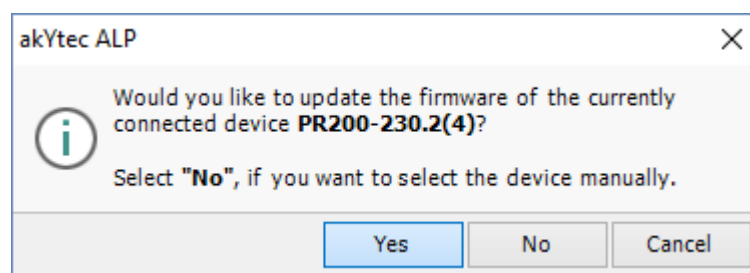
Ensure the power supply of the device and extension modules (if any) and the safe connection between the PC, the device and the extension modules (if any) during the update process.

You can also update the firmware manually using the menu item **Device > Firmware update**. This way the firmware can be repaired when the firmware damage is detected (see respective user guide, table “Error indication”).



NOTE

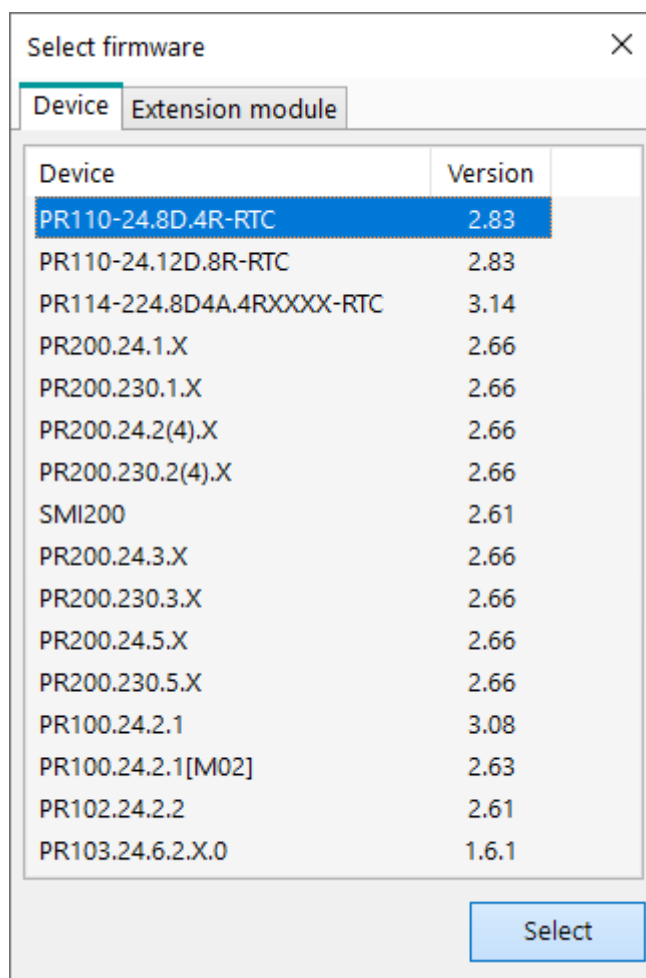
The user program will not be affected by firmware update.



If you select **Yes**, the firmware of the currently connected and recognized device will be updated (repaired).

If you select **No**, lists of devices and extension modules will be offered to select from. The opened window has two tabs: **Device** and **Extension Module**. This way a forced firmware update can be made.

Click **Select** to confirm the selection and start the update (repair) process. The message about the update result is shown upon the update completion.



Forced firmware update / repair

If the firmware is damaged (see respective user guide, table "Error indication") and device automatic recognition is not possible, a forced firmware update should be used. Proceed as follows:

1. set the device in the forced download mode (see the device user guide)
2. select the menu item **Device > Firmware update**, lists of devices and extension modules will be offered to select from
3. select the device (extension module)
4. click **Select** to confirm the selection and start the update (repair) process.

The message about the update result is shown upon the update completion.

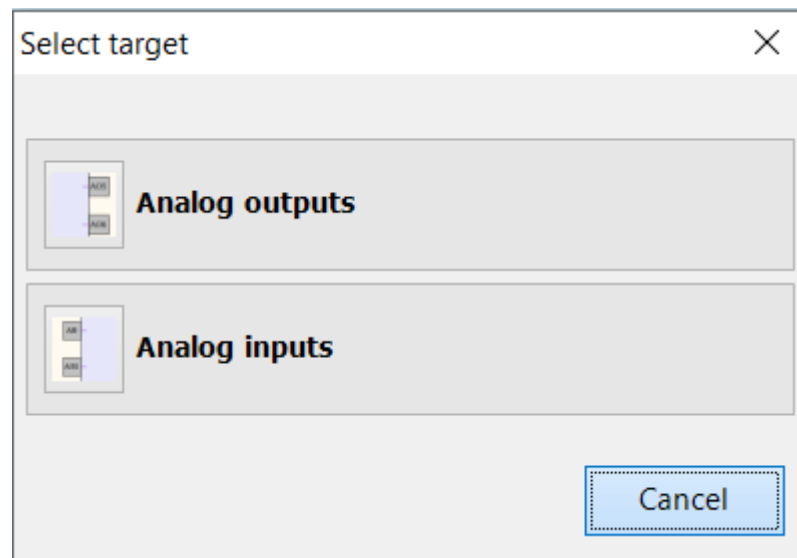
If the device and the extension module have incompatible firmware versions and the user program is uploaded to the device without the extension module connected, this may lead to an expansion module error being displayed. To fix the error, use forced firmware update for the expansion module as described, skipping step 1.

8.4 Calibration

Only general information about calibration of analog inputs or outputs is given in this section. For detailed information about calibration refer to the user guide of the device.

If calibration of analog inputs or outputs is necessary, use the menu item **Device → Calibration...**

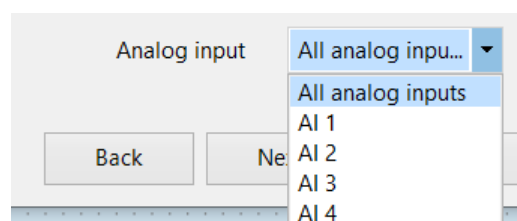
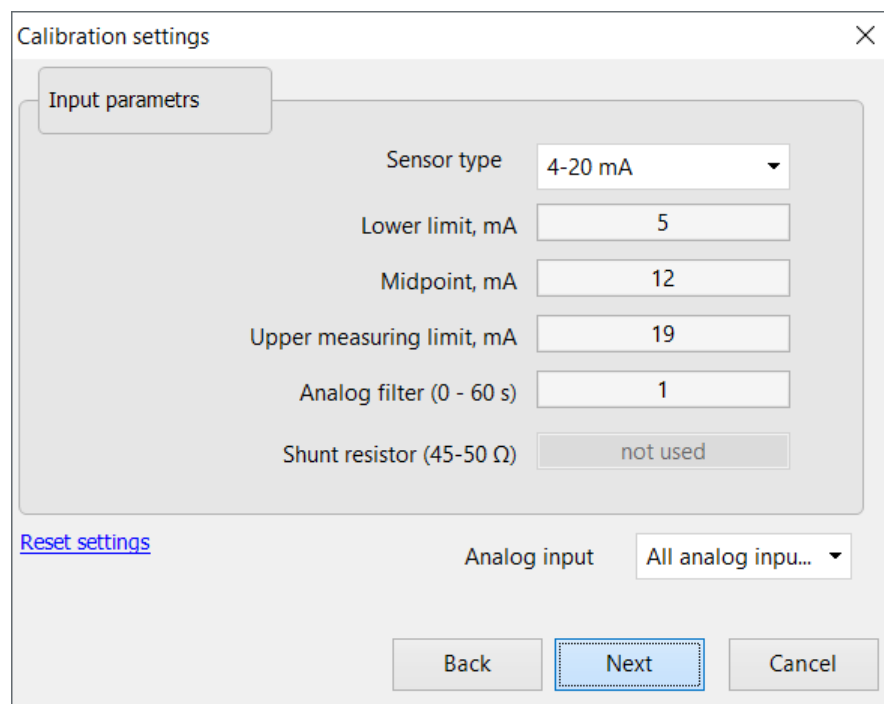
The item is active only if a device is connected. Select the calibration target (inputs or outputs) in the opened dialog.



After selecting the calibration target, device stops the execution of the program. The program starts again when the calibration completes successfully.

8.4.1 Input calibration

To calibrate inputs, connect a reference signal source to them. Start calibration, select the type of signal connected to the input and set the calibration parameters in the opened dialog.



Use the item **Reset settings** to apply the default settings for calibration.
Use the list **Select input** to select the input to calibrate, click the button **Next** and follow the instructions.

8.4.2 Output calibration

Before calibrating an analog output, prepare the appropriate measuring device, then start calibration and follow the instructions. Measure the signal at the output indicated at the top right of the window and enter the value in the input field.

Proceed the same way with the other outputs if needed. The message about the calibration results will appear after the completion of the calibration.

Lower limit calibration

Output A01

Step 1. Applied to the output: **5mA**.
Measure the output signal, enter the measured value in the input field and click "Next"

Measured value

Back Next Cancel

Upper limit calibration

Output A01


Step 2. Applied to the output: **19mA**.
Measure the output signal, enter the measured value in the input field and click "Next"

Measured value

Back Next Cancel

Proceed the same way with the other outputs if needed. The message about the calibration results will appear when the calibration is complete.

akYtec ALP

 Calibration has been successfully completed

OK

8.5 Change target device

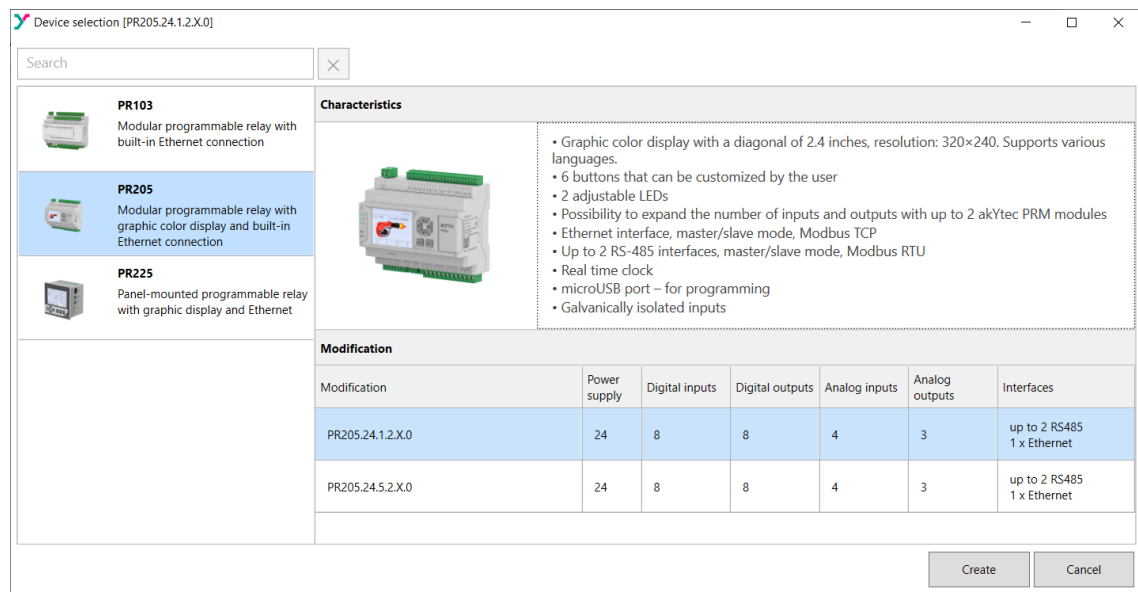
The **Change Target device** function is designed to transfer a project to another modification of a device from the same line of devices.

The change of target platform occurs according to the following rules:

- the canvas automatically adjusts to changes in the number of inputs/outputs.
- the user-configured I/O layout remains. New I/O are added after the existing I/O of the original project.
- Input/output connections whose data type has changed are broken.
- if the number of inputs/outputs increases but the data type of the original set of inputs/outputs does not change, then the connections remain.
- if the number of inputs/outputs decreases compared to the original, then the connections of remote inputs/outputs are broken.
- if extension modules were added to the original device, they are transferred to the target device, the connections for them remain.
- analog input/output settings are transferred (if there are analog outputs on the target device).
- communication interfaces are transferred without changes.
- all settings screens are transferred without any changes.
- all variables are transferred to the target device. If the variables are binded to parameters that do not exist in the target device, then a window will open with information about deleting the parameters and breaking the links with the variables bound to them;
- If the target device does not support the FB contained in the source project, an information window will appear notifying you of the need to delete the FB.

To change the target platform, you should:

1. Open the project that needs to be transferred to another device modification.
2. Select **File** → **Change Target device** from the menu. ALP will prompt you to save the project before changing the device. Next, a menu will appear with a list of available device modifications:



3. Select the new device. Click the **OK** button.
4. Check and restore broken connections, if any. You can check program's operation in the Simulator.
5. Save the modified project.

9 Plugins

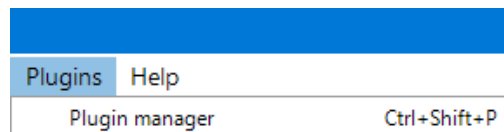
Plugins are used to create complex projects and integrate them with other akYtec services.
Plugins in ALP:

- Replication master. With this plugin you can load a program into the device without ALP and protect it from copying
- Exporting the device to akYtec Cloud. Allows you to export network variable configuration for polling in akYtec Cloud service

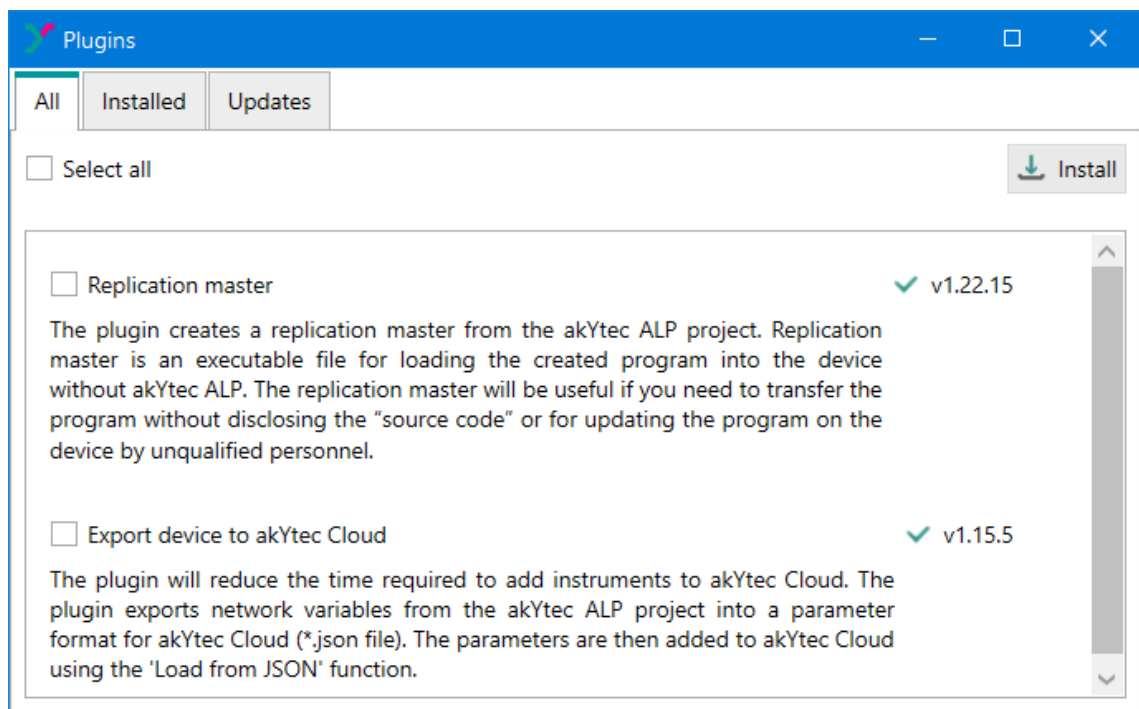
Installing plugins

To install plugins:

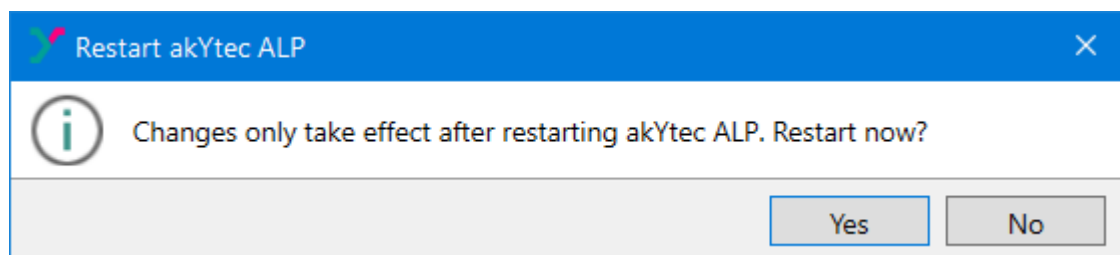
1. Select **Plugins** → **Plugin manager** at the main menu.



2. Check the required extensions in the **Plugins** window.



3. Click the **Install** button.
4. A green check mark ✓ next to the selected extensions shows that the installation is complete.
5. Close **Plugins** window. ALP restart dialog box appears.




After restarting ALP, the extensions will be ready for use.

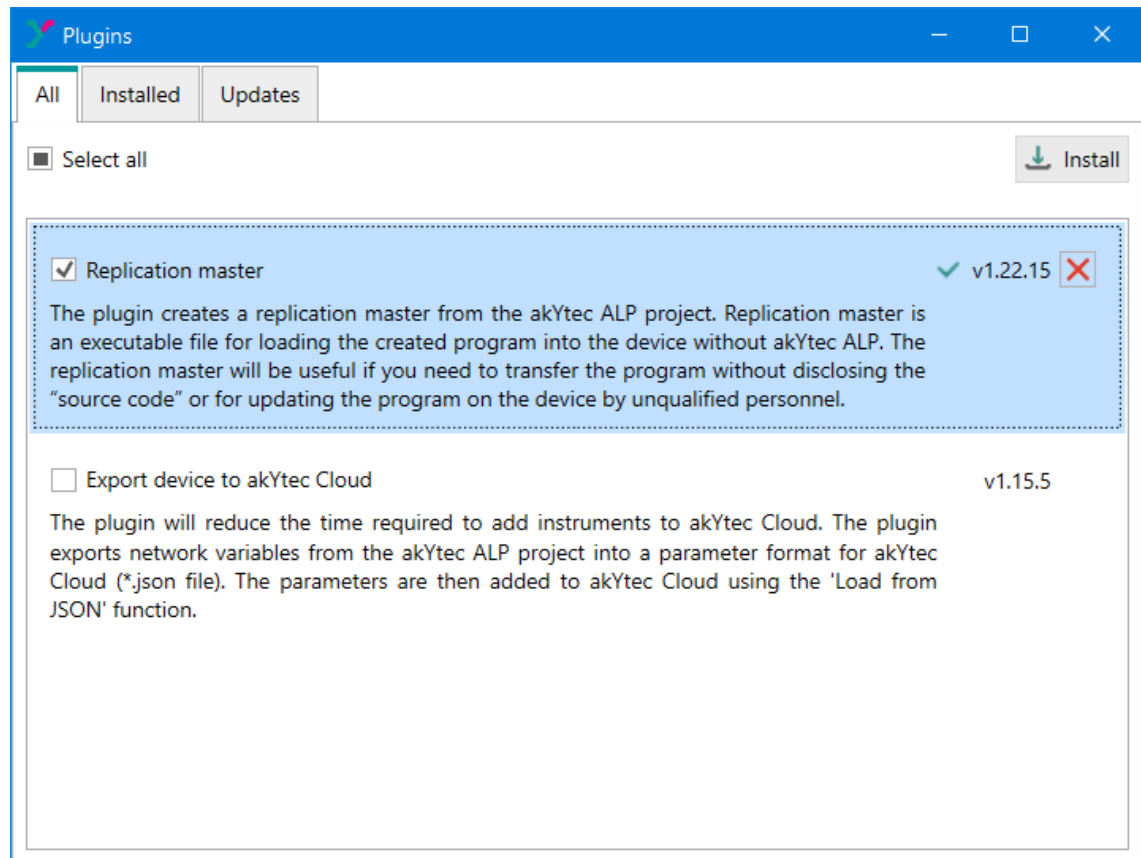
**NOTE**

If no extensions are installed, you should check if port 8084 is open.

Deleting extensions

To delete an extension:

1. Select **Plugins** → **Plugin Manager** at the main menu.
2. Check the box  next to the desired extension in the **Plugins** window.



3. The delete icon will change to the install icon.
4. Close **Plugins** window. ALP restart dialog box appears. After ALP is restarted, the deleted extensions will no longer be available in the main menu.

9.1 Replication master

The Replication master allows you to create a program file from an ALP project with one of the extensions:

*.exe – for Windows;

*.dll – for Linux.

The file can be used to load the program into the device without using the ALP.

**CAUTION**

The program can only be replicated between identical device modifications.

The result of writing a program to the device using the Replication master depends on whether the key is available in the device and the Replication master. The key can be written to the device using ALP.

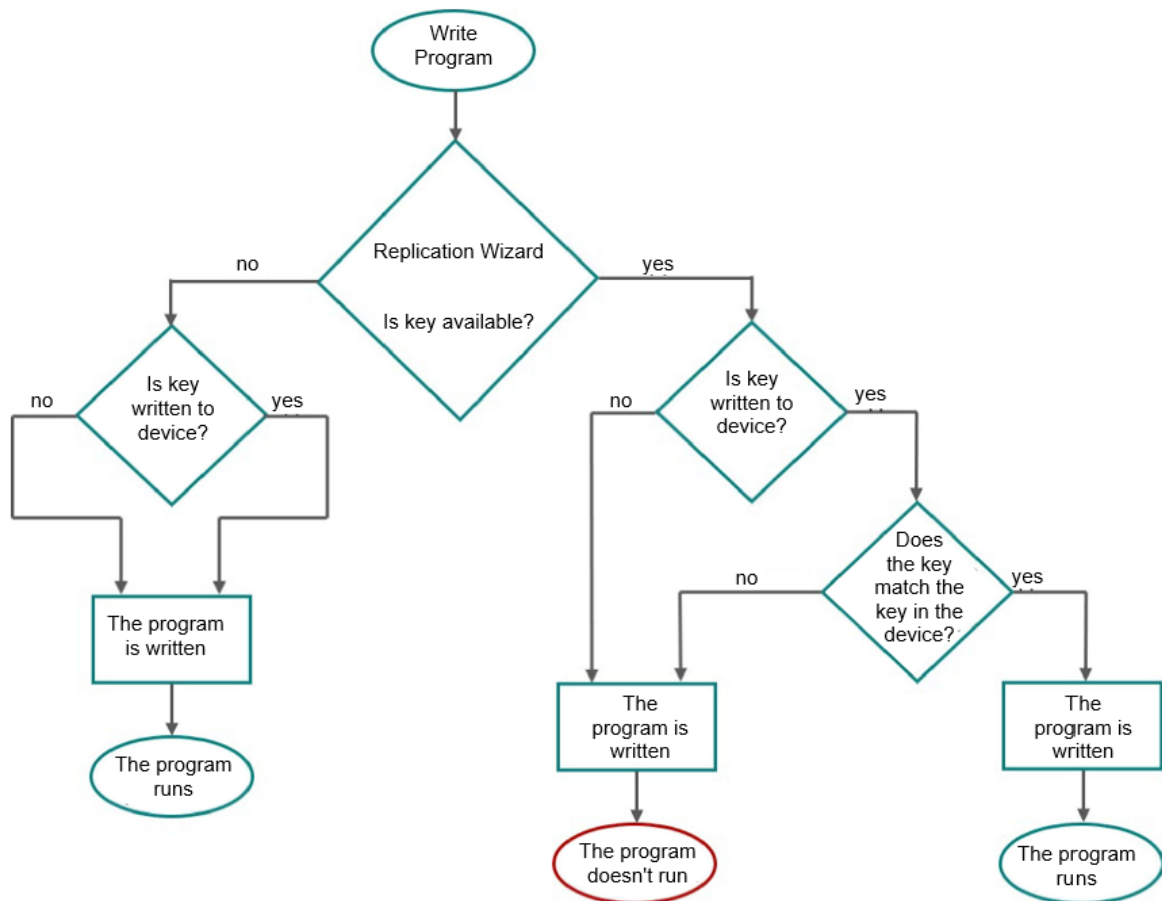


Fig. 9.1 Writing a program using the Replication master

Creating a Replication master

Before creating the Replication master for Windows, it is recommended to create an *.rtf text file with the project description. The project description will be displayed during the Replication master startup before the program is loaded into the device.

To create a Replication master for Linux, you must install the .NET 6 SDK. This can be done when installing ALP. To do this, check the **Install .NET6 SDK** checkbox in the installation window:

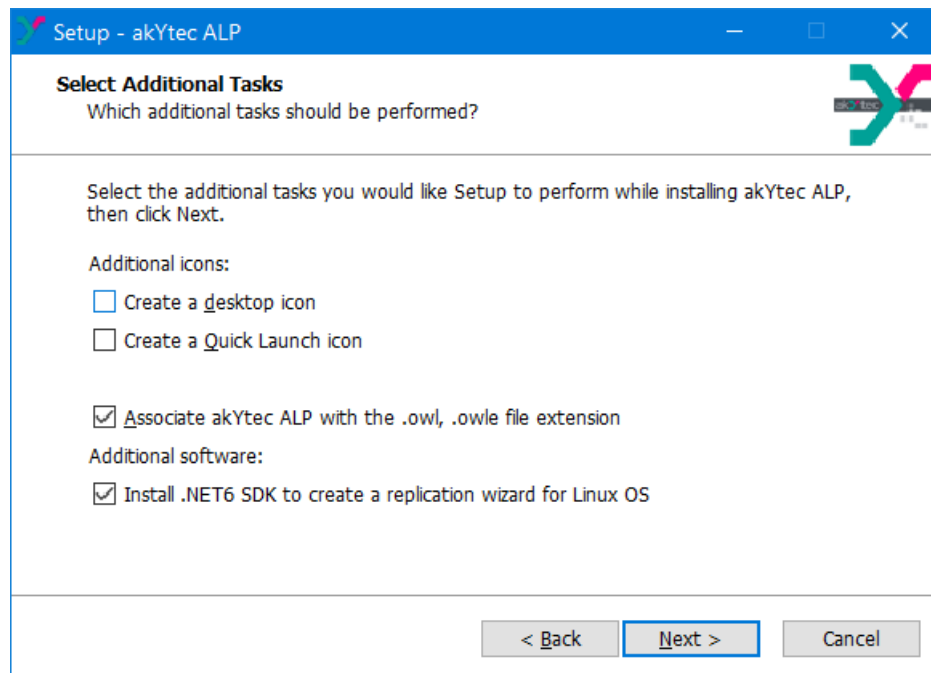


Fig. 9.2 ALP installation window

If the .NET 6 SDK has not been installed, an error will occur when creating the Replication master for Linux and the file will not be created. At the bottom of the window you will see a note about the installation being required and a link to download the package:

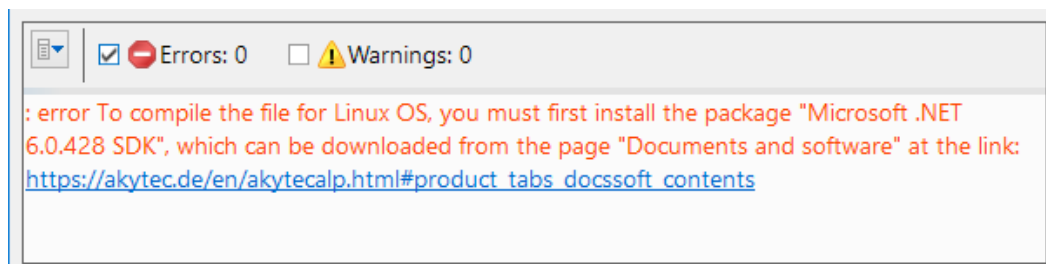
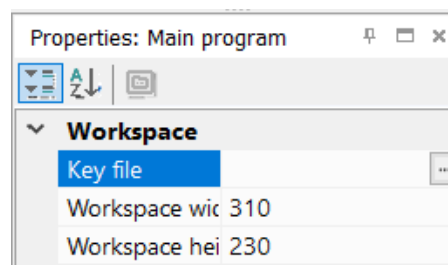


Fig. 9.3 File creation error

To create a Replication master:

1. In the **Key file name** field of the project properties panel, specify the file containing the key, stored on the PC. To download the program without a key, leave this field blank.



If after specifying the path to the key file, the key is deleted, the Replication master will not be created, and error information will be displayed at the bottom of the window:

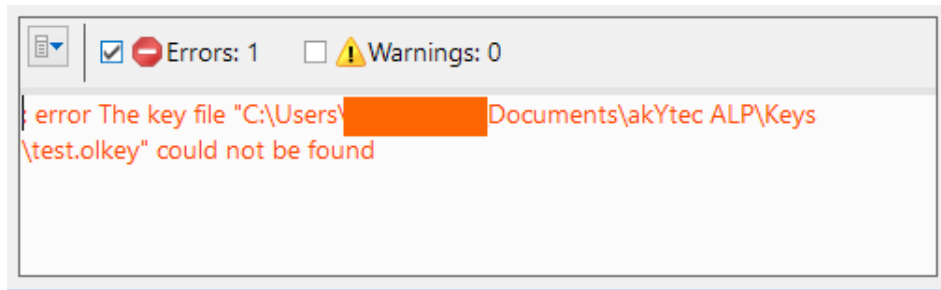


Fig. 9.4 File creation error

2. Select **Create Replication master** from the **Extensions** main menu.
3. A window will be opened for the Replication master creation. If a key is connected to the project, the message **User program is protected by a key!** will be displayed. Otherwise, the message **Device supports user program protection, but no key is selected for the project** will be summoned.

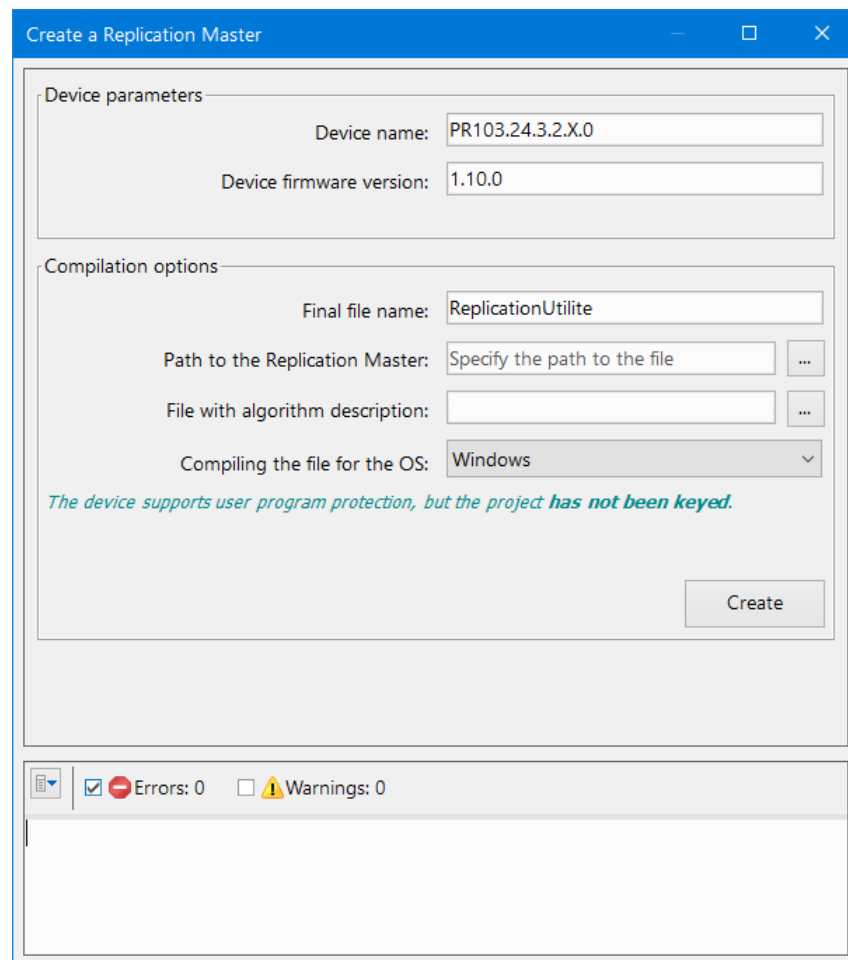


Fig. 9.5 Creating the Replication master

The **device parameters** are read from the device and filled in automatically.

In the compilation parameters window, fill in the fields:

- *Final file name* – name of new Replication master file
- *Path to the Replication master* – location of new file
- *File with algorithm description (For Windows)* – path to the text file with the project description in *.rtf format (optional)
- *Compiling the file for the OS* – target operating system

- Click the **Create** button. If there are no errors, a file with the specified name and extension *.exe or *.dll will be created.

**NOTE**

Creating a Replication master for Linux can take a long time.

**NOTE**

The *.dll file extension may not be displayed in Windows unless the option to display system files is enabled:

ReplicationUtilite

Fig. 9.6 Displaying a file with *.dll extension

The file creation report is displayed at the bottom of the window.

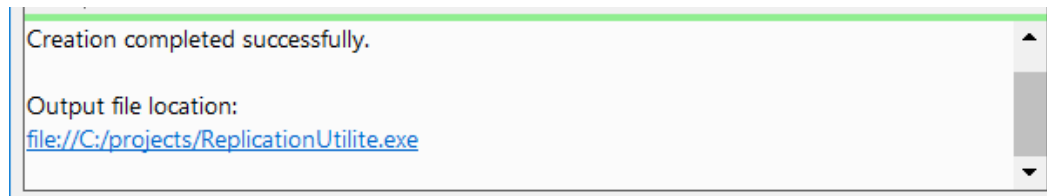


Fig. 9.7 File Creation Report

To hide or unhide the report display area, click the button. Information about warnings and errors is displayed if the corresponding checkboxes are checked.

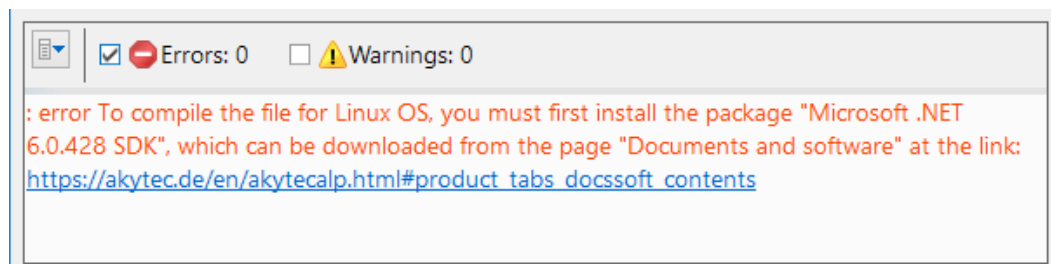


Fig. 9.8 Examples of warnings

9.1.1 Writing a program to the device via the Replication master on Windows

To write a program to the device using the Replication master, you should:

- Run the created Replication master file.

ReplicationUtilite.exe

Fig. 9.9 Replication Master File

**NOTE**

When the master opens, the project description text from the attached file with the *.rtf extension will be displayed. If the file was not attached when the master was created, the project description will be empty.

- Follow the instructions of the replication master.

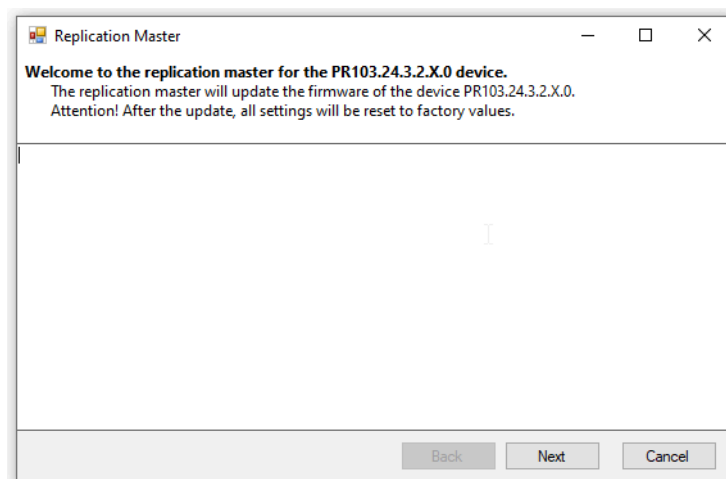


Fig. 9.10 Replication master window

3. In the Port settings section Replication master shows the number of the connected COM port. How to find out the port number is described in the section Connection to device.
4. If the device is password protected, enter the password in the Replication master window. If the password is missing or entered incorrectly, the **Next** button will be inactive. When the port is selected, connection status is shown in the lower right corner.

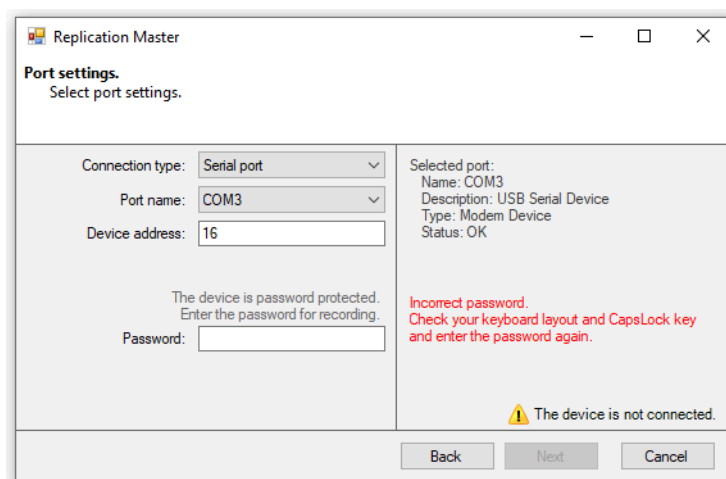


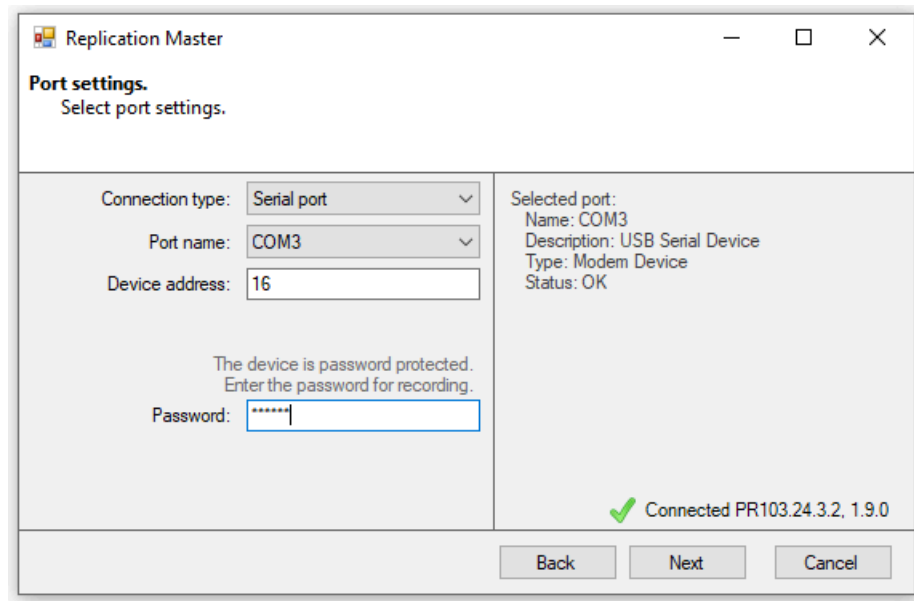


Fig. 9.11 Device connection error

In case of connection error, the error sign  and text will be displayed. When you hover over the sign  a tooltip will appear describing the error.

Error	Possible reasons
The device is not connected	<ul style="list-style-type: none"> – COM port selected incorrectly; – USB driver not installed; – connection break
Another device is required	The connected device does not match the project. You need to replace the device in the project and create a new replication master, or connect a suitable device

If there are no errors, there is a green check mark near the device name.



If the connection with the device is constantly interrupted (the check mark is blinking), then another application may be occupying the USB port. This may happen if an ALP program is running and a connection to the same port is configured in it. In this case, to load the project into the device, the other application should be closed or switched to OFFLINE.

5. If the connection is stable, you should click the **Next** button.

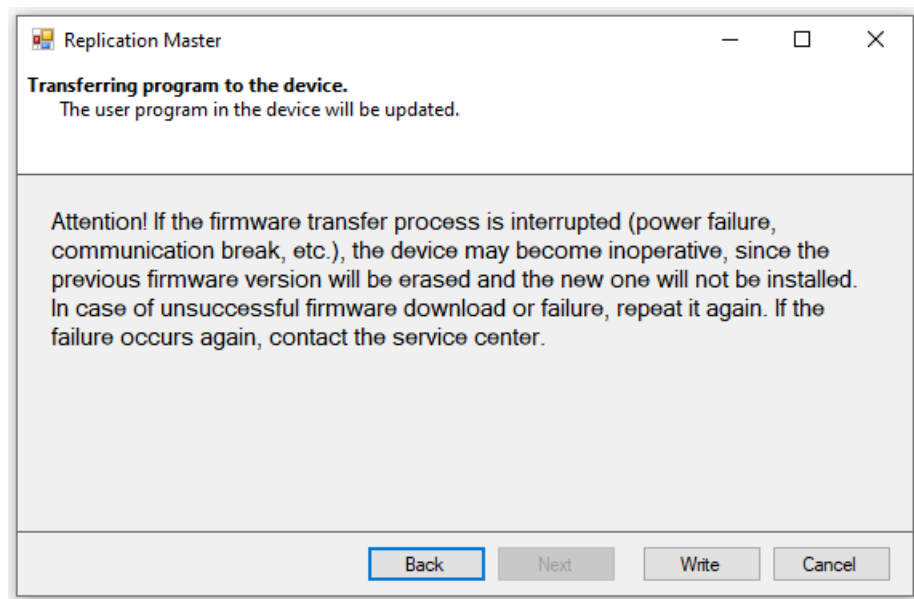


Fig. 9.12 Writing the program

6. To write the program into the device, click the **Write** button. The writing status line will be displayed at the bottom of the window. During the process, the connection between the PC and the device must not be interrupted. When the transfer is complete, the message will be displayed.

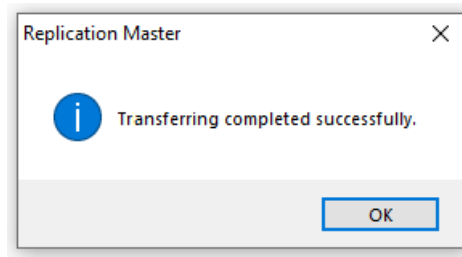


Fig. 9.13 Download complete

"Silent" installation mode**NOTE**

The "silent" installation mode is available when working with the Replication master version 1.21.14 and later.

The Replication master supports the "silent" firmware mode for devices. To launch it, use the following command line arguments:

- **/silent** – "silent" installation argument;
- **/password:** – password, required if the device was password protected;
- **/ip:** – IP address of the connected device (if the device is connected via the Ethernet interface);
- **/portname:** – COM port number (if the device is connected via USB interface);
- **1>> File_Name.txt** – text file displaying the firmware process in percentage;
- **2>> File_Name.txt** – text file displaying errors during the firmware process.

An example of a command to run Replication master file (*ReplicationUtilite.exe*) located on the desktop when connected via an Ethernet interface:

C:\Users\User_name\Desktop\ReplicationUtilite.exe /silent /IP:10.2.3.204

Example of a command to run Replication master file (*ReplicationUtilite.exe*) located on the desktop when connected via USB to a device with password 1234:

C:\Users\User_name\Desktop\ReplicationUtilite.exe /silent /portname:COM7 /password:1234

Example of a command to run the ReplicationUtilite.exe Replication master file located on the desktop when connected via USB to create a text file displaying errors during the firmware process:
C:\Users\User_name\Desktop\ReplicationUtilite.exe /silent /portname:COM7 2>> File_Name.txt

9.1.2 Writing a program to a device via the Replication master on Linux**PC requirements:**

- Ubuntu 22.04 or 24.04, x86/32 or x64 processor architecture;
- .NET 6.

**NOTE**

Not available for PR110 and PR114 devices.

To write a program to a device on Linux, open a command line and enter the command:

- **./ReplicationUtilite -ip:192.168.1.100** — when connecting the device via Ethernet, where 192.168.1.100 is the IP address of the device;
- **./ReplicationUtilite -portname:/dev/ttyACM0** — when connected via a serial port, where ACM0 is the port address.

**NOTE**

To determine the device port, use the commands:

- **lsusb** — for USB;
- **dmesg | grep tty** — for serial port.

An example of a command to launch the replication master file when connecting the device via a serial port with the address ACM0:


```
root@akYtec:~# ./ReplicationUtilite -portname:/dev/ttyACM0
```

Fig. 9.14 Command to write a program

If the actions are performed correctly, the command line will display the status line of the program loading into the device:

```
[#####------] 77% /
```

Fig. 9.15 Status bar

Table 9.1 Possible errors

Error	Solution
You do not have permission to run the file Permission denied	<ul style="list-style-type: none"> – contact the administrator; – use the command sudo ./ReplicationUtilite -portname:/dev/ttyACM0, where ACM0 is the port address
Entering an invalid command Incorrect string arguments are specified	Please enter a valid command to run the file
The device is password protected Incorrect string arguments are specified	Enter the command -password:.

9.2 Exporting the device to akYtec Cloud

Exporting a device to akYtec Cloud reduces the time to add a device configured in Slave mode to the akYtec Cloud service.


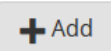
Working with akYtec Cloud requires a special device, a gateway, to be connected to the device. Let's assume that the gateway is already configured and connected to the control panel, the settings are given in the *Operation Manual* for the gateway.

To export the configuration of Modbus parameters of the device, proceed as follows:

1. Create a project with settings in Slave mode and network variables.
2. Select **Export Device to akYtec Cloud** from the **Plugins** main menu.
3. In the menu that opens, select the location and specify the file name in *.json format.

Further steps require a personal account in akYtec Cloud.

After logging in to your personal account:

1. Click the button .
2. On the page that opens, click the  button.
3. In the menu that appears, set the **ID**: IMEI / serial number of the network gateway (indicated on the gateway housing). In the **Device Type** drop-down list, select **PR103** in list **Programmable relays**. Enter the address of the device that was set in ALP. Fill in the remaining fields. Next, click the **Create** button.

Device registration ×

Device type*

ID*

Network address*

Serial number

Device name*

Categories

Time zone*

akYtec devices
▾

- ▢ Programmable controllers
- ▢ Programmable relays
 - ▢ PR103
 - PR103 over Ethernet - Autodetection
 - PR103 over Ethernet - Manual setting
 - ▢ PR200
 - ▢ PR102
 - ▢ PR100
- ▢ I/O Modules
- ▢ Arbitrary modbus device

Cancel
Create

4. On the **General/General Settings** tab, specify the polling rate and COM port settings of the device according to the settings in ALP. Click the **Save** button to apply the new settings.

Time zone*

Log retention period*

"Operational" polling period*

"Configuration" polling period*

"Manageable" polling period*

Offline period*

COM-port baud rate*

GMT+1:00
▾

Time on the device page will be shifted according to the time zone.

days

Not more than 90 days

sec

Polling interval for operation parameters

sec

Polling interval for configuration parameters

sec

Polling interval for manageable parameters

sec

The value must be greater than the minimum interval for polling parameters

9600
▾

5. On the page that appears, click the **Parameter Settings** button. Click on the **Import** drop-down list and select the **Load from JSON** option. In the menu that opens, select the previously created file in *.json format and click **Load Parameters**.

Export to JSON
Clear all parameters
Import... ▾
Settings ⚙

Parameter
All parameters

Import from JSON
Import from Codesys v.2.3

Register address	Unit of measurement	Data format

6. Modbus variables of the device will be added to akYtec Cloud.

10 Keyboard shortcuts

Keyboard shortcut	Action
Menu/File	
Ctrl + N	Create a new project
Ctrl + O	Open an existing project
Ctrl + Alt + S	Save an open project under a different name
Ctrl + S	Save an open project
Ctrl + P	Print
Ctrl + Shift + C	Open Component Manager
Menu/View	
Ctrl + Z	Undo last change
Ctrl + Y	Return (restore) a canceled action
Menu/Device	
Ctrl + F7	Transfer the application to the device
Ctrl + Shift + V	Open Variable Table
Ctrl + Shift + S	Open device configuration
Menu/Service	
Shift + F5	Go to simulation mode
F6	Start simulation
F7	Stop simulation
F8	Pause simulation
F10	Single cycle
Ctrl + F5	Go to debug mode
Menu/Plugins	
Ctrl + Shift + P	Open plugins manager
Menu/Help	
F1	Open Help
Insert panel	
Ctrl + Shift + F	Create an ST function
Ctrl + Shift + M	Create a macro
Ctrl + Shift + B	Create an ST function block
Ctrl + M	Create a macro from a selection
Usage location panel	
F5	Refresh usage locations
Keys for working with elements	
Ctrl + C	Copy an element
Ctrl + V	Paste from clipboard
Delete	Deleting a selected item

Keyboard shortcut	Action
Element resizing keys	
Ctrl + →	Increasing the width of a selected element
Ctrl + ←	Decreasing the width of a selected element
Ctrl + ↓	Increasing the height of a selected element
Ctrl + ↑	Decreasing the height of a selected element
Scaling the workspace	
Ctrl + Mouse wheel	When you rotate the mouse wheel away from you, the scale of the workspace increases. When you rotate the mouse wheel toward you, the scale of the workspace decreases
Ctrl + «+»	Increase scale
Ctrl + «-»	Decrease scale
Switch between tabs	
Tab + →	Switch between tabs
Tab + ←	

11 Program examples

These examples with simple tasks explain the creation of a circuit program in the ALP programming software.

- Light switch with automatic switch-off
- Mixer control
- Direct connection of PR103 to akYtec Cloud

11.1 Task 1: Light switch with automatic switch-off

The task is to switch the light on for a certain time, e.g. for a house entry.

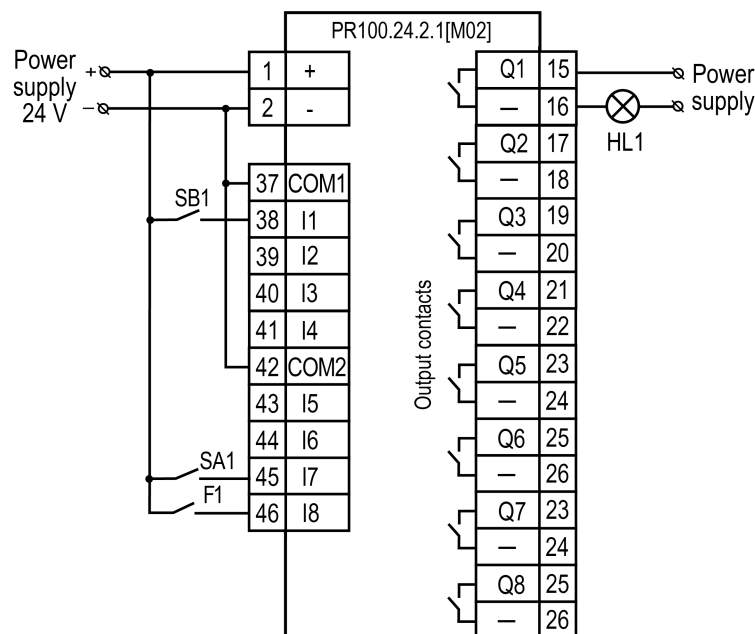
Task definition:

1. The light sensor F1 and the light button SB1 "TIME" are installed in front of the entrance door.
2. If the button SB1 is shortly pressed and the ambient light is insufficient, the light switches on for 1 minute – this time should be enough to find a key hole and to open the door.
3. If the button SB1 is pressed for 2 seconds, the light switches on for 3 minutes regardless of the ambient light – this mode can be useful for entrance cleaning.
4. Provide the possibility to control the light by commands from external devices or with the switch SA1 "CONST" regardless of the ambient light. This mode can be useful during the reception of guests or for further automation of the apartment as part of the "smart house" program.
5. Provide the possibility to switch on the light only at a certain time.

Device selection:

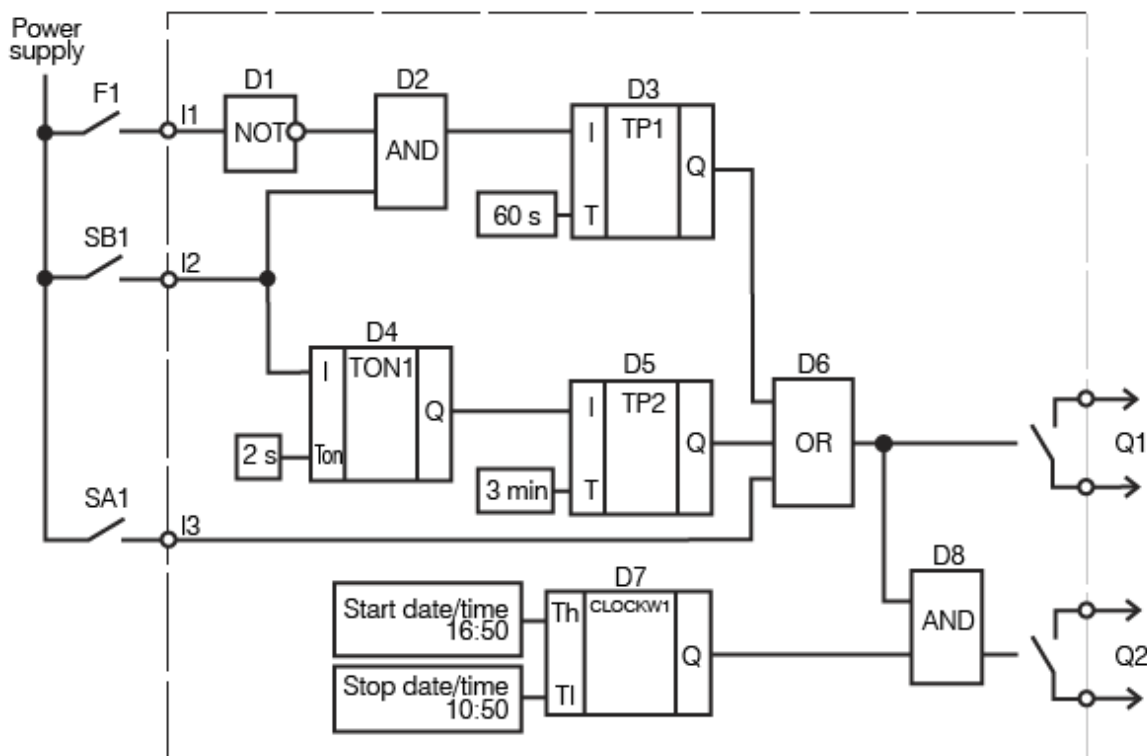
The control device must have minimum two digital inputs, one digital output and an integrated real-time clock to implement this task. These features can be provided by devices of PR100 series.

The task implementation with PR100.24.2.1(M02):



Circuit program

The circuit program can be implemented in the way shown in figure below.



Input I1 – connected to light sensor F1

Input I2 – connected to SB1 button

Input I3 – connected to SA1 switch

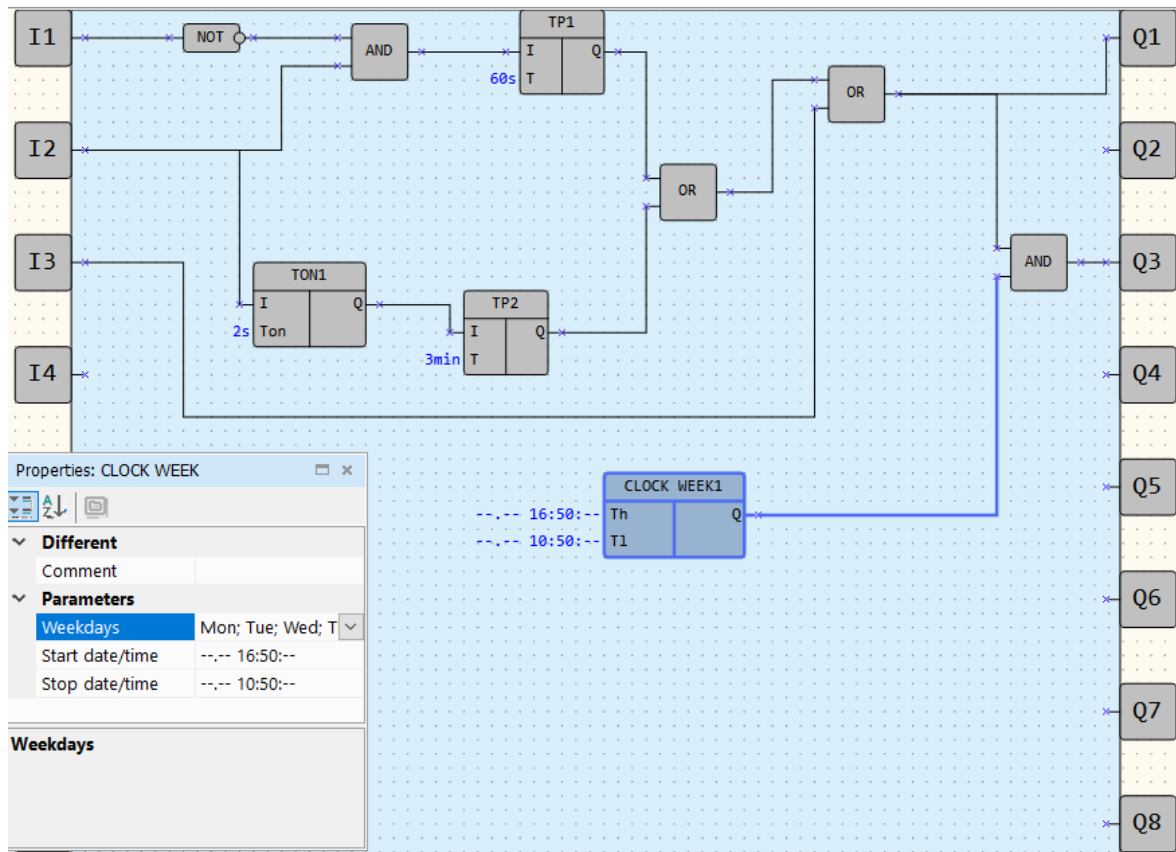
Output Q1 – output to implement the task points 1-4

Output Q2 – output to implement the task point 5

Program description:

1. If the button SB1 is shortly pressed (< 2 s), the logical AND (D2) is enabled. If the ambient light is insufficient, the first input of D2 is also **True** and the timer TP "Pulse" (D3) forms a pulse with 1 minute duration. This pulse activates the output Q1 over the logical OR (D6) and the light switches on for 1 minute.
2. If the button SB1 is pressed for > 2 s, the on-delay timer TON (D4) activates the timer TP "Pulse" (D5), a pulse with the duration of 3 minutes activates the output Q1 over logical OR (D6) and the light switches on for 3 minutes.
3. If the ambient light is sufficient, the contact of the sensor F1 closes, the logical AND (D2) is disabled and the timer TP "Pulse" (D3) is blocked.
4. If the switch SA1 "CONST" is closed, the output Q1 activates over the logical OR (D6) and the light is switched on constantly.
5. If you want to use the light only on certain weekdays at certain times, you can use the output Q2. With the weekly timer CLOCKW (D7) you can set the start and the stop time and the weekdays for lighting.

The circuit program created in ALP is shown in figure below.



11.2 Task 2: Mixer control

The task is to implement an industrial mixer with simple control functions.

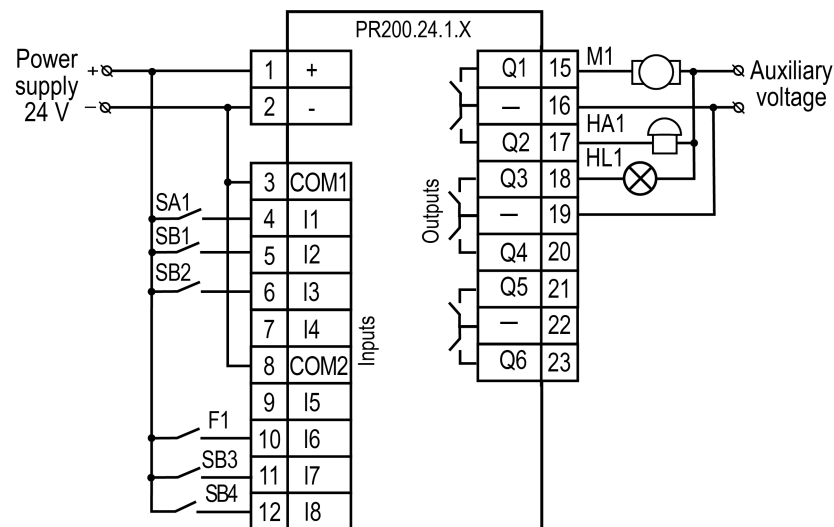
Task definition:

1. Automatic and Manual operation modes are required. The switch SA1 "MODE" is installed to switch between the modes.
2. In Automatic mode the operating cycle can be started with the button SB1 "START" and stopped automatically with the end of the cycle or manually with the button SB2 "STOP". The cycle duration is 5 minutes. During the cycle the motor of the mixer is on for 15 seconds and off for 10 seconds alternately. All settings can be changed in the program.
3. In Manual mode the motor can be started with the button SB1 "START" and stopped with the button SB2 "STOP".
4. When the motor is overloaded (overload switch F1), it switches off automatically. Signal lamp HL1 "Overload" indicates an operating error. Intermittent warning signal (HA1) sounds with 3-second interval.
5. The acoustic signal can be switched off with the button SB3 "RESET".
6. The button SB4 "CONTROL" is used for the functional test of the lamp HL1 and the acoustic signal HA1.

Device selection:

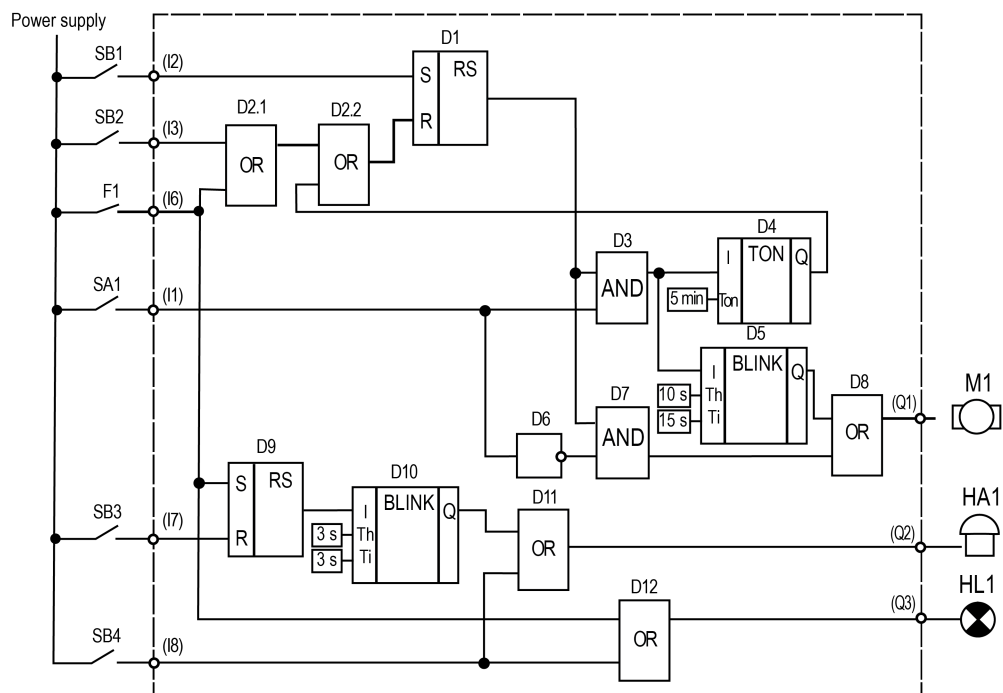
The control device must have minimum 6 digital inputs and 3 digital outputs to implement this task. These features can be provided by devices of PR200 series.

The task implementation with PR200.230.1.X:



Circuit program

The circuit program can be implemented in the way shown in figure below.



Input I1 – connected to SA1 “MODE” switch
 Input I2 – connected to SB1 “START” button
 Input I3 – connected to SB2 “STOP” button
 Input I6 – connected to F1 overload switch
 Input I7 – connected to SB3 “RESET” button
 Input I8 – connected to SB4 “TEST” button
 Output Q1 – connected to motor
 Output Q2 – connected to HA1 acoustic signal
 Output Q3 – connected to HL1 signal lamp

Program description:

1. Input I2 (SB1 "START")

If the button SB1 is pressed, the RS trigger D1 becomes **True** as long as there is no reset signal at the input R. Subsequent signal path depends on the state of the switch SA1 "MODE":

- If SA1 is open (Manual mode), the logical AND (D7) and the logical OR (D8) are enabled and the motor M1 (output Q1) is switched on.
- If SA1 is closed (Automatic mode), the logical AND (D7) is disabled and the start signal can only activate the pulse generator BLINK (D5) to start the operating cycle (15 s on / 10 s off) and the on-delay timer TON (D4) to stop it (in 5 min).

2. Input I3 (SB2 "STOP")

If the button SB2 is pressed or the switch F1 is activated, the RS trigger D1 is reset over the input R and the output Q1 is disabled.

3. Input I1 (SA1 "MODE")

- If the switch SA1 is open (Manual mode), the logical AND D3 is disabled and D7 is enabled, the timer D4 and the pulse generator D5 are disabled and the motor M1 can be only started with SB1 and stopped with SB2.
- If the switch SA1 is closed (Automatic mode), the logical AND D3 is enabled and D7 is disabled, thus the motor M1 can be only started by the pulse generator D5 (15 s on / 10 s off cycle) and stopped by the timer D4 in 5 minutes.

4. Input I6 (overload switch F1)

When the motor is overloaded, the F1 contact is closed, the RS trigger D1 is reset and the motor is stopped.

Concurrently the signal lamp HL1 is switched on over the logical OR (D12) and the acoustic signal HA1 is activated over the RS trigger D9. The pulse generator D10 provides an intermittent acoustic signal with the cycle 3 s on / 3 s off.

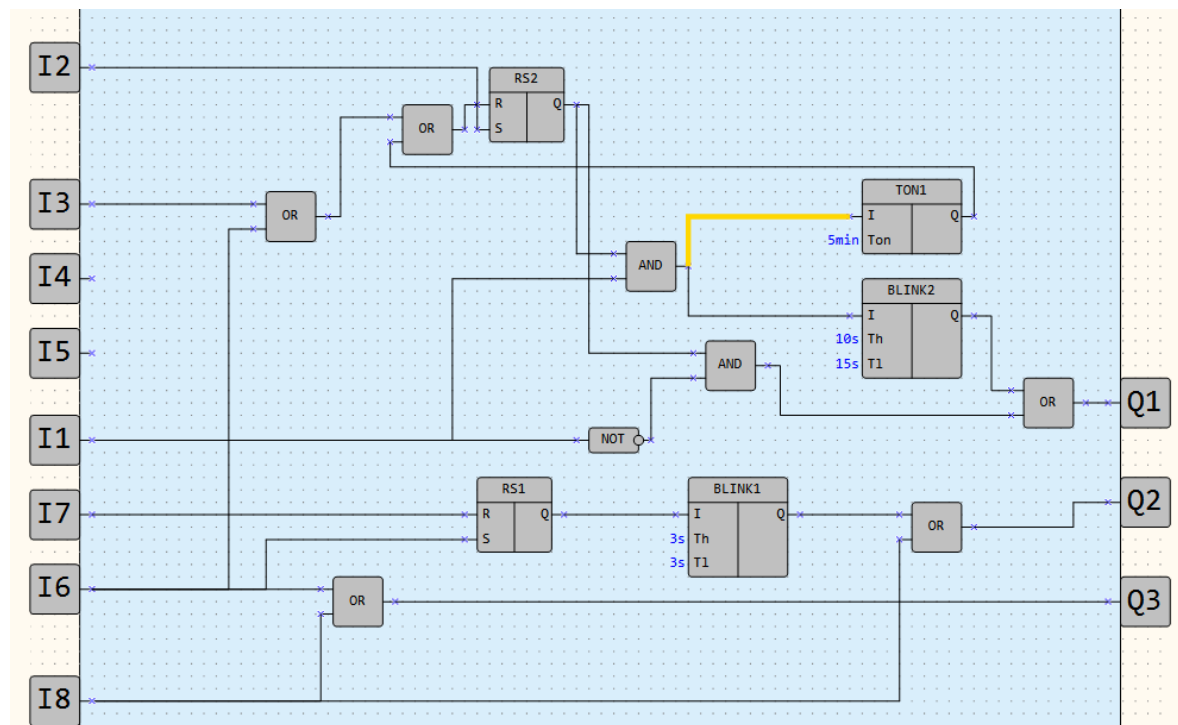
5. Input I7 (SB3 "RESET")

The button RESET is used to reset the acoustic signal HA1. If the button SB3 is pressed, the RS trigger D9 is reset and the pulse generator D10 for the acoustic signal HA1 is stopped.

6. Input I8 (SB4 "TEST")

The button TEST is used to test the acoustic signal HA1 and the signal lamp HL1. If the button SB4 is pressed, the logical ORs D11 and D12 are enabled, the outputs Q2 and Q3 activated, the acoustic signal and the lamp switch on.

The circuit program is shown in the figure below.



NOTE

1. The remaining two unused inputs and one output can be used for implementation of additional functions. For example, to switch between different time settings for automatic motor operation or to switch other operating parameters of the mixer.
2. The technological cycle of operation can be completely automated by implementation of an incremental counter (CT) to switch off the RS trigger D1.

11.3 Task3. Direct connection of PR103 to akYtec Cloud

Direct connection to akYtec Cloud is available for devices on the new platform

NOTE
For a list of

For a list of devices on the new platform, see the *About the program* section...

The device must be configured for operation and connected to the network.
To connect to akYtec Cloud you should:

1. Connect the device to the PC and create a project using network variables.
2. Set a password to access the device.
3. In the *settings* device window, allow access to akYtec Cloud.

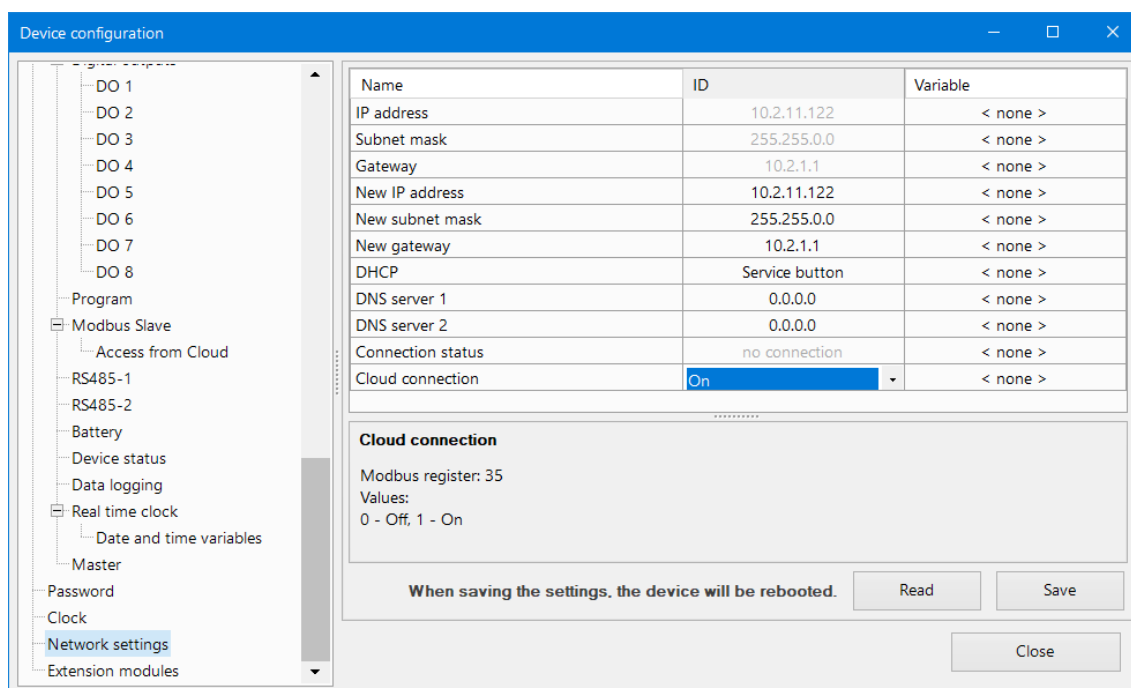


Fig. 11.1 Device configuration window

**NOTE**

The value of the **Cloud connection** status parameter can be read via Modbus or a user program variable can be linked. The parameter description is given in the table below..

Table 11.1 Possible states of the Cloud connection parameter

State	Value	Description
no connection	0	Cloud sharing is disabled
identification	1	Establishing connection to Cloud
normal operation	2	Cloud controls the device, no errors
network error	3	The device cannot establish a connection to the Cloud server
no password	4	The password for the device is not set

4. Allow remote access to Modbus registers.

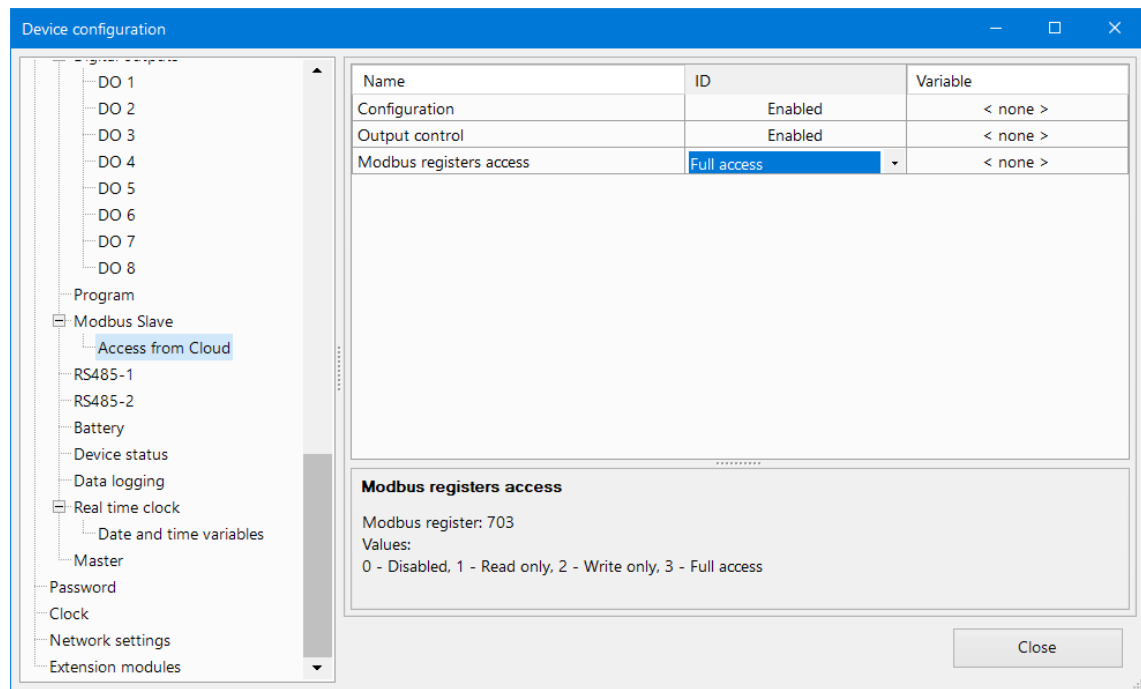



Fig. 11.2 Device configuration window

5. Load the program into the device (**Device** → **Load the program into the device**).
6. Go to the main page of the Cloud service. If you don't have an akYtec Cloud account, then go through the registration procedure.
7. Go to the **Administration** tab, open the **Devices** tab and click the **Add device** button ( **Add**).
8. A window will appear with a choice of the device type.

**NOTE**

Direct connection to Cloud is performed with PR103 as an example. To connect other second generation devices, follow similar steps.

Device registration

Device type*

ID*

Network address*

Serial number

Device name*

Categories

Time zone*

akYtec devices

- Programmable controllers
- Programmable relays
 - PR103
 - PR103 over Ethernet - Autodetection
 - PR103 over Ethernet - Manual setting
 - PR200
 - PR102
 - PR100
- I/O Modules
- Arbitrary modbus device

Cancel Create

Fig. 11.3 Selecting the device type

Option **PR103 via Ethernet – Autodetection**:

**NOTE**

With this method of adding a device, not only network variables are displayed, but also the entire tree of device parameters.

- Enter in the new window:
 - Identifier – device serial number
 - Network address – address 1, leave unchanged
 - Device name – name of the device;
 - Time zone – time zone in which the device is located
- Click **Add**. The basic settings interface of the device will open.
- Enter the password for the device. If necessary, you can change other settings (for example, the polling period).
- Click **Save** to apply the new settings.
- Cloud will connect to the device and read all parameters from it.

Option **PR103 via Ethernet – Manual setup**:

- Enter in the new window:
 - Identifier – device serial number
 - Network address – address 1, leaves unchanged
 - Device name – name of the device;
 - Time zone – time zone in which the device is located.
- Click **Add**. The general settings interface of the device will open.
- Enter the password for the device. If necessary, you can change other settings. Click **Create** to apply the new settings.
- Next, in the Parameters/Parameter settings tab, add the network parameters of the device using one of the available methods:
 - manually, in accordance with the network variables specified in PR103. It should be noted that the register addresses are specified in hexadecimal - therefore the values used differ from those given in ALP - 16384 (DEC) = 4000 (HEX).

- as a *.json file, if you use the Device Export extension in akYtec Cloud ALP. To add parameters, click on the Import drop-down list and select the Load from JSON option. In the menu that opens, select the previously created file in *.json format and click the Load parameters button.

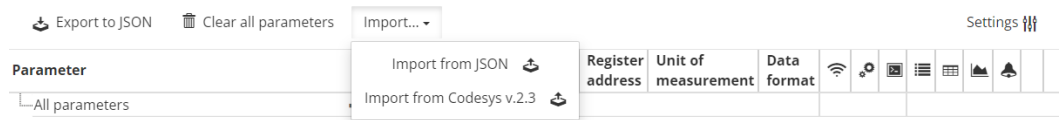


Fig. 11.4 Import parameters

If all settings were correct, then the **General Data** tab will show data from the device. If the device is not polled from Cloud, then you should check the network settings, the connection status to Cloud.

Connection status	normal operation
-------------------	------------------

Fig. 11.5 Example display of connection to Cloud

If the values are not written to the device from Cloud, you should check if the checkbox in the **Managed Parameter** column is checked:



Fig. 11.6 Controlled parameter

12 ST language

ST (Structured Text) is a high-level text language. It is one of the five languages supported by the IEC 61131-3 standard.

ALP allows you to create functions and function blocks in the ST language.

- Syntax
- Data types
- Language structure

12.1 Syntax

Keywords can be entered in upper and lower case characters. Spaces and tabs do not affect the syntax and can be used everywhere.

The names of variables, functions and function blocks should follow the rules:

- The name must not contain spaces or special characters (for example, !, @, etc.). The exception is the underscore character (_)
- The name must start with a letter
- The variable name can only contain letters of the Latin alphabet
- The name must not contain multiple underscore (_) characters in a row (i.e. the name **i__Test** is not valid, but the name **i_Te_st** is valid)
- Object names are case insensitive (**ITest** and **ITEST** will be interpreted as the same name)
- There are no restrictions on name length
- The name must not match one of the reserved keywords (eg VAR, INT, etc.)
- It is recommended to use Hungarian notation and lowerCamelCase style for variable names

12.1.1 Bitwise access to variables

Bitwise access is available for variables of type UDINT. To access a bit, use the following format:

Variable.Bit_number

Example

```
Variable.1 := TRUE;
```

```
Variable.31 := FALSE;
```

The bit number must be specified as an integer from 0 to 31. Using variables as bit numbers is not permitted.

12.1.2 Using functions in other ST program elements

A function can be called within another function or function block. To do this, you need to use the following format (informal call):

Function name (comma separated list of function inputs)

Example

```
FUNCTION rFun1: REAL;
```

```
VAR_INPUT
```

```
    rIn1 : REAL;
```

```
    rIn2 : REAL;
```

```
END_VAR
```

```
    rFun1 := rIn1 + rIn2;
```

```
END_FUNCTION
```

There is a function **rFun1**:

Calling **rFun1** from **rFun2** function will look like this:

```
FUNCTION rFun2: REAL;
```

```
VAR_INPUT
```

```
    rIn1_0 : REAL;
```

```
    rInf1_1 : REAL;
```

```
    rInf1_2 : REAL;
```

```
END_VAR
```

```
    rFun2 := rIn1_0 * rFun1(rInf1_1, rInf1_2);
```

```
END_FUNCTION
```

rFun2 function will return a number equal to the product of **rIn1_0** and the sum of **rInf1_1** and **rInf1_2**, which are specified at the input of this function.

12.1.3 Using one function block in another

Instances of one function block can be created in another function block, namely in the local variable declaration area in the format:

(short designation):(function block name)

Once a function block is instantiated, you can begin working with its data. The inputs of a block instance are externally writable. Outputs are for reading.

You can call an instance in different ways. As an example, consider the call to a counter for forward counting, which is created as a template when creating a function block:

1. Through a formal call:

```
FUNCTION_BLOCK fb2
```

```
VAR_INPUT
```

```
    xIn : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
    xAlarmMax : BOOL;
```

```
    udiQ : UDINT;
```

```
END_VAR
```

```
VAR
```

```
    fb1 : functionblock1; //declaration of a function block instance
```

```
END_VAR
```

```
    fb1 (U := xIn, Q => udiQ); //calling a function block instance
```

```
IF fb1.Q > 10 THEN
```

```
    xAlarmMax := TRUE;
```

```
END_IF
```

```
END_FUNCTION_BLOCK
```


**NOTE**

Pay attention to the specific operator for copying a value from the block's output variables ("=>").

- By accessing the inputs/outputs of a function block (variable declarations are similar):

```
fb1.U := xIn; //write data to FB input
fb1 (); //call a FB instance
udiQ := fb1.Q; //read the output of the FB instance
IF fb1.Q > 10 THEN
    xAlarmMax := TRUE;
END_IF
```

The **fb2** output will indicate that 10 pulses have been exceeded and the accumulated counter value.

**NOTICE**

You cannot declare an instance of a function block in the body of a function.

**NOTICE**

In an ST functional block, the maximum nesting of blocks is no more than 8.

**NOTICE**

ST function blocks do not support RETAIN type variables.

**NOTICE**

ST function blocks reserve space in ROM memory after they are added to the project library, regardless of whether they are used in the project or not.

12.1.4 Comments in ST editor

The ST editor in ALP supports the commenting feature. Two types of comments are available:

- Single-line. Its marker is a double slash – //
- Multiline. Its marker is:
 - (* – beginning of comment
 - *) – end of comment

12.1.5 Copying ST elements between projects

To copy ST components between projects:

- Select the ST elements that need to be copied on the original project diagram.
- Copy the selected elements using the keyboard shortcut **Ctrl+ C** or the context menu.
- Open the project to paste the copied elements.
- Paste elements into the second project diagram using the keyboard shortcut **Ctrl + V** or the context menu

As a result, templates of all copied elements will be added to the corresponding section of the project component library.

**NOTE**

- If the component library does not contain the group that was specified for the copied component, it will be created.
- If the copied component group is not specified, the template will be added to the **Other** folder.

When copying, all connections between all components that were included in the copy operation are preserved.

**NOTICE**

If any of the copied elements contains an error, the pasting of the elements into the project diagram will be canceled.

An error message will be displayed indicating the name of the incorrect element.

Other components will appear in the component library in the appropriate sections.

12.2 Documentation in the ST editor

ST editor supports documentation function.

The documentation marker is the triple slash “///”.

Documentation is added above objects: function/function block declaration, input/output variable declaration.

Tags for documentation are listed in the table below.

Tag	Description
<Description>... </Description>	Description of the program element (function, function block, input and output (function block only) variable)
<Author>...</Author>	Name of the creator of the function or function block
<GroupName>... </GroupName>	Group name for grouping a function or function block in a content library
<OutputDescription>... </OutputDescription>	Function output description

Documenting the function

```

///<Description>Resistance temperature detector (Pt1000)</Description>
///<OutputDescription>Temperature</OutputDescription>
///<Author>akYtec</Author>
///<GroupName>Temperature sensor</GroupName>

```

```

FUNCTION f_PT1000: REAL; // function for PT1000 RTD sensor

```

```

VAR_INPUT

```

```

    ///<Description>Resistance</Description>

```

```

    R : REAL;

```

```

END_VAR

```

```

VAR_OUTPUT

```

```

    ///<OutputDescription>Temperature</OutputDescription>

```

```

    Q : REAL;

```

```

END_VAR

```

Documenting the function block

```

///<Description>Counter for direct counting</Description>
///<Author>akYtec</Author>
///<GroupName>Timers and counters</GroupName>

```

```
FUNCTION_BLOCK fb_Counter
```

```
VAR_INPUT
```

```

    ///<Description>Pulse detector</Description>
    U : BOOL; //bool input variable

    ///<Description>Counter reset flag</Description>
    Res : BOOL; //bool input variable

    ///<Description>Preset counter value</Description>
    N : UDINT; //bool input value

```

```
END_VAR
```

```
VAR_OUTPUT
```

```

    ///<Description>Counter value</Description>
    Q : UDINT; //uint output value

```

```
END_VAR
```

12.3 Data types

Data types, supported in akYtec ALP:

Data type	Description	Valid range	Size
BOOL	Boolean	FALSE, TRUE	4 bytes
UDINT	Unsigned double integer	0...4294967295	4 bytes
REAL	Floating-point	$-1,2 \times 10^{-38} \dots 3,4 \times 10^{38}$	4 bytes
TIME	Time interval	T#0...4294967295ms T#0..4294967s T#0..71582m T#0..1193h T#0..49d T#0..49d17h02m47s295ms	4 bytes
DT	Time of day and date	DT#2000-01-01-00:00:00..2136-02-07-6:28:15	4 bytes
ARRAY	Array	—	—

The data type of a variable determines the type of information, the range of representations, and the set of allowed operations.

The variable can be used only after its declaration. It is possible to assign the value of one variable to another variable only if they are of the same type. Otherwise, type converter should be used.



NOTE

Converting a larger type to a smaller one can result in loss of information.

12.3.1 Reserved keywords

**NOTE**

In the table, words available for use are highlighted in **bold**.

ABS	END_REPEAT	READ_ONLY	THEN
ACTION	END_RESOURCE	READ_WRITE	TIME
AND	END_STEP	REAL	TIME_OF_DAY
ARRAY	END_STRUCT	REAL_TO_BOOL	TIME_TO_UDINT
AT	END_TRANSITION	REAL_TO_UDINT	TO
BEGIN	END_TYPE	REPEAT	TOD
BOOL	END_VAR	RESOURCE	TRANSITION
BOOL_TO_REAL	END_WHILE	RETAIN	TRUE
BOOL_TO_UDINT	ENO	RETURN	TYPE
BY	EXIT	SEL	UDINT
CASE	F_EDGE	SHL	UDINT_TO_BOOL
CD32	FALSE	SHR	UDINT_TO_DT
CONFIGURATION	FOR	SINT	UDINT_TO_REAL
CONTINUE	FROM	STEP	UDINT_TO_TIME
DATE	FUNCTION	STRING	UINT
DATE_AND_TIME	FUNCTION_BLOCK	STRUCT	ULINT
DC32	GET_DATE_TIME	SYS.BLINK	UNTIL
DINT	GET_TIME	SYS.CLOCK	USINT
DO	IF	SYS.CLOCKWEEK	VAR
DT	INITIAL_STEP	SYS.COMPARE_DATE_TIME	VAR_ACCESS
DT_TO_UDINT	INT	SYS.CT	VAR_CONFIG
DWORD	LINT	SYS.CTN	VAR_EXTERNAL
ELSE	LREAL	SYS.CTU	VAR_GLOBAL
ELSIF	LWORD	SYS.DTRIG	VAR_IN_OUT
EN	MOD	SYS.FTRIG	VAR_INPUT
END	NON_RETAIN	SYS.IS_LEAP_YEAR	VAR_OUTPUT
END_ACTION	NOT	SYS.RS	VAR_TEMP
END_CASE	OF	SYS.RTRIG	WHILE
END_CONFIGURATION	ON	SYS.SR	WITH
END_FOR	OR	SYS.TOF	WORD
END_FUNCTION	POW	SYS.TON	WSTRING
END_FUNCTION_BLOCK	PROGRAM	SYS.TP	XOR
END_IF	R_EDGE	TASK	

12.3.2 Arrays

Array declaration syntax

The following syntax is used to declare an array:

```
<Array name>:[Index1..IndexN]OF <ELEMENT TYPE>:=[Default value of the element1,
Default value of the element2, Default value of the elementN];
```

Where:

Index1 - number of the first element of the array;

IndexN - number of the last element of the array.

Rule for indices:

- The IndexN number cannot be less than or equal to the Index1 number;
- Indexes must be of integer type only and only positive.

Examples of array declaration:

```
Mass1: ARRAY[0..1] OF UDINT; //declaration of an array consisting of
two elements
UDINT type
Mass1: ARRAY[0..1] OF UDINT :=[1, 5]; //initialization of array elements
when declaring an array
Mass1: ARRAY[0] OF UDINT :=[1, 5]; /error: array dimension must be
specified as <uint>...<uint>
Mass1: ARRAY[] OF UDINT; //error: array dimension must be specified in
<uint>...<uint>
```

Syntax for accessing array elements

To access array elements, use the following syntax:

```
<Array name>[Index1];
```

Examples of working with array elements:

```
VAR:=Mass1[5]; //assigning the value from the fifth element
of the array to the variable Var
VAR:=Mass1[5]+5*2;
VAR:=UDINT_TO_BOOL(Mass1[5]); //assigning the converted
value to the variable Var
from the five element
VAR:=(Mass1[i]); //assigning the value from the i element
of the array to the variable Var,
where i is a variable (local/input)
```

The following rules apply when working with arrays:

1. An array can only be one-dimensional.
2. The number of array elements is limited to 32768 elements.
3. Array elements can be of the following types: BOOL, UDINT, FLOAT, TIME, and DT.
4. Array support only for local variables that are declared in a block
VAR...END_VAR.
5. The sequence of default values for array elements is enclosed in square brackets.
6. When initializing array elements, the number of default values must match the number of array elements.

Example:

VAR

```
Mass1: ARRAY[0..1] OF UDINT:= [1,5]; //no error
Mass1: ARRAY[0..1] OF UDINT:= [1,5,7]; //error
Mass1: ARRAY[0..1] OF UDINT:= [1]; //error
```

END_VAR

7. Bitwise access to array elements is not supported.

Example:

```
Mass1: ARRAY[0..1] OF UDINT:= [1,5];
Mass1[1].1 := TRUE; //error
```

8. Repeating a sequence of array elements is not supported.

Example:

```
Mass1: ARRAY[0..1] OF UDINT:= [a, b, c(N)]; //where (N) stands
for repetition
sequences a, b, c N times)
Mass1: ARRAY[0..1] OF UDINT:= [1,5,7(2)]; //error
```

9. Repeating a sequence of array elements is not supported.

Example:

```
Mass1[1, 5]:=1; //error: assigning value 1 to first and fifth
array elements
```

Example of using arrays.

12.4 Language structures

ST language structures include:

- arithmetic operations
- bit operations
- data type conversion operations
- logical operations
- relational operations
- assignment operation
- IF – ELSIF – ELSE statement
- CASE statement
- RETURN statement
- FOR statement
- WHILE statement
- REPEAT – UNTIL statement



NOTE

When writing expressions, it is permissible to use variables (input, output and local) and constants.

12.4.1 Operations

12.4.1.1 Arithmetic operations

Operation	Operator	Data types	Example
addition	+	IN, OUT: UDINT/ REAL	OUT := IN1 + IN2 + ...
multiplication	*		OUT := IN1 * IN2 * ...

Operation	Operator	Data types	Example
subtraction	-		OUT := IN1 - IN2
division	/		OUT := IN1 / IN2
modulo	MOD		OUT := IN1 MOD IN2
absolute value	ABS (IN)	IN, OUT: REAL	OUT := ABS (IN1)
exponentiation	POW (IN, N) IN – base N – exponent	IN, N, OUT: REAL	OUT := POW (IN1, N)

The result of the arithmetic operation is the mathematical result of the expression.

The priority of an operation determines the order of its execution in the expression. Parentheses are allowed to define the calculation order in arithmetic expressions.

12.4.1.2 Bit operations

Operation	Operator	Data types	Example
Bitwise shift left	SHL (IN, N)	IN, OUT: UDINT N: 1..32	OUT := SHL (IN1, N)
Bitwise shift right	SHR (IN, N)		OUT := SHR (IN1, N)
Decoder. Converts binary code to positional code	DC32 (IN)	IN, OUT: UDINT	OUT := DC32 (IN1)
Encoder Converts positional code to binary code	CD32 (IN)	IN, OUT: UDINT	OUT := CD32 (IN1)

12.4.1.3 Data type conversion operations

Operation	Operator	Data types	Example
UDINT to REAL	UDINT_TO_REAL (IN)	IN: UDINT OUT: REAL	OUT := UDINT_TO_REAL (IN)
UDINT to BOOL	UDINT_TO_BOOL (IN)	IN: UDINT OUT: BOOL	OUT := UDINT_TO_BOOL (IN)
UDINT to TIME	UDINT_TO_TIME (IN)	IN: UDINT OUT: TIME	OUT := UDINT_TO_TIME (IN)
UDINT to DT	UDINT_TO_DT (IN)	IN: UDINT OUT: DT	OUT := UDINT_TO_DT (IN)
REAL to UDINT	REAL_TO_UDINT (IN)	IN: REAL OUT: UDINT	OUT := REAL_TO_UDINT (IN)
REAL to BOOL	REAL_TO_BOOL (IN)	IN: REAL OUT: BOOL	OUT := REAL_TO_BOOL (IN)
BOOL to REAL	BOOL_TO_REAL (IN)	IN: BOOL OUT: REAL	OUT := BOOL_TO_REAL (IN)
BOOL to UDINT	BOOL_TO_UDINT (IN)	IN: BOOL OUT: UDINT	OUT := BOOL_TO_UDINT (IN)

Operation	Operator	Data types	Example
TIME to UDINT	TIME_TO_UDINT (IN)	IN: TIME OUT: UDINT	OUT := TIME_TO_UDINT (IN)
DT to UDINT	DT_TO_UDINT (IN)	IN: DT OUT: UDINT	OUT := DT_TO_UDINT (IN)

12.4.1.4 Logical operations

Operation	Operator	Data type	Example
Logical negation	NOT	IN, OUT: BOOL	OUT := NOT IN1
Boolean multiplication	AND &		OUT := IN1 AND IN2 OUT := IN1 & IN2
Boolean addition	OR		OUT := IN1 OR IN2
logical (bitwise) "exclusive OR"	XOR		OUT := IN1 XOR IN2

12.4.1.5 Relational operations

Operation	Operator	Data types	Example
greater than	>	IN: UDINT/REAL OUT: BOOL	OUT := IN1 > IN2
greater than or equal to	>=		OUT := IN1 >= IN2
equal to	=		OUT := IN1 = IN2
less than or equal to	<=		OUT := IN1 <= IN2
less than	<		OUT := IN1 < IN2
not equal to	<>		OUT := IN1 <> IN2

12.4.1.6 Operation priorities



NOTE

The operations in the table are ordered **from highest to lowest priority**. The higher the priority of an operation, the sooner it is executed.

Operation	Operator
Brackets	(expression)
Calling a function and function block	Example: fb1(); function1 := ... ;
Bit operations	
Unary minus	–
Logical negation	NOT
Exponentiation	POW
Multiplication	*
Division	/
Modulo	MOD
Addition	+
Subtraction	–

Operation	Operator
Relational operations	>.< <=, >=
Equal to	=
Not equal to	<>
Conjunction Logical multiplication "AND"	& AND
Exclusive OR	XOR
Disjunction Logical addition "OR"	OR

12.4.2 Assignment operation

The paired symbol ":= " is used to indicate assignment. The right and left sides of the expression must contain operands of the same type (automatic type casting is not provided). On the left side of the expression (receiving side) only a variable can be used. The right side can contain an expression or a constant.

12.4.3 IF statement

The **IF** operator allows you to test one or more conditions, and, if at least one of the conditions is true, execute the specified expression conditions. After executing the expressions, the operator exits the statement – that is, the remaining conditions are no longer checked.

Let's consider the operator's work using the example of signaling that the temperature value exceeds the permissible limits:

```
FUNCTION_BLOCK fb //function block name

VAR_INPUT //declaration of input variables
    rTemp : REAL;
END_VAR

VAR_OUTPUT //declaration of output variables
    xHigh : BOOL;
    xLow  : BOOL;
END_VAR

VAR //declaration of local variables
    rHighTemp : REAL := 20;
    rLowTemp  : REAL := 10;
END_VAR

//coding area

IF rTemp > rHighTemp THEN
    xHigh := TRUE;
ELSIF rTemp < rLowTemp THEN
    xLow  := TRUE;
ELSE
```

```

    xHigh := FALSE;
    xLow  := FALSE;
END_IF

```

END_FUNCTION_BLOCK

If the condition in the **IF** statement is true (the value of the variable **rTemp** is greater than **rHighTemp**), then the variable **xHigh** will be assigned the value **TRUE** and the statement will exit the statement (the next condition will not be checked). If the condition is not met, then the next condition placed in the nested **ELSIF** statement will be checked. If the condition in **ELSIF** is satisfied (the value of the variable **rTemp** is less than **rLowTemp**), then the variable **xLow** will be assigned the value **TRUE** and the statement will exit the statement (the next condition will not be checked). If none of the conditions in **IF** and **ELSIF** are met (that is, the temperature value is within acceptable limits), then the expressions placed in the nested **ELSE** statement will be executed – the **assignment of value FALSE to the xHigh and xLow variables**.

The use of nested **ELSIF** and **ELSE** statements is optional. An arbitrary number of **ELSIF** statements can be placed inside an **IF** statement.

The construction allows nesting, that is, inside one **IF** there can be another one, etc. Also inside the **IF** operator, loops and the **CASE** operator can be used.

12.4.4 CASE statement

The **CASE** operator allows you to compare the value of a given integer variable (selector) with a set of constants or integer values (labels), and if there is a match, execute the expressions specified for this label. After executing the expressions, the operator exits the statement.

Example:

```

FUNCTION_BLOCK fb1 //function block name

    VAR_INPUT //declaratio of input variables
        udiSel : UDINT;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        xOut1 : BOOL;
        xOut2 : BOOL;
        xOut3 : BOOL;
        xOut4 : BOOL;
    END_VAR

    //coding area

    xOut1 := FALSE;
    xOut2 := FALSE;
    xOut3 := FALSE;
    xOut4 := FALSE;

    CASE udiSel OF
        0:
            xOut1 := TRUE;
        1..3:
            xOut2 := TRUE;
    END_CASE

```

```

        4, 6:
            xOut3 := TRUE;

ELSE
    xOut4 := TRUE;

END_CASE

```

END_FUNCTION_BLOCK

If the **udiSel** value is:

- Equal to 0, then **xOut1** will take the value **TRUE**;
- Falling into the range 1..3, then **xOut2** will take the value **TRUE**;
- Equal to 4 or 6, then **xOut3** will take the value **TRUE**;
- Not falling into any of the specified values, then **xOut4** will take the value **TRUE**;

As can be seen from the example, a label can include several values, listed, separated by commas “4, 6”, or the range “1..3”. In this case, values of one of the labels should not coincide with the values of the others. Also, when specifying a range of values, the beginning of the range must be less than its end.

The nested **ELSE** statement is optional; the expressions placed in it are executed if the selector value does not match any of the labels.

The actions provided to handle each of the **CASE** statement cases can use loops, **IF** statements, and **CASE** statements.

12.4.5 RETURN statement

The **RETURN** statement allows you to exit a program object.

Usage example:

```

IF xDone THEN
    RETURN;
END_IF;

```

```

udiCounter := udiCounter + 1;

```

If the variable **xDone** takes the value **TRUE**, then the expression “**udiCounter := udiCounter + 1**” will not be executed will be (like everything that will be located below in the body of the program).

12.4.6 FOR statement

The **FOR** operator is used to organize a loop with a predetermined number of iterations. It is usually used for operations on arrays of data.

The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.

As an example, consider the implementation of bubble sort from smallest to largest:

```

FUNCTION_BLOCK MaxI_MinI
//Maximum and minimum numbers using bubble sort from smallest to largest

VAR_INPUT //declaration of input variables
    udiX1, udiX2, udiX3, udiX4, udiX5 : UDINT;
//Adding the required number of input variables and defining the data type
END_VAR

VAR_OUTPUT //declaration of output variables
    udiMaxI, udiMinI : UDINT;
END_VAR

```

```

VAR //declaration of local variables
    udiI, udiJ, udiN, udiK : UDINT;
    audiX : ARRAY [1..5] OF UDINT;
//Specifies the range of the array and the data type of the array
END_VAR

//coding area
//array declaration
audiX[1] := udiX1;
audiX[2] := udiX2;
audiX[3] := udiX3;
audiX[4] := udiX4;
audiX[5] := udiX5;
udiN := 5; // the number of numbers to sort is specified (array size)

FOR udiI := 1 TO udiN-1 DO
    FOR udiJ := 1 TO udiN-udiI DO
        IF audiX[udiJ] > audiX[udiJ+1] THEN
            udiK := audiX[udiJ];
            audiX[udiJ] := audiX[udiJ+1];
            audiX[udiJ+1] := udiK;
        END_IF;
    END_FOR;
END_FOR;

udiMaxI := audiX[udiN]; //the last (maximum) number of the array is displayed
udiMinI := audiX[1]; //the first (minimum) number of the array is displayed

END_FUNCTION_BLOCK

```

The **udiI** variable is called the loop iterator (counter). This variable must be a signed integer type (**UDINT**). After each execution of the loop body, the iterator value changes - by default to **+1**. The user can set the iterator “step” using the nested **BY** operator. After this, the transition immediately occurs to the next iteration - that is, the entire cycle is executed “from beginning to end”. The structure will look like this:

```

// udiI will take the values 1, 4, 7, 10, etc.
FOR udiI := 1 TO udiN-1 BY 3 DO
    .. // loop code
END_FOR

```

The loop iterator (counter), its start and end values, and its increment are integer values. They are calculated before entering the loop, and changing the values of the variables included in any of these expressions will not change the number of iterations.

You can exit a loop early using the **EXIT** operator. Example structure:

```

FOR udiI := 1 TO udiN-1 DO
    IF audiX[2] > 100 THEN
        EXIT;
    ELSE

```

```

        .. // loop code
    END_IF
END_FOR

```

You can skip a loop step using the **CONTINUE** operator.

```

FOR udiI := 1 TO udiN-1 DO
    IF udiI = 3 THEN
        CONTINUE;
    ELSE
        .. // loop code
    END_IF
END_FOR

```

12.4.7 WHILE statement

The **WHILE** operator is used to create a loop with an unknown number of iterations. The loop will terminate if the condition being tested returns **FALSE**. In this case, the condition is checked **BEFORE** the expression is executed (a loop with a precondition). Thus, if the condition immediately returns **FALSE**, then the loop will not be executed even once.

The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.

Example

```

WHILE rVar < 100 DO
    rVar := rVar + 1;
END_WHILE

```

The result of executing this loop (with the initial value of the variable **rVar** := 10) will be the number 100.

You can exit a loop early using the **EXIT** statement.

You can skip a loop step using the **CONTINUE** operator.

12.4.8 REPEAT UNTIL statement

The **REPEAT** statement is used to create a loop with an unknown number of iterations. The loop will terminate if the condition being tested returns **TRUE**. In this case, the condition is checked **AFTER** the expression is executed (a loop with a postcondition). Thus, if the condition immediately returns **TRUE**, then the loop will be executed once.

The **IF** and **CASE** operators, as well as other loop operators, can be used inside a loop.

Example:

```

REPEAT
    IF rVar > 100 THEN
        EXIT;
    END_IF;
    rVar := rVar + 1;
UNTIL rVar > 180
END_REPEAT;

```

The result of executing this loop (with the initial value of the variable **rVar** := 10) will be the number 101.

To exit a loop early, you can use the **EXIT** operator.

You can skip a loop step using the **CONTINUE** operator.

12.5 System functions

GET_TIME function

The **GET_TIME** function returns a **TIME** type value (4 bytes) containing the time elapsed since the device was last turned on, in milliseconds.

Sample

```
VAR
    Time_1 : TIME := T#0ms;
    Time_2 : TIME := T#0ms;
    Q : BOOL := FALSE;
END_VAR

IF Time_1 = T#0ms THEN
    Time_1 := GET_TIME();
END_IF

Time_2 := GET_TIME();

IF (Time_2 - Time_1) >= T#1000ms THEN
    Q := NOT Q;
    Time_1 := T#0ms;
    Time_2 := T#0ms;
END_IF
```

GET_DATE_TIME function

The **GET_DATE_TIME** function returns a **DT** type value (4 bytes) containing real time clock data, in seconds since 00:00:00 01/01/2000, taking into account the time zone set in the device.

Sample

```
VAR
    Ton_UDINT : UDINT;
    Ton_DT : DT;
END_VAR

Ton_DT := GET_DATE_TIME();

Ton_UDINT := DT_TO_UDINT(Ton_DT);
```

SYS.COMPARE_DATE_TIME function

The **SYS.COMPARE_DATE_TIME** function compares two UDINT values given as input using a given date/time mask and returns a UDINT value that evaluates to:

- 1 – value 1 is greater than value 2;
- 2 – value 1 is less than value 2;
- 0 – value 1 is equal to value 2.

The comparison is made by the number of seconds starting from 00:00:00 01.01.2000.

Sample

```
FUNCTION udiCompare: UDINT;
```

```

VAR_INPUT
    udiValue1: UDINT; //value 1
END_VAR

VAR
    udiValue2 : UDINT; //value 2
    udiMask : UDINT := 63; //date/time mask
END_VAR

udiValue2 := DT_TO_UDINT (GET_DATE_TIME ());
udiCompare := SYS.COMPARE_DATE_TIME (udiValue1, udiValue2, udiMask);

```

END_FUNCTION

Mask = 63 (0b111111) – all bits are used:

- 0 bit – if 1, then seconds are used.
- 1 bit – if 1, then minutes are used.
- 2 bits – if 1, then the hours are used.
- 3 bits – if 1, then days are used.
- 4 bits – if 1, then months are used.
- 5 bits – if 1, then years are used.

SYS.IS_LEAP_YEAR function

The **SYS.IS_LEAP_YEAR** function returns a BOOL value containing data on whether the UDINT number supplied to the function's input corresponds to a leap year (1 - leap year, 0 - not).

Sample

```
FUNCTION xLeapYear: BOOL; //function name and output data type
```

```

VAR_INPUT //declaration of input variables
    udiYear : UDINT; //year being checked
END_VAR

xLeapYear := SYS.IS_LEAP_YEAR (udiYear);

```

END_FUNCTION

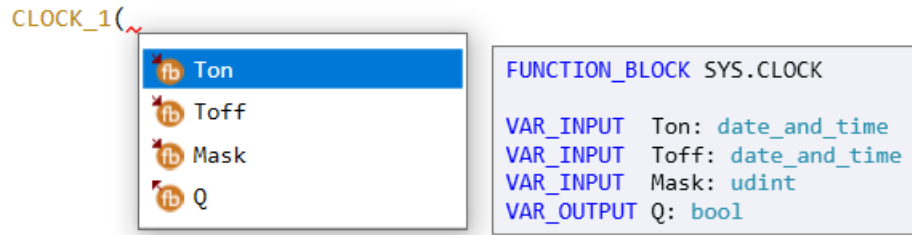
12.6 System Function Blocks

The ST editor supports the following system function blocks:

- Triggers
- Timers
- Generators
- Counters

The use of system function blocks in other program elements is similar to using custom function blocks.

For each system function block, the editor provides a hint on how to use it.



12.6.1 Triggers

- RS trigger reset dominant (SYS.RS)
- SR trigger set dominant (SYS.SR)
- Rising edge (SYS.RTRIG)
- Falling edge (SYS.FTRIG)
- D-trigger (SYS.DTRIG)

12.6.1.1 RS trigger reset dominant (SYS.RS)

The RS trigger reset dominant (SYS.RS) is used to switch with state fixation during the receipt of short pulses at the corresponding input. The output Q will show HIGH level signal at the edge of the signal from the input S.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
R	BOOL	RESET input
Outputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK RS_trigger //Function block name
```

```
VAR_INPUT //declaration of input variables
```

```
    R_in : BOOL;
```

```
    S_in : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT //declaration of output variables
```

```
    Q_out : BOOL;
```

```
END_VAR
```

```
VAR //declaration of local variables
```

```
    RS_1: SYS.RS;
```

```
END_VAR
```

```
//code area
```

```
RS_1(R := R_in, S := S_in, Q => Q_out);
```


END_FUNCTION_BLOCK

To prevent simultaneous receipt of signals at both inputs, the signal of input R takes priority.

12.6.1.2 SR trigger set dominant (SYS.SR)

The SR trigger set dominant (SYS.SR) is used to switch with state fixation during the receipt of short pulses at the corresponding input. The output Q will show HIGH level at the edge of the signal from the input S.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
R	BOOL	RESET input
Outputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK SR_trigger //Function block name
```

```
VAR_INPUT //declaration of input variables
```

```
    S_in : BOOL;
```

```
    R_in : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT //declaration of output variables
```

```
    Q_out : BOOL;
```

```
END_VAR
```

```
VAR //declaration of local variables
```

```
    SR_1: SYS.SR;
```

```
END_VAR
```

```
//code area
```

```
SR_1(S := S_in, R := R_in, Q => Q_out);
```

END_FUNCTION_BLOCK

To prevent simultaneous receipt of signals at both inputs, the signal of input S takes priority.

12.6.1.3 Rising edge (SYS.RTRIG)

The rising edge (SYS.RTRIG) is used when it is necessary to have a reaction to a change in the state of a digital input signal. Output Q generates a single pulse on the rising edge of the input I.

Designation	Data type	Description
Input		
I	BOOL	Trigger input
Output		
Q	BOOL	Trigger output

```

FUNCTION_BLOCK R_trigger //Function block name

    VAR_INPUT //declaration of input variables
        RT_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        RT_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        RTrig_1: SYS.RTRIG;
    END_VAR

    //code area

    RTrig_1(I := RT_in, Q => RT_out);

END_FUNCTION_BLOCK

```

12.6.1.4 Falling edge (SYS.FTRIG)

The falling edge (SYS.FTRIG) is used when it is necessary to have a reaction to a change in the state of a digital input signal. Output Q generates a single pulse on the leading edge of the input I.

Designation	Data type	Description
Input		
I	BOOL	Trigger input
Output		
Q	BOOL	Trigger output

```

FUNCTION_BLOCK F_trigger //Function block name

    VAR_INPUT //declaration of input variables

        FT_in : BOOL;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        FT_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        FTrig_1: SYS.FTRIG;
    END_VAR

```

```
//code area
```

```
FTrig_1(I := FT_in, Q => FT_out);
```

```
END_FUNCTION_BLOCK
```

12.6.1.5 D-trigger (SYS.DTRIG)

D-trigger (SYS.DTRIG) is used to generate a pulse to turn on the output for the time interval of the pulse at the D input. Output interval synchronizes with the clock frequency at the C input.

At the Q trigger output, a HIGH level signal will appear on the front of the clock pulses at the C input if there is a HIGH level signal at the D input. Output Q returns to the LOW level signal on the front of the clock pulses at the C input if there is a HIGH level signal at the D input.

Input S forces output Q to a HIGH level state.

Input R is the priority input and sets output Q to LOW level.

Designation	Data type	Description
Inputs		
S	BOOL	SET input
D	BOOL	Trigger input
C	BOOL	Clock frequency
R	BOOL	RESET input
Outputs		
Q	BOOL	Trigger output

```
FUNCTION_BLOCK D_trigger //Function block name
```

```
VAR_INPUT //declaration of input variables
```

```
    S_in : BOOL;
```

```
    D_in : BOOL;
```

```
    C_in : BOOL;
```

```
    R_in : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT //declaration of output variables
```

```
    Q_out : BOOL;
```

```
END_VAR
```

```
VAR //declaration of local variables
```

```
    DTrig_1: SYS.DTRIG;
```

```
END_VAR
```

```
//code area
```

```
DTrig_1(S := S_in, D := D_in, C := C_in, R := R_in, Q => Q_out);
```

```
END_FUNCTION_BLOCK
```

12.6.2 Timers

- Pulse (SYS.TP)
- ON-delay timer (SYS.TON)
- OFF-delay timer (SYS.TOF)
- Timer (SYS.CLOCK)
- Weekly timer (SYS.CLOCKWEEK)

12.6.2.1 Pulse (SYS.TP)

The pulse (SYS.TP) is used to generate a pulse to turn on the output for a specified time interval. A HIGH level signal appears at the output Q of the block at the edge of the input signal I. After starting, the output Q does not respond to a change in the value of the input signal during the interval T. When the interval T has expired, the output signal is reset to LOW level.

Designation	Data type	Description
Inputs		
I	BOOL	Turning on the timer
T	TIME	Pulse duration
Outputs		
Q	BOOL	Timer output

```
FUNCTION_BLOCK TP_timer //Function block name
```

```
    VAR_INPUT //declaration of input variables
```

```
        I_in : BOOL := FALSE;
```

```
    END_VAR
```

```
    VAR_OUTPUT //declaration of output variables
```

```
        Q_out : BOOL;
```

```
    END_VAR
```

```
    VAR
```

```
        TP_1: SYS.TP;
```

```
    END_VAR
```

```
    //code area
```

```
    TP_1(I := I_in, T := T#1000ms);
```

```
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days
```

```
    Q_out := TP_1.Q;
```

```
END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK TP_timer //milliseconds
```

```
    VAR_INPUT //declaration of input variables
```

```
        I_in : BOOL := FALSE;
```

```
        T_in : UDINT := 5000; //milliseconds
```

```
    END_VAR
```

```
    VAR_OUTPUT //declaration of output variables
```

```

        Q_out : BOOL;
    END_VAR

    VAR
        TP_1: SYS.TP;
        T_time: TIME;
    END_VAR

    //code area

    T_time := UDINT_TO_TIME(T_in);
    TP_1(I := I_in, T := T_time, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.2.2 ON-delay timer (SYS.TON)

The ON-delay timer (SYS.TON) is used to delay the signal transmission.. The timer output Q will produce a HIGH level signal with a delay relative to the input signal front I for time more than T and will turn off at the input signal fall.

Designation	Data type	Description
Inputs		
I	BOOL	Timer start (on rising edge)
T	TIME	Delay on power-on
Outputs		
Q	BOOL	Timer output

```

FUNCTION_BLOCK TON_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TON_1: SYS.TON;
    END_VAR

    //code area

    TON_1(I := I_in, T := T#1000ms);
    //where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days

```

```

    Q_out := TON_1.Q;

END_FUNCTION_BLOCK

FUNCTION_BLOCK TON_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        Ton_in : UDINT := 5000; //milliseconds
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        TON_1: SYS.TON;
        Ton_time: TIME;
    END_VAR

    //code area

    Ton_time := UDINT_TO_TIME(Ton_in);
    TON_1(I := I_in, T := Ton_time, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.2.3 OFF-delay timer (SYS.TOF)

The OFF-delay timer (SYS.TOF) is used to delay the output off. The timer output Q will show a HIGH level signal on the rising edge of the signal at the input I, the countdown of the off-delay time T will start on each falling edge of the input signal. After the input signal is off, the output will show a LOW level signal with a delay of T.

Designation	Data type	Description
Inputs		
I	BOOL	Timer start (on falling edge)
T	TIME	Delay on power-on
Outputs		
Q	BOOL	Timer output

```

FUNCTION_BLOCK TOF_timer //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
    END_VAR

```

```

VAR_OUTPUT //declaration of output variables
    Q_out : BOOL;
END_VAR

VAR
    TOF_1: SYS.TOF;
END_VAR

//code area

TOF_1(I := I_in, T := T#1000ms);
//where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days
Q_out := TOF_1.Q;

END_FUNCTION_BLOCK

FUNCTION_BLOCK TOF_timer //Function block name

VAR_INPUT //declaration of input variables
    I_in : BOOL := FALSE;
    Tof_in : UDINT := 5000; //milliseconds
END_VAR

VAR_OUTPUT //declaration of output variables
    Q_out : BOOL;
END_VAR

VAR
    TOF_1: SYS.TOF;
    Tof_time: TIME;
END_VAR

//code area

Tof_time := UDINT_TO_TIME(Tof_in);
TOF_1(I := I_in, T := Tof_time, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.2.4 Timer (SYS.CLOCK)

The timer (SYS.CLOCK) is used to generate a pulse to turn on the Q output according to the real-time clock. The output turn-on time T_{on} and turn-off time T_{off} are set as timer parameters.

Designation	Data type	Description
Inputs		
Ton	DT	Turn-on time
Toff	DT	Shutdown time
Mask	UDINT	Selection of quantities to be used
Output		
Q	BOOL	Timer output

**NOTE**

Specifying the Mask variable is optional.

If the value of the Mask variable is not specified, the block defaults to a Mask = 63 (0b111111),

Where:

Mask = 63 (0b111111)

0 bit – if 1, then seconds are used

1 bit – if 1, then minutes are used

2 bits – if 1, then the hours is used

3 bits – if 1, then days are used

4 bits – if 1, then months are used

5 bits – if 1, then years are used

```
FUNCTION_BLOCK CLOCK_timer //Function block name
```

```
VAR_INPUT //declaration of input variables
```

```
    I_in : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT //declaration of output variables
```

```
    Q_out : BOOL;
```

```
END_VAR
```

```
VAR
```

```
    CLOCK_1: SYS.CLOCK;
```

```
END_VAR
```

```
//code area
```

```
CLOCK_1(Ton := DT#2023-09-28-7:20:55, Toff := DT#2023-09-28-12:30:59);
```

```
Q_out := CLOCK_1.Q;
```

```
END_FUNCTION_BLOCK
```


12.6.2.5 Weekly timer (SYS.CLOCKWEEK)

The weekly timer (SYS.CLOCKWEEK) is used to generate a pulse to turn on the output Q according to the real-time clock, taking into account the days of the week. The time of turning on T_{on} and turning off T_{off} from the output Q and the days of the week of operation are set as timer parameters.

Designation	Data type	Description
Inputs		
Ton	DT	Turn-on time
Toff	DT	Shutdown time
DayOfWeekMask	UDINT	Selecting the days to use
DateTimeMask	UDINT	Selection of quantities to be used
Outputs		
Q	BOOL	Timer output

**NOTE**

Specifying the DayOfWeekMask and DateTimeMask variables is optional.

If the value of the DayOfWeekMask variable is not specified, then the block defaults to a DayOfWeekMask = 127 (0b1111111),

If the value of the DateTimeMask variable is not specified, then the block defaults to a DateTimeMask = 63 (0b111111),

Where:

DayOfWeekMask = 127 (0b1111111)

0 bit – if 1, then Mondays are taken into account

1 bit – if 1, then Tuesdays are taken into account

2 bits – if 1, then Wednesdays are taken into account

3 bits – if 1, then Thursdays are taken into account

4 bits – if 1, then Fridays are taken into account

5 bits – if 1, then Saturdays are taken into account

6 bits – if 1, then Sundays are taken into account

DateTimeMask = 63 (0b111111)

0 bit - if 1, then seconds are used

1 bit - if 1, then minutes are used

2 bits - if 1, then hours are used

3 bits - if 1, then days are used

4 bits - if 1, then months are used

5 bits - if 1, then years are used

```
FUNCTION_BLOCK CLOCKWEEK_timer //Function block name
```

```
VAR_INPUT //declaration of input variables
```

```
    I_in : BOOL;
```

```
END_VAR
```

```
VAR_OUTPUT //declaration of output variables
```

```
    Q_out : BOOL;
```

```
END_VAR
```

```
VAR
```

```
    CLOCKWEEK_1: SYS.CLOCKWEEK;
```

```
END_VAR
```

```

//code area

CLOCKWEEK_1 (Ton := DT#2023-09-28-7:20:55, Toff := DT#2023-09-28-12:30:59);
Q_out := CLOCKWEEK_1.Q;
END_FUNCTION_BLOCK

```

12.6.3 Generators

– Pulse generator (SYS.BLINK)

12.6.3.1 Pulse generator (SYS.BLINK)

The pulse generator (SYS.BLINK) is used to form rectangular pulses. At the output Q of the generator, pulses are formed with specified parameters of the duration of the on (Th – HIGH level signal) and off (Tl – LOW level signal) state for the duration of the control signal at the input I (HIGH level signal).

Designation	Data type	Description
Inputs		
I	BOOL	Work permission
Th	TIME	Duration of a logical unit
Tl	TIME	Logical zero duration
Outputs		
Q	BOOL	Generator output

```

FUNCTION_BLOCK BLINK_generator //Function block name

VAR_INPUT //declaration of input variables
    I_in : BOOL := FALSE;
END_VAR

VAR_OUTPUT //declaration of output variables
    Q_out : BOOL;
END_VAR

VAR
    BLINK_1: SYS.BLINK;
END_VAR

//code area

BLINK_1(I := I_in, Th := T#1000ms, Tl := T#1000ms);
//where ms is milliseconds, s is seconds, m is minutes, h is hours, d is days

Q_out := BLINK_1.Q;

```

```

END_FUNCTION_BLOCK

FUNCTION_BLOCK BLINK_generator //Function block name

    VAR_INPUT //declaration of input variables
        I_in : BOOL := FALSE;
        Th_in : UDINT := 5000; //milliseconds
        Tl_in : UDINT := 5000; //milliseconds
    END_VAR

    VAR_OUTPUT //declaration of output variables
        Q_out : BOOL;
    END_VAR

    VAR
        BLINK_1: SYS.BLINK;
        Th_time: TIME;
        Tl_time: TIME;
    END_VAR

    //code area

    Th_time := UDINT_TO_TIME(Th_in);
    Tl_time := UDINT_TO_TIME(Tl_in);
    BLINK_1(I := I_in, Th := Th_time, Tl := Tl_time, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.4 Counters

- Threshold counter with self-reset (SYS.CT)
- Universal counter (SYS.CTN)
- Threshold counter (SYS.CTU)

12.6.4.1 Threshold counter with self-reset (SYS.CT)

The threshold counter with self-reset (SYS.CT) is used to count a specified number of pulses N (input N is the pulse number setting). At the output Q of the counter, a pulse of the HIGH level signal with the duration of the device working cycle (cycle time) will appear if the number of pulses arriving at the input C reaches the set value N.

Designation	Data type	Description
Inputs		
C	BOOL	Counter input
N	UDINT	Counter setting
Outputs		
Q	BOOL	Signaling of the setpoint reached (for one cycle)

```

FUNCTION_BLOCK CT_counter //Function block name

```

```

VAR_INPUT //declaration of input variables
    C_in : BOOL;
    N_in : UDINT := 10;
END_VAR

VAR_OUTPUT //declaration of output variables
    Q_out : BOOL;
END_VAR

VAR //declaration of local variables
    CT_1: SYS.CT;
END_VAR

//code area

CT_1(C := C_in, N := N_in, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.4.2 Universal counter (SYS.CTN)

The universal counter (SYS.CTN) is used for direct and indirect counting. The "direct counting" operation is performed by the rising edge of the pulse at the direct counting input U, which increases the value of the output signal Q. Incoming pulse on the input D ("decrease counting") decrease the value of the output Q. If a logical "1" signal comes to the input R, the output of the counter Q is set to the value of the input N.

Designation	Data type	Description
Inputs		
U	BOOL	Direct Counting
D	BOOL	Countdown
R	BOOL	Reset the output state to value N
N	UDINT	Setpoint
Outputs		
Q	UDINT	Cumulative value of pulses

```

FUNCTION_BLOCK CTN_counter //Function block name

VAR_INPUT //declaration of input variables
    U_in : BOOL;
    D_in : BOOL;
    R_in : BOOL;
    N_in : UDINT := 10;
END_VAR

VAR_OUTPUT //declaration of output variables

```

```

        Q_out : UDINT;
    END_VAR

    VAR //declaration of local variables

        CTN_1: SYS.CTN;
    END_VAR

    //code area

    CTN_1(U := U_in, D := D_in, R := R_in, N := N_in, Q => Q_out);

END_FUNCTION_BLOCK

```

12.6.4.3 Threshold counter (SYS.CTU)

The threshold counter (SYS.CTU) is used to count the number of incoming pulse on the C input. A pulse of HIGH level signal will appear at the Q counter output if the number of incoming pulses on the input reaches the set value at the N input (N is the setpoint).

Designation	Data type	Description
Inputs		
U	BOOL	Direct Counting
R	BOOL	Reset the counter state to 0
N	UDINT	Setpoint
Outputs		
Q	BOOL	Signaling of setpoint reached

```

FUNCTION_BLOCK CTU_counter //Function block name

    VAR_INPUT //declaration of input variables
        C_in : BOOL;
        R_in : BOOL;
        N_in : UDINT := 10;
    END_VAR

    VAR_OUTPUT //declaration of output variables

        Q_out : BOOL;
    END_VAR

    VAR //declaration of local variables
        CTU_1: SYS.CTU;
    END_VAR

    //code area

```

```
CTU_1(C := C_in, R := R_in, N := N_in, Q => Q_out);  
  
END_FUNCTION_BLOCK
```