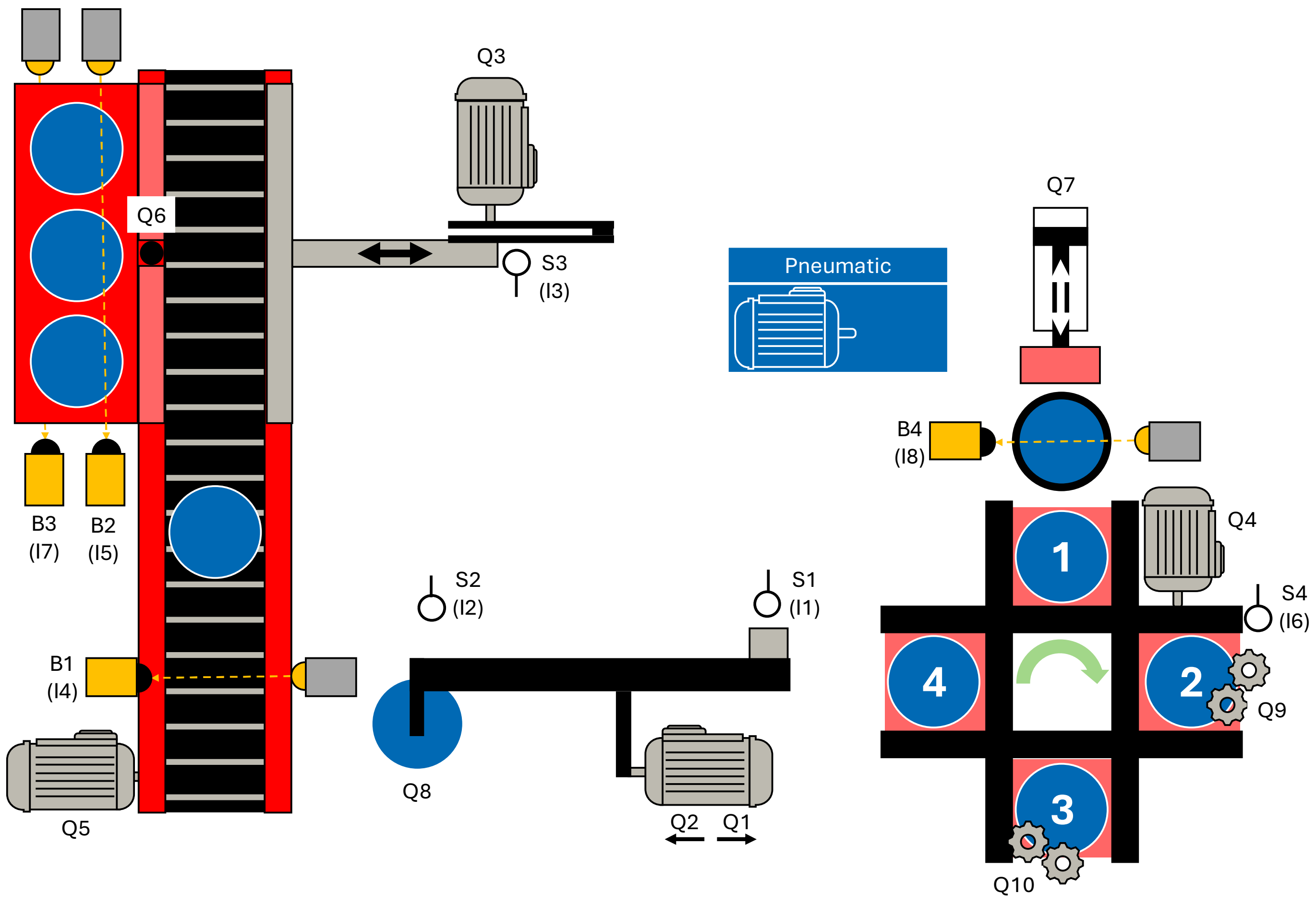
fischertechnik

GROLLMUS
KNOWLEDGE | SOFTWARE | AUTOMATION

# 24V production line

Commissioning (software)



## Table of contents

# 8 Commissioning (software)

## 8.1 Introduction

Once you have ensured that the hardware and wiring have been checked and are free of errors, you can start commissioning the previously created user program. As far as possible, the program should be commissioned and tested section by section. A procedure based on the principle of "transfer everything and wait and see what happens" is less suitable for this. A checklist can be a suitable aid for this.

When switching on the devices and commissioning the user program, a careful and considered approach must be taken to ensure that any remaining hardware or program errors cannot lead to danger for people, the environment or system components at any time.

In addition to the familiar test functions (e.g. monitoring table / monitoring list), the following functions are available for commissioning the software

- Observe (program status),
- Reference lists (cross-references)

available.

The creation of backup files is mandatory. Any changes made during commissioning or the subsequent optimization phase must of course be updated in the system documentation, symbols, etc.

The commissioning of a user program can be simplified by dividing it into individual sub-functions (structured programming). In cyclical "MAIN", only certain sub-functions are called up and thus put into operation step by step.

This often requires signal states that are created in other parts of the program that have not yet been put into operation. The signal states (e.g. outputs, flags, variables in data blocks) can be simulated in watch tables using the "Watch/control variables" function.

## 8.2 Program status

After loading the user program into the PLC, the current program status can be tracked on the PG. The program status allows you to monitor the program sequence. The values of the operands and the logic results (VKE) are displayed. This allows you to find and correct errors in the program logic, for example.

The program module to be monitored must be open in the workspace. Pressing the button in the function bar of the workspace establishes an online connection and activates the display of the program status. The display of the statuses depends on the selected programming language.

### 8.2.1 Program status in FUP

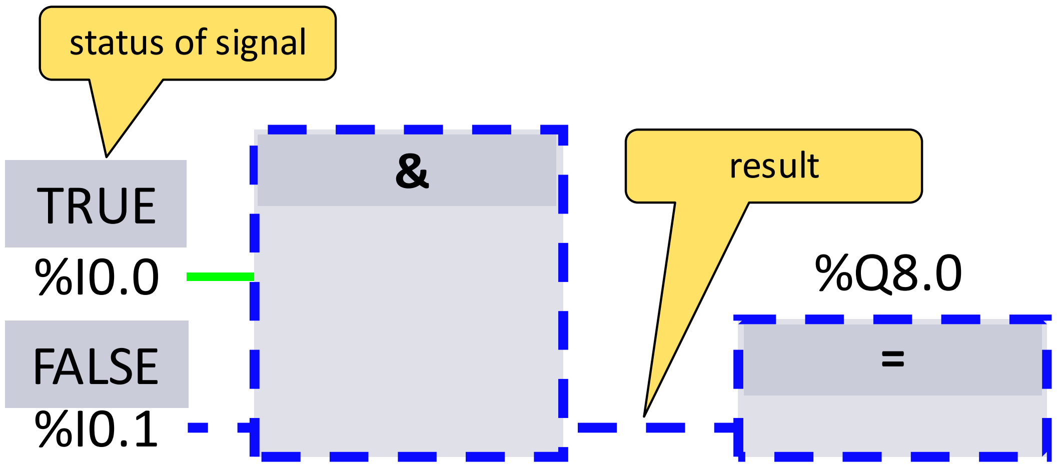
The following table summarizes the assignment of colors, display and status in the program editor.

| Representation | Status |
|---|---|
| Green solid | fulfilled |
| Blue dashed | not fulfilled |
| Gray solid | unknown or not run through |
| Black | not wired |
| Parameters in a frame with a saturation of 100% | Value is current |
| Parameters in a frame with a saturation of 50% | Value comes from a previous cycle. The program point was not run through in the current cycle. |

Table 1 Status in the FBD program editor

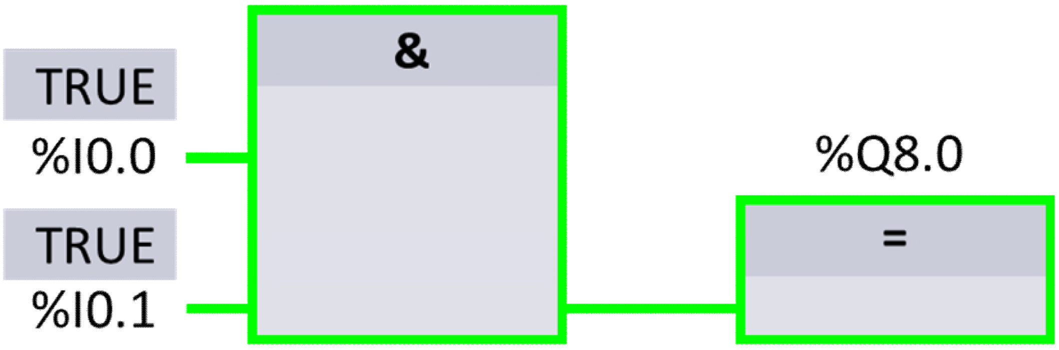
The pictures below show examples of the different program status displays in functional plan view.

Blue dashed - condition not fulfilled



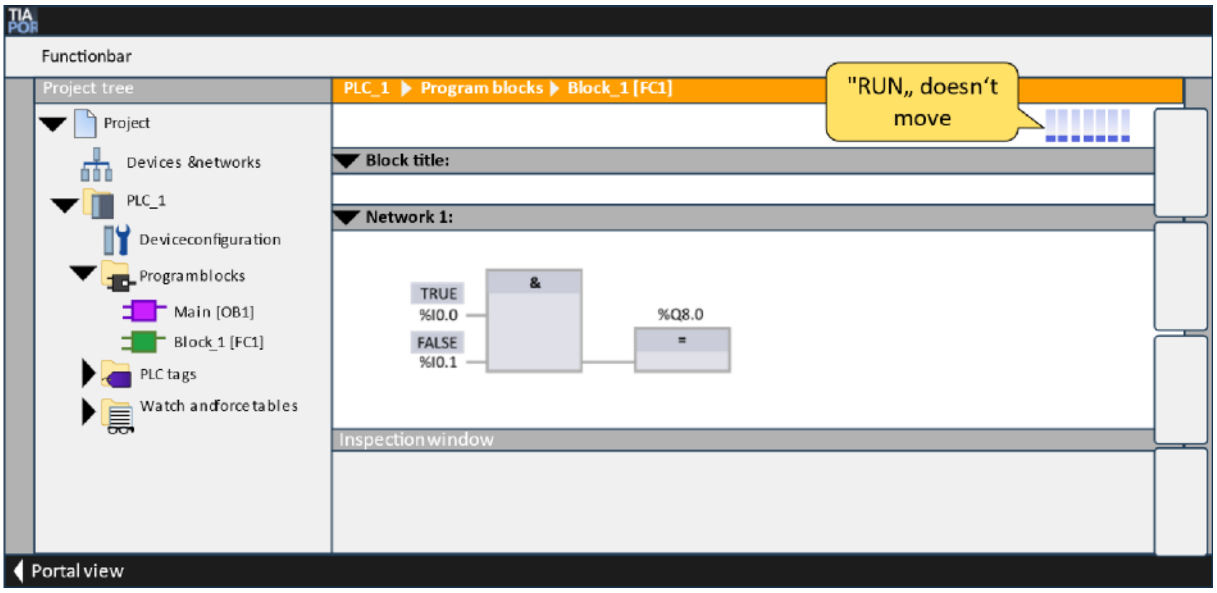
Picture 1 Program status display in FBD / condition not fulfilled

## Green solid - condition fulfilled



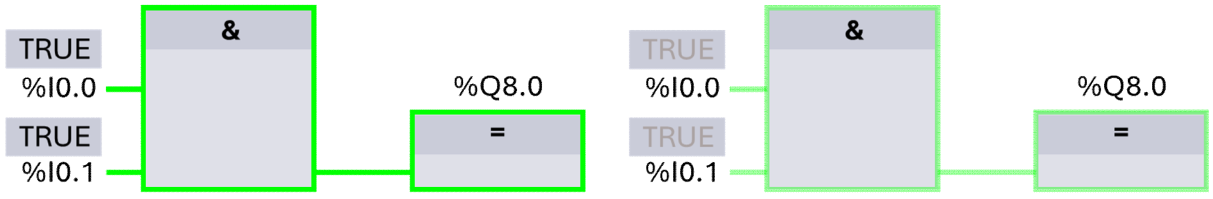
Picture 2 Program status display in FBD / condition fulfilled

## Gray solid - unknown or not run through



Picture 3 Program status display in FBD / block is not being processed

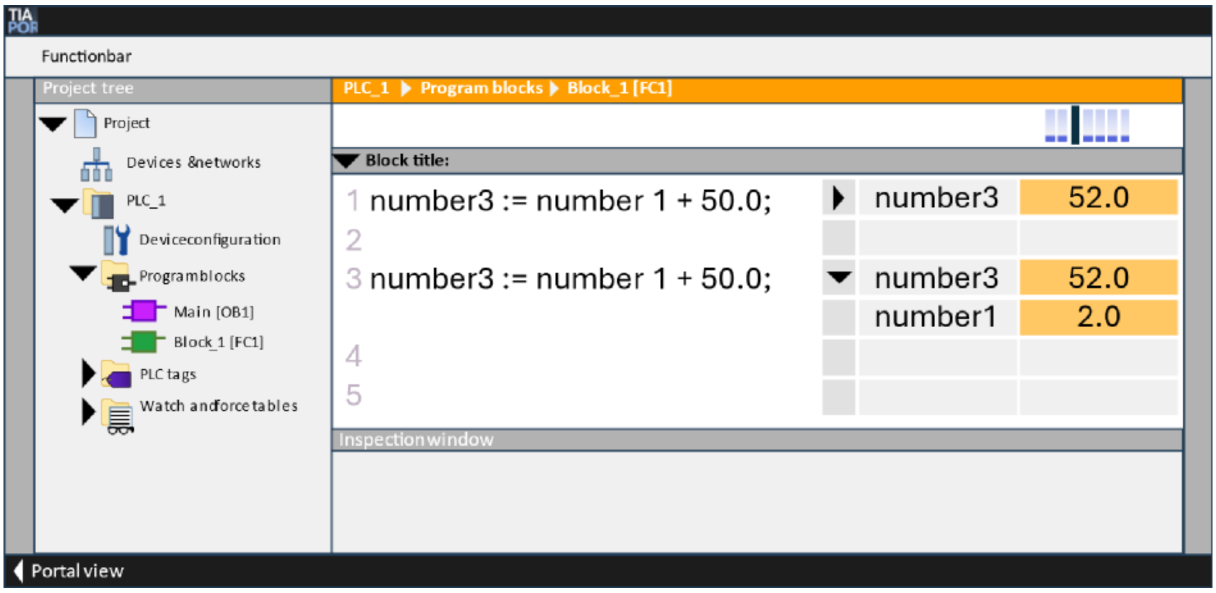
## Parameters in a frame with a saturation of 100 % or 50 %



Picture 4 Value is current or from previous cycle

### 8.2.2 Program status in SCL

The program status display is updated cyclically and shown in a table. The table is displayed directly next to the SCL program and you can read the program status for each program line. You can move the table to the right or left and expand it line by line using the triangle ▶ . The table contains the variable name and its value.



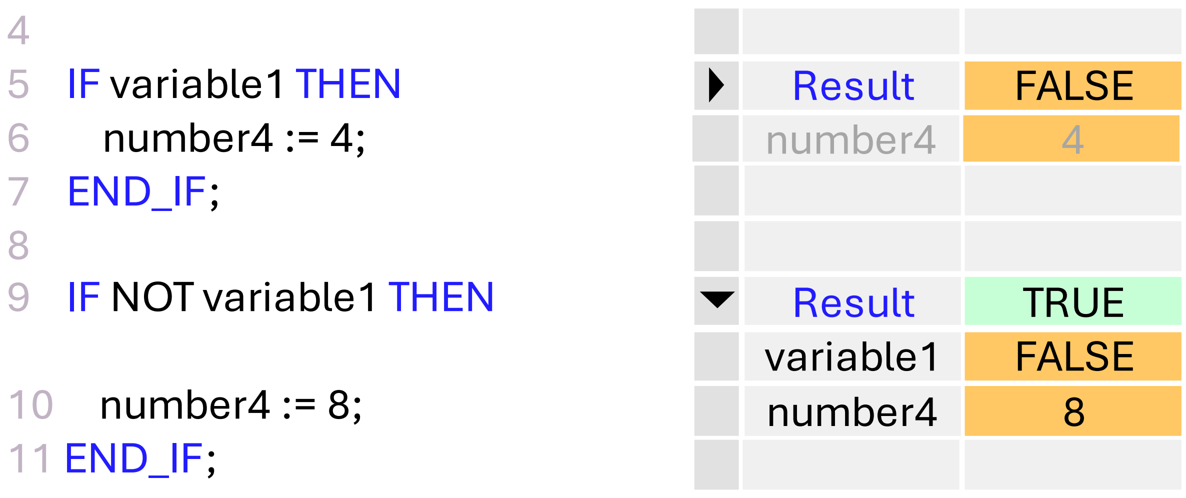
```
1 number3 := number 1 + 50.0;
2
3 number3 := number 1 + 50.0;

4
5
```

| | | |
|---|---|---|
| ▶ | number3 | 52.0 |
| | | |
| ▼ | number3 | 52.0 |
| | number1 | 2.0 |
| | | |
| | | |

Picture 5 Program status display in SCL / line 3 expanded

If the status does not originate from the current cycle, for example because the IF statement is not currently fulfilled, both the variable and the status are displayed in gray.
If the result of an IF statement is fulfilled, this is highlighted in green.



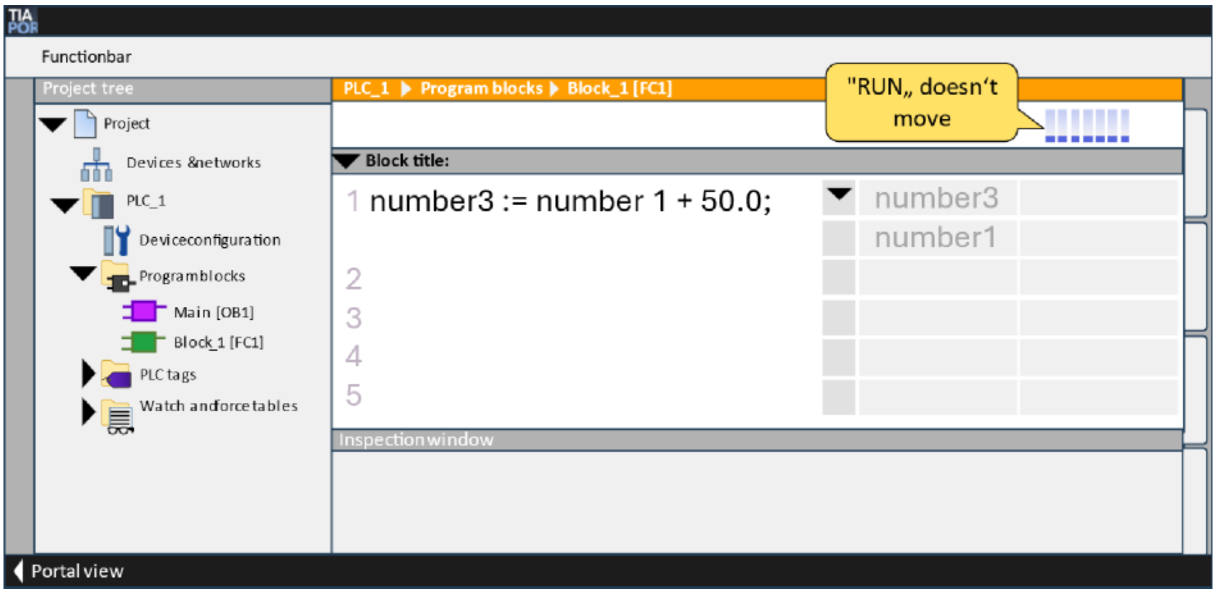
```
4
5  IF variable1 THEN
6     number4 := 4;
7  END_IF;
8
9  IF NOT variable1 THEN

10    number4 := 8;
11 END_IF;
```

| | | |
|---|---|---|
| | | |
| ▶ | Result | FALSE |
| | number4 | 4 |
| | | |
| | | |
| ▼ | Result | TRUE |
| | variable1 | FALSE |
| | number4 | 8 |
| | | |

Picture 6 Program status display in SCL / IF instruction

A gray status window is a sign that the program code has not yet been executed. This may be because the IF statement, for example, is only processed conditionally and this has not yet been fulfilled. However, if the "RUN bar" does not move either, this is an indication that the block is not being processed because it has not been called or the controller is in STOP.



```
number3 := number 1 + 50.0;
```

Picture 7 Program status display in SCL / block is not being processed

## 8.3 Cross-references

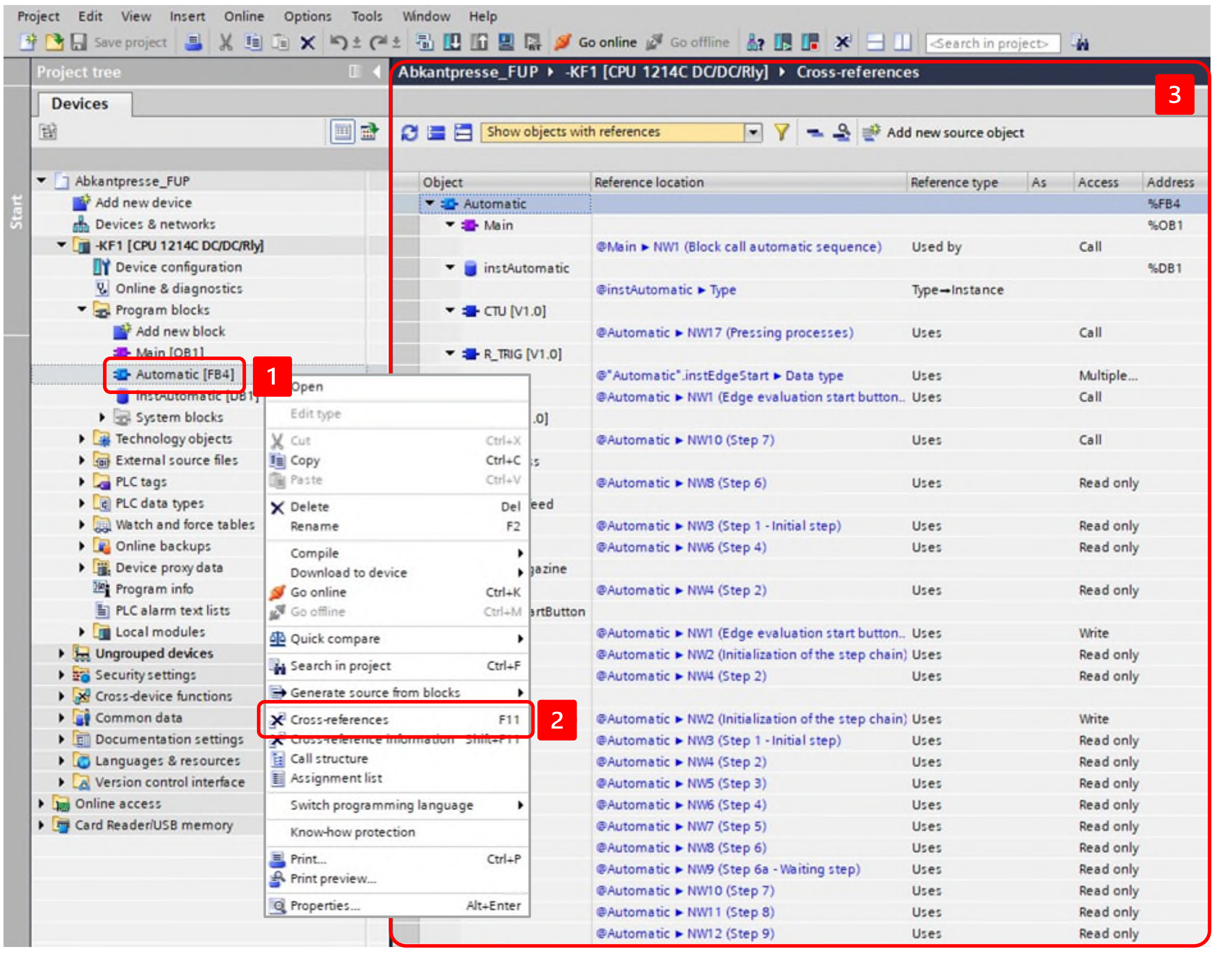
The cross-references provide an overview of the use of objects and devices within the project. The cross-reference list can be used to display the relationships and dependencies between the objects.
The list depends on the area from which you query the cross-references.

The cross-references are opened in the TIA portal in the context menu of the corresponding object via "Cross-references". Alternatively, the shortcut F11 can also be used.

The object for which the cross-references are listed (=source object) is shown in the first line of the list.

### Source object Module

The "Automatic" function block in the project navigation was selected as the source object in the following screen.
The cross-references list the location of the call (here: "MAIN") and the assigned instance data block (here: "instAutomatik"), as well as all variables and functions used in the block.

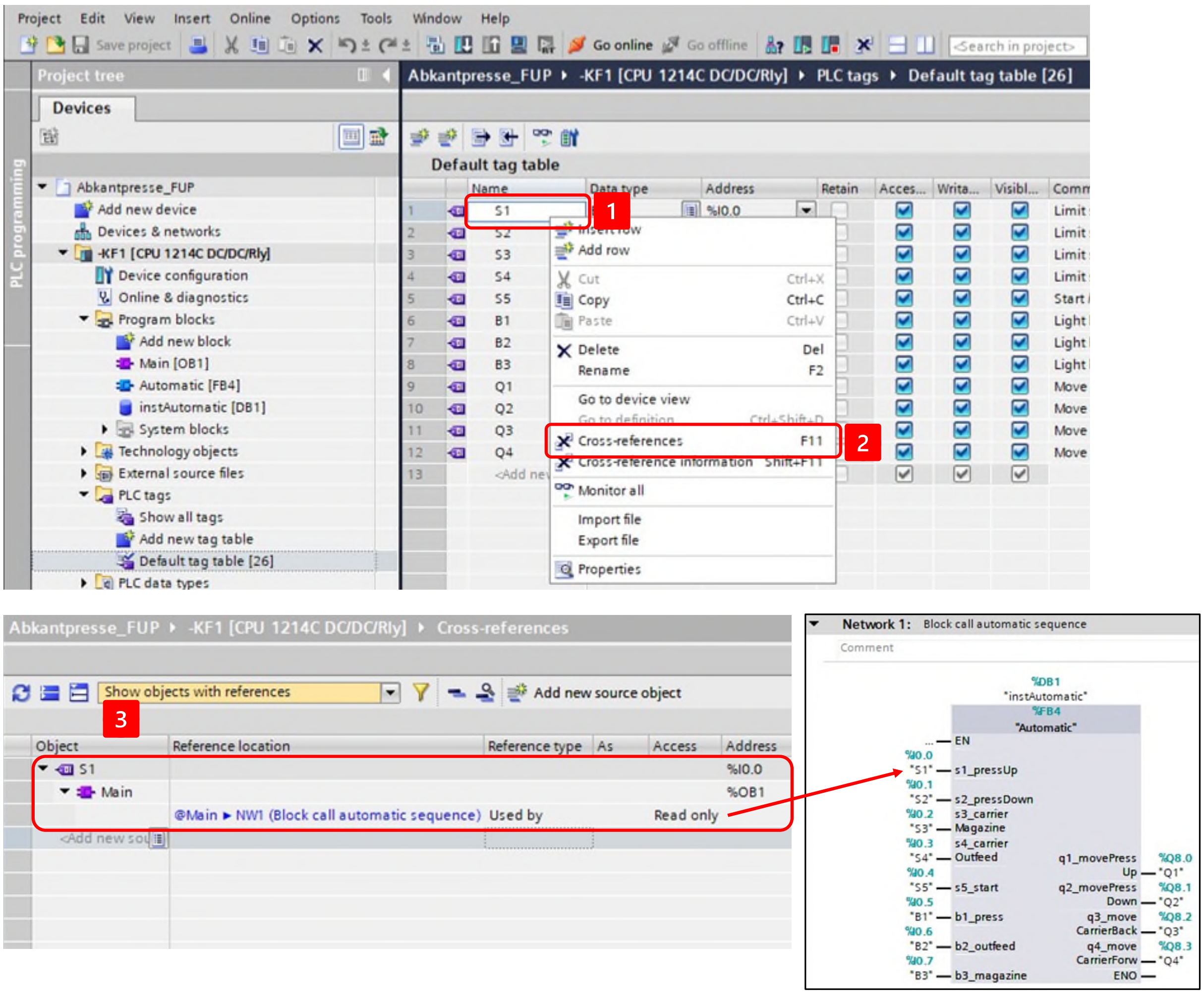


| Object | Reference location | Reference type | As | Access | Address |
|---|---|---|---|---|---|
| Automatic | | | | | %FB4 |
| Main | | | | | %OB1 |
| | @Main ▸ NW1 (Block call automatic sequence) | Used by | | Call | |
| instAutomatic | | | | | %DB1 |
| | @instAutomatic ▸ Type | Type→Instance | | | |
| CTU [V1.0] | | | | | |
| | @Automatic ▸ NW17 (Pressing processes) | Uses | | Call | |
| R_TRIG [V1.0] | | | | | |
| | @"Automatic".instEdgeStart ▸ Data type | Uses | | Multiple... | |
| | @Automatic ▸ NW1 (Edge evaluation start button... | Uses | | Call | |
| .0] | | | | | |
| | @Automatic ▸ NW10 (Step 7) | Uses | | Call | |
| s | | | | | |
| | @Automatic ▸ NW8 (Step 6) | Uses | | Read only | |
| eed | | | | | |
| | @Automatic ▸ NW3 (Step 1 - Initial step) | Uses | | Read only | |
| | @Automatic ▸ NW6 (Step 4) | Uses | | Read only | |
| gazine | | | | | |
| | @Automatic ▸ NW4 (Step 2) | Uses | | Read only | |
| rtButton | | | | | |
| | @Automatic ▸ NW1 (Edge evaluation start button... | Uses | | Write | |
| | @Automatic ▸ NW2 (Initialization of the step chain) | Uses | | Read only | |
| | @Automatic ▸ NW4 (Step 2) | Uses | | Read only | |
| | | | | | |
| | @Automatic ▸ NW2 (Initialization of the step chain) | Uses | | Write | |
| | @Automatic ▸ NW3 (Step 1 - Initial step) | Uses | | Read only | |
| | @Automatic ▸ NW4 (Step 2) | Uses | | Read only | |
| | @Automatic ▸ NW5 (Step 3) | Uses | | Read only | |
| | @Automatic ▸ NW6 (Step 4) | Uses | | Read only | |
| | @Automatic ▸ NW7 (Step 5) | Uses | | Read only | |
| | @Automatic ▸ NW8 (Step 6) | Uses | | Read only | |
| | @Automatic ▸ NW9 (Step 6a - Waiting step) | Uses | | Read only | |
| | @Automatic ▸ NW10 (Step 7) | Uses | | Read only | |
| | @Automatic ▸ NW11 (Step 8) | Uses | | Read only | |
| | @Automatic ▸ NW12 (Step 9) | Uses | | Read only | |

Picture 8 Cross-references of a function module

## Source object global variable

If a global variable is selected, the cross-references for this variable can also be displayed. The cross-references can be opened from any usage location of the variable, so they can be opened directly in the program or, as shown below, by selecting the variable from the PLC variable table.



| | Name | Data type | Address | Retain | Acces... | Writa... | Visibl... | Comm |
|---|---|---|---|---|---|---|---|---|
| 1 | S1 | | %I0.0 | | ✓ | ✓ | ✓ | Limit |
| 2 | S2 | | | | ✓ | ✓ | ✓ | Limit |
| 3 | S3 | | | | ✓ | ✓ | ✓ | Limit |
| 4 | S4 | | | | ✓ | ✓ | ✓ | Limit |
| 5 | S5 | | | | ✓ | ✓ | ✓ | Start |
| 6 | B1 | | | | ✓ | ✓ | ✓ | Light |
| 7 | B2 | | | | ✓ | ✓ | ✓ | Light |
| 8 | B3 | | | | ✓ | ✓ | ✓ | Light |
| 9 | Q1 | | | | ✓ | ✓ | ✓ | Move |
| 10 | Q2 | | | | ✓ | ✓ | ✓ | Move |
| 11 | Q3 | | | | ✓ | ✓ | ✓ | Move |
| 12 | Q4 | | | | ✓ | ✓ | ✓ | Move |
| 13 | <Add nev | | | | ✓ | ✓ | ✓ | |

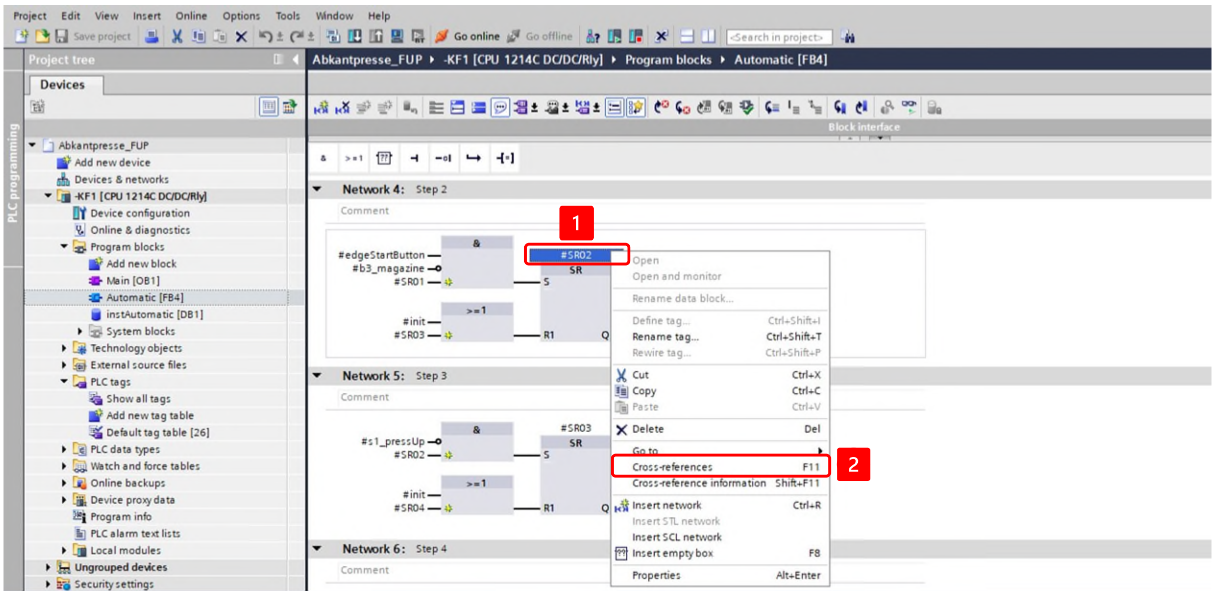
| Object | Reference location | Reference type | As | Access | Address |
|---|---|---|---|---|---|
| S1 | | | | | %I0.0 |
| Main | | | | | %OB1 |
| | @Main ▸ NW1 (Block call automatic sequence) | Used by | | Read only | |
| <Add new sou | | | | | |

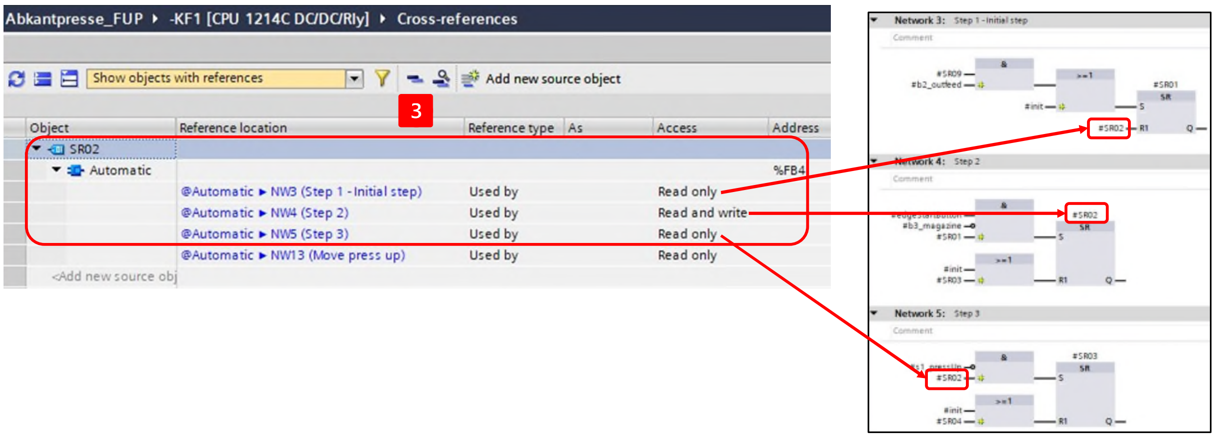
Picture 9 Cross-references of a global variable

For the variable "S1" selected as the source object, for example, the read access to the variable in the first network of the MAIN is shown as the usage point in the above image when the block is called.

## Source object local variable

If a local variable is selected, the cross-references for this variable can also be displayed. The cross-references can be opened from any usage location of the variable, so they can be opened directly in the definition, in the block interface or, as shown below, by selecting the variable in the program editor.




Abkantpresse_FUP ▸ -KF1 [CPU 1214C DC/DC/Rly] ▸ Cross-references
Show objects with references
Add new source object
3
Network 3: Step 1 - Initial step
Network 4: Step 2
Network 5: Step 3
#SR02

| Object | Reference location | Reference type | As | Access | Address |
|---|---|---|---|---|---|
| SR02 | | | | | |
| Automatic | | | | | %FB4 |
| | @Automatic ▸ NW3 (Step 1 - Initial step) | Used by | | Read only | |
| | @Automatic ▸ NW4 (Step 2) | Used by | | Read and write | |
| | @Automatic ▸ NW5 (Step 3) | Used by | | Read only | |
| | @Automatic ▸ NW13 (Move press up) | Used by | | Read only | |
| <Add new source obj | | | | | |

Picture 10 Cross-references of a local variable

The selected variable "SR02" has four usage locations. Three of these four usage locations are directly assigned to the step chain:

- Step marker
- Reset the previous step
- Set the following step if the transition condition is fulfilled