

4 Editors

Editors

4.1 CFC Editor

4.1.1 CFC Editor, General Information

The CFC editor is used to program objects in the **Continuous Function Chart** () language available in the standard languages of IEC 61131-3.

The **programming language** for a new object is specified when the object is created using the .

The CFC editor is a graphical editor.

The editor is located in the lower section of the window that opens when a CFC object is edited.

The upper section contains the

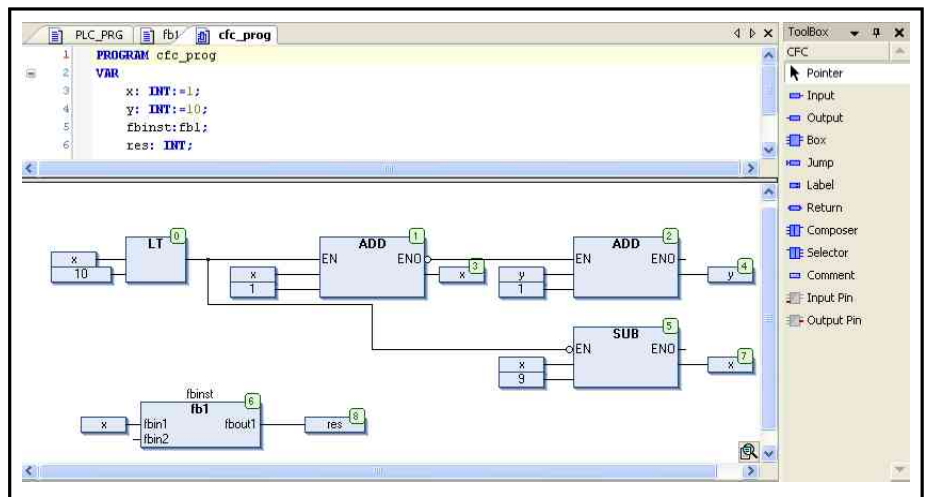


Fig.4-1: CFC editor

In contrast to the network-based editors, the CFC editor allows free of elements which enables feedbacks to be inserted directly for example. The processing sequence is determined by a list of the currently inserted elements which can also be modified.

and are ready to be added: Function block (operators, functions, function blocks, programs), input, output, comment, label, jump, composition selector.

The input and output pins of the elements can be connected using the mouse to draw a line between them.

The connection line is automatically created so that it covers the shortest possible distance between the elements.

The connection lines are also adjusted automatically when an element is moved. Also see:

The size of the working sheet is adjustable:



To do this, use the button in the lower right corner of the window and select one of the zoom factors from the list. Alternatively, select the list entry ... to enter any desired factor.

Call the [commands, page 268](#), to work in the CFC editor from the context menu or the CFC menu available by default when the CFC editor is active.

Editors



The **CFC Editor** under **Tools ▶ Options ▶ IndraLogic 2G ▶ CFC Editor** connects elements whose unassigned connections are in contact with each other.

4.1.2 CFC - Continuous Function Chart - Language

"Continuous Function Chart" is a graphical programming language, an extension of the IEC 61131-3 standard languages based on the (FBD). However, in contrast to that language, CFC allows the free positioning of elements to insert feedback for example. To generate CFC program blocks in IndraLogic, see: CFC Editor.

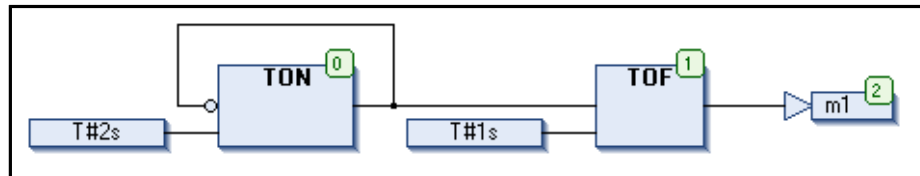


Fig.4-2: Example of a CFC network

4.1.3 Cursor Positions in CFC

In the CFC editor, possible cursor positions are indicated by default with gray shading when the cursor is moved over the inserted elements. Click on one of these shaded areas and hold down the mouse button. The color of this area changes to red. When you release the mouse button, the position becomes the current cursor position, i.e. the respective element is selected and displayed in red.

There are three categories of cursor positions. In the following, the possible positions that are not yet in use are each indicated by gray shading:

1. When the cursor is positioned on a text, it is displayed with a blue background and can be edited.

The button can be used to open the input assistance. After the element has been inserted, three question marks "???" appear to represent the text. These have to be replaced by a valid identifier. Afterwards, when the cursor is positioned on the name of a variable or a box parameter, a tooltip is displayed that contains the type of the variable/parameter and a related comment in a second line if available.

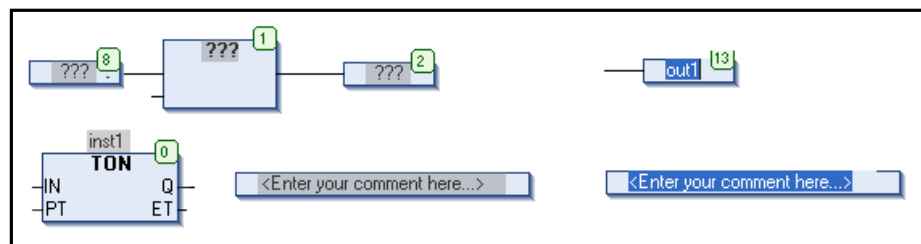


Fig.4-3: Possible cursor positions and two examples of selected text in the CFC

2. If the cursor is positioned on the **body of an element** (function block, input, output, comment, label, jump, composition, selector), the body is displayed in red and can be moved with the mouse.

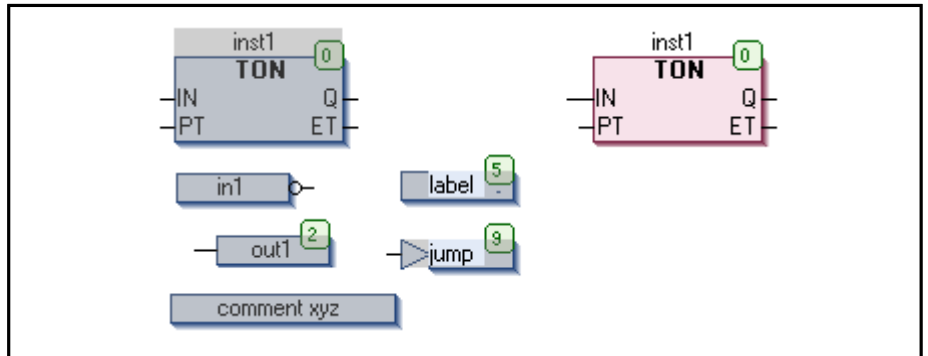


Fig.4-4: Possible cursor positions and an example of a selected function block (box)

- If the cursor is positioned on an **input** or **output** pin of an element, the pin is displayed in red and can be negated, set or reset.

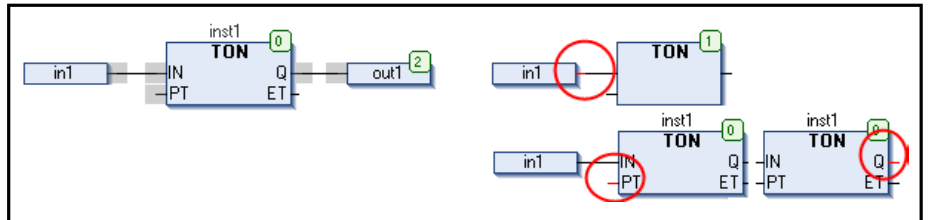


Fig.4-5: Possible cursor positions and examples of selected input and output pins

4.1.4 CFC Elements/Tools

The graphical elements for programming in the are in a toolbox.
The toolbox usually opens automatically in a window to the right of the working sheet when the CFC editor is active, but it can also be opened explicitly using the command **View ▶ Tools**

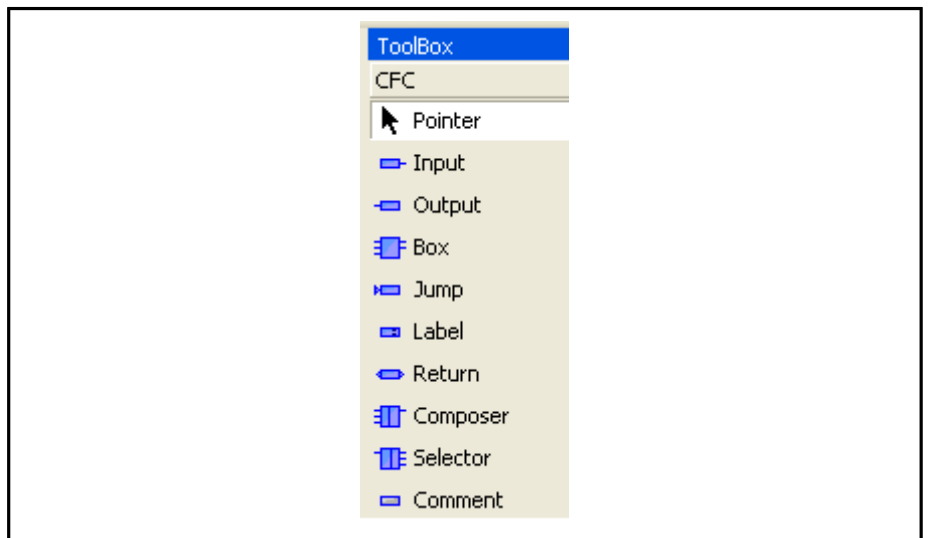


Fig.4-6: CFC tools, standard configuration




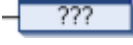

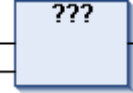



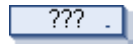



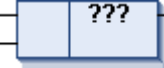
The desired element is selected in the toolbox and in the editor window via drag&drop.


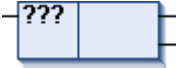



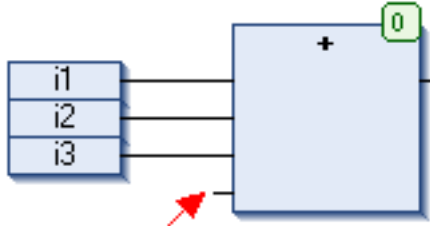

In addition to the programming elements, there is also a "Pointer" entry, usually at the top of the tool list. While this item is selected, the cursor appears as arrow and elements already inserted can be selected to and edit them.



Editors

CFC elements:

	Input		The text represented by "???" can be selected and replaced by a variable name or a constant. The input assistance can be used to select a valid identifier.
	Output		
	Function block (Box)		<p>A function block element is used to insert an operator, a function, a function block or a program. The "???" have to be replaced by the name of the desired operator, etc. The input assistance supports the selection of an available object.</p> <p>For function blocks, "???" are also displayed above the function block symbol. These have to be replaced with the name of the function block instance.</p> <p>If the function block is inserted in place of an existing function block (the existing function block is selected and the new one inserted) and if it has a different minimum number of inputs, these are automatically added. If the function block has a lower maximum number of inputs, the final inputs of the previously existing element are deleted.</p>
	Jump		<p>The jump element is used to specify a position at which the program execution is continued. This position has to be defined by a "label" (see below).</p> <p>The "???" text has to be replaced as well with the name of the label.</p>
	Label		<p>A label indicates a position to which the program execution can jump using a jump element (see above).</p> <p>In online mode, a RETURN label is automatically placed at the end of a CFC function block.</p>
	Return		<p>Note that in online mode in the CFC editor, a RETURN element is automatically placed in front of the first line and after the last element in the function block. In step-by-step processing, a jump is made to the RETURN element at the end before the function block is exited.</p>
	Composer		<p>A composition element is used to handle inputs from a function block of the type of a structure. The composer displays the structure components and provides them to the programmer in CFC. To do this, the composer element has to have the same name as the respective structure (replace the "???") and has to be connected to the function block instead of to an input element.</p>

	Selector		<p>In contrast to the composition element, a selector element is used to handle the output of a function block of the type of a structure. The selector displays the structure components and provides them to the programmer in CFC. To do this, the selector has to have the same name as the respective structure (replace the "??") and has to be connected to the function block instead of to an output element.</p>
	Comment		<p>This element can be used to insert a comment in CFC. Replace the placeholder text in the element with the comment text. A line break can be added with the shortcut <Ctrl> + <Enter>.</p>
	Function block input (Input Pin)		<p>Depending on the function block type, more inputs can be added to an inserted function block element. To do this, the function block element has to be selected and the function block input element has to be dragged onto the function block body.</p>
	Function block output (Output Pin)		<p>Depending on the function block type, more outputs can be added to an inserted function block element. To do this, the function block element has to be selected and the function block output element has to be dragged onto the function block body.</p>

Example: Composer A CFC program cfc_prog uses an instance of the function block "fublo1" which has an input variable "struvar" of the type of a structure. The composition elements can be used to address the structure elements.

Definition of structure "stru1":

```
TYPE stru1 :
STRUCT
  ivar:INT;
  strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Function block "fublo1", declaration and implementation:

```
FUNCTION_BLOCK fublo1
VAR_INPUT
  struvar:STRU1;
END_VAR
VAR_OUTPUT
  fbout_i:INT;
  fbout_str:STRING;
END_VAR
VAR
  fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

Editors

CFC program "cfc_prog", declaration and implementation:

```
PROGRAM cfc_prog
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
END_VAR
```

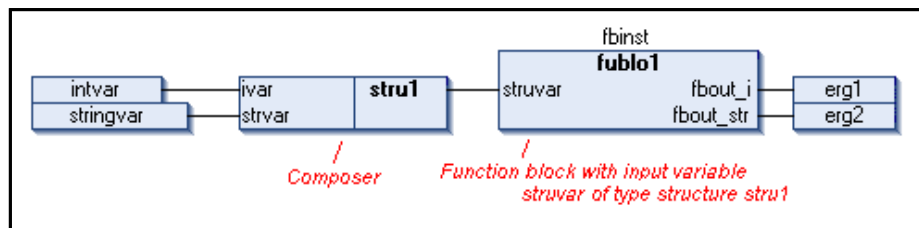


Fig.4-7: Example: Composer

Example: Selector

A CFC program cfc_prog uses an instance of the function block "fublo2", which has an output variable "fbout" of the structure type "stru1". The selector elements can be used to access the structure elements:

Definition of structure "stru1":

```
TYPE stru1 :
STRUCT
  ivar: INT;
  strvar: STRING := 'hallo';
END_STRUCT
END_TYPE
```

Function block "fublo2", declaration and implementation:

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
  fbin1: INT;
  fbin2: DWORD := 24354333;
  fbin3: STRING := 'hallo';
END_VAR
VAR_INPUT
  fbin : INT;
END_VAR
VAR_OUTPUT
  fbout : stru1;
  fbout2: DWORD;
END_VAR
VAR
  fbvar: INT;
  fbvar2: STRING;
END_VAR
```

Program "cfc_prog", declaration and implementation:

```
VAR
  intvar: INT;
  stringvar: STRING;
  fbinst: fublo1;
  erg1: INT;
  erg2: STRING;
  fbinst2: fublo2;
END_VAR
```

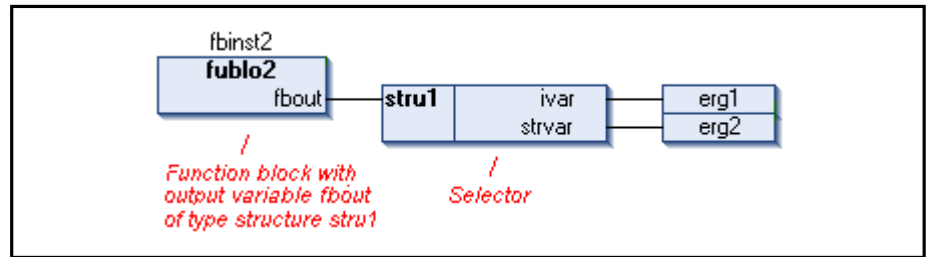


Fig.4-8: Example: Selector

4.1.5 Inserting and Arranging Elements

for programming in the CFC editor can be accessed in the toolbox which opens in a window by default when the CFC editor is active.

Insert To insert an element, select it by clicking on it in the tools window, then hold down the mouse button and drag it to the editor window. While dragging the element, the cursor appears as arrow with an attached rectangle and plus sign. After the mouse button has been released, the element is inserted.

Selecting To select an element to edit or move it for example, note the possible in the element body, on the inputs or outputs or in the text. The element is selected by clicking on the body and it is displayed in red by default.

By pressing the <Ctrl> key, other elements can be selected additionally. Alternatively, hold down the left mouse button and draw a dotted frame around the elements to be selected. The elements are displayed as "selected" when the mouse button is released. Use the "Select all" command, located in the context menu by default, to select all elements simultaneously. The arrow keys can be used to move the highlighted selection to the next possible cursor position. The sequence depends on the processing sequence of the elements indicated by the element numbers; see below.

If an input pin is selected and <CTRL> + <left arrow> is pressed, the respective output pin is selected. If an output pin is selected and <CTRL> + <left arrow> is pressed, the respective output (can also be multiple) is selected.

Creating boxes (function blocks) To replace an existing box, it is sufficient to replace the currently inserted identifier with the desired new name. Consider that the number of input and output pins is adjusted according to the definition of the POU's. This causes that the existing assignments can be deleted.

Moving To move an element, select it by clicking on the body (see possible) and then hold the mouse button down while dragging it to the desired position. After the mouse button has been released, the element is inserted at that position. Alternatively, use the "Cut" and "Paste" commands.

Linking Use the mouse to draw connection lines between outputs and inputs of elements. The shortest possible connection is created, taking existing elements and connection lines into consideration. If a connection line is displayed in light gray, it is covered by another line.

Copy To copy an element, select it and use the "Copy" and "Paste" commands.

Editing text After an element is inserted, the text section is represented by "???". To replace this placeholder with a valid identifier (name of a POU, a label, an instance, etc.), click on the text to select it and open an input field. The button can also open the

Deleting A selected element can be deleted with the "Delete" command located in the context menu by default or with the key.

Editors

Execution sequence, element numbers

The sequence in which the elements of a CFC network are executed in on-line mode is indicated by the element numbers in the upper right corner of each element (function block, output, jump, return, label). The processing begins with the element with the lowest number, "0".

The execution sequence can be modified using located by default in the "Execution sequence" submenu in the CFC menu.

The execution sequence has to be set according to topological arrangement or data flow.

When inserting an element, the number is automatically assigned in topological sequence (from left to right and from top to bottom). If the sequence has already been modified, the new element receives the number of its topological successor and all of the higher numbers incremented by one.

When an element is moved, the number remains the same.



Remember that the sequence affects the result and has to be changed in certain cases.

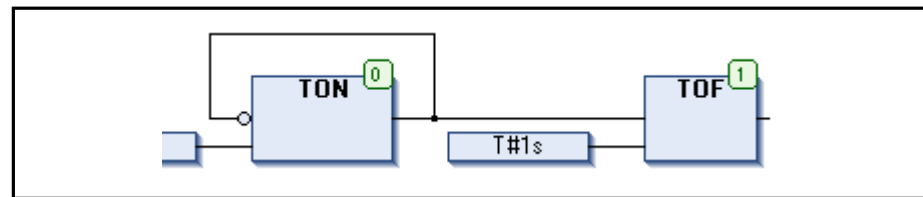


Fig.4-9: Example of element numbering

Changing the size of the working sheet in the editor window

To create more space in the editor window around an existing CFC chart, the size of the workspace ("working sheet") can be changed. It can be changed all elements with the mouse or with the "Cut&Paste" commands (see above).

Alternatively, use a special dialog to specify the dimensions of the working sheet. This can be useful for very large charts.

().

4.1.6 CFC Editor in Online Mode

In online mode, the CFC editor provides views for and for variable values from the control. Debugging functionality (breakpoints, step-by-step execution, etc.) is available. See below.

Note that the editor window for an CFC object also contains a declaration editor in the upper section.

Monitoring

The current variable value is displayed in a small window behind the variable (Inline monitoring).

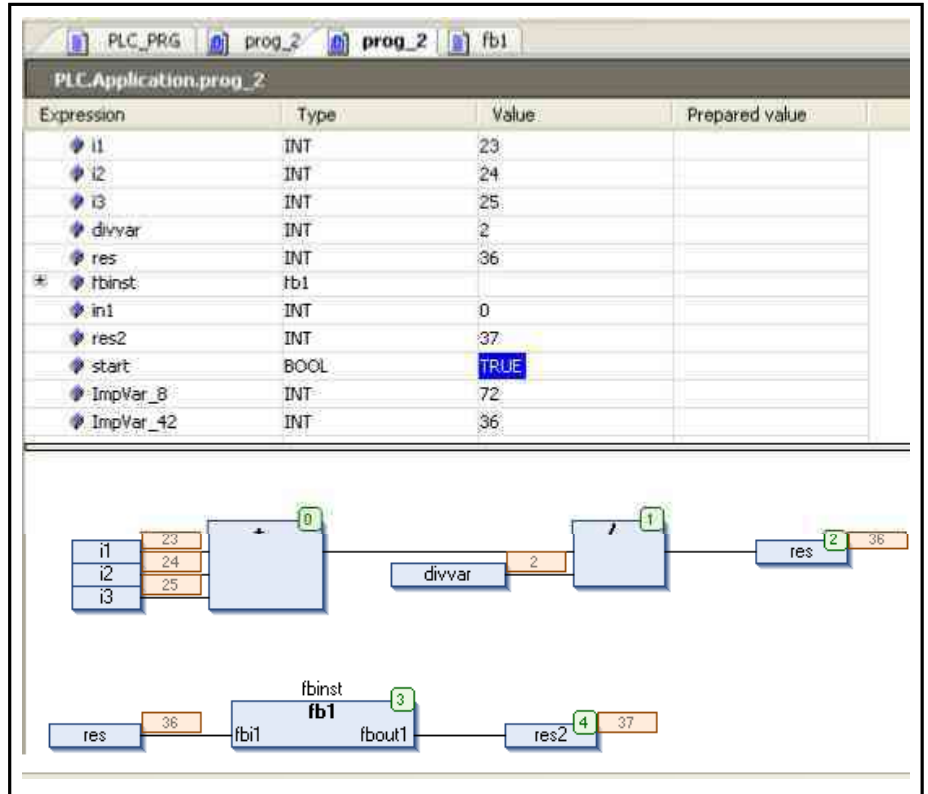


Fig.4-10: Example: Online view of a program PLC_PRG

Note for the online view of the function block that monitoring is only possible in the view of an instance. No values are displayed in the view of the basic implementation. Instead, the "Value" column contains the text "<The value of the expression cannot be read>" and three question marks appear in the respective Inline monitoring fields in the implementation section.

Breakpoint positions in the CFC editor:

at the positions in a function block at which a variable value can change or where the program sequence branches or another function block is called. The red circles in the following figure show possible breakpoint positions:

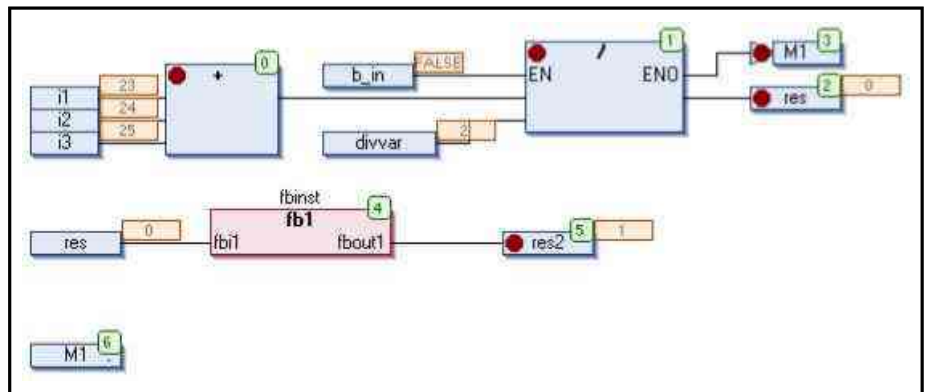


Fig.4-11: Breakpoint positions in the CFC editor



Editors



A breakpoint is automatically set in all that can be called.

Thus, the following applies: If a method defined by an interface is called, breakpoints are set in all methods of function blocks that implement this interface and in all function blocks derived that define the method.

4.2 Data Source Editor

4.2.1 Data Server, Overview

to set up a point-to-point connection between this application (on the local device) and a remote data source.

This way, **data**, i.e. variables provided by other devices (not currently possible) from OPC servers can be in the local application for example.

Several different data sources can be assigned to one data server.



The data server does not replace the network variable functionality used in IndraLogic 1.x.

It only allows another type of data exchange.

Based on defined variable lists, network variables are exchanged in broadcast mode.



Bit accesses used in visualizations that are executed using a data server connection, only function if they contain literal offset specifications (i.e. no defined constants).

A **data server** in the Project Explorer as object below an application. Only one data server may be set up per application. The necessary data server libraries are automatically integrated into the project.

The **data source(s)** that the data server should manage, has to be as objects below the data server.

The data source editor provides dialogs for the individual data sources.

The following will be configured:

- The variables to be considered
- The types of access to be possible
- The settings to communicate with the data source

4.2.2 Configuring a Data Source

can be configured with regard to the selection of variables and the communication with the actual data source.

Open the configuration dialog either with "Open" or by double-clicking on the data source object in the Project Explorer.

The dialog has two tabs: Data Source Items and Communication.

Data Source Items

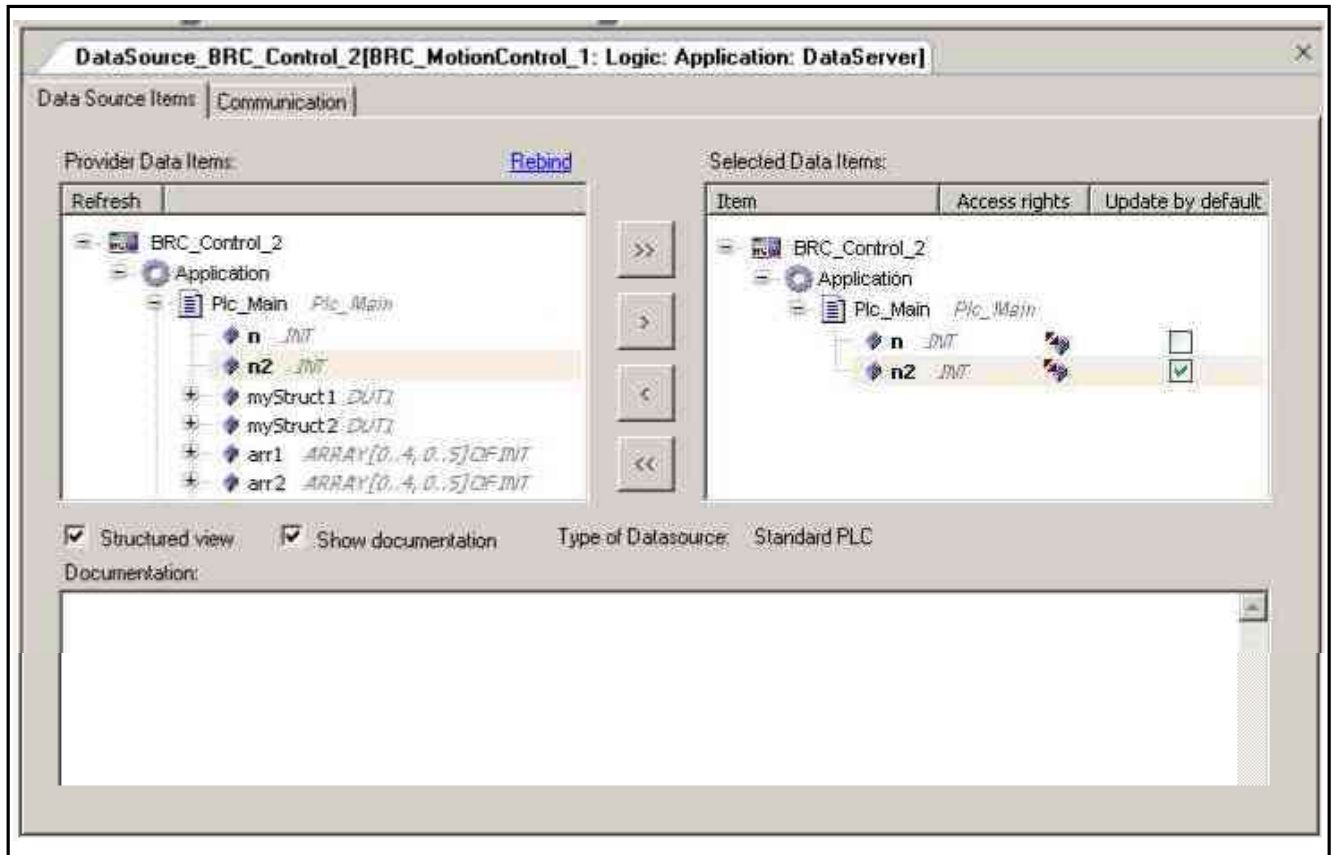


Fig.4-12: "DataSource" dialog, Data Source Items

Here, specify which variables from the remote data source can be used in the local application and which access rights apply in that case.



Bit accesses used in visualizations that are executed using a data server connection, only function if they contain literal offset specifications (i.e. no defined constants).

All data source items are displayed in the **Provider Data Items** window. Use the **Refresh** button above the item list to refresh the display. Code has to be generated for the respective application in advance so that the list of items can be created!

If the **Structured view** option is enabled, the items are displayed in a tree structure. Otherwise, they are displayed non-hierarchically in a "flat" arrangement. The function blocks and variables used by the data source application appear in the "Application" node. If function blocks and data types are instantiated in that case, they and their components are displayed below the "Data Types" node. Note that the data type function blocks are not displayed in a flat arrangement!

If the **Show documentation** option is enabled, a description of the currently selected variable appears in the **Documentation** window if provided by the data source.

To select the variables that the local application is supposed to access:

Select the variables in the "Provider Data Items" window and either double-click or use the **>** button (individual items) or the **>>** button (all items) to

Editors

move them into the "Selected Data Items" window. The < and << buttons can be used to remove the items from the list again.

In the **Structured** view, **limit the selection** to certain function block components if there are **function blocks** and **data types**. Proceed as follows:

In the "Provider Data Items" window, select the desired variables in the "Application" node. If instances of function blocks and data types are included, all components are automatically provided as well. If this is not desired, sharing can be limited to certain components by selecting them explicitly in the "Data Types" node.

Example of component selection

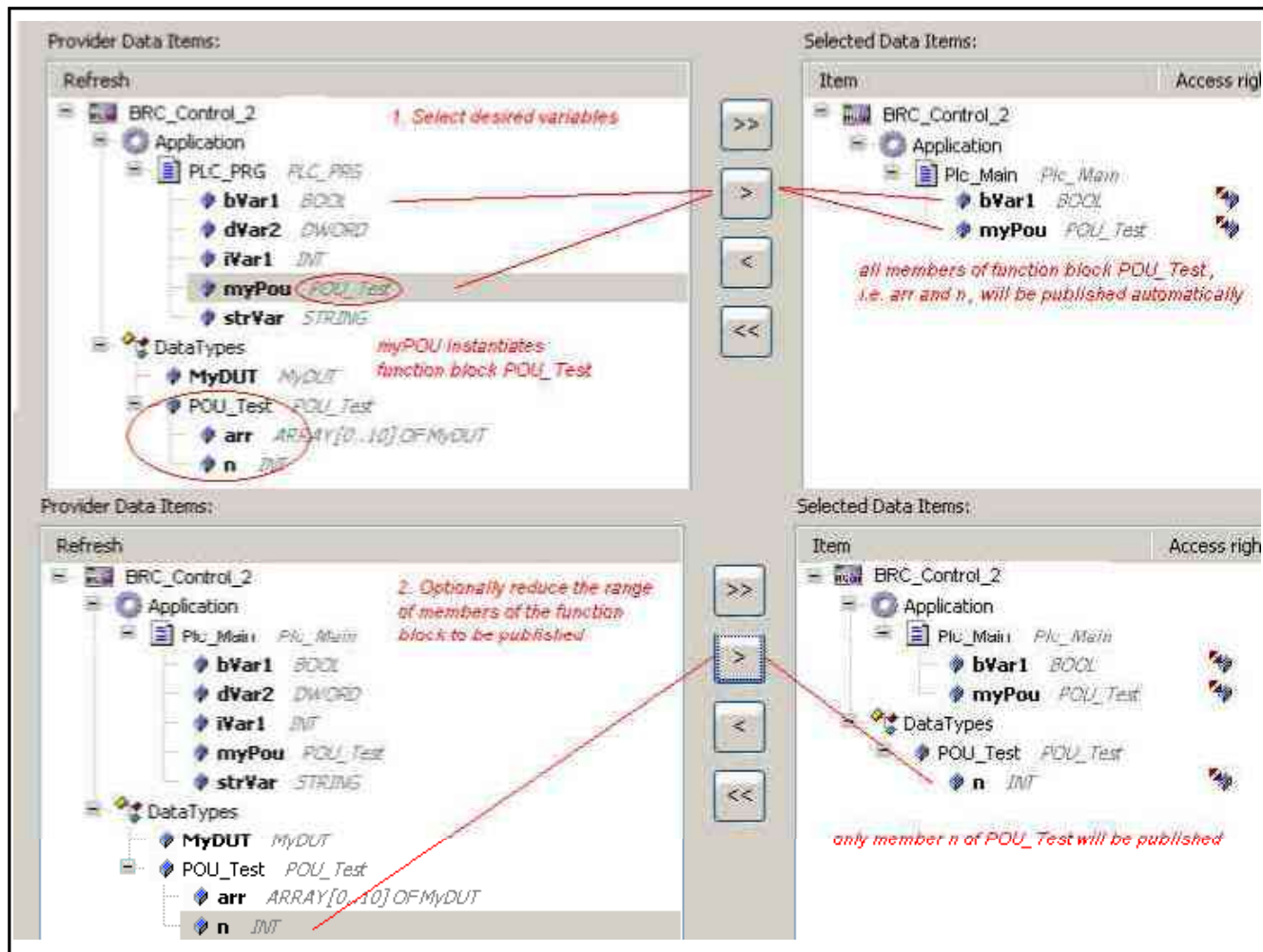


Fig.4-13: Example of component selection

Access rights

By default, "read and write" access rights are assigned for each variable at first; see the **Access rights** column. To change access rights to "Read-only", click on the icon in this column to select it and click again to "switch" it.

"Read and write" icon:

"Read-only" icon:

Read: The variable is updated in the local application with its current value in the data source as soon as its value in the data source changes.

Write: The variable is updated in the data source with its current value in the local application as soon as its value in the application changes.

Editors

Updating by default

By default, the selected variables are always only updated in the currently active visualization. If a variable is also used in another function block in the application, it has to be updated explicitly there using the corresponding function of the data server interface. (In this case, please contact your current software supplier).

The **Update by default** button can switch between selecting and deselecting all variables and the manual selection of individual variables. The selected variables are updated each cycle.



Note that for each variable with the enabled option **Update by default**, data is permanently exchanged between the data source and the PLC on which the data server runs.

This can decrease the refresh rate of the variables if this option is enabled for too many variables.

Communication

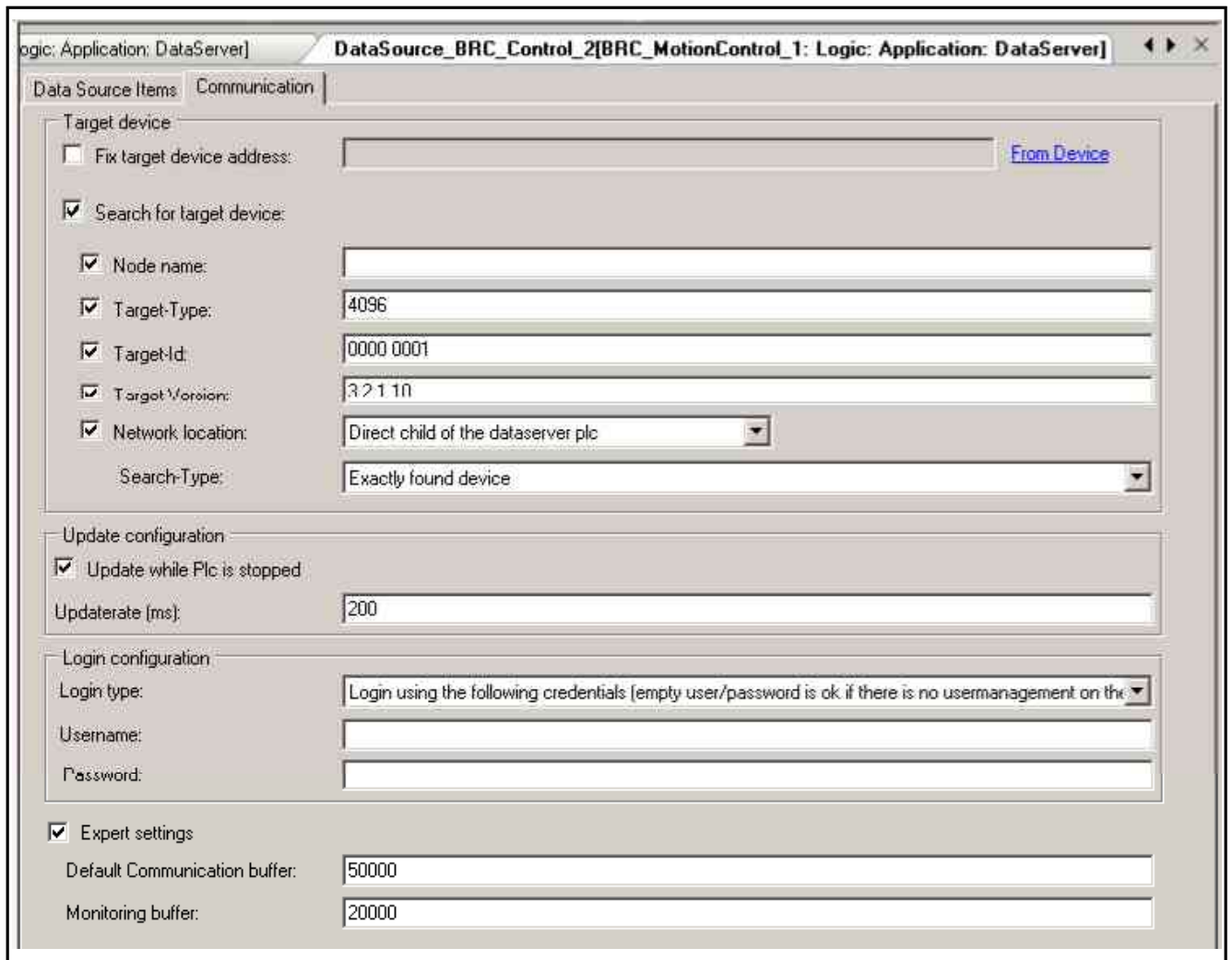


Fig.4-14: "DataSource" dialog, communication parameters

Here, configure the parameters to communicate with the data source on another device. Note the related information stated above.

1. **Enter data source device:** Specify either a fixed address or criteria for the current search for an available control.

Editors

- Click on "**From Device**" to automatically enter the data from the respective, currently connected data source device as configured in the
- **Fix target device address:** If this option is enabled, the communication takes place using the address entered here. Enter the address of the device where the data source is located either manually or using "From Device".



Fig.4-15: Example for entering the address of the data source device

- **Search for target device:** Instead of using a fixed address (see above), do a current search for a control. To do this, define certain search criteria (by placing checks):
 - **Node name:** e.g. WST06
 - **Target system type:** Control type number, e.g. "4096"
 - **Target ID:** e.g. "0000 0001"
 - **Target Version:** e.g. "1.0.0.0"
 - **Network location:** Select the location of the target device from the list:
 - Direct child of the data server PLC: In this case, the runtime system that operates the data server has to provide a switching node that the IEC can access. In addition, the preprocessor constant CMPROUTER_AVAILABLE has to be set in the description file of the device on which the data server is located to signal possible use of the router library.
 - Direct child of the node with address: If this option is selected, enter the address of the father node in the additional field that appears.
 - Direct child of the data server PLC or of the node with address: This selection combines both options. Enter the address of the father node in the additional field that appears and note the related requirements at the router (see the first option).
 - **Search mode:** Select a search mode from the list:
 - First device found:** The first control in the device tree that meets the criteria is used.
 - Exactly found device:** Only a control that exactly meets the criteria entered is used.

2. **Setting the transfer parameters:**

Login configuration: Select one of the following options from the list to manage the login on the target device and the credentials required:

- Login using the following credentials (empty user/password is ok if there is not user management on the PLC):

Credentials required by the target device has to be entered in the **User name** and **Password** fields. If the target device does not have user management, the fields can be left empty.

Editors

- Login with credentials defined during runtime:
Select this option if the user is requested to enter valid credentials during runtime using login dialog in a visualization.
- Do not perform a login on the device:
Select this option when using an old target device without user management so that no login is required.

Update configuration:

Update while PLC is stopped:

If this option is enabled, the data source items are updated at the set update interval, even if the control is stopped. Otherwise, updates are not performed while the control is in "Stop" state.

Update rate (ms): Update interval for the variables in milliseconds.
Default: 200 ms.

Expert settings: If this option is enabled, the dialog is extended by the following settings:

- **Size of the communication buffer:** Buffer size for communication data in bytes.
Default: 50000.
- **Size of the monitoring buffer:** Buffer size for the monitoring data in bytes.
Default: 20000.

4.2.3 Adding a Data Server

A **DataServer** object can be added to an application with the `addDataServer` command.

Alternatively, a data server can be dragged into the Project Explorer from the library (PLC objects). Only one data server can be set up per application and the object cannot be renamed.



Fig.4-16: "Add Object" dialog, adding a data server

The necessary data server libraries in the global project library manager are automatically added with the data server object and a **data server task** is created in the task configuration of the application.

Editors



The task priority has to be specified.

The command **addDataSources** can be used to add one or multiple **data source** objects to the application.

4.2.4 Adding a Data Source

A **Data Source** can be assigned to the **Application** of an application using the **addDataSources** command.

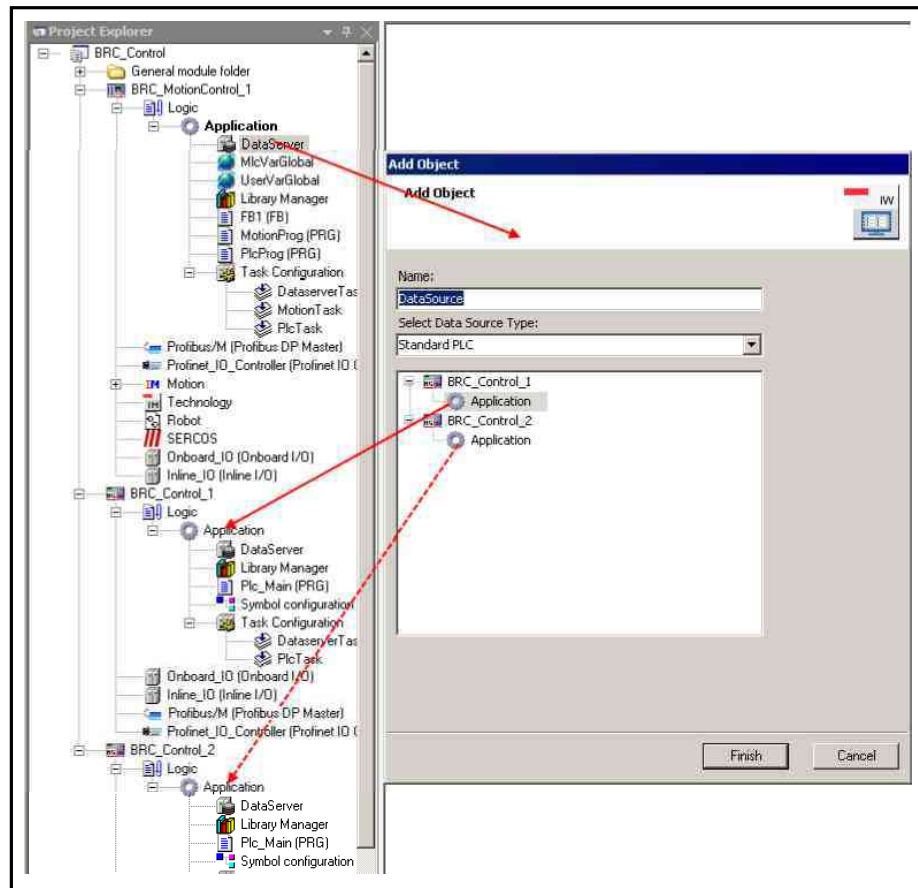


Fig.4-17: "Add Object" dialog, adding a data source

The "Add Object" menu provides a selection list (**Select Data Source Type:**) to set the desired data source type.

Currently available "Default PLC":

In preparation ####: OPC server.

The data sources that are currently available for the type set are then displayed in the window below the list. Select a data source and specify a symbolic **name**.

After confirming with **Finish**, the data source object with the symbolic name is added to the device tree below the data server for the application.

Example:

Data server and data sources

The Project Explorer currently contains three device entries: BRC_MotionControl_1, BRC_Control_1 and BRC_Control_2

Editors

The application below BRC_MotionControl_1 would like to use data from the BRC_Control_1 and BRC_Control_2 devices.

First, add **DataServer** to all of the desired applications for all three controls ().

Then go to the data server of the **BRC_MotionControl_1** control.

Select the data sources that correspond to this data server ().

Now both sources should have been added to the Project Explorer as shown in the figure below and can be

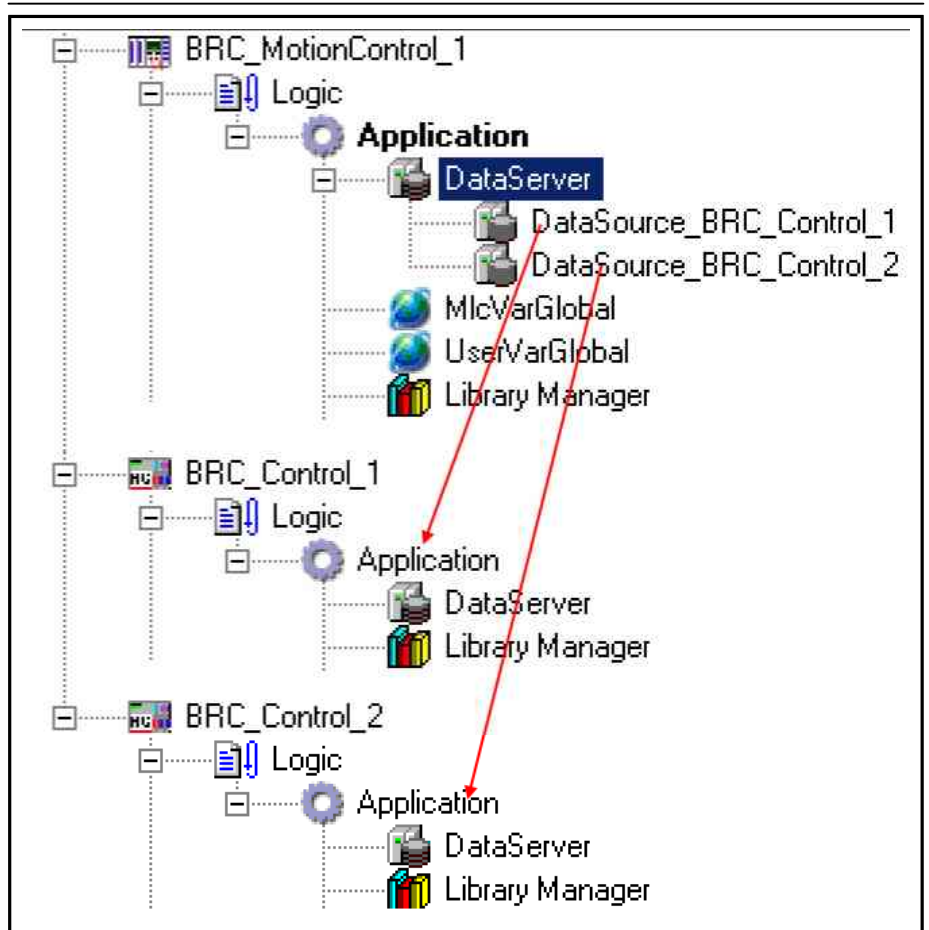


Fig.4-18: Example of data server and data sources

4.2.5 Using Data Sources in Visualizations

All visualizations assigned to an application can use all that are also assigned to the application.

for data exchange can be used when configuring the properties of a visualization element.

They can be entered manually or using the input assistant or "Intellisense" function.

For manual entry, the entire variable path has to be entered based on the following syntax:

<Device name for the data source>.<Function block name>.<Variable name>.

Editors



Bit accesses used in visualizations that are executed using a data server connection, only run if they contain literal offsets (i.e. no defined constants).

Example:

Using a data source variable in the properties of a visualization element

See the "Properties" dialog for a visualization element below. The variable "iCount" from a defined data source is used as text variable. This data source is the device "Device2" and the application "App1" located there along with function block "prg1".



Fig.4-19: Example for using a data source variable in the properties of a visualization element



By default, the selected variables are always only updated in the currently active visualization.

If a variable is also used in another function block in the application, it has to be updated explicitly there using the corresponding function of the data server interface. (In this case, please contact your current software supplier).

4.3 Declaration Editor

4.3.1 Declaration Editor, General Information

The declaration editor is a **text editor** to declare variables.

Behavior and appearance are thus determined by the current text editor **settings** in **IndraWorks** > **Tools** > **Options** > **IndraLogic 2G** > **Text Editor** dialog and in **IndraWorks** > **Tools** > **Customize** > **Commands** > **Edit** dialog. The default settings for colors, line numbers, tab widths, indents, etc. can be made there.

..., either textual or tabular view appears or it can be switched between the views via the buttons "Textual / Tabular" at the right margin of the editor window.

Editors

Usually, the declaration editor is used in connection with a programming language editor, i.e. it appears in the upper section of the window that opens if an object is edited or displayed in offline or

The declaration header describes the POU type (e.g. PROGRAM, FUNCTION_BLOCK, FUNCTION ...) and can be extended by POU global pragma attributes.

In online mode, the declaration editor is structured as a

Variable declarations can also be made in a global variable list and in data type objects (DUTs), but which work with their own editors.

Also refer to the general information on the

Text declaration editor

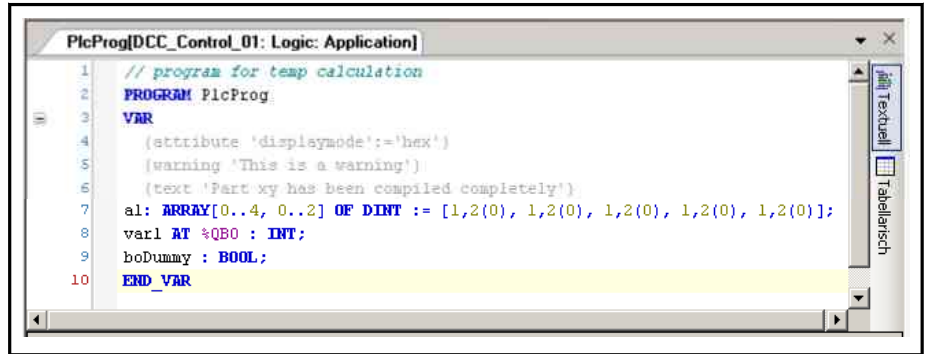


Fig.4-20: Text editor view

Behavior and appearance of the textual editor are determined by the current and "Customize" dialog. The default settings for colors, line numbers, tab widths, indents, etc. can be made there.

The usual Windows functions are available and even those of the function are supported if necessary.

Note that **function block selection** is possible by pressing <Alt> while selecting the desired section of text with the mouse.

Table editor

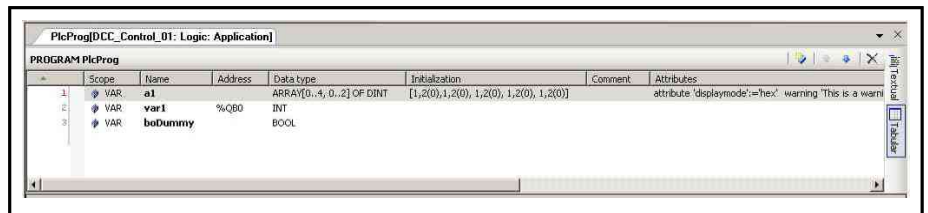


Fig.4-21: Tabular editor view

The tabular version of the editor displays columns for common definitions re: "Validity range" (visibility), "Name", "Address", "Data type", "Initialization", "Comment" and "Attributes". The individual declarations are inserted as numbered lines.

The declaration header can be edited in the "Edit Declaration Header" that opens with the command of the same name from the context menu.

Editors

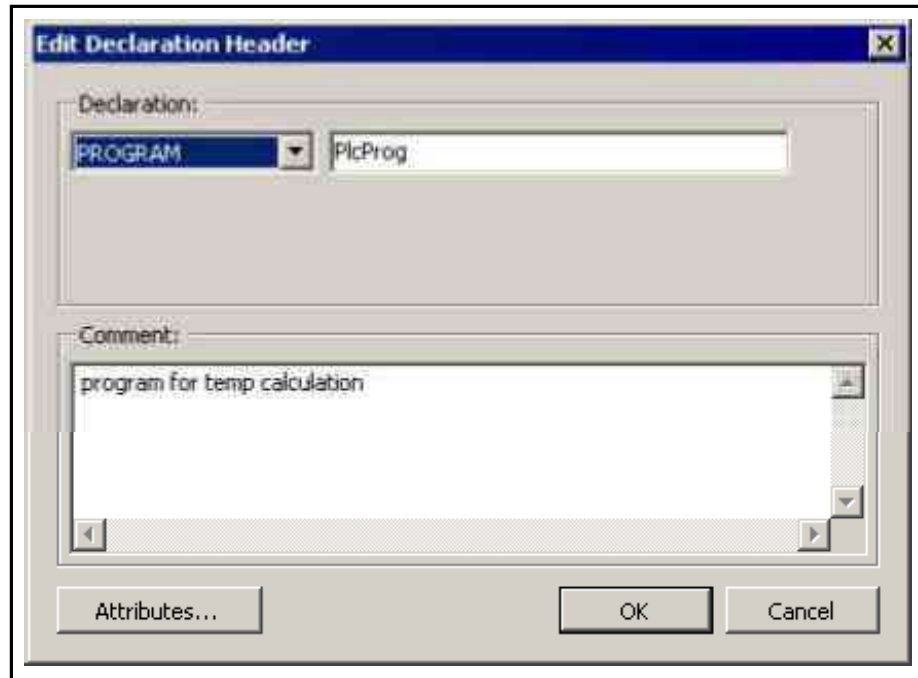


Fig.4-22: "Edit Declaration Header" dialog

Type (from the selection list) and name of the POU object can be entered below "Declaration". A comment can be entered (line breaks with <CTRL> + <Enter>). With the "Attributes" button, the "Attributes" dialog can be opened to enter pragmas and attributes.

To insert a **new declaration line** above an existing one, select the existing line and click on "Insert" from the toolbar or the context menu.

To add a new declaration at the very bottom of the table, click on the last existing line and use the "Insert" command as well.

The newly inserted declaration is provided with the validity range "VAR" and the latest data type entered. The input field opens automatically for the obligatory variable "Name". A valid identifier has to be entered there and then closed either with <Enter> or by clicking to a different section of the view using the mouse.

Each table cell opens the respective input option with a double-click on the cell. To enter the validity range, a drop-down list opens from which the desired range and fitting attribute (flag) can be selected.

Enter the "data type" directly or click on ">" button to reach the
or the

Enter the "initialization value" directly or click on the array and use the dialog of the same name especially useful for structured variables via .

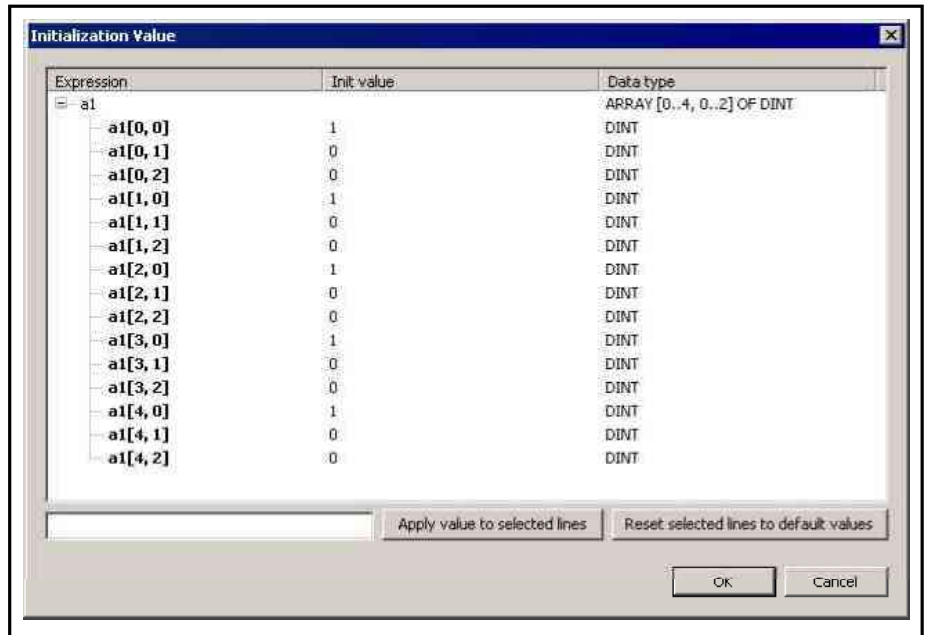


Fig.4-23: "Initialization Value" dialog

All expressions of the variables are represented with their current initial values. Select the desired ones and edit the value in the field below the list. Then apply it via the button "Apply value to selected lines". The default initialization can be restored via the button "Reset selected lines to default values".

Line breaks in the input fields of the comment column can be inserted with <Ctrl> + <Enter>.

The **Attributes** entries are made in the "Attributes" dialog in which multiple attributes and pragma statements can be entered as text. They have to be entered without {} and each in an individual line. The example shown in the following figure corresponds to the textual view that can be seen in the figure "Textual editor view".

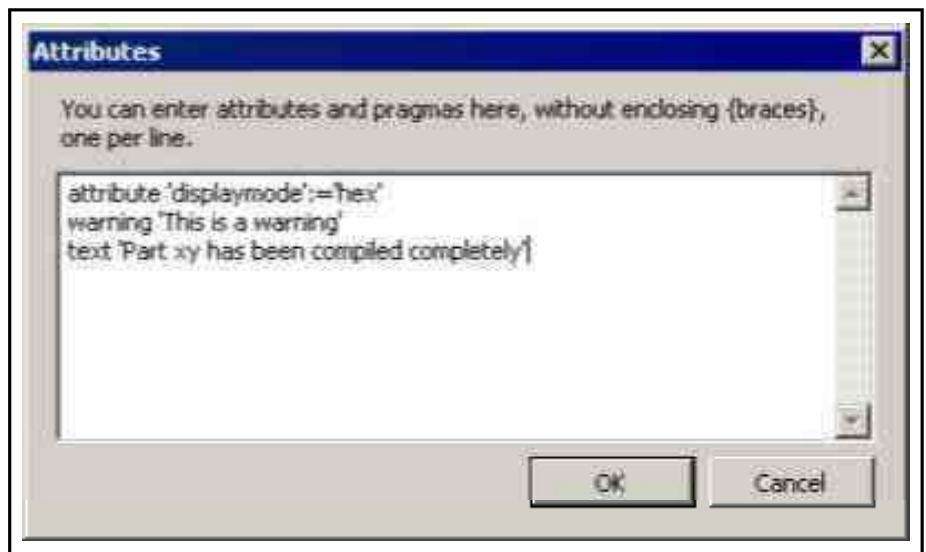






Fig.4-24: "Attributes" dialog


Each variable is declared in an individual line. The lines are numbered.



Editors

The order of the lines (line numbers) can be changed using the commands  "Move up" or  "Move down" (context menu or toolbar). The currently selected line is moved down by one position.

The list of declarations can be sorted by a column by clicking on the respective column header. The column specified for the sorting is labeled with an arrow:  (sorted in ascending order) or  (sorted in descending order). Each click in the column header switches between ascending and descending.

To delete one or multiple declarations, select the respective lines and click on `` or use the "Delete" command from the context menu or toolbar .

4.3.2 Declaration Editor in Online Mode

After login into the target system, each project object that was already displayed in a window in offline mode is automatically displayed in the online view.

The online view of the declaration editor displays a table like those used in

The header always contains the current object path:

`<Device name>.<Application name>.<Object name>`.

The table displays the data type and the current value for each expression (watch expression), and - if it is currently set - the prepared value for




or

dialog.

Alternatively, enter the value directly (after a click) in the respective field in the "Prepared value" column.

Handling a Boolean variable is even easier: Use the Return key or Space bar to toggle the prepared value according to the following sequence: If the prepared value was TRUE, enter FALSE -> TRUE -> Space. Otherwise, if the prepared value was FALSE, enter TRUE -> FALSE -> Space.

If a declaration function block follows a declaration (here an instance of a function block), a plus or minus sign has to be precede. Click on this sign to see the declaration function block.

Symbols indicate if the respective variable is an input , output  or "normal"  variable.

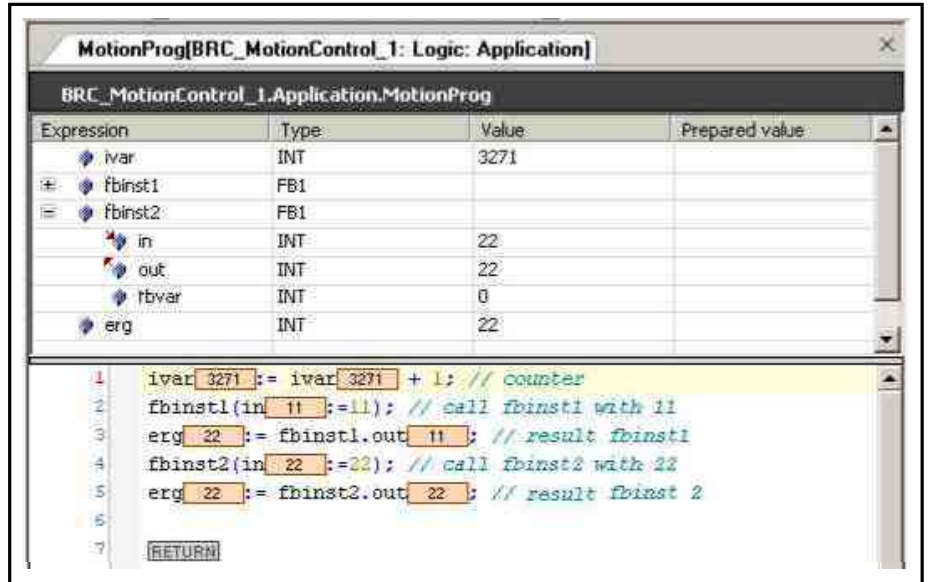


Fig.4-25: Declaration editor in the upper section of a program object, online view

4.4 Device Editor

4.4.1 Device Editor, General Information

The device editor provides dialogs to configure a managed in the Project Explorer.

By default, the **device dialog** can be opened by double-clicking on the control when it is selected in the Project Explorer.

The title of the main dialog is the device name (e.g. DCC_MotionControl1) and, depending on the device type, it contains a combination of the follow subdialogs:

- : List of the applications on the control.
- : Displays PLC log file.
The tab can be switched off and on via **Options ► IndraLogic 2G ► General Settings ► Enable PLC Logger**.
-
-
- : General information on the device (name, vendor, version, etc.)

4.4.2 Applications

This tab is used to display and, if required, to delete applications that are currently located on the control.

Editors

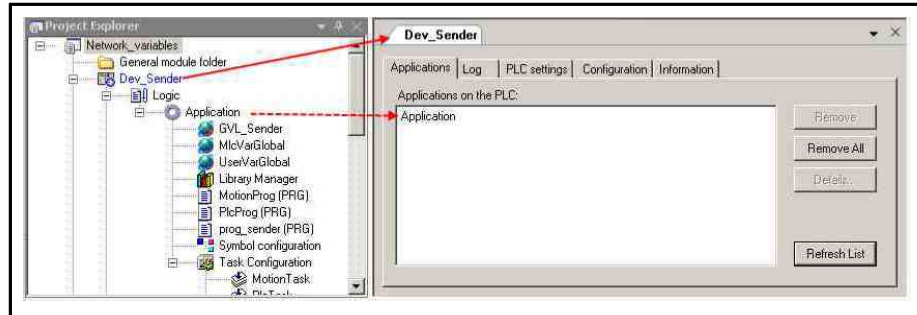


Fig.4-26: Device dialog, Applications

Applications on the PLC: List of the applications that were found on the control during the most recent "scan" (with "Refresh"). If a scan was not yet carried out or is not possible because a gateway for a connection was not configured, a message is output.

Refresh List: The control is searched for applications and the list is refreshed accordingly.

Remove or Remove All: The currently selected applications or all applications are deleted from the control.

If an application is loaded to the control, the following is checked first:

- The list of applications on the control is compared with those available in the project. If the lists do not match, the corresponding dialogs appear, either for loading the applications not already present on the control or for deleting other applications on the control.
- The "externally implemented" function blocks in the application to be loaded are checked to see if these are also available on the control. If not, a corresponding message ("Unresolved reference(s)") is output in a message box and in the message window.
- The parameters (variables) of the function blocks in the application to be loaded are compared with those in the function blocks of the same name in the application already present on the control (signature check). If they do not match, a corresponding message ("Invalid signature(s)") is output in a message box and in the message window.

4.4.3 Log

This tab is used to display the control logbook, i.e. to display events recorded on the target system.

The tab can be switched off and on via **Options ► IndraLogic 2G ► General Settings ► Enable PLC Logger**.

Displayed are:

- Events at system start and shuts down (loaded components with version)
- Application download and loading the boot project
- Customer-specific entries
- Log entries from I/O drivers
- Log entries from the data server

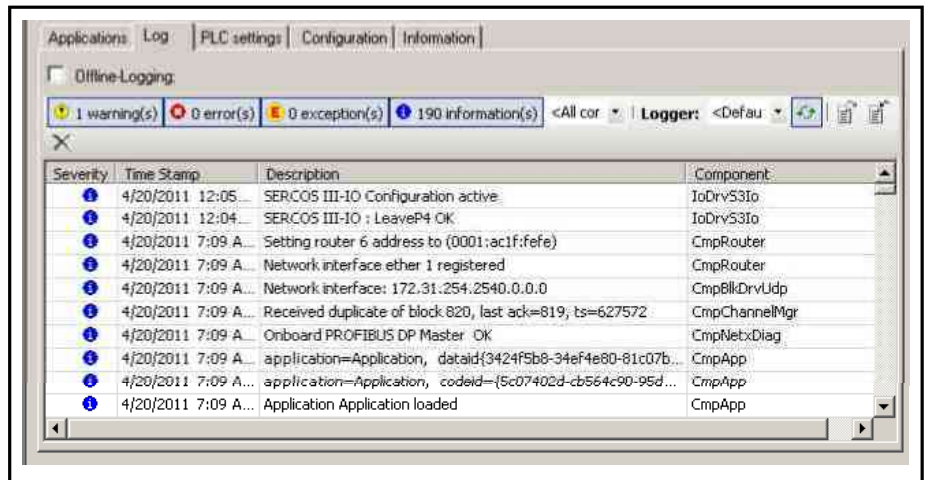


Fig.4-27: Device editor, Log

A logbook entry is displayed with the following information:

Severity (Scaling): There are four categories for the severity of the event: Warning(s), error(s), exception(s), information. The display for each category can be displayed or hidden by using the corresponding button in the bar above the list. The respective number of log entries are displayed for the respective category on the button.

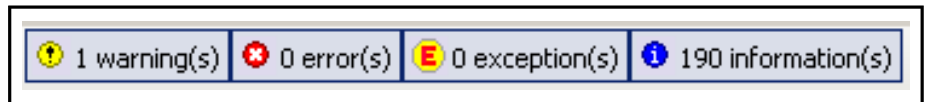


Fig.4-28: Severity of event

Time Stamp: Date and time, e.g. "12.01.07 09:48:00"

Description: Description of the event, e.g. Import function failed of <CmpFile-Transfer>

Component: Name of the respective component.

Com Name (component name): Here, it can be searched for an individual component in a selection list to display only logbook entries that are related to it. The default setting is "All components".

Logger: The selection list contains the available recordings. The default setting is "<Default Logger>" specified by the target system, currently the same as "PlcLog" for the IndraLogic runtime system.

Not yet available: The log list is automatically updated.

Currently, the list has to be refreshed using the button.

The content of the list can be exported into an XML file. To do this, press the

button to open the standard dialog for saving files. The file filter is set to "xml files (*.xml)". The log list is stored in the selected directory with the file name entered and the extension ".xls". If the "Offline logging" option is enabled, even actions that are not related to the connection with the control are recorded. However, at this time it can only be implemented in the safety version of the programming system.



Please see the information on the diagnostic possibilities in your **System Description!**

Editors

4.4.4 PLC Settings

This tab is used to specify how the control behaves in the "Stop" state.

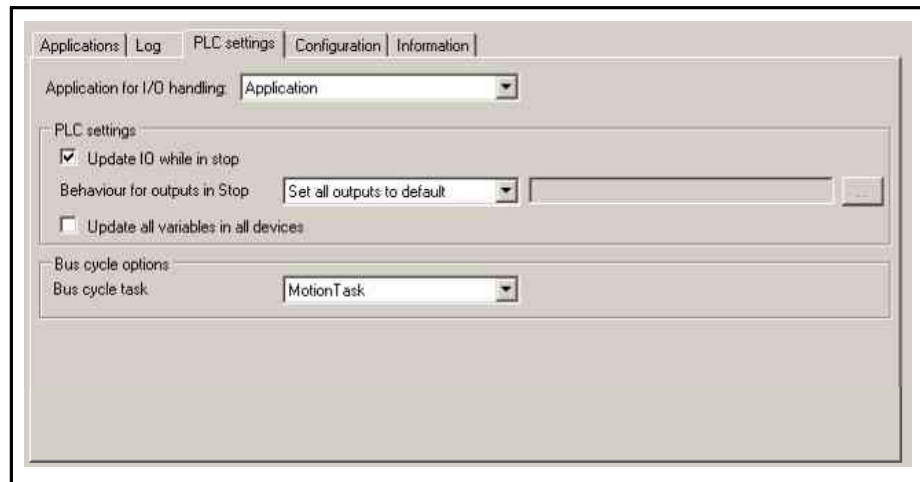


Fig. 4-29: Device editor, PLC settings

Application for I/O handling:

If several applications are available for the device, they are listed here. The default application, which is automatically created with a default project, is always entered first.

PLC settings:

Update IO while in stop: If this option is enabled (default), the values for input and output channels are updated if the PLC goes into the "Stop" state.

Output behavior in "Stop" state:

The selection list provides the following options for handling the values of the output channel when the control goes into "Stop" state:

- **Retain current values:** The current values are retained.
- **Set all outputs to default:** The default values from the I/O mapping are assigned.
- **Execute program:** The handling of the output values can be controlled using a program in the project. This program name can be entered and it is then executed when the control goes into "Stop" state. The "..." button can be used to open the input assistance to facilitate the program selection.

Bus cycle options:

Bus cycle task: The selection list provides all tasks defined in the task configuration of the active application (e.g. "Motion task", "PLC task", etc.).



Check the data of the tasks of your application and then enter the desired task!

Select a specific task to control the bus cycle or select the "Unspecified" setting if the task with the shortest cycle time, i.e. the fastest task, is to be used here.

4.4.5 Floating Point Exceptions in the PLC program

Editors



This feature applies only for IndraLogic XLC/IndraMotion MLC for 12VRS or higher.

It is preset, that the floating point exceptions are blocked in the PLC program. This ensures compatibility for existing user projects.

For the IndraMotion MTX and IndraMotion MLD, handling of floating point exceptions in the PLC cannot be configured. The dialog in the device editor is omitted

Exceptions can occur during floating point calculations.

Example: Division by Zero

For this purpose, the PLC provides a default handing. That means that the PLC goes into STOP state and the floating point exception is displayed.

The user can select whether

- the PLC goes into "Stop" state (default reaction) or whether
- the PLC remains in the "Run" state and continues the calculation for the PLC project when a floating point exception occurs.

Background information

If the handling of floating point exceptions is blocked and if an exception occurs during a floating point calculation, the floating point processor (FPU) automatically uses a value allowing the calculation to continue in most of the cases.

Example:

At floating point calculations, a division by 0 results in an infinite value. This does not always work out and depends on the calculation method in the user program. Continuing the calculation can thus also cause an invalid reaction of the user program.



For new PLC projects, The floating point exception handling should be at least switched on in the engineering phase.

Types of floating point exceptions

In the IEEE standard for floating point calculations, different types of floating point exceptions are defined. These are supported by floating point processors according to IEEE standard.

- Overflow
- Underflow
- Divide by zero
- Invalid
- Inexact (precision)
- Denormalized.

It is not reasonable to execute an exception handling via software for all floating point exceptions. If "Inexact" exists permanently and the FPU hardware rounds according to a preset algorithm, no software handling can be selected for the "Inexact" floating point exceptions.

Dialog description

The device dialog opens in the right window when double-clicking on the control in the PLC project. There is a new "Configuration" tab.

Editors

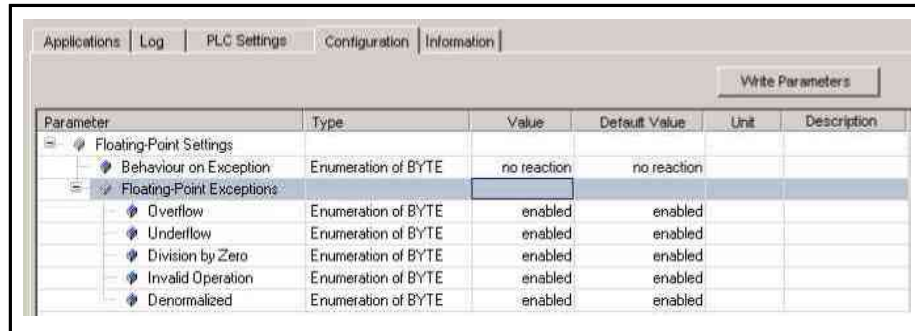


Fig.4-30: Device dialog, here IndraMotion MLC L65, "Configuration" tab

There are no floating point settings in the "Configuration" tab. The floating point settings are divided into two components:

- Behavior on Exception
- Floating Point Exceptions

In the component "Behavior on Exception", there are the following selection options:

- no reaction
- PLC in Stop, no reaction

No reaction: All exception conditions occurring at floating point calculations are ignored.

The PLC program is still executed (PLC remains in Run).

The floating point exceptions set in the dialog are not relevant in this case.

PLC in STOP: If enabled exception conditions occur, the PLC is switched to Stop.

Floating point exceptions set to the "enabled" state in the dialog are considered.

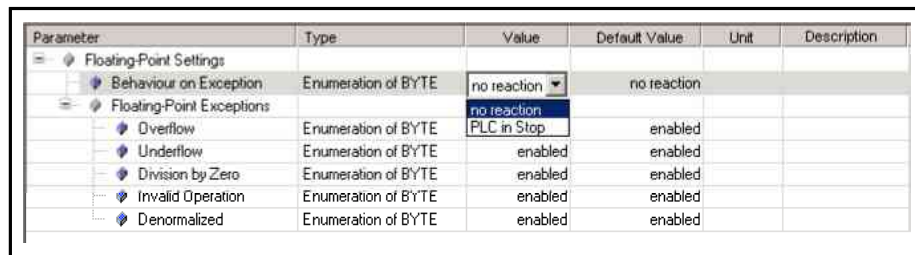


Fig.4-31: Floating point settings, behavior on exception

The floating point exceptions displayed in the dialog are only relevant if "PLC in Stop" is set as behavior.

In the presetting, all floating point exceptions are enabled. However, the use can disable any floating point exceptions:

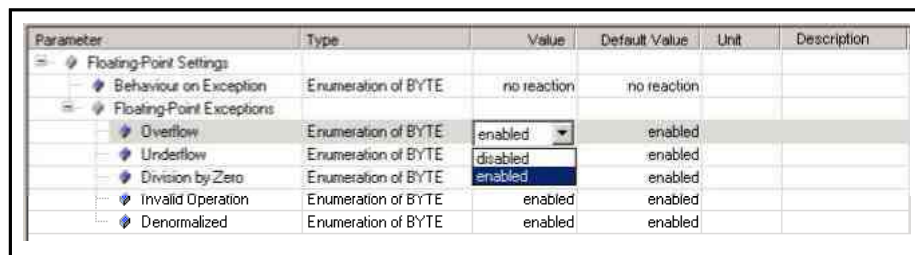


Fig.4-32: Floating point settings, floating point exceptions, overflow

Editors



Fig.4-36: Device editor, Information

4.5 Data Type Editor

(DUT, data unit types) can be created in the data type editor. It is text editor and thus the appearance and behavior of the current settings are specified in the

The data type editor window opens if an existing DUT object is opened for editing.

The editor already contains the predefinition of an which can be modified into a simple structure definition or into a definition of another data type, e.g. an enumeration.

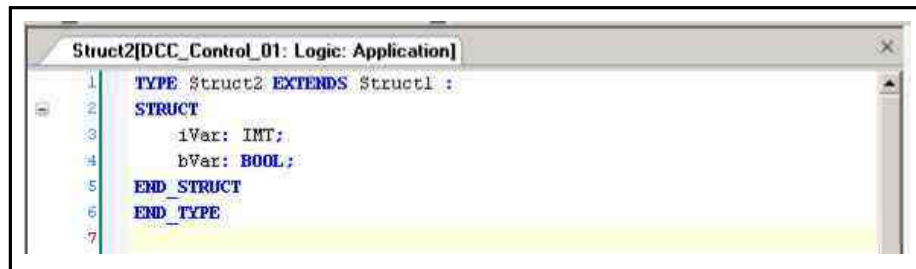


Fig.4-37: DUT editor window with structure

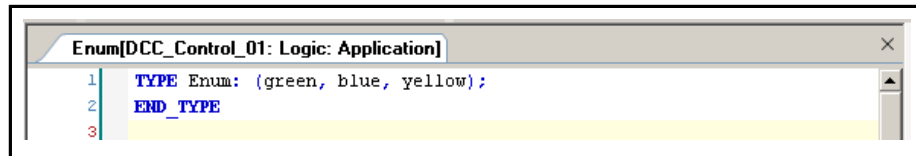


Fig.4-38: DUT editor window with enumeration type

For an enumeration type, remove the lines "STRUCT" and "END_STRUCT".

A setting for an automatic opening of the editors can be made by inserting a data type using the main menu **Tools** ▶ **Options** ▶ **IndraLogic 2G** ▶ **General Settings**.



Editors

A description of the initialization of arrays is located in _____ or in _____ the _____.

A description of the initialization of structures is located under _____.

A description of the initialization of enumerations is located under _____ subrange types.

A description of variable initialization using the desired expressions is located under _____.

4.6 FBD/LD/IL Editor

4.6.1 FBD/LD/IL Editor, Overview

There is a **combined editor** for the programming in the following languages: FBD (_____), LD (_____) and IL (_____).

That means that a **common set of commands and elements** is used and the three programming languages can be internally converted from one to another. The programmer can **switch** into one of the the other editor views (_____) at any time, even in online mode.

However, note that a few special elements **cannot be converted** and can only be displayed in the corresponding language. There are also constructs that cannot be converted between IL and FBD without ambiguity. For this reason, they are "normalized" when reconverted to FBD, i.e. they are reset. This applies to the: negation and explicit/implicit assignments for function block inputs and outputs.

Behavior, appearance and menus are defined in the _____ . See the special setting options there for displaying comments and addresses in the editor.

If required, the menu structure can be reconfigured in the dialog under **IndraWorks ► Tools ► Customize ► Commands**.

The editor opens in a **split window** if an object is to be edited in FBD/LD/IL.

The upper section contains the _____ .

The **programming language** for a new object is specified when the object is created using the _____ .

- _____
- _____

4.6.2 Programming Languages in the FBD/LD/IL Editor

Function Block Diagram - FBD

The function block diagram is a graphically oriented programming language. It works with a list of networks, where each network contains a structure that displays a logical or arithmetic expression, a call of a function block, a jump or a return instruction.

Editors

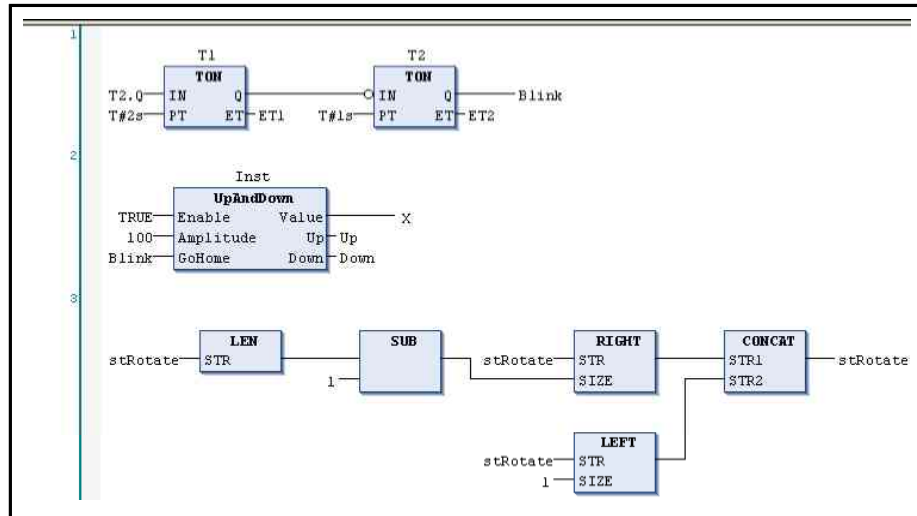


Fig.4-39: Example of a few FBD networks

Ladder diagram - LD

The ladder diagram is a graphically oriented programming language, similar in principle to an electrical circuit.

On one hand, the ladder diagram is suitable for constructing logical switches, but on the other hand, networks as in FBD can also be created. For this reason, LD is very suitable to control calls of other function blocks.

The ladder diagram consists of a series of networks. A network is limited on the left and right sides by a left and a right vertical **electrical power supply**. Between these is a circuit diagram made up of contacts, coils, optional function blocks (POUs) and connection lines.

On the left side, each network consists of a series of contacts transmitting the states "ON" or "OFF" from left to right. These states correspond with the Boolean values TRUE and FALSE. Each contact has a Boolean variable. If the variable is TRUE, the status is transmitted across a connection line from left to right. Otherwise, the right connection receives the value OFF.

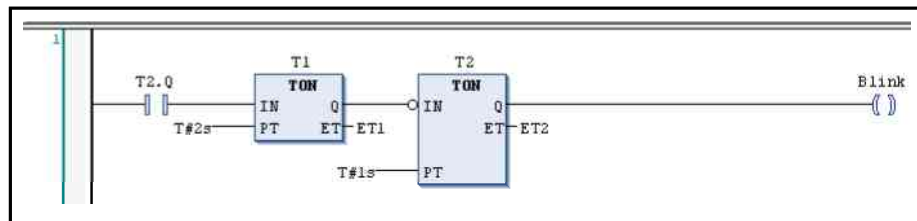


Fig.4-40: Example of an LD network

Instruction list - IL

The instruction list is a IEC61131-compliant programming language that is similar to an assembler language.

It supports accumulator-based programming. All IEC 61131-3 operators are supported as well as multiple inputs, multiple outputs, negations, comments, setting/resetting outputs and conditional/unconditional jumps.

Each instruction is primarily based on loading the values to the accumulator (LD instruction). Afterwards, the corresponding operation is executed using the first parameter from the accumulator. The result of the operation is written to the accumulator again from where the user should save it to using an ST instruction.

Editors

For programming conditional executions or loops, the instruction list supports relational operators (EQ, GT, LT, GE, LE, NE) and jumps. Jumps can be unconditional (JMP) or conditional (JMPC / JMPCN). For conditional jumps, a check is made to see if the value in the accumulator is TRUE or FALSE.

Syntax: An instruction list (IL) consists of a series of **instructions**. Each instruction begins on a new line and includes an **operator** and, depending on the type of operation, one or multiple **operands** separated by commas.

An identifier (**label**) can be located in front of an instruction followed by a colon (:). It identifies the instruction and can be used as a **jump target** for example.

The last element in a line has to be a **comment**. Empty lines can be inserted between instructions.

Empty lines can be located between instructions.

```

1 PROGRAM IL
2 VAR
3   inst1: TON;
4   dwVar: DWORD;
5   dwRes: DWORD;
6   t1: TIME;
7   tout1: TIME;
8   inst2: TON;
9   bVar: BOOL;
10 END_VAR

```

LD	bVar		variable
ST	inst1.IN		starts timer with risin.
JMPC	m1		
CAL	inst1(
	PT:=t1,		
	ET:=>tout1)		
LD	inst1.Q		is TRUE, PT seconds aft.
ST	inst2.IN		starts timer with risin.


```

m1:
LD   dwVar
RDD  230
ST   dwRes

```

Fig.4-41: Program example in the IL table editor

The IL editor is a table editor that is integrated into the

-
-

4.6.3 Modifiers and Operators in IL

In the language, the following operators and modifiers can be used.



Editors

Modifiers:

C	With JMP, CAL, RET	The instruction is only executed if the result of the preceding expression is TRUE.
N	With JMPC, CALC, RETC	The instruction is only executed if the result of the preceding expression is FALSE.
N	else	Negation of the operand (not the accumulator)

Fig.4-42: Modifiers

The following table includes all IL operators with their possible modifiers and meanings:

The "**accumulator**" always contains the preceding value of the previous operation.

Operator	Modifier	Description	Example
LD	N	Loads the (negated) value of the operand to the accumulator	LD iVar
ST	N	Saves the (negated) contents of the accumulator to the operands	ST iErg
S		Sets the operands (type BOOL) to TRUE if the accumulator content is TRUE	S bVar1
R		Sets the operands (type BOOL) to FALSE if the accumulator content is FALSE	R bVar1
AND	N,(Bit-by-bit AND of the accumulator value and the (negated) operand	AND bVar2
OR	N,(Bit-by-bit OR of the accumulator value and the (negated) operand	OR xVar
XOR	N,(Bit-by-bit exclusive OR of the accumulator value and the (negated) operand	XOR N, (bVar1,bVar2)
NOT		Bit-by-bit negation of the accumulator value	
ADD	(Addition of the accumulator value and the operands. Result in the accumulator	ADD (iVar1,iVar2)
SUB	(Subtraction of the operand of the accumulator value. Result in the accumulator	SUB iVar2
MUL	(Multiplication of the accumulator value and the operand. Result in the accumulator	MUL iVar2
DIV	(Division of the accumulator value by operands. Result in the accumulator	DIV 44
GT	(Check whether the accumulator value is greater than the operand value. Result (BOOL) in the accumulator; >	GT 23
GE	(Check whether the accumulator value is greater than or equal to the operand value. Result (BOOL) in the accumulator; >=	GE iVar2
EQ	(Check whether the accumulator value is equal to the operand value. Result (BOOL) in the accumulator; =	EQ iVar2
NE	(Check whether the accumulator value is not equal to the operand value. Result (BOOL) in the accumulator; <>	NE iVar1
LE	(Check whether the accumulator value is less than or equal to the operand value. Result (BOOL) in the accumulator; <=	LE 5
LT	(Check whether the accumulator value is smaller than the operand value. Result (BOOL) in the accumulator; <	LT cVar1
JMP	CN	Unconditional (conditional) jump to the specified label	JMPN next
CAL	CN	(Conditional) call of a program or function block (if the accumulator value is TRUE)	CALC prog1
RET		Exiting the function block and returning to the calling function block	RET

Operator	Modifier	Description	Example
RET	C	Conditional - if the accumulator value is TRUE - exiting the function block and returning to the calling function block	RETC
RET	CN	Conditional - if the accumulator value is FALSE - exiting of the function block and returning to the calling function block	RETCN
)		Evaluation of the postponed operation	

Fig.4-43: Operators and modifiers

- Information on
- Information on using **multiple operands, complex operands**, function, method, function block, program, action **calls** and **jumps**; see

The following shows an example program with some modifiers:

```

1  AND      TRUE      load TRUE to accumulator
   ANDN    bVar1     execute AND with negated value of bVar1
   JMP     m1        if accum. is TRUE, jump to label "m1"

   LDN     bVar2     store negated value of bVar2...
   ST     bRes      ... in bRes

2  m1:
   LD     bVar2     store value of bVar2...
   ST     bRes      ... in bRes

```

Fig.4-44: Example of an IL program

4.6.4 Working in the FBD and LD Editors

Networks are the basic units of FBD and LD programming. Each network contains a structure that displays a logical or arithmetic expression, a call of a programming function block (function, function block, program, etc.), a jump or a return instruction.

When a new object is created, the editor window already contains an empty network.



Behavior, appearance and menus are defined in the . See the special setting options there for displaying comments and addresses in the editor.

Tooltip with information on variables or function block parameters

If the cursor is pointing to a variable or a function block parameter, its data type is shown in a tooltip.

If it is defined, the address and symbol comment are also shown as well as the operand comment in double quotation marks in a second line.

Inserting and arranging elements:

- The most important **commands for working in the editor** are always located in the context menu.
- The programming units (program elements) are **inserted** either by using the "Add" commands that are available by default in the or by dragging and dropping the element from the in the editor window. Inserting from the menu is based on the current cursor position and the present selection (multiple selections are also possible). When inserting from the toolbox, the possible insertion positions are displayed with position markers highlighted in green when the mouse is dragged over the element in the editor window and the element is inserted at this marker with a click.



Editors

- The commands **Cut**, **Copy**, **Insert** and **Delete** available in the "Edit menu" by default can be used to arrange the elements and networks.
 - The possible **insertion positions** are described in the
 - Inserting EN/ENO function blocks is handled differently in the FBD editor and in the LD editor. For more information, refer to

(Adding EN/ENO function blocks is not supported in the IL editor.)
- Navigating in the editor:**
- The **arrow keys** can be used to jump between adjacent within and across networks.
 - The <Tab> key can be used to jump to the next within the network.
 - <Ctrl>+<Pos1> focuses the start of the document and highlights the first network.
 - <Ctrl>+<End> focuses the end of the document and highlights the last network.
 - <PageUp> moves a page up and highlights the top rectangle.
 - <PageDown> moves a page down and highlights the top rectangle.
- Selecting:**
- An element can be selected via mouse click or using the arrow or tab keys to move to the corresponding cursor position.
 - Multiple non-adjacent elements can be selected by holding down the <Ctrl> key while selecting the desired elements one after the other.
 - Multiple adjacent elements can be selected by holding down the <Shift> key while selecting two contacts, one at the beginning and one at the end of the desired network section. To cut (copy) a network section and re-insert it, hold down the <Ctrl> key while selecting two contacts that define the section. Then, the elements located between the contacts are automatically included.
- Opening a function block** If a function block was added in the editor, open it with a double-click or using the command "Search symbol - Go to definition" of the context menu.

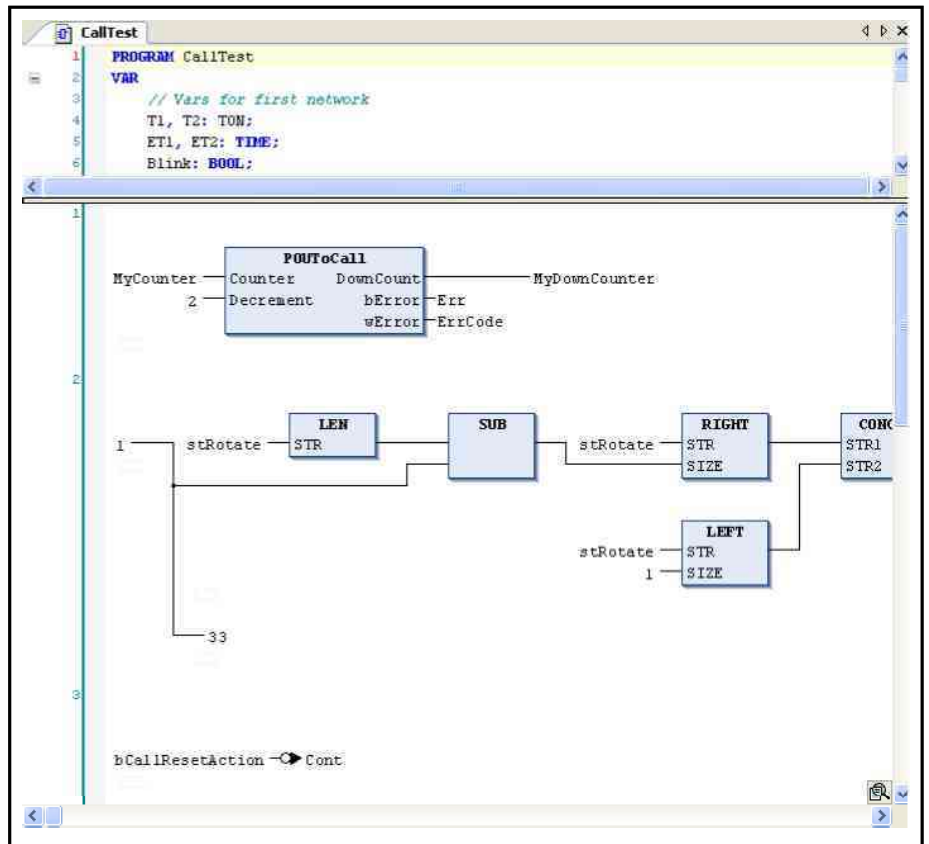


Fig.4-45: Example of an FBD editor window

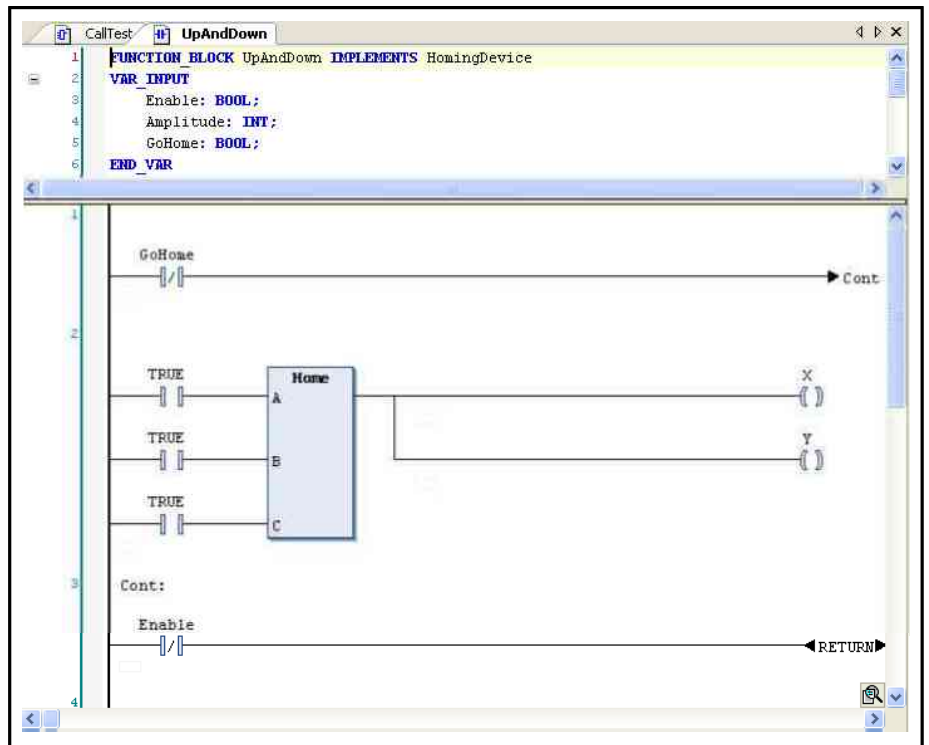


Fig.4-46: Example of an LD editor window

For information on the programming language, see:

-



Editors

4.6.5 Working in the IL Editor

The IL () editor is a **table editor** in contrast to the pure text editor used in IndraLogic 1.x. The **network** structure of FBD or LD programs can be sufficient, although when considering switching back and forth among the FBD, LD and IL views of a program, it might be advantageous to purposely structure an IL program using networks.



Behavior, appearance and menus are defined in the . See the special setting options there for displaying comments and addresses in the editor.

Tooltip with information on variables or function block parameters

If the cursor is pointing to a variable or a function block parameter, its data type is shown in a tooltip.

If it is defined, the address and symbol comment are also shown as well as the operand comment in double quotation marks in a second line.

Inserting and arranging elements:

- The most important **commands for working in the editor** are available in the context menu.
- Programming units, i.e. **elements**, are inserted at the current cursor position using the "Add" commands located in the by default.
- The commands **Cut, Copy, Insert** and **Delete** available by default in the "Edit menu", can be used to arrange the elements.
- The possible **cursor positions** are described in
- Information on the IL programming language can be found in

The following shows how the IL table editor is structured, how the table is navigated and how complex operands, calls and jumps are used.

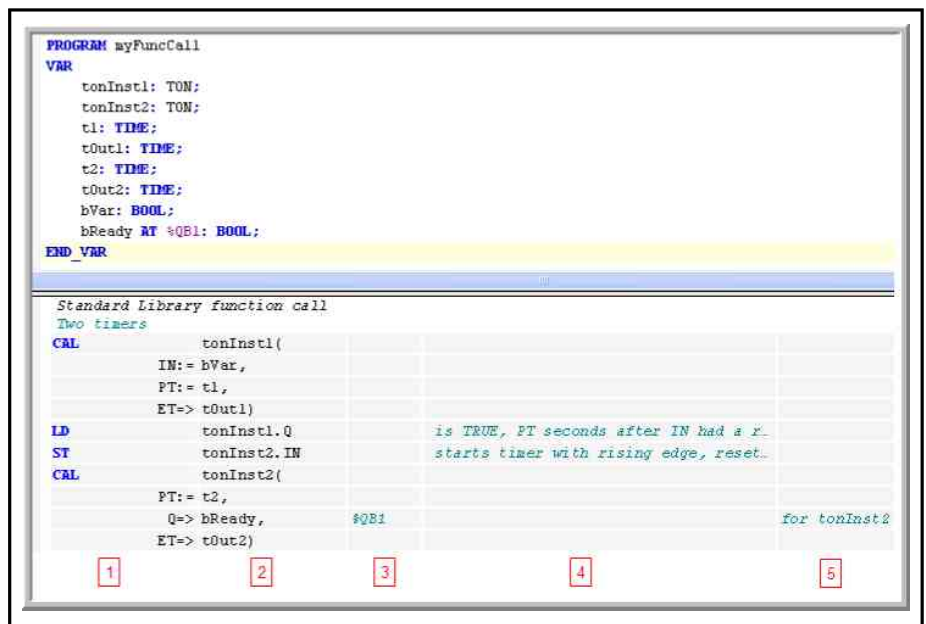
Structure of the IL table editor:

Each program line is in one table line divided into individual **fields** by the following table columns:

Column	Contains...	Description
1	Operator	This field contains the IL operator (LD, ST, CAL, AND, OR, etc.) or a function name. If a function block is called, the corresponding parameters have to be entered here as well. In this case, ":@" or "=>" has to be entered in the prefix field.
	Prefix	This field contains ":@" or "=>" if the parameter for a function block call is in the operator field.

Column	Contains...	Description
2	Operand	This field contains exactly one operand or the name of a jump label. If more than one operand ("AND A, B, C" or function calls with multiple parameters), they have to be entered in the following lines in which the operator field is empty. In this case, enter a comma in the postfix field to separate the parameters from each other (see the following figure, "IL table editor").
	Postfix	per operator or in case of function calls, this field contains the separating comma or the opening or closing parenthesis. If a or is called, the corresponding opening and closing parentheses have to be added.
3	Address	This field contains the address of the operand as it was defined in its declaration. The field cannot be edited and can be shown or hidden in the view using the
4	Symbol comment	This field contains the comment that might have been entered for the operand in its declaration. The field cannot be edited and can be shown or hidden in the view using the
5	Operand comment	This field contains the comment for the current program line. This can be edited here. The display of the comment can be shown or hidden using the

Fig.4-47: IL table editor



For explanations on the columns 1...5, see table "IL table editor" above

Fig.4-48: IL table editor

Navigating in the table:

- <Up> and <Down> arrow keys: Jump to the field above or below
- <Tab>: Jump to the next field within the line (to the right)
- <Shift> <+> <Tab>: Jump to the previous field within the line (to the left)
- <Space bar>: Opens the editing frame for the currently selected field. Alternatively, click on the field. If necessary, the input assistance dialog can be accessed with the button. An open input field can be exited with <Enter> or <Esc> which discards the new entries.

Editors

- <Ctrl> <+> <Enter>: Inserts a new line below the current line
- : Deletes the line in which the cursor is currently positioned.
- **Cut, copy, paste:** To copy one or multiple lines, select at least one field in the line(s) and use the "Copy" command. To cut a line, use the "Cut" command. "Paste" adds the previously copied or cut line(s) above the current line. If the cursor is not currently positioned in any line, it is pasted at the end of the network.
- <Ctrl>+<Pos1> focuses the start of the document and highlights the first network.
- <Ctrl>+<End> focuses the end of the document and highlights the last network.
- <PageUp> moves a page up and highlights the top rectangle.
- <PageDown> moves a page down and highlights the top rectangle.

Multiple operands (extendable operators):

If the same is used with multiple operands, there are two programming options:

1. The operands are entered in successive lines separated by commas in the postfix field. Example:

4		
	LD	7
	ADD	2,
		4,
		7
	ST	iVar

Fig.4-49: Multiple operands (extendable operators), e.g. 1

2. The instruction is repeated in successive lines. Example:

3		
	LD	7
	ADD	2
	ADD	4
	ADD	7
	ST	iVAR

Fig.4-50: Multiple operands (extendable operators), e.g. 2

Complex operands:

If a complex operand is used, an open parenthesis has to be entered in the prefix field, the further operand entries in the following lines and the closing parenthesis in a separate line, also in the prefix field.

Example: A string is rotated; each cycle by one character

Code in Structured Text:

```
stRotate := CONCAT(RIGHT(stRotate,(LEN(stRotate) - 1)), (LEFT(stRotate, 1)));
```

Code in IL:

4	LD	stRotate	
	RIGHT(stRotate	
	LEN		
	SUB	1	
)		
	CONCAT(stRotate	
	LEFT	1	
)		
	ST	stRotate	

Fig.4-51: Complex operands

Function calls: The function name is entered in the operator field. The input parameters have to be used as operands in a preceding LD operation. The return value of the function is written in the accumulator, but...

Note the following limitation with regard to the IEC standard: A **function call with multiple return values is not possible**. Only one single return value can be used for the following operation.

Example:

Function X7 is called. "25" is transferred as a input parameter and the return value is assigned to the variable "Ave":

Code in Structured Text:

```
Ave := GeomAverage(X7, 25);
```

Code in IL:

LD	X7	
GeomAverage	25	
ST	Ave	

Function block calls, program calls:

here. Enter the name of the function block instance and the program name in the operand field. The opening parenthesis is entered in the same line in the postfix field. The input parameters are entered individually in the following lines as follows:

Operator field: Parameter name

Prefix field: „:=“ for input parameters; "=>" for output parameters

Operand field: Current parameter

Postfix field: "," if more parameters are to be entered; ")" after the last parameter

If there are calls without parameters, the postfix field has to contain "()".

Example: Call of POUToCall with two input and two output parameters

Code in Structured Text:

```
POUToCall(Counter := MyCounter, Decrement:=2, bError=>Err, wError=>ErrCode);
```

Code in IL:

Editors

```

1  PROGRAM IL_EXAMPLE
2  VAR
3      bErr: BOOL;
4      wResult: WORD;
5  END_VAR
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
```

Editors

grammed with the command "JMPC" in the operator field instead of "JMP". The execution of the jump depends on the loaded value.

Example: Jump instruction: if bCallRestAction is "TRUE", the program execution jumps to the network with the "Cont" label.

Code in IL:

LDN	bCallResetAction	
JMPC	Cont	

4.6.6 Cursor Positions in FBD, LD and IL

IL editor:

This is a text editor structured as a table. Each table cell is a possible cursor position. See also:

FBD and LD editor:

These are graphical editors. See below: examples (1) to (15) show the possible cursor positions: Text, input, output, contact, coil, return jumps, connection line between elements and networks.

Setting the cursor corresponds with "selecting" an element or text. Actions such as cutting, copying, pasting, deleting and other editor-specific commands can be used at the current cursor position or at the currently selected element.

Also refer to

-

In FBD, the current cursor position is displayed with a dashed frame around the respective element. In addition, texts and function blocks as well as coils and contacts are displayed with a blue or red background.

In LD, coil and contact symbols appear in red when the cursor is positioned there.

The cursor position determines which element is provided in the context menu for the

Possible cursor positions:

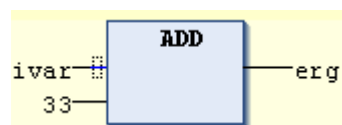
(1) Each text field:

In the following figure, the possible cursor positions are displayed at left with a red frame. At right, a function block is shown in which the cursor is currently positioned in the "AND" field.

Note that instead of the variable name, the addresses can also be displayed if the corresponding option is enabled in the



(2) Each input:



(3) Each operator, function or function block:

352/697

Bosch Rexroth AG

DOK-IWORKS-IL2GPRO*V12-AP01-EN-P

Rexroth IndraWorks 12VRS IndraLogic 2G PLC Programming System

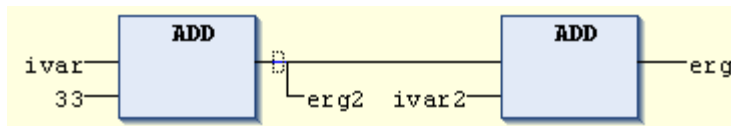
Editors



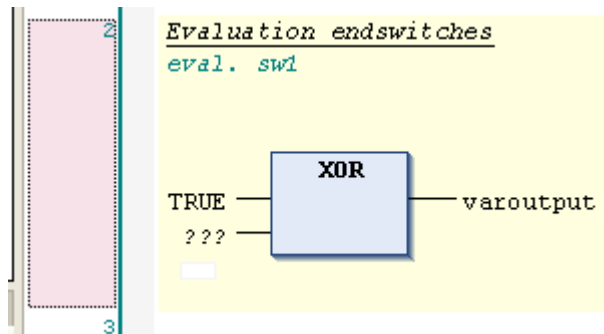
(4) Each output followed by an assignment or a jump:



(5) The position in front of a branch to an assignment, a jump or a return instruction:



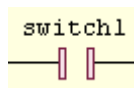
(6) The cursor position located farthest to the right or anywhere in the network where there are no other cursor positioned. This selects the entire network:



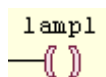
(7) The line cross directly in front of an assignment:



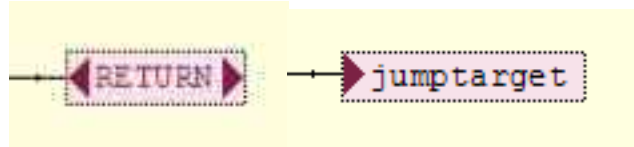
(8) Each contact



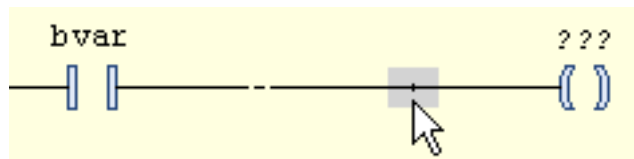
(9) Each coil



(10) Each return and jump:



(11) The connection line between contacts and coils.



(12) Line branch or "subnetworks" within a network

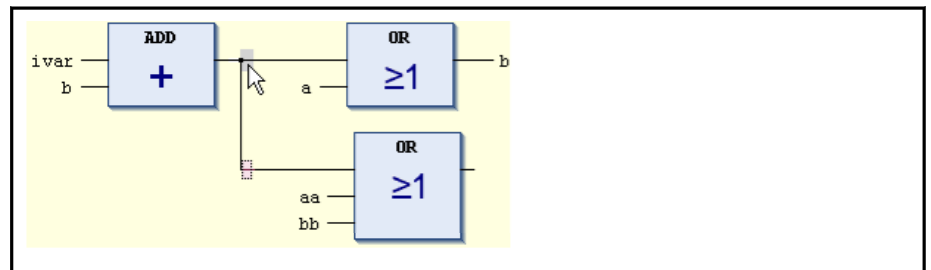


Fig.4-53: Line branches

(13) Connection line between parallel contacts (pos. 1-4)

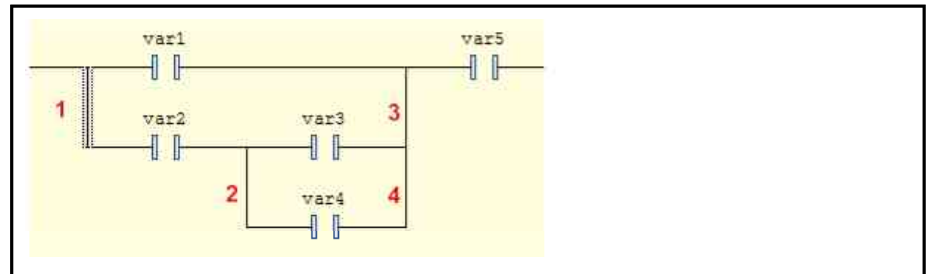


Fig.4-54: Connection line between parallel contacts

(14) In front and behind networks

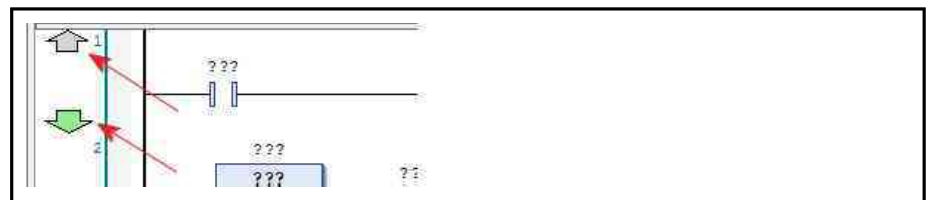


Fig.4-55: In front and behind networks

New networks can be added at the left side of the editor. Inserting a new network in front of an existing network is only possible in front of network 1.

(15) Beginning or end of a network

Editors

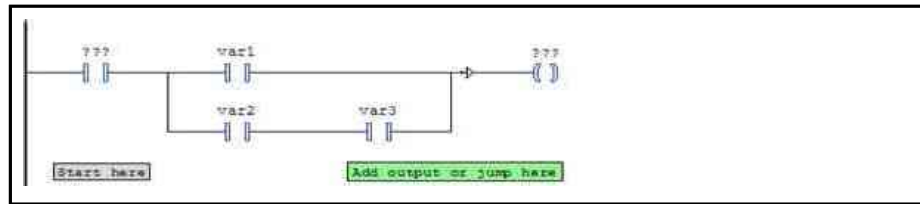


Fig.4-56: Beginning or end of a network

Contacts and function blocks can be inserted at the beginning of a network on the "Start here" field. The elements return, jumps and outputs can be inserted at the end of a network on "Add output or jump here" field.

4.6.7 FBD/LD/IL Menu

the FBD/LD/IL menu is available in the menu bar by default. The menu provides the following commands for programming in the editor:

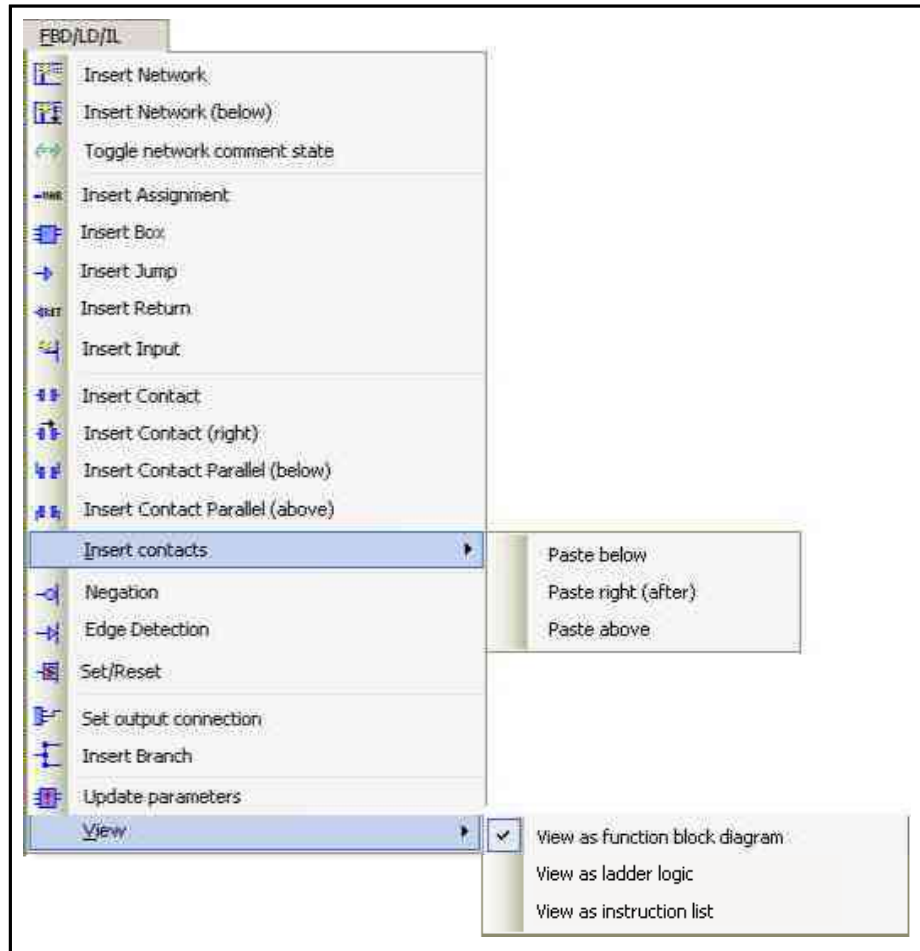


Fig.4-57: FBD/LD/IL menu, here subset of the ladder diagram

A description of the individual commands can be found under

If required, the menu structure can be reconfigured in the dialog under **IndraWorks ► Tools ► Customize ► Commands**.

4.6.8 FBD/LD/IL Elements

Editors

FBD/LD/IL tools

provides a toolbox in a separate view from which the individual programming elements can be inserted into the editor window via drag&drop. By default, the toolbox is opened automatically next to the editor window, but if necessary, it can also be opened explicitly using the command in the "View" menu.

Depending on **which editor view is currently open**, select the corresponding elements in the toolbox. Elements that are not available are shaded in gray (see the description for the respective).

The "tools" (elements) are sorted in **categories**: General (general elements such as network, assignment, etc.), logical operators, mathematical operators, function blocks (e.g. R_TRIG, F_TRIG, RS, SR, TON, TOFF, CAD, CDU), ladder diagram elements and POU's.

The "POU's" category provides all POU's created by the user below the same application as the FBD/LD/IL function block opened in the editor. If a bitmap is assigned to a POU in "Properties", it appears in front of the POU name. Otherwise, the standard icon appears to label the POU type. The list is automatically updated if POU's are added or removed below the application.

Open the category, i.e. display the elements, by clicking on the button with the category name. See the following figure: "General" category is expanded and the others are collapsed. The figure shows an example for inserting an assignment from the toolbox via drag&drop:

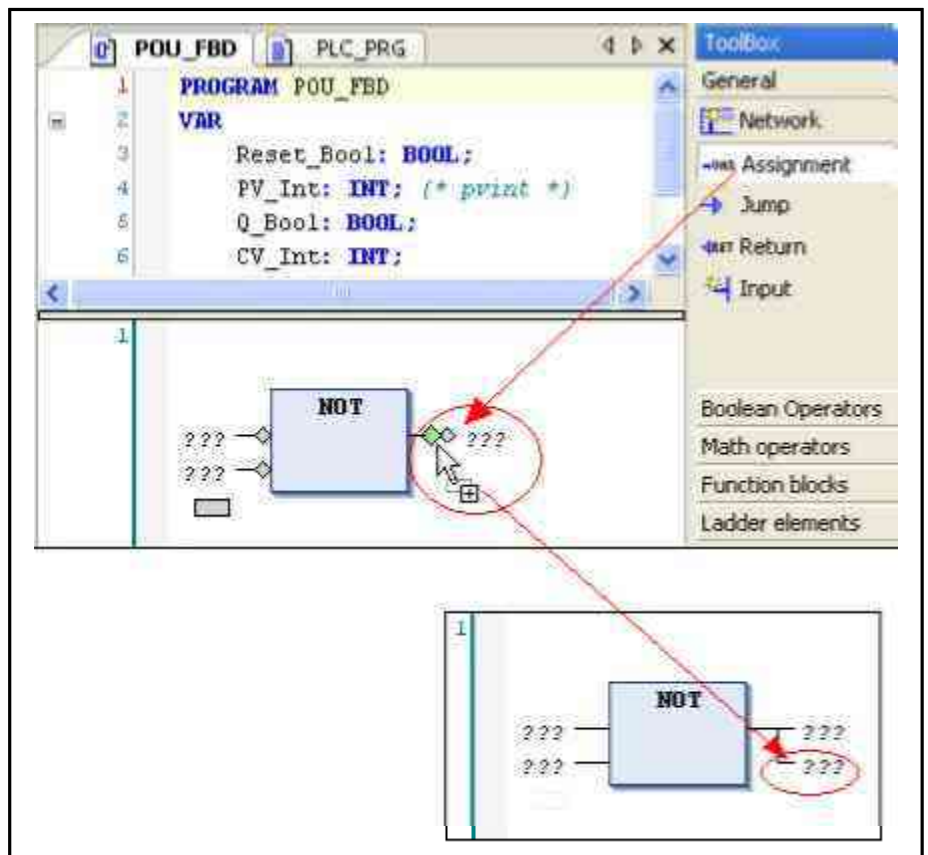


Fig.4-58: Example of inserting from the toolbox

To **insert** an element in the editor, click on it in the toolbox to select it and hold down the mouse button while dragging it into the editor window. The

Editors

possible insertion positions are indicated by **position labels** (usually diamonds) while dragging the element across the editor window with the mouse button pressed. The closest possible insertion position lights up in **green**. When releasing the mouse button, the element is inserted at the position displayed in green.

If a function block element is inserted at a position where another element is already located, the new element replaces the old one. In principle, the inputs and outputs already assigned to the old element are maintained.

Network in FBD/LD/IL

... program.

In the FBD/LD/IL editor, the networks are arranged in a top-down list. Each network has a consecutive network number at the left side and contains logical or arithmetic expressions, program, function or function block calls and a jump or return instruction.

editor also uses the network element based on the common editor environment with the FBD and LD editor. If a POU is originally programmed in FBD or LD and is then converted to IL, the network structure is maintained. If a program is originally created in IL, it consists of at least one network that can contain all instructions. However, in consideration of a planned conversion to FBD or LD, it might be advantageous to structure the IL program using several networks.

A network can optionally be provided with a title, a comment or a **jump label** that can be used as a jump target for a jump from another network.

The availability of title and comment fields can be switch on or off in the

If this function is switched on, the input field for the title can be opened by clicking in the network directly below the upper limit. Click directly below the title to open a text field to enter a comment. The comment can contain multiple lines. Line breaks can be inserted with <Enter>. End the comment text input with <Ctrl>+<Enter>. Whether and how the network comment is displayed in the editor, is defined in the same "Options" dialog.

To insert a label that can be used as a jump target for a jump from another network **jump label** command. After a jump label is defined, it is displayed below the title or comment field, or, if these fields are not present, directly below the upper limit of the network.

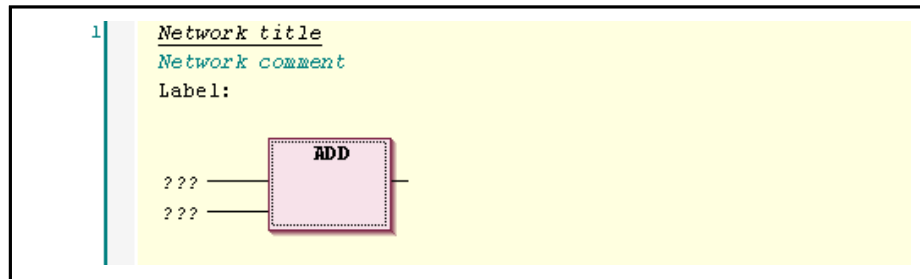


Fig.4-59: Position of title, comment and jump label in a network

A network can be commented out, i.e. the entire network can be redefined as a comment (' ') and therefore excluded from processing.

When a network is selected () the default commands for **copy**, **cut**, **paste** and **delete** can be used.



If you right-click on a title, comment or label (cursor position 6), only that item is selected, not the entire network. In this case, using the "default commands" does not affect the network itself.

See the description for **adding** a network in .
Consider the option of " in a network"


RET Network:

In online mode, an additional empty network is automatically represented below the existing networks. Instead of a network number, it is labeled with "RET".

It represents the position at which it is returned to the calling function block during processing and provides a possible **breakpoint position**.

Assignment in FBD/LD/IL

In FBD or LD, an assignment is inserted as a line depending on the current either directly in front of the input (cursor position 2), directly behind an output (cursor position 4) or at the end of the network (cursor position 6). In an LD network, an assignment is inserted as a

After the insertion is complete, the character string "???" can be replaced by the name of the variable that is to be assigned. The input assistance can be accessed with the  button.

In , an assignment is programmed using the "LD" and "ST" instructions.

See also the .

Jump in FBD/LD/IL

In or , a jump is inserted as a line depending on the current either directly in front of an input (cursor position 2), directly behind an output (cursor position 4) or at the end of the network (cursor position 6).

After the insertion is complete, the character string "???" can be replaced by the name of the label that is to be used as the jump target, i.e. the network, to which the jump is to be made.

In , a jump is programmed using a "JMP" instruction.

See also the .

Label in FBD/LD/IL

network has a text input field below the field for the network can be defined. This label is an optional identifier for the network and can be entered as an address for a . It can consist of any character string.

Editors

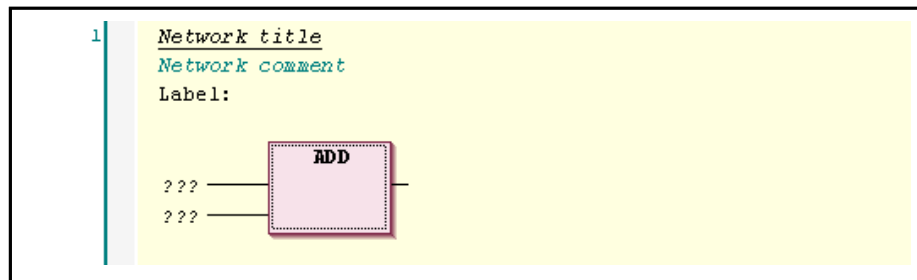


Fig.4-60: Label for network

Function Block Call in FBD/LD/IL

A network is a complex element and can represent additional functions such as a timer, counter, arithmetic operations or programs, IEC functions and IEC function blocks.

Such a function block can have any desired inputs and outputs and can originate either from a library or directly from a project. However, at least one input and one output has to provide Boolean values.

Usage in FBD and LD

A function block call can be inserted in the left section of an LD network (as a coil) or in an FBD network using the "Add FB call" or "Add empty block" commands or (for standard blocks) from the

Usage in IL

In an IL program, a CALL instruction with parameters is inserted to call a function block.

You can get an **update** of the block parameters (inputs, outputs) in current code - in case the function block interfaces have changed - by using the command `CALL_UPDATE`; the function block does not need to be inserted again.

RETURN Instruction in FBD/LD/IL

Use a RETURN instruction to exit an FBD, LD, or IL function block.

In a FBD or LD network, the RETURN instruction can be placed parallelly with or subsequent to the preceding elements. As soon as the input of the RETURN instruction is TRUE, the function block processing is interrupted immediately.

For inserting elements, please see

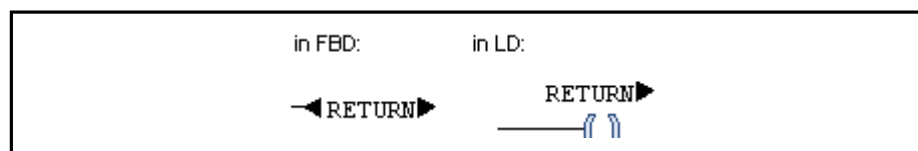


Fig.4-61: RETURN element

In IL, the CALL instruction is used for the same purpose.

Line Branching/Post-Interconnection in FBD/LD/IL

FBD and LD

In an FBD or LD network, a line branch or post-interconnection splits a processing line starting from the current cursor position. It is carried out in two arms (subnetworks), from top to bottom one after another. Each subnetwork can be branched more. Thus, multiple branching is possible within one network.

Editors

Each subnetwork receives a label icon (rectangle) that can be selected () to execute actions on the subnetwork such as cutting and pasting.

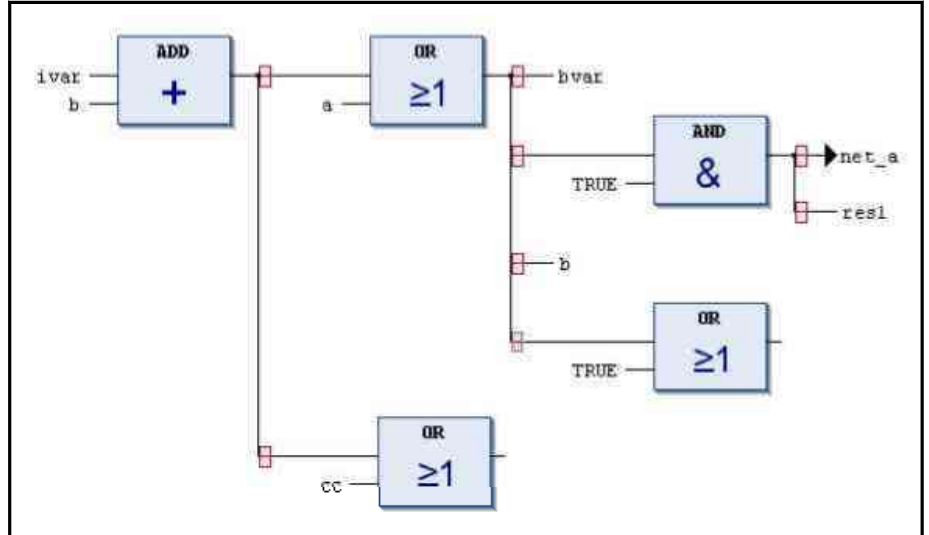


Fig.4-62: Examples for labeling networks within a network.

The "Insert branch" command is used to add a branch to the FBD. Alternatively, the element can also be obtained from the . Possible insertion positions can be found under .



Cut/Copy&Paste are currently not possible for subnetworks.

See the example in the following figure: A branch was inserted at the output of the SUB function block. This creates two subnetworks. Each can be selected at a subnet label. Then, an ADD function block was added to each subnetwork

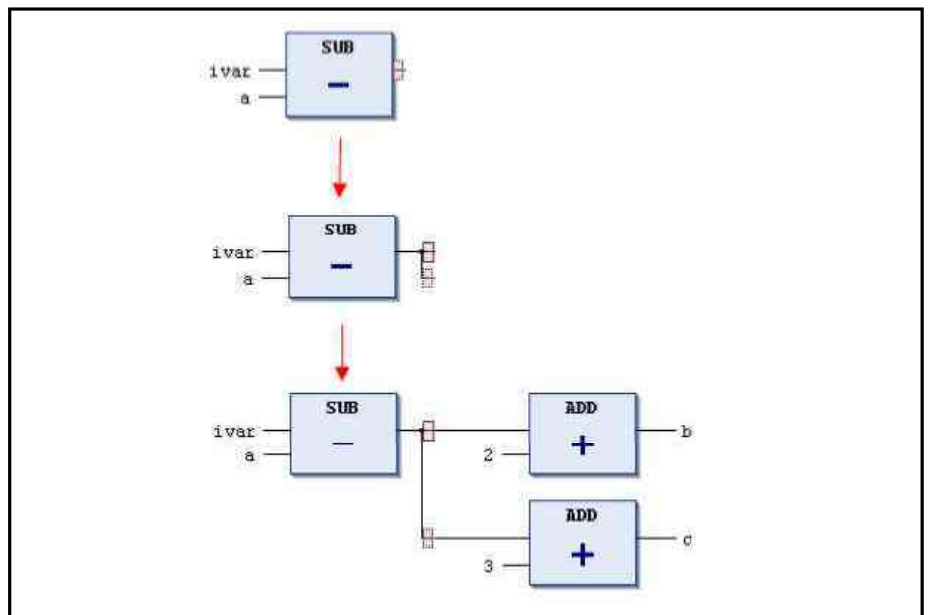


Fig.4-63: Example in FBD, Inserting a branch

To delete a subnetwork, all its elements have to be deleted first. That are all elements on the right from the subnet label. The label can then be selected and deleted with "Delete" or . Refer to the following figure: The three-

Editors

input OR element has to be deleted before the label of the lower subnetwork can be selected and deleted.

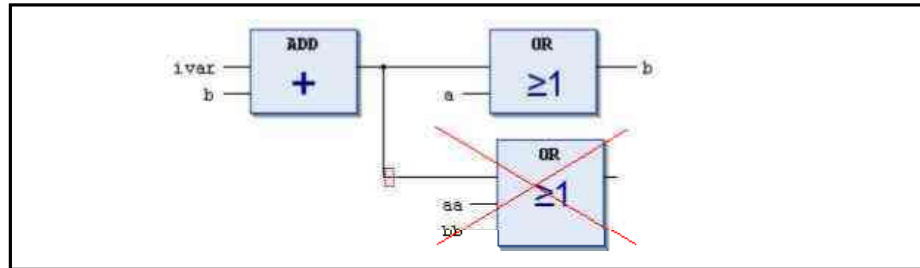


Fig.4-64: Deleting branch or subnetwork

Execution in online mode:

The individual subnetworks in online mode are processed from left to right and from top to bottom.

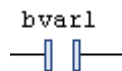
IL (instruction list)

In IL, line branching and post-interconnection are returned by a respective sequence of instructions. See also the description of

Contact in FBD/LD/IL

This is a pure LD element.

contains one or multiple contacts in its left section. A contact is displayed as follows:

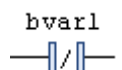


A contact transfers the condition "ON" (TRUE) or "OFF" (FALSE) from left to right until it finally reaches a coil in the right section of the network. For this purpose, a **Boolean variable** that contains the condition is assigned to the contact.

Several contacts can be arranged in a **sequence** or in **parallel**. If there are two parallel contacts, only one has to have the value TRUE for "ON" to be transferred to the right. If contacts are connected in **series**, **all** contacts have to contain the value TRUE for the "ON" to be transferred to the right from the last contact.

This way, LD can be used to program an electric circuit in parallel or in series.

A contact can also be **negated**. This is indicated by a slash in the contact icon.



A negated contact only transfers the incoming condition (TRUE or FALSE) if the Boolean variable assigned to it has the value FALSE. The directly provides negated contact elements.

A contact element can be **inserted** into an LD network either by using the commands



Editors

-
-
-
-

or directly via drag&drop from the

FBD, IL

editor view, the commands for inserting contacts are not available. However, contacts that were previously inserted in the LD view are displayed using the corresponding FBD elements or IL instructions.

Coil

This is a pure **LD element**.

Any desired number of coil elements can be inserted on the right side of a LD network. A coil is displayed as follows:



Several coils can only be arranged in **parallel**. A coil transfers the value delivered from the left to the right and copies it into its assigned **Boolean variable**. Its input value can be "ON" (TRUE) or "OFF" (FALSE).

A coil can also be **negated** indicated by a slash in the icon:



of the incoming signal is copied to the Boolean variable assigned to the coil. Therefore, a negated coil will only transfer an "ON" signal if this variable has the value FALSE.

command located in the FBD/LD/IL menu by default or via drag&drop from the toolbox, the "Ladder diagram elements" category.

See also

FBD/IL:

editor view, the commands for inserting contacts are not available. However, contacts that were previously inserted in the LD view are displayed using the corresponding FBD elements or IL instructions.

Set/Reset in FUP/KOP/AWL

FBD and LD

can be set or reset. The output or the coil icon are identified with an S (set) or R (reset).

See also

Set: If the value TRUE is provided to a set output or a set coil, the output and coil become TRUE and remain TRUE. The value can no longer be overwritten as long as the application is running.

Reset: If the value TRUE is provided to a reset output or a reset coil, output and coil become FALSE and remain FALSE. The value can no longer be overwritten as long as the application is running.

Editors

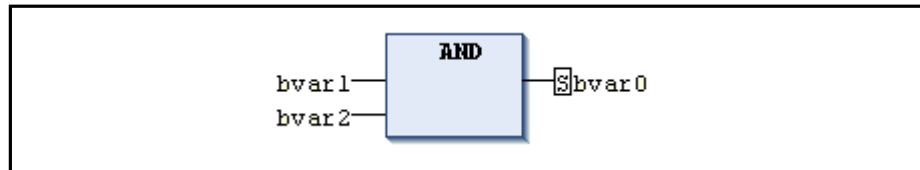


Fig.4-65: Example of a set output in FBD

In the LD editor, set and reset coils can be inserted directly via drag&drop from the toolbox from the "Ladder diagram elements" category.

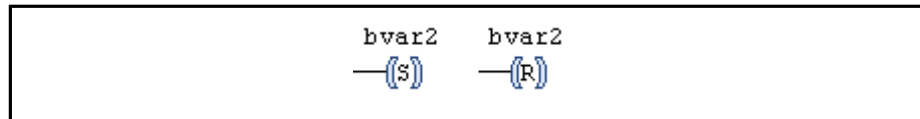


Fig.4-66: Examples of a set coil and reset coil

See for information on set and reset coils.

- IL In an instruction list, the operators and are used to set or reset an operand.

Set/Reset Coils

Coils can also be defined as set or reset coils. A set coil is identified by an "S" in the coil icon: (S). A set coil never overwrites the value TRUE of the related Boolean variable, i.e. a variable with the truth value TRUE retains this value.

A reset coil is identified by an "R" in the coil icon: (R). A reset coil never overwrites the value FALSE of the related Boolean variable, i.e. if this variable has the truth value FALSE, this value remains FALSE.

In the LD editor, use the mouse to drag set and reset coils directly from the toolbox ("Ladder diagram elements":) into the editor.

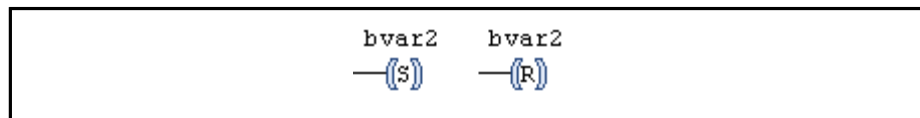


Fig.4-67: Example - Set coil, reset coil

4.6.9 FBD/LD/IL Editor in Online Mode

In the online mode of the FBD/LD/IL editor, there are views for and for and variables and expressions on the control.

(breakpoint, step-by-step execution, etc.) is also available.

Monitoring

, it is available in the FBD and LD editor views as small monitoring windows after each variable or in a separate column in the IL table editor.

These windows display the respectively current value on the target system (**Inline monitoring**).

That also applies to function block inputs and outputs that are not assigned.

The Inline monitoring variable of a window displays a small red triangle in the upper left corner if the variable is currently



Editors

a blue triangle in the left bottom corner if the variable is currently forced and prepared to cancel forcing:

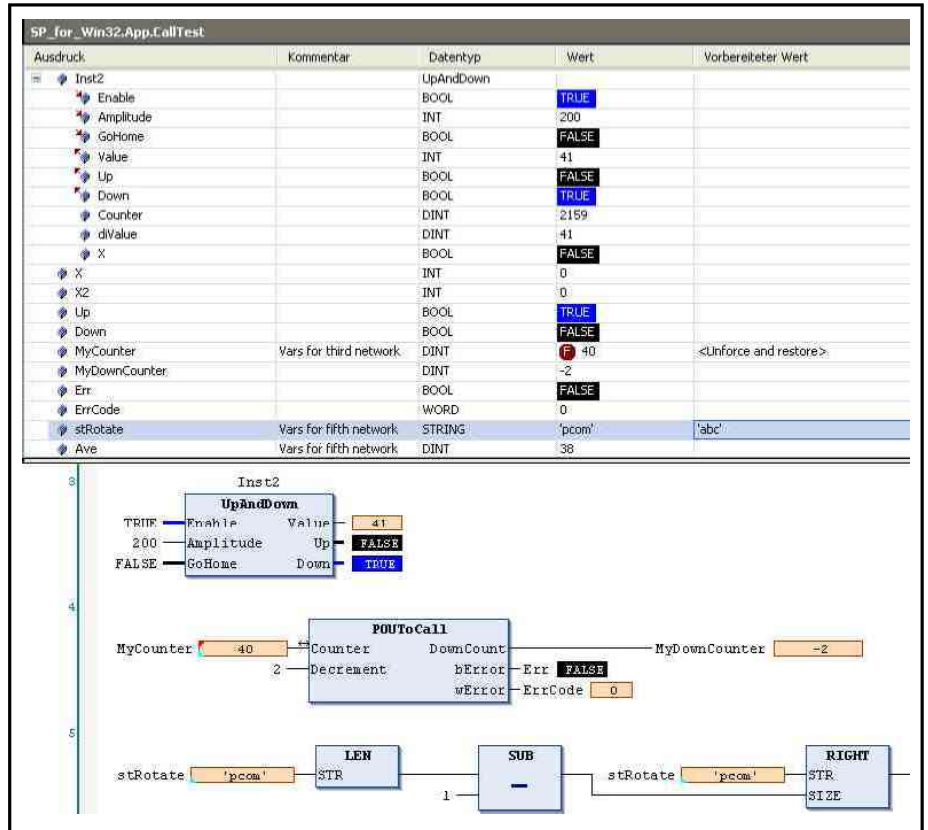


Fig.4-68: Online view of an FBD program

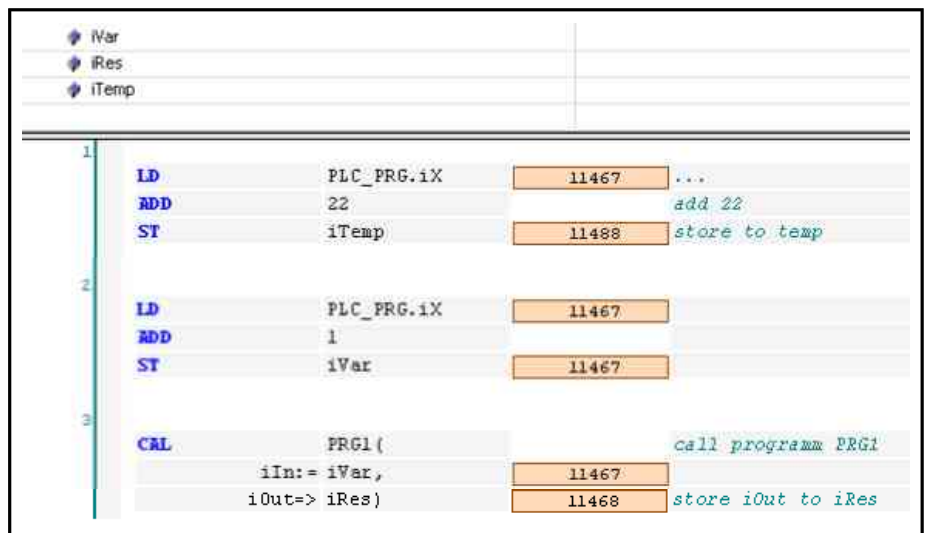


Fig.4-69: Example: Online view of an IL program

In the online view of a ladder diagram (LD), the connection lines are colored: Connections with the value TRUE are displayed with a bold blue line, connections with the value FALSE as bold black line, and in contrast, connections with unknown or analog values are displayed as normal (thin black line). (Note: the value of the connections is calculated from the value of the variables. This is not an actual sequence check.)

Editors

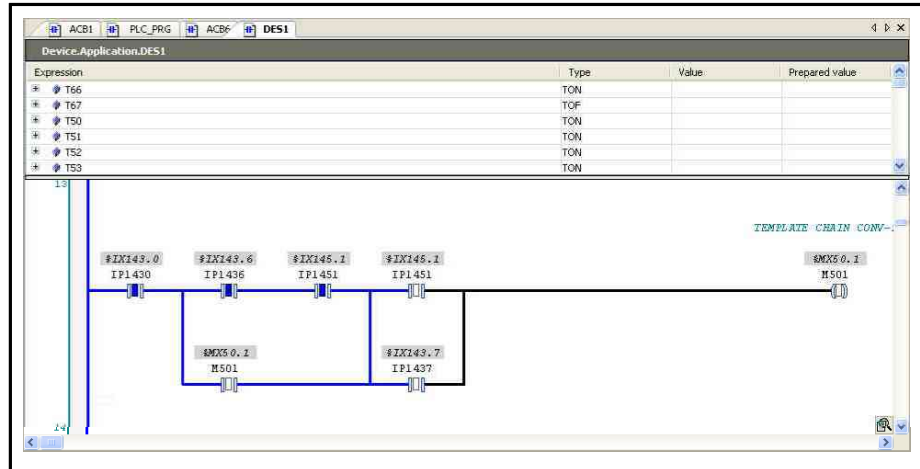


Fig.4-70: Example - Online view of an LD program

A function block can be opened via double-click or by using the command "Search symbol - Go to definition" of the context menu.

Forcing variables

In addition to be able to enter a prepared value for a variable in the declaration section in each editor, you can also click on a variable in the implementation section, which opens a dialog in which the prepared value can be entered in the FBD/LD/IL editor in online mode

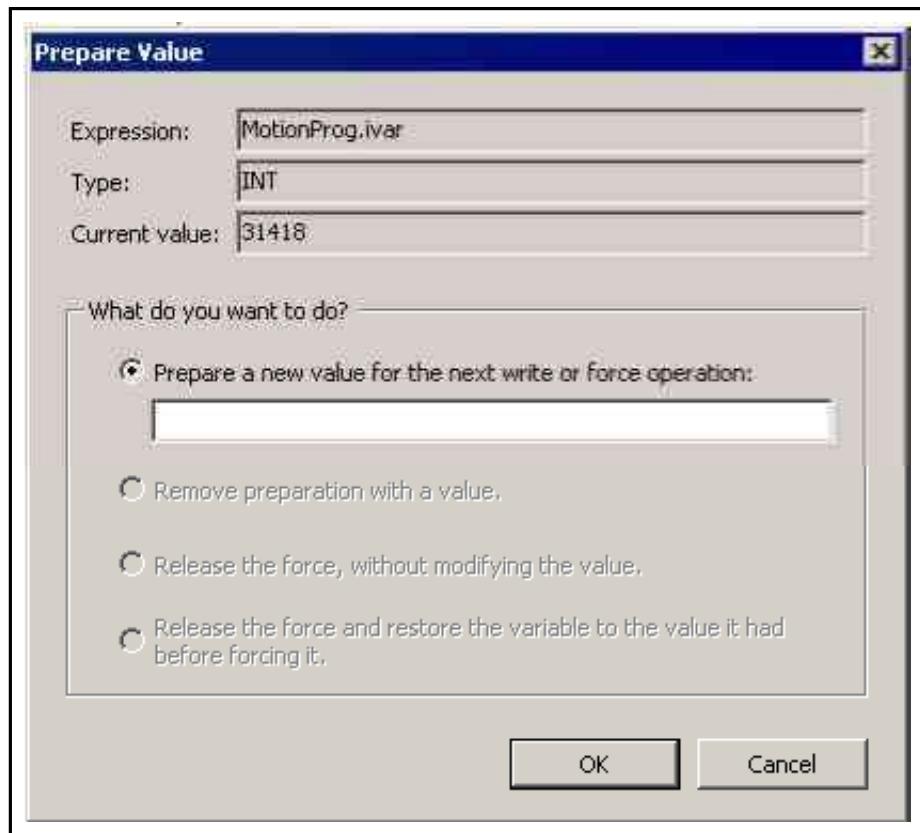


Fig.4-71: Dialog - Prepare Value

The complete path for the variable in the device tree is shown along with its type and current value. By enabling the corresponding item, decide whether



**Breakpoint or breakpoint positions:
(not yet available for the IL editor)**

- a new value should be prepared that has to be entered into the editing field
- a prepared value should be removed
- a currently forced variable should be released
- a currently forced variable should be released and its original value restored before it is forced

The selected action is carried out when the "Force value" command (Debug menu item) is called or when <F7> is pressed.

Possible positions that can be selected for a breakpoint () for debugging purposes are always the positions where variable values can change (instructions), where a program branches or where another function block is called. That are the following positions:

- On the complete network. That causes the breakpoint to be set at the first possible position in the network
- On a function block (box) if the function block contains an assignment. That is thus not possible for operators such as ADD, DIV). Refer to the following note:



A breakpoint can **currently** not be set on the first function block in the triangle. If the breakpoint is set on the complete network, this breakpoint position label is automatically transferred in online mode to the first function block.

- On assignments
- At the end of the function block at the return position to the calling function block. In online mode, an empty network automatically appears at this place which is labeled with "RET" instead of a network number.

The currently possible positions can be seen in the selection list in the . A network containing an active breakpoint is labeled with the breakpoint icon (red filled circle) on the right next to the network number and with a red-shaded rectangle by storing the first breakpoint position possible in the network. Disabled breakpoint positions are displayed by an empty, red circle or a red rectangular frame.

Editors

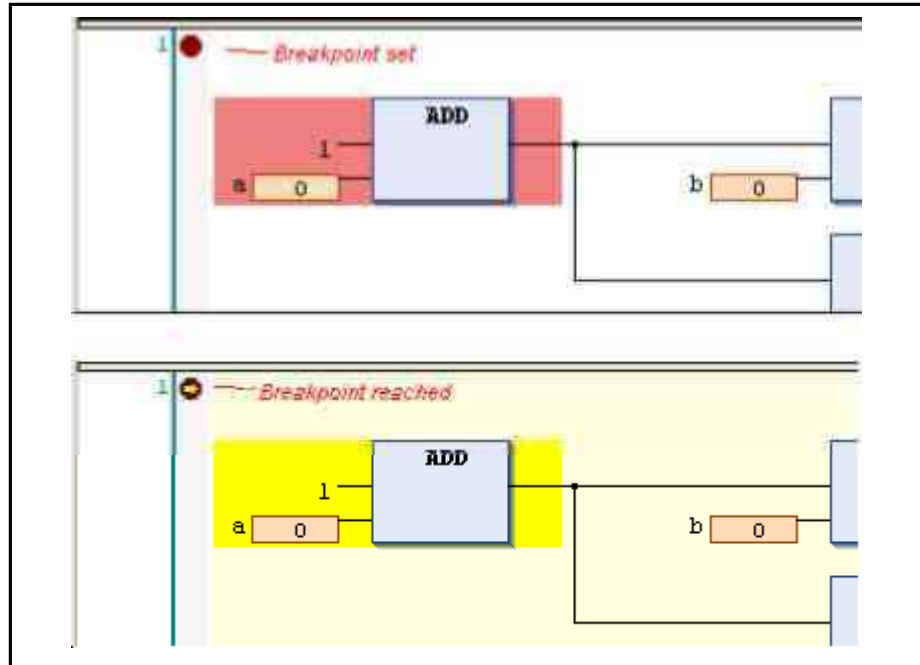


Fig.4-72: Example: Breakpoint set or reached

As soon as a breakpoint position is reached during the incremental processing or the program sequence, a yellow arrow appears in the breakpoint icon and the red shading changes to yellow.

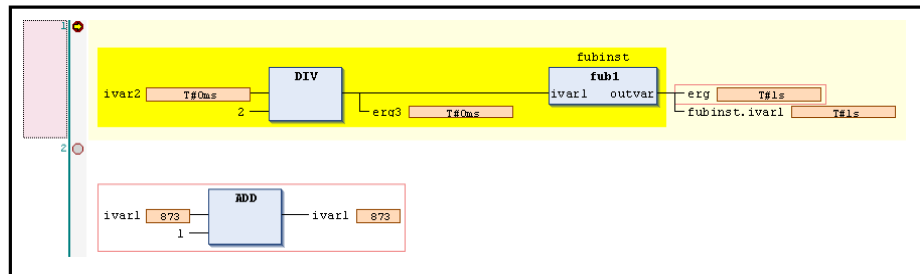


Fig.4-73: Breakpoint positions in FBD

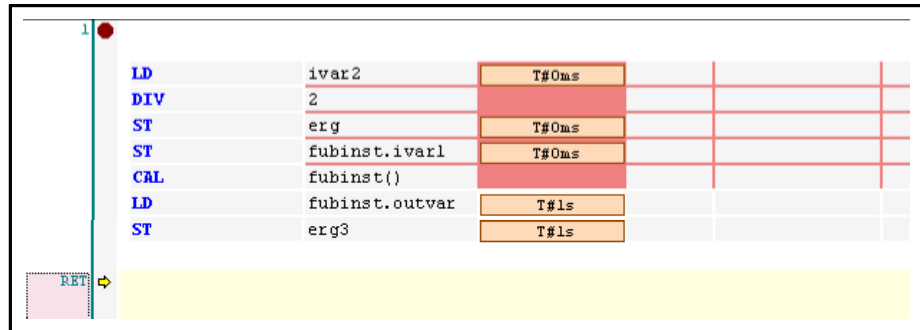


Fig.4-74: Breakpoint positions in IL



A breakpoint is automatically set in all methods that can be called. For this reason, if a method managed by an interface is called, breakpoints are set in all methods that appear in function blocks that implement this interface as well as in all derived function blocks that "record" for this method.

If a method is called by a pointer to a function block, the breakpoints are set in the method of the function block and in all of the derived function blocks that record for the method.

4.7 GVL Editor

The editor for global variable lists (GVL editor) is a declaration editor to create and edit settings in the . It functions based on the current settings in the and is displayed in online mode as described for the .

The declaration has to begin with "VAR_GLOBAL" and end with "END_VAR". These keywords are automatically available.

Valid of global variables are inserted in between.

```

Gvl1[DCC_Control_01: Logic: Application]
1  VAR_GLOBAL
2      glob_iVar: INT;
3      glob_bVar: BOOL;
4      glob_stVar: STRING;
5  END_VAR
6
    
```

Fig.4-75: GVL editor

4.8 Library Manager

4.8.1 Library Manager, General Information

The library manager integrates and manages libraries in the project.

Installing libraries, like defining library storage locations (repositories), is done in the .

This dialog can be opened using the command of the same name in the "Tools" menu (by default) or in the "Library Manager" dialog.

In the project, the library manager can be inserted with either global access in the "General module" folder or application-specific with the "Add" dialog.

At each of these positions, only one library manager per level can be inserted. The entry has the name given in the "Add object" dialog.

Editors

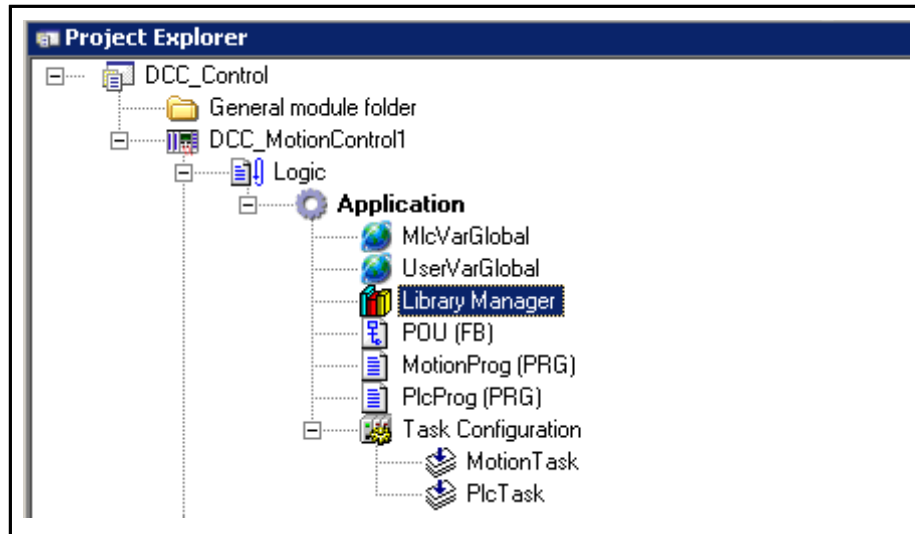


Fig.4-76: Library manager in the Project Explorer

The library manager opens via "Open" or a double-click.

Compiling errors with regard to the library manager are output in the "message window".

For general information on the

4.8.2 Editor Window of the Library Manager

The library manager opens via "Open" or double-click on the object entry.

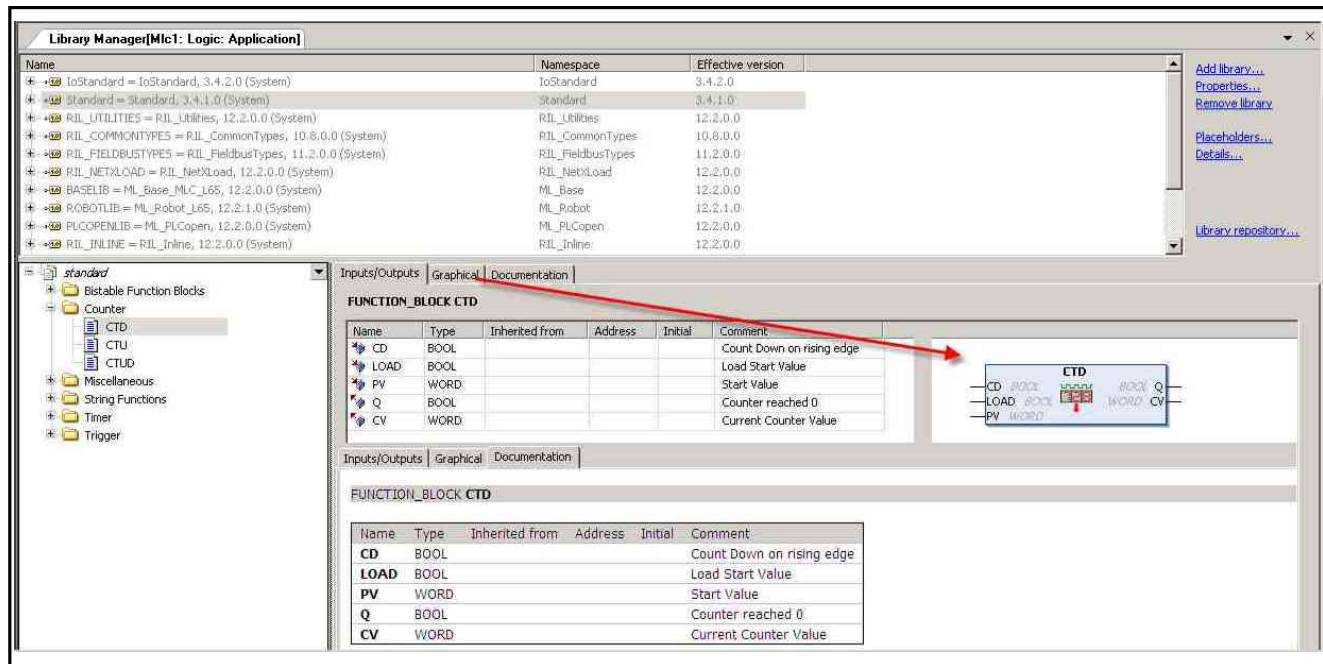


Fig.4-77: Library manager editor window

Structure of the editor window

The upper section of the library manager displays the **currently included in the project**. The following information is given:

cur-



Editors

Name: Title, version and the company name (optional) as defined in the "Summary" dialog in the when the library was created.

Namespace: The default for the library namespace is <LibraryName> unless it was explicitly defined with another namespace in the library information of the library project.

has to precede the function block identifier if there is supposed to be unambiguous access to a function block that is present in multiple instances in the project.

The default namespace in an included library can also be modified for local use in the project. This is done in the "Properties" dialog, see

Further information on

Effective version: indicates the version of the library currently used in the project according to the definition in the

Libraries automatically added to the project are displayed in a gray font. Those that are manually added (Add library...) are shown in black.

An icon in front of the library name indicates the type of library:

	IndraLogic 2G library (contains version information)
	Referenced library, automatically included.
	The referenced library file could not be found or is not a valid library file (see the corresponding message in the "library manager" category in the message window). In this case, see:

If a library has dependencies on other libraries (), these - when they are found - are also automatically included and displayed with a preceding symbol in a subbranch of the item. Such a subtree can be expanded or collapsed either with a plus or a minus sign. For an example, see the "ML_Base" library.

Name	Namespace	Effective version
IoStandard, 3.1.3.2 (System)	IoStandard	3.1.3.2
Standard, 3.0.1.0 (System)	Standard	3.0.1.0
RIL_CommonTypes, 9.5.0.0 (System)	RIL_CommonTypes	9.5.0.0
RIL_NetLoad, 9.3.0.0 (System)	RIL_NetLoad	9.3.0.0
ML_Base, 10.2.0.0 (System)	ML_Base	10.2.0.0
CmpErrors, * (System)	CmpErrors	3.1.3.0
SysTypes, * (System)	SysTypes	3.1.2.0
RIL_COMMONTYPES - RIL_CommonTypes, 9.5.0.0 (System)	RIL_CommonTypes	

Fig.4-78: Referenced libraries

In the lower left section of the editor, the function blocks of the currently selected library are also displayed in a tree structure. The usual sorting and search functions are available in a menu bar.

The following tabs are found in the lower right section:

Documentation: The components of the currently selected library function block at the left are displayed in a table with (variable) name, data type and the comment that might have been included in the declaration when the . Adding such a comment is an easy way to automatically provide the user with documentation of a function block.

Editors

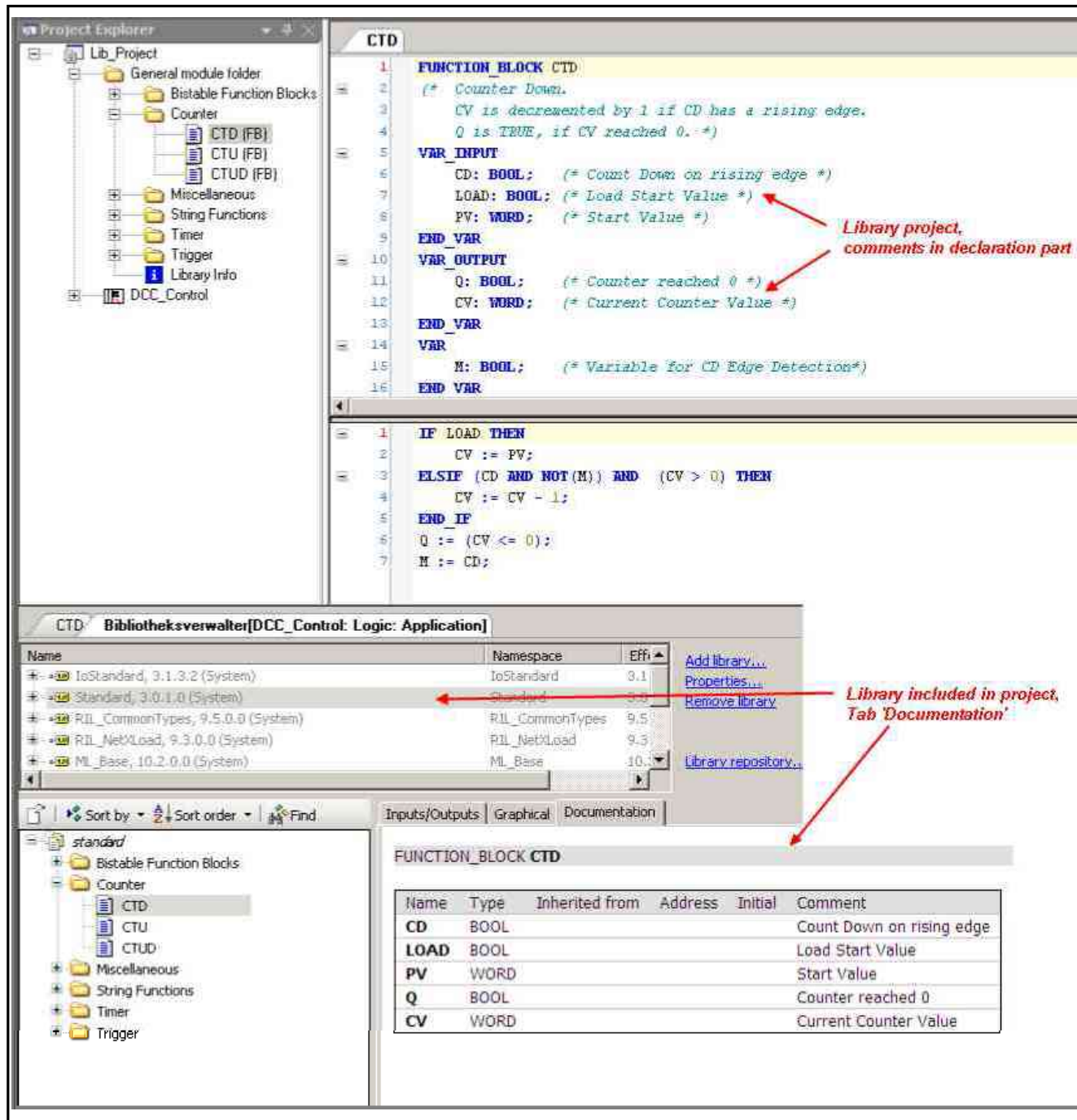


Fig.4-79: Example of commented library function blocks

Inputs/outputs: The components of the currently selected library function block at the left are displayed in a table with (variable) name, data type, address, initial value and comment as defined in the library.

Graphical: Graphical display of the function block.

Buttons and commands in the library editor window

The following commands are available in the editor window if one or multiple entries in the tree of included libraries are selected. Some of them can also " menu which appears in the menu bar by default when working in the library manager:



Add library..., page 227, to include a library in the project. Prerequisite: the library has to be installed in the system.

Properties..., page 230, for settings on the namespace, and - in case the library appears later as a "referenced library" in another project - for settings of version management, display and access.

Remove library: The libraries currently selected in the list of included libraries are removed.

Library repository..., page 185, to define storage locations and install or uninstall libraries.

A corresponding message appears if an attempt is made to add a library that is already included in the project.

4.8.3 Library Manager Menu

The currently available commands of the library manager are located at the right margin of the opened "Library Manager" dialog.

is highlighted in the Project Explorer, the "Library Manager" menu appears in the menu bar by default.

Installing libraries, like defining library storage locations (repositories), is done in the

4.8.4 Library Information

Icon:

In addition to the project properties, the "Project Information" dialog (Library Info, Project Information) also contains other information (e.g. access rights, version number, author, company, statistics on the project objects, etc.). It has already been added as an object in the POU window in a "default project". In this case, the "Standard.library" library.

They are required by the user even if he does not want to

Editors

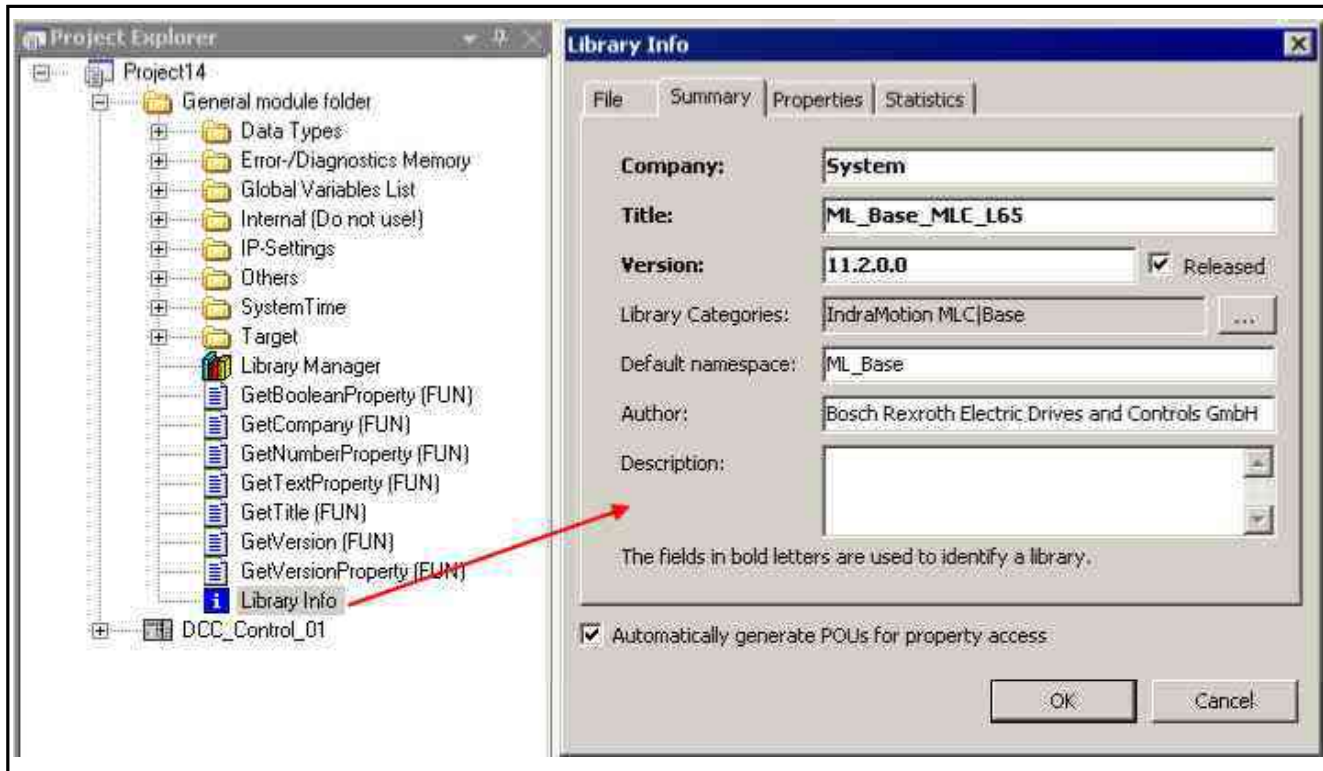


Fig.4-80: "Library Info" dialog

To open the dialog, double-click on the object in the Project Explorer or use the "Open" command.

Note the option to access library information externally using the and additionally created functions (see below).

Automatically generate POUs to access the properties:

If this option is enabled, the POU objects of type function are automatically created in the POUs window which can be used to access project properties from the application program.

In this case, special functions are generated for the 'Company', 'Title' and 'Version' properties (GetCompany, GetTitle, GetVersion).

To explicitly access defined properties, a corresponding function is available for each property type (GetTextProperty, GetBooleanProperty, GetNumberProperty, GetVersionProperty). In this case, call the corresponding function and transmit the "Properties" key (as defined in the tab) as input so that the property value is returned.

For an example, refer to

Example:

The following property is defined in the "Properties" tab: Key = nProp1, Type = Number, Value = 333. To receive the value in the application program, call the "GetNumberProperty" function; e.g.. showprop:=GetNumberProperty("nProp1"). In this case, "showprop" has to be declared as DINT.

There are four tabs available for the information categories "File", "Summary", "Properties" and "Statistics".



1. File

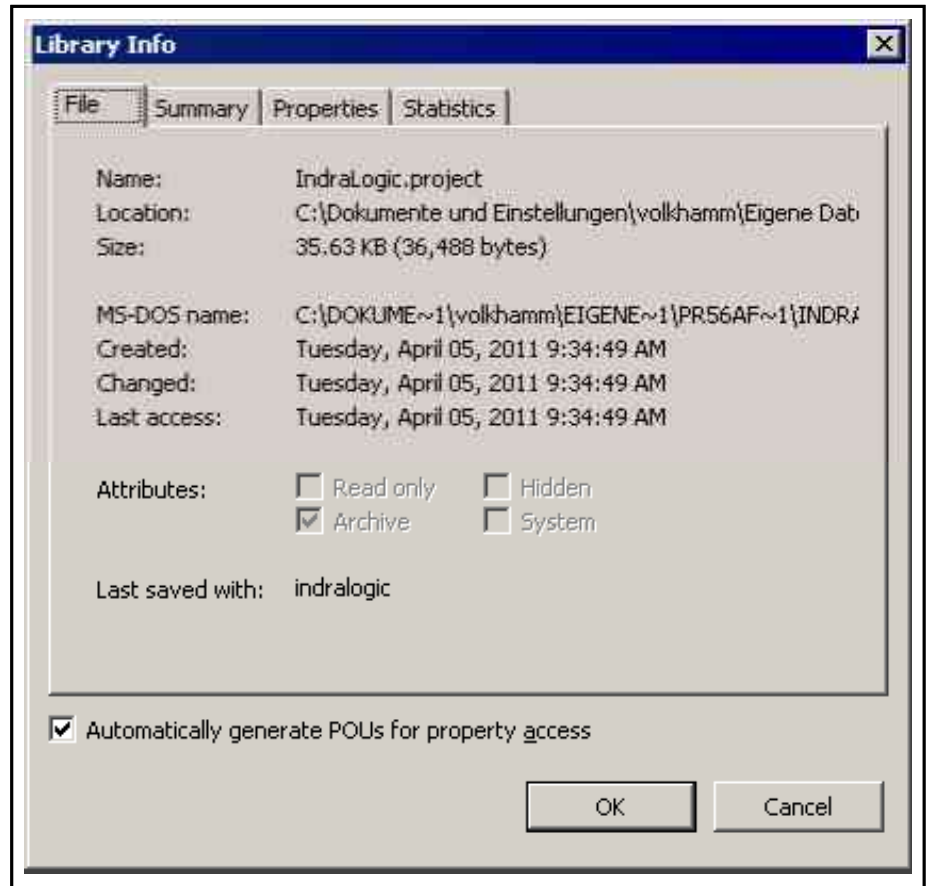


Fig.4-81: "Library Info" dialog, File

The following project file properties are displayed: Name, Location, Size in KB, MS-DOS name, Created, Changed, Last Access and name of the profile with which the file was saved last.

In addition, it can be seen which of the following file attributes are currently set: Read-only, Hidden (i.e. not visible in the Explorer by default), Archive (prepared for archiving), System (system file). By default, these attributes cannot be edited here (see the file attributes in the Windows Explorer).

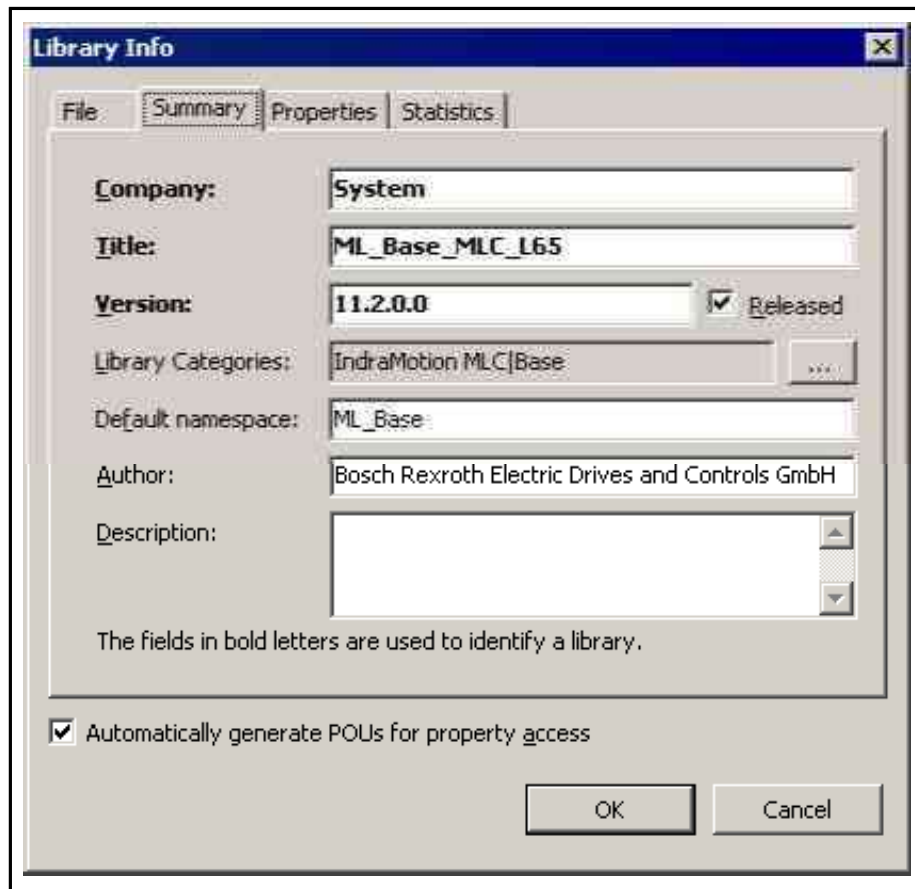


Fig.4-82: "Library Info" dialog, Summary

The following library can be stored here: **Title**, **Version**, a **Default namespace**, the **Author**, the name of the **Company** and a short **Description**. This information is automatically available as a "Key" in the 'Properties' tab (see below).

Note the following for library projects, page 83: If a project is to be used in other projects as a library, at least the following **has to be entered** here: a **Title**, a **Version** number and the **Company** name.

A library file that includes this information can be used on the projects. In addition to category, the company name is used for sorting in the "Library Repository" dialog. As an option, "Default namespace", "Author" and a short "Description" can also be specified to be saved as library project information.


If a **default namespace** is not defined, the name of the library file is automatically the namespace.

Assignment to a category:

Assigning a library to a category is also used for sorting.

If no category is explicitly given in the library information, the library is assigned to the "Other" category. If it should belong to another category, that category has to be defined.

One or multiple external **description files** in XML format are used to define library categories. To assign the library, either one of these files can be called to provide the category or another library file can be called that already includes this information on the categories from a description file.

Use the  button to open the **Library Categories** dialog:

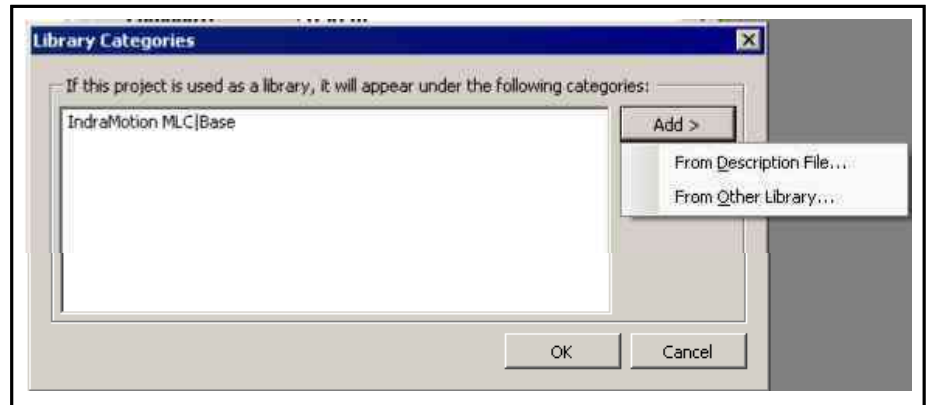


Fig.4-83: Dialog to select library categories

<Add> opens a menu to select whether the category information should be retrieved

- **From Description File...** or
- **From Other Library...**

In both cases, the default dialog for browsing for a file appears. The filter is set to *.libcat.xml or *.library.

The categories from the category description file or the library are listed in the window.

Delete those ones not needed with <Delete>.

Further categories from other sources can be added in the same way. If the list contains all of the desired categories, confirm with <OK> to close the dialog and to enter the categories in the 'Library Categories' field in the 'Library Info' dialog.

Editors

3. Properties

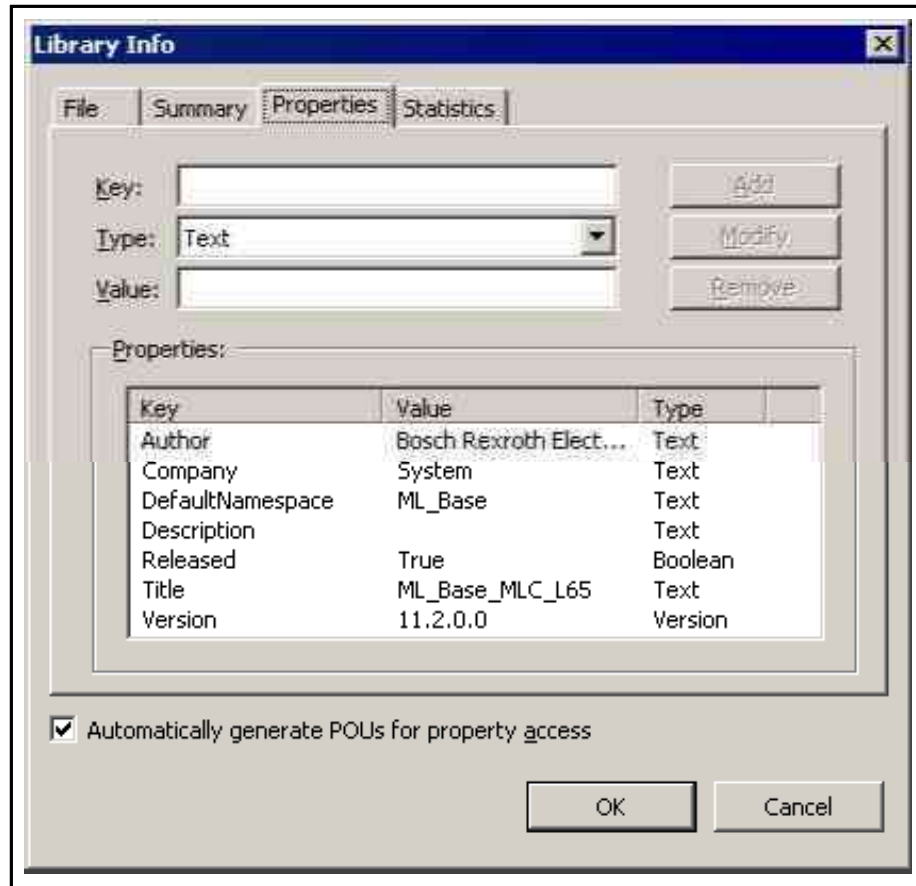


Fig.4-84: "Library Info" dialog, Properties

Define the key for specific file properties here. These can then be used in user-specific external programs to externally control the respective properties.

tab is provided as "key". The property names are used as key names, "Text" is automatically set as the data type and the "Values" consist of the texts entered in the "Summary" tab. However, other keys can be added explicitly (see the following).

Adding a key: In the **Key** field, enter a name and select the desired data type from the **Type**: list (possible data types: text, date, number, true/false, version). In the **Value**: field, enter the desired value which has to be compatible with the data type. Use the **Add** button to apply the new key to the **Properties** list.

Editing a key: Select the key in the "Properties" list from the selections in the "Key" column and modify the key attributes as desired in the input fields above the list. Then use the **Modify** button to apply the changes to the "Properties" list and, for the keys affected, **to the "Summary" (!) tab as well.**

Removing a key: Select the key in the "Properties" list from the selections in the "Key" column and click on **Remove.**

4. Statistics:

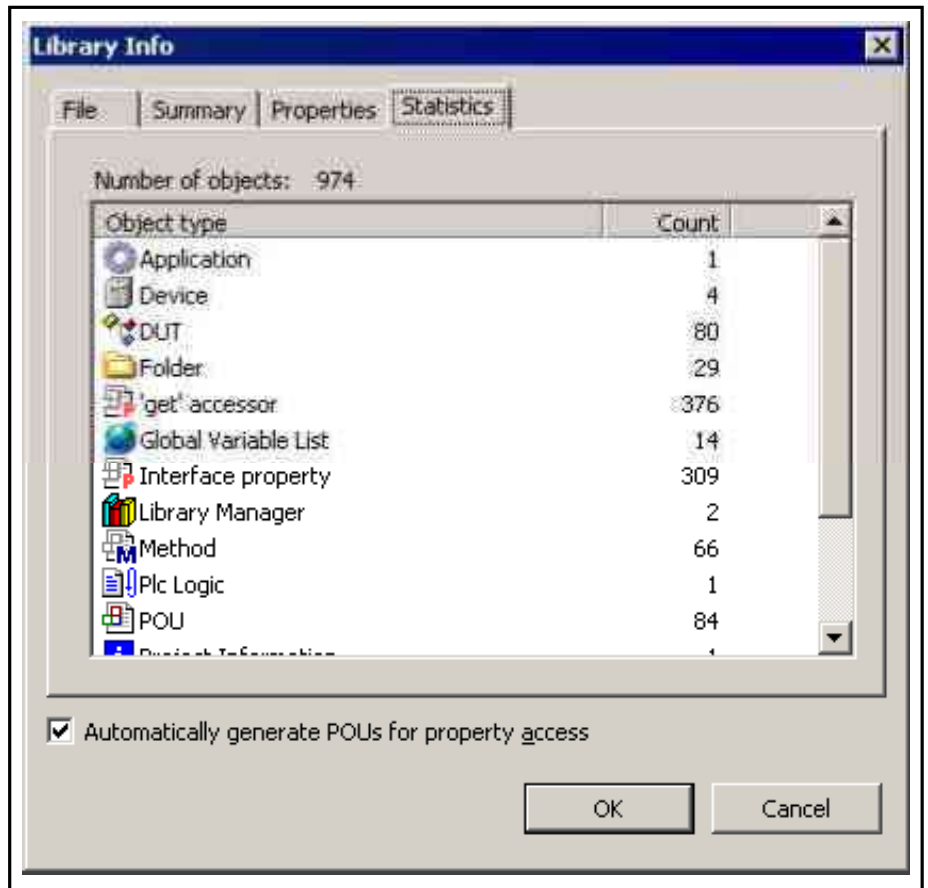


Fig.4-85: "Library Info" dialog, Statistics

This dialog shows the **number of objects** in total that can be used per **object type** (Number).

4.8.5 Library Info and Access Functions

Library Info and Access Functions, General Information

Information is saved in the "Library Info Object" of the library when creating a library.



As each library contains this object, the **namespace** followed by a "." has to be specified for differentiation purposes.

The user can access part of this information via automatically generated functions.

As example library, the BASELIB, namespace ML_Base is used in the characteristic ml_base_mlc_l65.

The following functions might be interesting for the user:

Functions without input parameters

-
-
-

Functions with input parameters (key)

-
-

378/697 Bosch Rexroth AG

DOK-IWORKS-IL2GPRO*V12-AP01-EN-P
Rexroth IndraWorks 12VRS IndraLogic 2G PLC Programming System

Editors

- and
-

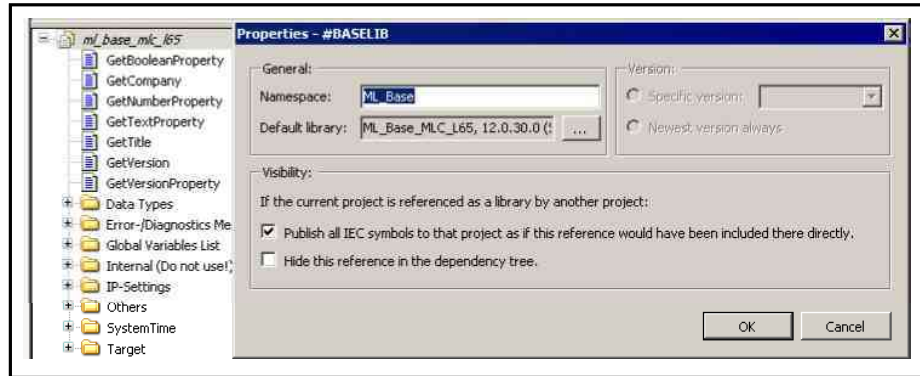


Fig.4-86: Access functions of the ML_Base library

GetCompany

Brief description The **GetCompany** function responses with the group affiliation of the library.

Library	Range
ML_Base	

Fig.4-87: Library assignment

Interface description

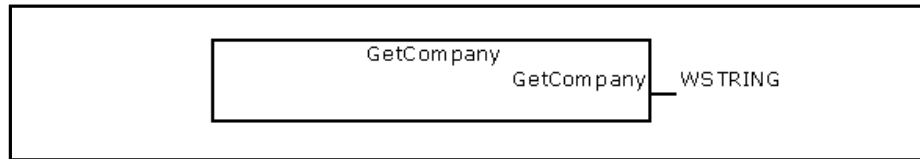


Fig.4-88: "GetCompany" function

	Name	Type	Comment
Function value	GetCompany	WSTRING	Group affiliation of the library, e.g. "system"

Fig.4-89: "FUN GetCompany" interface

Implementation example:

Example in ST:

```
wsText := ML_Base.GetCompany();
```

GetTitle

Brief description The **GetTitle** function replies with the library title.

Library	Range
ML_Base	

Fig.4-90: Library assignment

Interface description

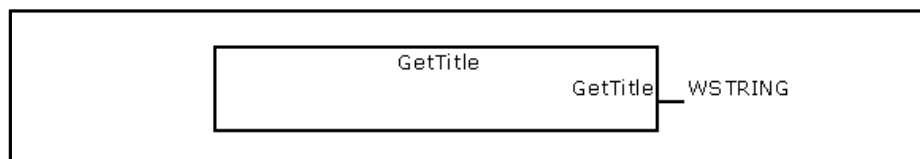


Fig.4-91: "GetTitle" function

	Name	Type	Comment
Function value	GetTitle	WSTRING	Library title

Implementation example: *Fig.4-92: "FUN GetTitle" interface*
Example in ST:

```
wsText := ML_Base.GetTitle();
```

GetVersion

Brief description The **GetVersion** function replies with the library version.

Library	Range
ML_Base	

Interface description *Fig.4-93: Library assignment*

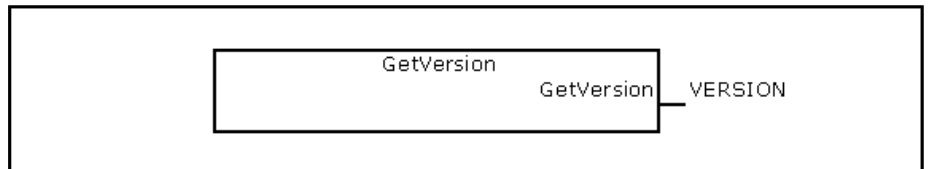


Fig.4-94: "GetVersion" function

	Name	Type	Comment
Function value	GetTitle	VERSION	Exact library version

Implementation example: *Fig.4-95: "FUN GetVersion" interface*
Libraries also develop further.
This information ensures that a certain functionality is available.
This function can thus replace the version function in IndraLogic 1.x.
Example in ST:

```
VAR
uVersion: VERSION;
END_VAR

uVersion := ML_Base.GetVersion();
```



VERSION is a system-individual structure to which the

- major build number,
- minor build number,
- service pack number and a
- patch number

of the library are assigned.

GetBooleanProperty

The "GetBooleanProperty" function outputs "Released" TRUE with the key. Otherwise, it outputs FALSE.

Library	Range
ML_Base	

Fig.4-96: Library assignment

Editors

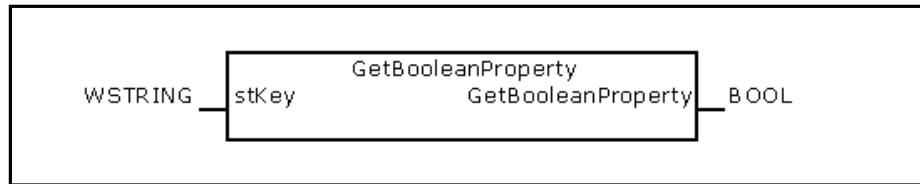


Fig.4-97: "GetBooleanProperty" function

	Name	Type	Comment
VAR_INPUT	stKey	WSTRING	Key of information to be read
Function value		BOOL	Desired information

Fig.4-98: "FUN GetBooleanProperty" interface

GetNumberProperty

Irrespective of the key, the "GetNumberProperty" function outputs "0".

Library	Range
ML_Base	

Fig.4-99: Library assignment

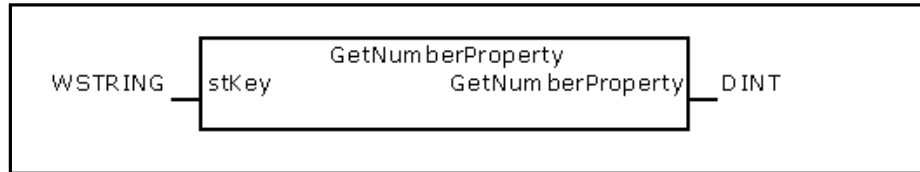


Fig.4-100: "GetNumberProperty" function

	Name	Type	Comment
VAR_INPUT	stKey	WSTRING	Key of information to be read
Function value		DINT	Always "0"

Fig.4-101: "FUN GetNumberProperty" interface

GetVersionProperty

Brief description The "GetVersionProperty" function reads the version number from the currently available library.

Library	Range
ML_Base	

Fig.4-102: Library assignment

Interface description

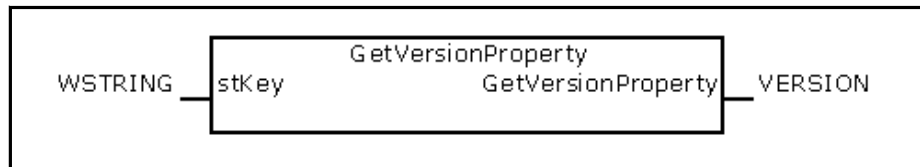


Fig.4-103: "GetVersionProperty" function

	Name	Type	Comment
VAR_INPUT	stKey	WSTRING	Key of information to be read
Function value		VERSION	Desired information

Fig.4-104: "FUN GetTextProperty" interface

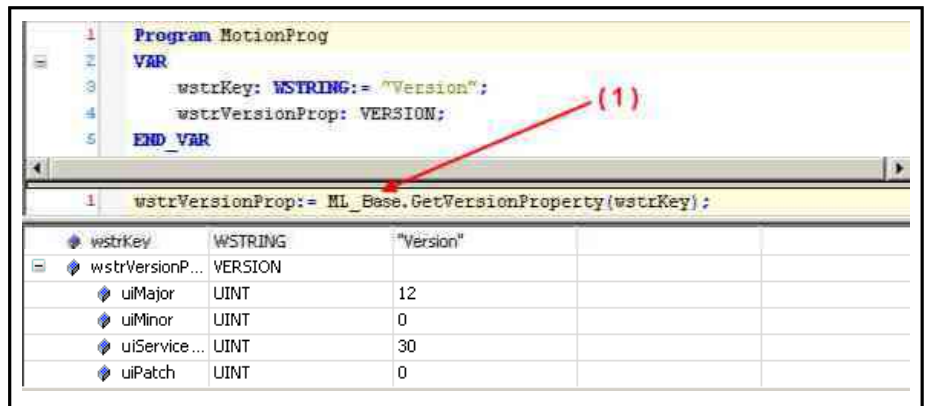


VERSION is a system-individual structure to which the

- major build number,
 - minor build number,
 - service pack number and a
 - patch number
- of the library are assigned.

Implementation example:

Libraries also develop further.
This information ensures that a certain functionality is available.
This function can thus replace the version function in IndraLogic 1.x.



(1) Namespace
Fig.4-105: Implementation example in ST

GetTextProperty

Brief description

The "GetTextProperty" function reads the subsequent information from the currently available library.

- Author
- Title
- DefaultNamespace
- Company

Library	Range
ML_Base	

Fig.4-106: Library assignment

Interface description

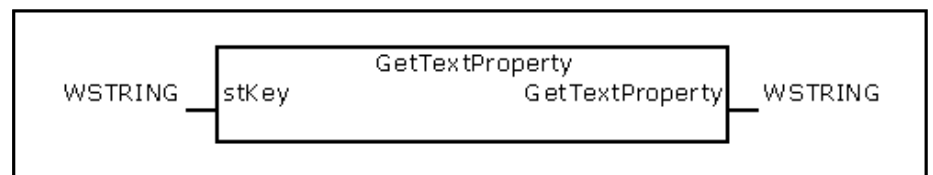


Fig.4-107: "GetTextProperty" function

	Name	Type	Comment
VAR_INPUT	stKey	WSTRING	Key of information to be read
Function value		WSTRING	Desired information

Fig.4-108: "FUN GetTextProperty" interface

Editors

Implementation example: The subsequent table contains the keywords and function values.

Keyword	Function value
Author	Bosch Rexroth Electric Drives and Controls GmbH
Title	ML_Base
DefaultNameSpace	ML_Base
Company	System

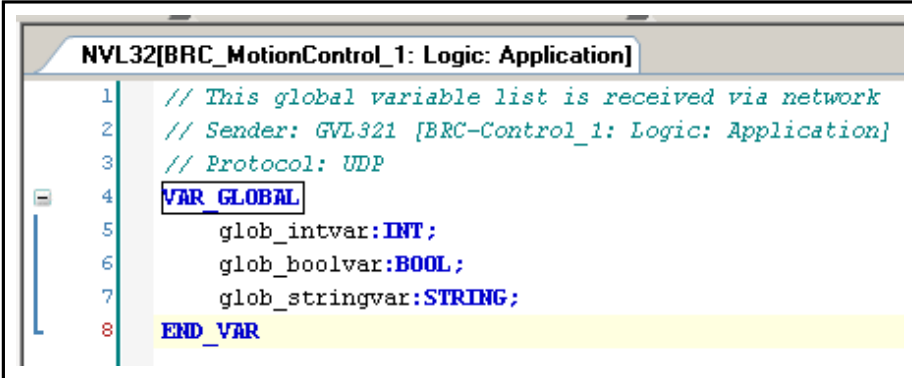
Fig.4-109: Keywords and function values

4.9 Network Variable List Editor

The NVL editor is a declaration editor to create . It works based on the current mode and generally described for the and in online .

The declaration of the network variables have to begin with the keyword "VAR_GLOBAL" and end with "END_VAR". These keywords are automatically specified.

Valid of global variables are inserted in between.



```

1 // This global variable list is received via network
2 // Sender: GVL321 [BRC-Control_1: Logic: Application]
3 // Protocol: UDP
4 VAR_GLOBAL
5     glob_intvar:INT;
6     glob_boolvar:BOOL;
7     glob_stringvar:STRING;
8 END_VAR
    
```

Fig.4-110: NVL editor

4.10 Recipe Manager

4.10.1 Recipe Manager, General Information

The **Recipe manager** provides functions to manage user-defined variable lists called **recipe definitions**.

A recipe definition is a table with a variety of columns (**Recipes**) and lines(**Variables**) in which value sets can be defined for the variables. Use such recipes to set and monitor the variables on the control.

Recipe definitions can be read out of the control and described.

Recipe definitions can be saved in files and loaded again from the files. These actions can be taken using visualization elements that have to be configured accordingly.

Editors



If the recipe manager is located on another control than the application to which the recipes are applied, the data server is used to write or read the recipes.

Reading and writing take place synchronously in this case, i.e. all variables in a recipe definition are updated at the same time.

Call `g_RecipeManager.LastError` after reading and writing are completed to check whether the transfer was successful (`g_RecipeManager.LastError=0` in this case).

The recipe manager is added in the Project Explorer below an application.

To do this, highlight the application node and select **Add ► Recipe manager** in the context menu.

The "Recipe manager" object is also available in the "PLC objects" library in the "VI logic objects" folder.

When creating the recipe manager, the following dialog opens which contains the initial settings for the recipe manager.

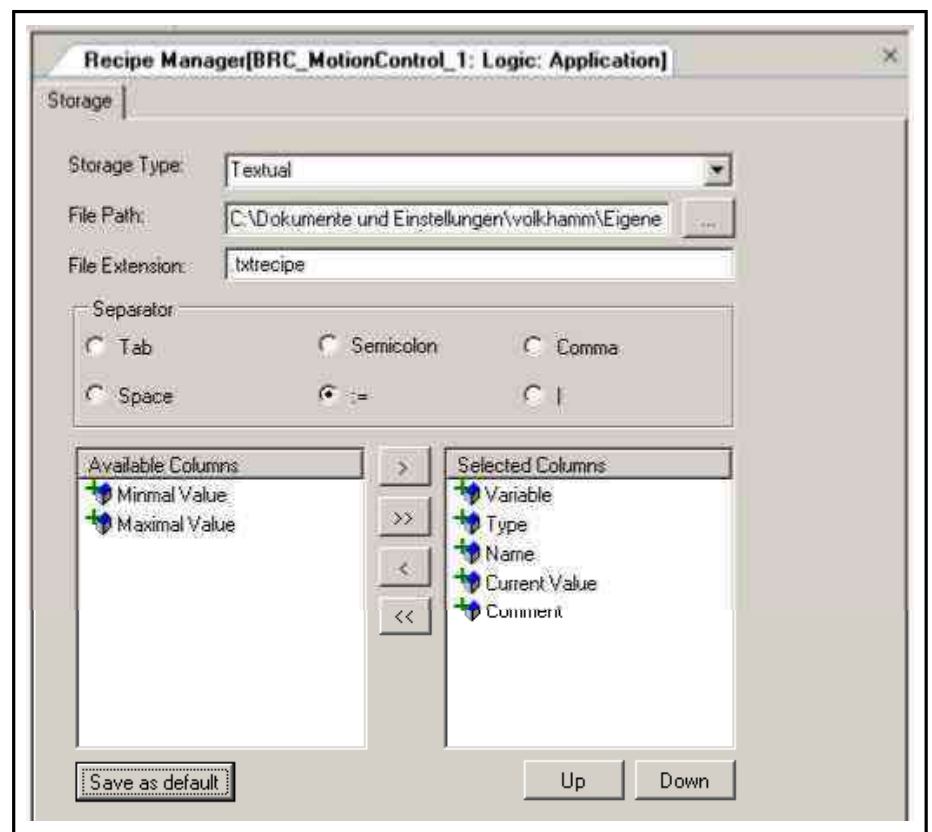


Fig.4-111: Initial settings of the recipe manager

Storage Type: Recipe definitions can be stored as text or binary information.

File Path: Storage location; can be selected.

Separator: Between the stored data elements; can be selected.

Available / Selected Columns: The desired recipes can be selected from the available columns (recipes). The columns can be changed in their sequence.

The initial settings can be saved as "defaults".

Multiple recipe definitions can be added below the "Recipe Manager" object.

Editors

A recipe definition along with the recipes internally defined is displayed as table in the editor and it can be edited there.

4.10.2 Recipe Definition

The **recipe manager** manages one or multiple recipe definitions.

A **recipe definition** contains a user-defined list of variables and one or multiple **recipes** (value sets) for these variables.

By using different recipes, one set of variables on the control can be assigned in a single action using another set of values.

Recipe definition

A recipe definition must be added as object below the "Recipe Manager" object in the Project Explorer. To do this, highlight the "Recipe Manager" object and select **Add ▶ Recipe definition...** in the context menu.

The "Recipe Definition..." object is also available in the "PLC objects" library in the "VI logic objects" folder.

When a recipe definition object is highlighted, the related editor can be opened by double-clicking on the object.

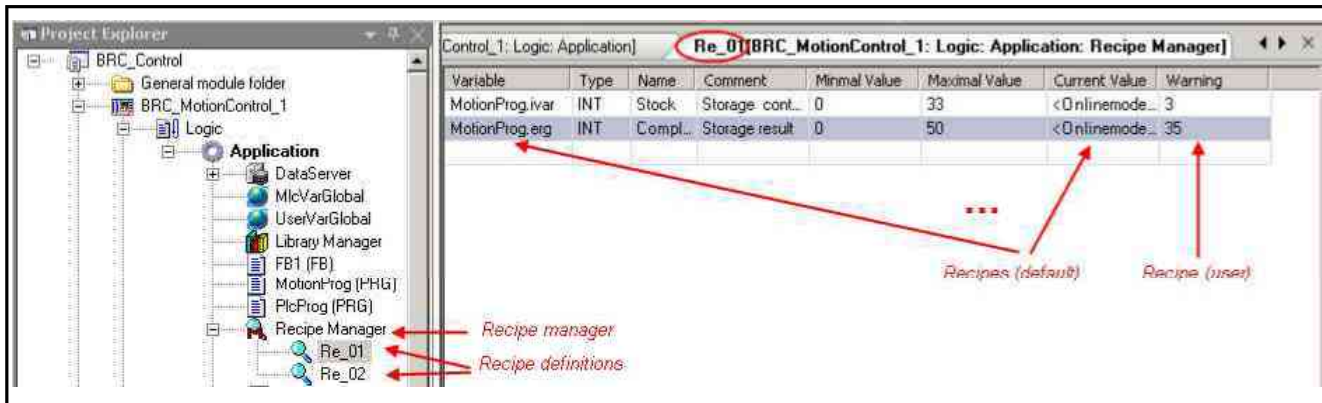


Fig.4-112: Recipe definition, editor window

The title bar of the editor window includes the name of the recipe definition.

Multiple project variables for which one or multiple recipes are to be created can be entered in a table.

At first, the editor contains only one empty line.

Variable	Enter the path of a project variable, e.g. "MotionProg.ivar", into the field in the "Variable" column. In editing mode, double-click to access the field. The input assistance can also be accessed with the button.
Type	Variable data type, entered automatically.
Name	A symbolic name can be added to the variable (optional).
Minimal Value	Minimum value range that may be written on the variable.
Maximal Value	Maximum value range that may be written on the variable.
Current Value	Current value in online mode.
Recipe name	For each recipe within the recipe definition, there is a table column titled with the recipe name.

To add another line at the end of the list for a variable entry, use the menu option **VI logic recipe definition ▶ Add variable** in the main menu. To delete



one or multiple highlighted lines, use the menu option **VI Logic recipe definition ▶ Remove variables**. Alternatively, execute both menu options via the context menu.



For more information on the recipe commands, see

Recipes

Create or remove a recipe in offline mode.

- To create a recipe, click on **VI Logic recipe definition ▶ Add recipe** in the main menu.
- To remove a recipe, click on **VI Logic recipe definition ▶ Remove recipe** in the main menu.

Another column is then added at the right end of the table, titled with the . The fields in the recipe column can then be filled with variable values. This way, several value sets can be prepared for the same variables.

In online mode, the recipes can be managed via "Visualization elements" (creating, reading, writing, saving in a file, loading from a file); see

The following actions can be taken with regard to a recipe:

Create recipe (=Add recipe)	A new recipe is created in the indicated recipe definition.
Read recipe	The current values of the variables in the indicated recipe definition are read by the control and written into the indicated recipe. During this procedure, the values are saved implicitly (in a file on the control) while they are displayed in the recipe definition table in the IndraLogic recipe manager. The recipe managed in IndraLogic is updated with the values from the control.
Write recipe	The values of the indicated recipe - as stored in the recipe manager - are written on the corresponding variables in the control.
Save recipe	The values of the indicated recipe are saved in a file with the extension "*.txtrecipe". The file name has to be defined. To do this, the default dialog to save a file opens. ATTENTION: The recipe files used implicitly for clipboard functions while reading and writing may not be overwritten. That means that the new file has be named different from <Recipe name>.<Recipe definition name>.txtrecipe!
Load recipe	The recipe saved in a file (see "Save recipe" above) can be loaded from this file again. The default file selection dialog opens to select the file. The filter is automatically set to include the file extension "*.txtrecipe". After it is loaded, the display of the respective recipe is updated in the IndraLogic recipe manager.
Delete recipe (=remove recipe)	The indicated recipe is deleted.

4.10.3 Recipe Manager in Online Mode

In preparation

Editors

4.11 ST Editor

4.11.1 ST Editor, Overview

The ST editor is used to program objects in the IEC Structured Text programming language (ST) or Extended Structured Text which provides additional functions with regard to the IEC 61131-3 standard.

The ST editor is a text editor.

It works based on the current settings in the Colors, line numbering, tab widths, indents, etc. can be set there.

Note that **function block selection** in texts is possible by pressing <Alt> while selecting the desired section of text with the mouse.

The ST editor opens in the lower section of the editor window which also contains the in the upper section.

Note that if syntax errors are made while editing. Corresponding messages are output in the message window. This window is always refreshed when the editor window gets the input focus again (e.g. if the cursor is placed in another window and then moved back in the ST editor).

4.11.2 ST Editor in Online Mode

In online mode, the editor for Structured Text (ST editor) provides views for , i.e. displays and for and values onto variables and expressions.

(breakpoints, single step processing, etc.) are available. See

- regarding how a prepared value can be input for variables in online mode.
- Note that the editor window for an ST object also contains the declaration editor in the upper section.

See also online mode in the

Monitoring If the **Inline monitoring** function is not explicitly disabled (), the current variable value is displayed in a small window after the variable.

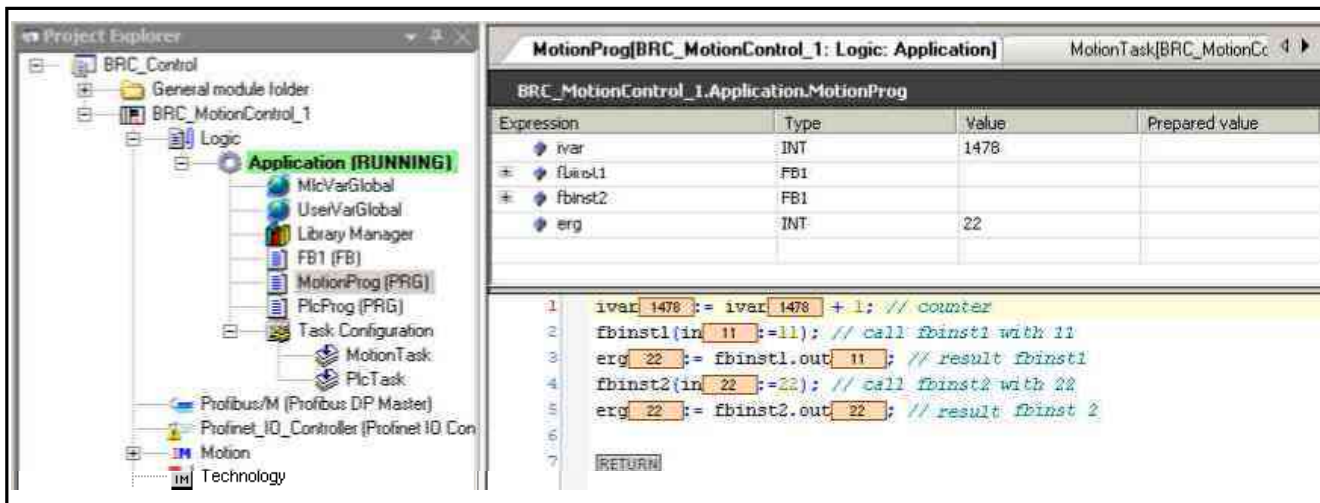


Fig.4-113: Example: Online view of a MotionProg program with Inline monitoring

Online view of a function block: Monitoring is only possible in the view of an instance. No values are displayed in the view of the basic implementation. Here, the "Value" column

Editors

contains the text "<The value of the expression cannot be read>" and three question marks appear in the respective Inline monitoring fields in the implementation section.

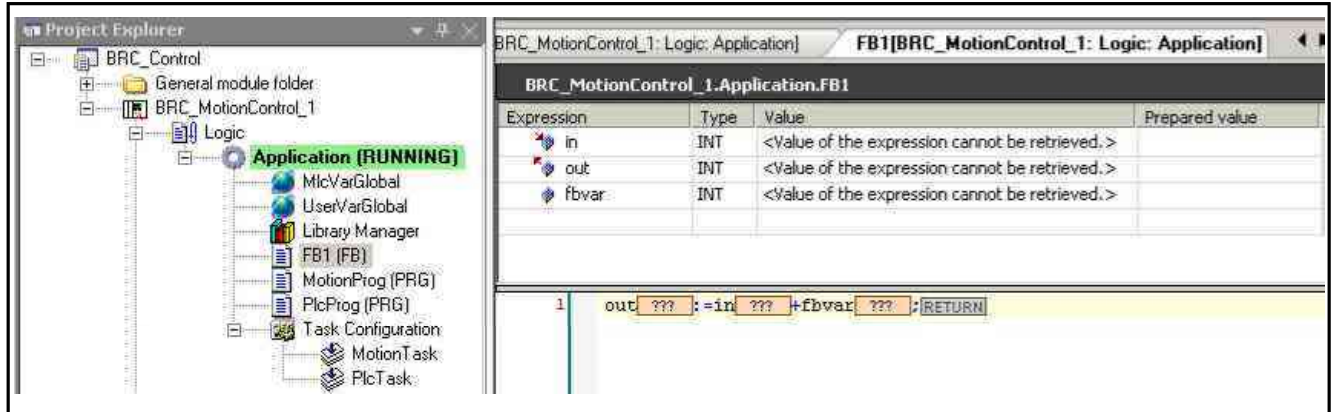


Fig.4-114: Example: Online view of the implementation of the function block FB1

Forcing variables

In addition to a prepared value for a variable in the declaration section in each editor, there is the option in the ST editor in online mode to click in the implementation section (instruction) on the **monitoring box** of a variable which opens a dialog in which the prepared value can be entered.

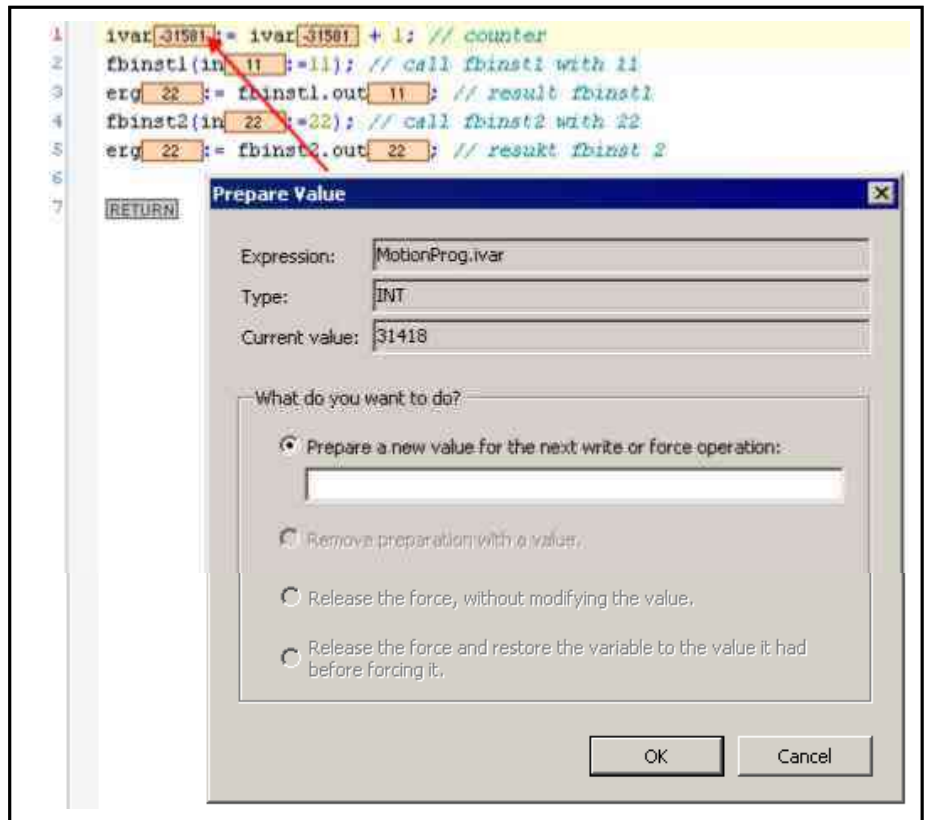


Fig.4-115: Dialog - Prepare Value

The path, data type and current value from the variable are shown. By enabling the corresponding item, decide whether

- a new value should be prepared that has to be entered into the editing field
- a prepared value should be removed

Editors

- a currently forced variable should be released
- a currently forced variable should be released and its original value restored before it is forced

The selected action is executed after the "Force values" command is called. To do this, click on **Debug ▶ Force values** in the main menu.

Breakpoint positions in the ST editor:

at the positions in a function block at which a variable value can change or where the program sequence branches or another function block is called.

In the following descriptions, "{BP}" indicates a possible breakpoint position:

- **Assignment:** At the beginning of a line.
Note: If you use `:=` there is are further breakpoint positions within a line.
- **FOR loop:** 1. Before the initialization of the counter. 2. Before the counter is checked. 3. Before an instruction.
{BP} FOR i := 12 TO {BP} x {BP} BY 1 DO
{BP} [statement1]
...
{BP} [statementn-2]
END_FOR
- **WHILE loop:** 1. Before testing the condition. 2. Before an instruction.
{BP} WHILE i < 12 DO
{BP} [statement1]
...
{BP} [statementn-1]
END_WHILE
- **REPEAT loop:** Before testing the condition.
REPEAT
{BP} [statement1]
...
{BP} [statementn-1]
{BP} UNTIL i >= 12
END_REPEAT
- **Calling a program or function block:** At the beginning of the line.
- **There is always a breakpoint position at the end of a function block.** At incremental processing (step-by-step), this position is reached after a RETURN instruction.

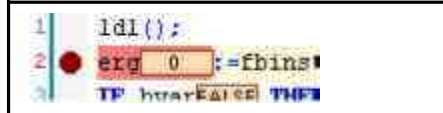
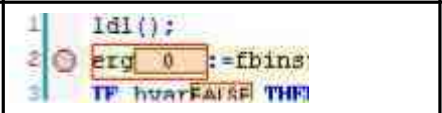
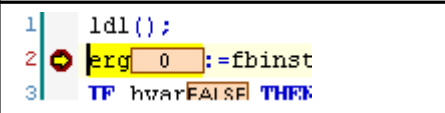
Breakpoint in online mode:	Disabled breakpoint:	Program stop at a breakpoint:
		

Fig.4-116: Breakpoint representation in ST



A breakpoint is automatically set in all methods that can be called. Thus, the following applies: If a method defined by an interface is called, breakpoints are set in all methods of function blocks that implement this interface and in all function blocks derived that define the method.

4.11.3 Structured Text ST/ExST

Structured Text ST/Extended Structured Text ExST

Structured text is a programming language that is comparable with other high level languages such as C or PASCAL which allows complex algorithms to be developed.

The program code consists of a combination of **IF..THEN..ELSE** and **WHILE..DO** that can be executed conditionally (IF..THEN..ELSE) or in loops (WHILE..DO).

Example:

```
IF value < 7 THEN
  WHILE value < 8 DO
    value:=value+1;
  END_WHILE;
END_IF;
```

"Extended Structured Text (ExST)" is an IndraLogic-specific extension with regard to the IEC 61131-3 standard for Structured Text (ST).

Examples:

-
-

Expressions

An "expression" is a construct that returns a value after being evaluated.

Expressions consist of **constants** and **variables**. **Function calls** can also be used as expressions.

An operand can be a constant, a variable, a function call or another expression.

Examples:

```
33 // Constant
ivar // Variable
fct(a,b,c) // Function call
a AND b // Expression
(x*y) / z // Expression
real_var2 := int_var; // Assignment, see below
```

Evaluation of expressions

The evaluation of an expression is carried out by processing the operators according to specific rules for the order of operation (priority of tasks). The operator with the highest order is processed first, the operator with the next highest order, etc., until all operators have been processed.

Operators with equal priority are processed from left to right.

In the following table, the ST operators are listed in order of their priority:

Operation	Symbol	Priority
Bracketing	(Expression)	Highest priority



Editors

Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negation	-	
Complementation	NOT	
Multiplication	*	
Division	/	
Modulo	MOD	
Addition	+	
Subtraction	-	
Comparisons	<,>,<=,>=	
Equality	=	
Inequality	<>	
Bool AND	AND	
Bool XOR	XOR	
Bool OR	OR	Lowest priority

Assignment as expression As an extension with regard to the IEC 61131-3(ExST) standard, IndraLogic allows the usage of assignments as expressions.

Examples:

```
int_var1 := int_var2 := int_var3 + 9; // int_var1, int_var2 will get the value of (int_var3 + 9)
real_var1:= real_var2 := int_var;    // correct assignments,
int_var := real_var1 := int_var;     // real_var1, real_var2 will get the value of int_var
// this will lead to an error, data type mismatch!

IF b := (i= 1) THEN
  i := i + 1;
END_IF;
```

Instructions

Control the instructions on how the given are to be processed. The following instructions can be used in ST:

Instruction	Example
Assignment ()	A:=B; CV := CV + 1; C:=SIN(X);
and using the function block output	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q;
	RETURN;



IF, page 393	D:=B*B; IF D<0.0 THEN C:=A; ELSE D=0.0 THEN C:=B; ELSE C:=D; END_IF;
	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;
	J:=101; FOR I:=1 TO 100 BY 2 DO IF ARR[I] = 70 THEN J:=I; EXIT; END_IF; END_FOR;
	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
	EXIT;
	CONTINUE;
	label: i:=i+1;
	JMP label;
Empty instruction	;

Fig.4-117:

Assignment Operators

On the left side of an assignment is an operand (variable, address) to which the value of the expression on the right side is assigned using the assignment operator :=.

392/697

Bosch Rexroth AG

DOK-IWORKS-IL2GPRO*V12-AP01-EN-P

Rexroth IndraWorks 12VRS IndraLogic 2G PLC Programming System

Editors

Also refer to

- with the same effect.

Example:

```
Var1 := Var2 * 10;
```

After this line is executed, "Var1" is equal to ten times "Var2".

Note the following functionalities which are new compared to IndraLogic V1.x:

Further assignment operators that are not described in the 61131-3 standard (ExST):

Set operator "S=": The value is "set". If it was once TRUE, it remains TRUE.

Program:

```
a S= b;
```

"a" receives the value from "b". If it was once set to TRUE, it remains TRUE even if "b" becomes FALSE again.

Reset operator "R=": The value is "reset". If it was once FALSE, it remains FALSE.

Program:

```
a R= b;
```

"a" receives the value from "b". If "b" was once set to FALSE, it remains FALSE irrespective of the value of "a".



Note this behavior in case of multiple assignments. All "Set" and "Reset" assignments always refer to the **last** element of the assignment.

Example:

```
a S= b R= fun1(par1,par2);
```

In this case, "b" receives the output that results when "fun1" is "reset",

BUT: "a" does **not** receive the "Set" result from "b", but the "Set" result from "fun1" instead.

Assignment as expression, extension of the IEC 61131-3 standard (ExST):

Note that an assignment can be used as an

Calling a Function Block in ST

A (abbreviated as "FB") is called in ST based on the following syntax:

Syntax:

```
<FB instance name>(FB input variable:=<value or address>|, <further FB input variable:=<value or address>|...further FB input variables);
```

Example:

In the following example, a timer FB (TON) with assignments for the parameters IN and PT is called.

Afterwards, the output variable Q is assigned to variable A.



Editors

The timer FB is instantiated in "TMR:TON;"

The output variable is addressed based on the syntax "<FB instance>.<FB variable>"

Program:

```
TMR(IN := %IX5, PT := 300);
A:=TMR.Q
```

Detailed information:

- ,
- .

RETURN Instruction

The RETURN instruction can be used to exit a function block, e.g. dependent on a condition.

Syntax:

```
RETURN;
```

Example:

```
IF b:=TRUE THEN
RETURN;
END_IF;
a:=a+1;
```

IF Instruction

If the value of "b" is TRUE, instruction a:=a+1 is not executed, but the function block is exited immediately instead.

A condition can be tested with an IF instruction and, depending on this condition, instructions can be executed.

Syntax:

```
IF <Boolean_expression1> THEN
<IF_instructions>
{ELSIF <Boolean_expression2> THEN
<ELSIF_instructions1>

ELSIF <Boolean_expression n> THEN
<ELSIF_instructions-1>
ELSE
<ELSE_instructions>}
END_IF;
```

The section in curly brackets {} is optional.

If <Boolean_expression1> returns TRUE, only the <IF_Instructions> are executed and none of the other instructions.

Otherwise, the Boolean expressions that begin with <Boolean_expression2> are tested successively until an expression returns TRUE. Then, all instructions between this expression and the next ELSE or ELSIF instruction is evaluated and executed accordingly.

If none of the Boolean expressions returns TRUE, only the <ELSE_instructions> are evaluated.

Example:

```
IF temp<17
THEN heating_on := TRUE;
```



Editors

```
ELSE heating_on := FALSE;
END_IF;
```

Here the heating is switched on if the temperature falls below 17 degrees. Otherwise, it remains switched off.

CASE Instruction

A CASE instruction can be used to combine several conditional instructions with the same condition variables in one construct.

Syntax:

```
CASE <Var1> OF
<value1>: <Instruction 1>
<value2>: <Instruction 2>
<value3, value4, value5>: <Instruction 3>
<value6 .. value10>: <Instruction4>
...
<value n>: <Instruction n>
ELSE <ELSE Instruction>
END_CASE;
```

A CASE instruction is processed according to the following scheme:

- If the variable <Var1> has the value <value i> hat, the instruction <instruction i> is executed.
- If <Var1> does not have any of the specified values, the <ELSE instruction> is executed.
- If the same instruction is to be executed for multiple variable values, write these values separated by commas and thus develop a common instruction.
- If the same instruction is to be executed for the complete value range of the variables, write the starting and end values separated by two dots to develop the common instruction.

Example:

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
   BOOL3 := FALSE;
2:   BOOL2 := FALSE;
   BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
   BOOL3:= TRUE;
ELSE
   BOOL1 := NOT BOOL1;
   BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR Loop

Repeating procedures can be programmed with the FOR loop.

Syntax:

```
VAR INT_Var :INT; END_VAR

FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO
<instructions>
END_FOR;
```

The section in curly brackets {} is optional.

- The <instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before the <instructions> are executed so that the <instructions> are never executed if the starting value <INIT_VALUE> is greater than the end value <END_VALUE>.



Editors

- When <instructions> is executed, <INT_Var> always increases by <stepSize>.
- The step size can have any integer value.
- If this specification is missing, the step size is set to the default value "1". This ensures that the loop comes to an end at some point, since <INT_Var> can only become higher.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
  Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Assuming that variable "Var1" was preset with a value of "1", it assumes the value "32" after the FOR loop.



If <END_VALUE> equals the limit value of the counter <INT_VAR>, e.g. if the counter - as used in the example above - is of type SINT and if <END_VALUE> equals 127, an endless loop results.

For this reason, <END_VALUE> may not receive a value that is equal to the limit value of the counter!

Extension with regard to the IEC 61131-3 standard (ExST):

WHILE Loop

The can be used in a FOR loop.

The WHILE loop can be used like the FOR loop, except that the abort condition can be any Boolean expression. That means that when the condition entered is met, the loop is executed.

Syntax:

```
WHILE <boolean expression> DO
  <instructions>
END_WHILE;
```

- The <Instructions> are executed repeatedly as long as the <Boolean expression> returns TRUE.
- If the <Boolean expression> is already FALSE at the first evaluation, the <instructions> are never executed.
- If the <Boolean expression> never assumes the value FALSE, the <instructions> are repeated endlessly causing a runtime error.



The programmer has to ensure that no endless loops are caused by modifying the condition in the instruction part of the loop, e.g. increasing or decreasing a counter.

Example:

```
WHILE counter<>0 DO
  Var1 := Var1*2;
  Counter := Counter-1;
END_WHILE;
```

In a sense, the WHILE and REPEAT loops are more powerful than the FOR loop, since the number of loop cycles does not have to be known before the loop is executed.

In some cases, you will only be able to work with these two types of loops.

Editors

However, if the number of loop cycles is known, a FOR loop is preferred, since it does not cause endless loops.

Extension with regard to the IEC 61131-3 standard (ExST):

The `REPEAT` instruction can be used in a WHILE loop.

REPEAT Loop

The REPEAT loop differs from the WHILE loop in the abort condition being checked only after the loop has been executed. As a result, this loop completely runs at least once irrespective of the abort condition.

Syntax:

```
REPEAT
  <instructions>
UNTIL <Boolean expression>
END_REPEAT;
```

- The <Instructions> are executed as long as the <Boolean expression> returns TRUE.
- If the <Boolean expression> is already TRUE at the first evaluation, the <instructions> are executed once.
- If the <Boolean expression> never assumes the value TRUE, the <instructions> are repeated endlessly causing a runtime error.



The programmer has to ensure that no endless loops are caused by modifying the condition in the instruction part of the loop, e.g. increasing or decreasing a counter.

Example:

```
REPEAT
  Var1 := Var1*2;
  Counter := Counter-1;
UNTIL
  Counter=0
END_REPEAT;
```

Extension with regard to the IEC 61131-3 standard (ExST):

The `REPEAT` instruction can be used in a REPEAT loop.

CONTINUE Instruction

As an extension with regard to the IEC 61131-3 standard (ExST) the CONTINUE instruction is supported within `FOR`, `WHILE` and `REPEAT` loops.

CONTINUE causes the execution to jump to the beginning of the next loop cycle.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
  INT1:=INT1/2;
  IF INT1=0 THEN
    CONTINUE; // to avoid division by zero
  END_IF
  Var1:=Var1/INT1; // only executed, if INT1 is not "0"
END_FOR;
Erg:=Var1;
```

EXIT Instruction

If the EXIT instruction appears in a `FOR`, `WHILE` or `REPEAT` loop, the innermost loop is ended irrespective of the abort condition.

JMP Instruction

Editors

The JMP instruction can be used for an unconditional jump to a program line marked by a jump label.

Syntax:

```
<label>:  
...  
JMP <label>;
```

The <jump label> is any unique identifier positioned at the beginning of a program line.

The jump instruction JMP is followed by a jump target input which has to match the name of a defined jump label.

When the JMP instruction is reached, there is a return jump to the program line with label that matches the jump target.



The programmer has to ensure that no endless loops are caused by modifying the jump, e.g. by linking it to a condition.

Example:

```
i:=0;  
llabel: i:=i+1;  
IF (i<10) THEN  
JMP llabel;  
END_IF;
```

As long as the variable i initialized with 0 has a value lower than 10, the conditional jump instruction in the example above causes a repeated return jump to the program line with the jump label and thus, repeated processing of the instructions between the label and the jump instruction. Since these also contain the incrementation of the variable i, it is ensured that i violates the jump condition (exactly in the ninth query) and continues in the program sequence.

The same functionality can be achieved by a WHILE or REPEAT loop in the example. In general, jump instructions can and should be avoided, since they decrease the readability of the code.

Comments in ST

Comments can be placed anywhere in the declaration editor or ST editor.

The following options are available to insert comments in Structured Text.

1. A comment begins with **(*** and ends with ***)**. This allows to insert comments with a length over several lines.

Example:

```
(*This is a comment.*)
```

2. One line comment (extension with regard to the 61131-3 standard): The comment starts with **//** and ends at the end of the line.

Example:

```
// This is a comment
```

3. **Nested comments:** Comments can be inserted in other comments.

Example:

```
(*  
a:=inst.out; (* to be checked *)  
b:=b+1; // increment  
*)
```

Editors

In this example, the comment that begins with the first open parenthesis does not end with the closing parenthesis after "checked", but with the parenthesis in the last line instead.

4.12 Symbol Configuration Editor

The symbol configuration can create symbol descriptions for project variables used to address these variables ("items") an external location, e.g. an OPC server.

Refer also to the

The symbols for an application can be configured in the **symbol configuration editor** that opens by double-clicking on the symbol configuration object in the "Devices" window.

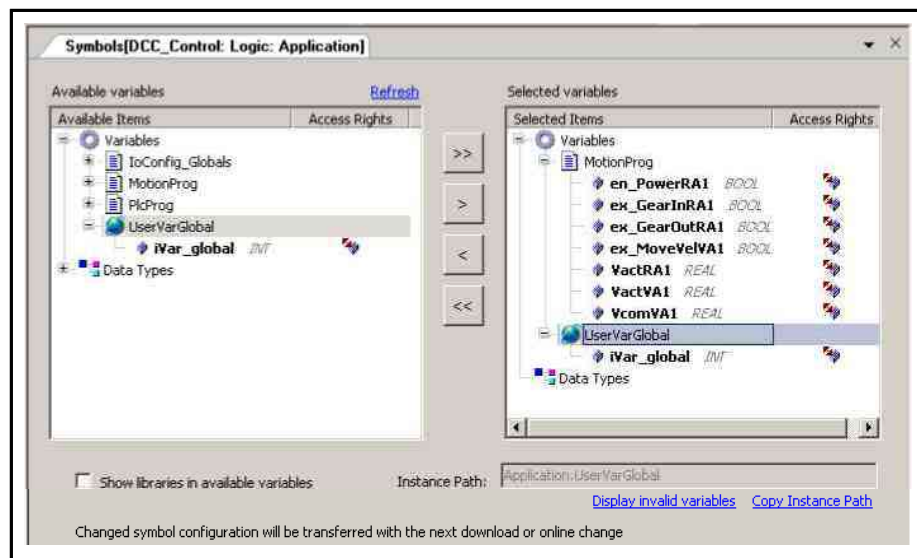


Fig.4-118: Symbol configuration editor

In the left section of the editor, the currently **available variables** of the application are displayed in a tree structure with one node for variables and one node for data types.

Indented below this node are the names of the blocks, global variable lists (GVL) and data types, and in turn, below these are the names, data types and access rights for the individual variables available here ("Items"). Use the <Refresh> button to update the tree with the currently valid variables.



Under "Available variables" only those variables used in the program are displayed.

POU variables are only visible if the respective POU is called in the task editor.

GVL variables are only visible if these variables are **explicitly** used in the program.

, a non-compiled object can be explicitly compiled and loaded to the control. Thus, all variables declared in the object are available. Furthermore, the is available. This results in the provision of unused variables in the symbol configuration.




In the right section of the editor window, the variables currently selected for the symbol configuration (**Selected variables**) are also displayed in a tree



structure. For each one of these "items", a symbol definition is created and exported to a symbol file. The <display invalid variables> button highlights all variables in red that are currently invalid in the application, e.g. since their declaration was deleted.

To add a variable or node currently selected in the left tree or in the right tree or to remove an item from there, use the **single arrow buttons** between the two windows. The **double arrow buttons** move the entire respective tree without being selected first.

The access rights for a selected "item" can be modified in the right window by clicking on the symbol in the **Access Rights** column.

Each click switches to the next possible symbol and the access right represented by it:  read and write access,  write-only access,  read-only access.

The option "**Show libraries in available variables**" specifies whether library variables are displayed.

The **Instance Path** for a currently selected "item" is displayed in a field of the same name in the lower section of the dialog. It can be selected and copied to the clipboard using the <Copy Instance Path> button which can be useful to enter a long path into an input field.

If the symbol configuration was modified in online mode, the modified application can be loaded to the control using the <Download> button.

4.13 Sequential Function Chart Editor (SFC)

4.13.1 Sequential Function Chart Editor (SFC), Overview

The SFC editor is used for programming POU's in the IEC 61131-3 language. The language is selected when a new POU object is created in the project using the dialog under ... ► **Application** ► **Add** ► **POU**. The SFC editor is a graphical editor.

General settings on behavior and appearance are made in the

The SFC editor is located in the lower section of the window that opens when an SFC object is edited.

The upper section contains the

Editors

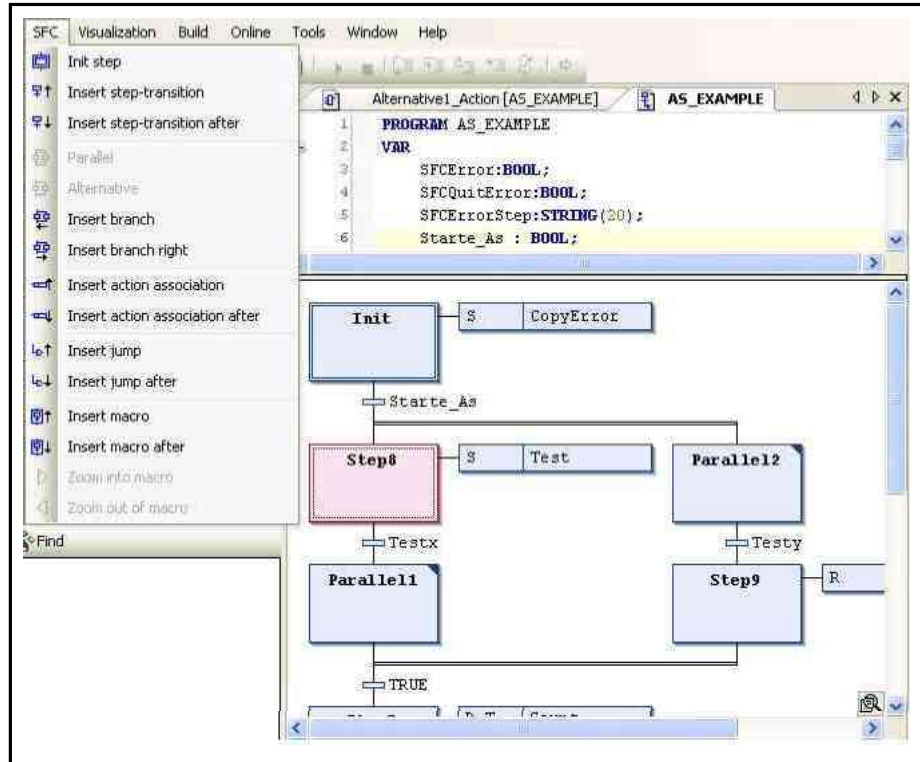


Fig.4-119: SFC editor

used in an SFC diagram are available by default in the SFC menu as soon as the SFC editor is active.

They have to be arranged as a sequence of steps connected by transitions. Parallel and alternative branches are both possible. Also see:

window.

Among other properties, the minimum and maximum activity time for a step can be defined there.

can be used to monitor the processing of an SFC diagram (e.g. step status, time monitoring, reset, etc.).

Call the commands for working in the SFC editor from the context menu or the SFC menu available by default in the menu bar when the SFC editor is active.

Comparison: IndraLogic 1.x / IndraLogic 2G

- Basically, the same functionality is available: IndraLogic 1.x SFC projects can be imported.
- Editing has become easier, since each element can be selected and repositioned individually (see ...)
- It is not necessary that the syntax in the SFC diagram is absolutely correct while programming.
- There is only one step type which combines the types used in ... can now only be provided as POU objects and are always assigned to a step in the step properties.
- ... can be used to structure an SFC diagram.

4.13.2 SFC - Sequential Function Chart

Editors

The sequential function chart (SFC) is a graphically-oriented language that allows to describe the chronological sequence of individual activities in a program.

For this purpose, the actions, which are independent programming objects written in one of the available languages, are assigned to steps. The processing sequence is controlled by transitions.

(See
).

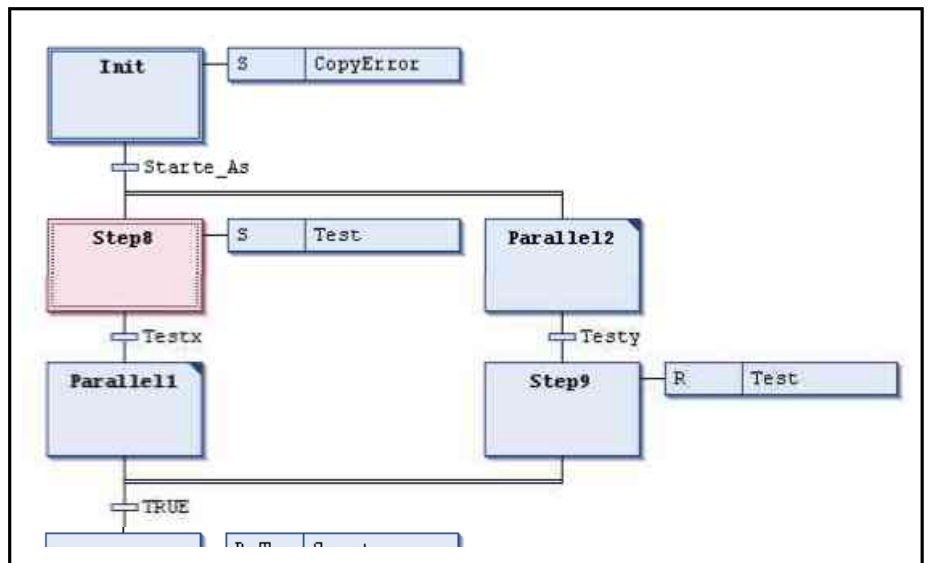


Fig.4-120: Example of a step sequence in an SFC module

4.13.3 Cursor Positions in SFC

A possible cursor position in an SFC is displayed by default with gray shading in the diagram when moving the cursor over the elements.

There are two categories of cursor positions: Texts and function block bodies. In the following figures, note the possible positions displayed with gray shading:

1. Texts

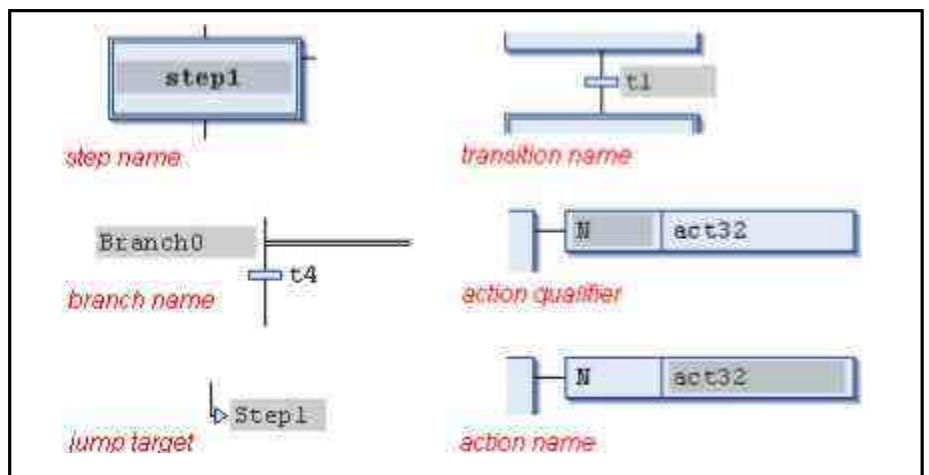


Fig.4-121: Possible cursor positions, texts:

Editors

Click on a text cursor position and the text can be edited:

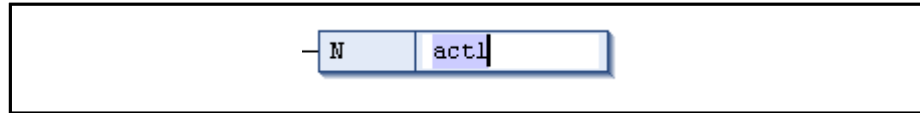


Fig.4-122: The action name was selected for editing

2. Element bodies

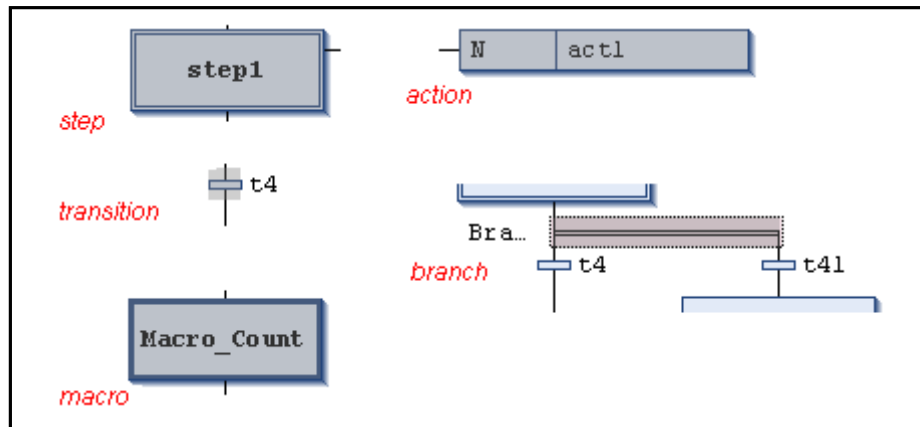


Fig.4-123: Possible cursor positions, element bodies:

Click on one of the gray shaded areas and the **element is selected**. It is outlined by a dotted frame and displayed with red shading.

():

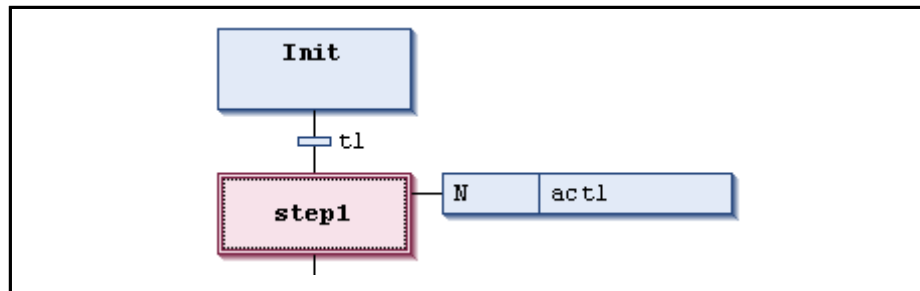


Fig.4-124: Selected step element

4.13.4 Working in the SFC Editor

By default, a new SFC POU contains an Init step and a subsequent transition. The following describes how further elements can be added, positioned and edited:

Possible cursor positions: See

Navigation: Use the arrow keys to jump from the currently selected element to the next or previous one.

Adding elements

can be added using the corresponding default commands in the "SFC" menu.

Double-clicking on a step, transition or action element already added that does not contain a reference on an existing project object opens a dialog to assign such a reference.



Editors

Selecting elements An element or text area can be selected by clicking on a possible . Use the arrow keys to switch to the adjacent element. The selected element changes its color to red. ().

Steps and transitions can be individually selected, repositioned, cut, copied, added or deleted.

Multiple selection is possible as follows:

- Hold down the <Ctrl> key and click subsequently on the individual elements to be selected.
- Press the left mouse button and to draw a rectangle (dotted line) around the elements to be selected.
- Use the command **Edit ▶ Select all**.

Editing texts

the editing field in which the text can be edited opens immediately. If a text area is selected with the arrow keys, the editing field has to be opened explicitly by pressing the <space bar>.

Editing assigned actions

Double-clicking on the action assignment in a step element (input, step or output action) or on a transition element with an assigned action opens the action in the respective editor. For example, double-click on the triangle displaying an assigned output action in a step element.

Cutting, copying and inserting elements

Select the elements and use the respective command, "Cut", "Copy" or "Insert" (from the "Edit" menu in the default setting) or use the respective keyboard commands.

Note the following behavior compared to that in IndraLogic 1.x:

- Add one or multiple element(s) and the clipboard content is added in front of the currently selected position. If nothing is currently selected, the content is added to the end of the diagram.
- If a branch is added and the currently selected element includes branching, the branch added is positioned at the left of the existing branch.
- If an action (list) is added while a step is currently selected, the new actions are added at the beginning of the action list for the step or if none exists yet, an action list is created.

- Incompatible elements during cutting/copying:

If an action (list) and an element that is not the step to which the action belongs are selected simultaneously, a message appears: "The current selection contains incompatible elements. No data is sent to the clipboard."

The selection is not saved, cannot be inserted at another position or copied.

- Incompatible elements while adding:

If there is no step or another action list when inserting an action list, an error message appears: "The contents of the clipboard cannot be inserted at the current selection."

If you try to add an element such as a step, a branch or a transition while an action list is selected, the same error message appears.

Deleting elements: Select the element(s) and use the "Delete" command or the key.



Editors

- Deleting a step also deletes the assigned action list.
- Deleting the initial step automatically makes the subsequent step to the initial step, i.e. the "Initial step" option in the step properties is enabled.
- Deleting the horizontal line in front of a branched area deletes all branches.
- Deleting all elements of a branch deletes the branch.

4.13.5 SFC Element Properties

The properties of an SFC element can be displayed and edited in the "Properties" window.

Open this window via the command
(**View ▶ Other Windows ▶ Element Properties**).

Which properties are displayed depends on the currently selected element. They are summarized in groups and the groupings can be opened and closed using the plus and minus signs.

Note that it can be specified in the "View" tab of the regarding whether a property can be displayed next to the element in the SFC diagram.

The following includes all possible properties:

Name	Element name, by default "<Element><consecutive number>" , e.g. step name "Step0", "Step1", branch name "Branch0" etc.
Comment	Element comment (text), e.g. "Counter reset". Line breaks can be added with <Ctrl>+<Enter>.
Symbol	<p>For each SFC element, an implicit variable is created with the same name as the element. Configure whether this flag variable is to be exported to the symbol configuration and which access rights should apply for the symbol in the control.</p> <p>Double-click on the field in the "Value" column or select the field and press the <space bar> to open a selection list to set one of the following access options:</p> <ul style="list-style-type: none"> • No access: The symbol is exported to the symbol configuration. It is not possible to access this symbol. • Read: The symbol is exported to the symbol configuration and can be read on the control. • Write: The symbol is exported to the symbol configuration and can be written on the control. • Read/write: Combination of reading and writing. <p>By default, nothing is entered (empty field in the selection list). That means that no symbol is exported to the symbol configuration.</p>

Fig.4-125: General properties

Initial step	<p>The first step in an SFC diagram is preset. Note: If this property is enabled for another step, it has to be disabled for the step that had this property before to prevent compilation errors.</p> <p>Also note the command in this context.</p>
Times:	Timeouts in steps can be monitored using the "SFCErrror" .
<ul style="list-style-type: none"> • Minimum active 	<p>Minimum time period that the step should be active.</p> <p>Possible values: Time specifications based on IEC syntax (e.g. t#3s) or Variable of type TIME with initial value: t#0s.</p>

<ul style="list-style-type: none"> Maximum active 	<p>Maximum time period that the step should be active.</p> <p>Possible values: Time specifications based on IEC syntax (e.g. #8s) or Variable of type TIME with initial value: #0s.</p>
<p>Actions:</p>	<p>Define the to be executed when the step is active.</p> <p>Note the .</p>
<ul style="list-style-type: none"> Step entry 	<p>This action is executed after the step was enabled ("input action").</p>
<ul style="list-style-type: none"> Step active 	<p>This action is executed when the step is active and any possible input actions have already been processed.</p>
<ul style="list-style-type: none"> Step exit 	<p>If the step is disabled, this action is executed in the following cycle ("output action").</p>

Fig.4-126: Special features



Refer to the status of a step or an action from the respective for more information.

Furthermore, you also obtain information on timeouts from .

4.13.6 SFC Elements/Toolbox

The graphical elements for programming in the SFC editor can be added using the corresponding commands available by default in the when the editor is active.



The SFC toolbox is #### in preparation ####.

In addition, refer to the help page for .

The following elements are available and described below:

-
-
-
-
-
-
-
-

Step, initial step and subsequent transition

Icons:

is represented by a rectangle (box) that contains the step name and that is connected with the subsequent transition via a line.

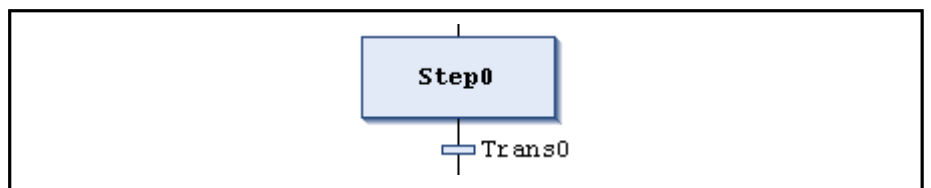


Fig.4-127: Step and subsequent transition:

The frame of the initial step consists of a double line.