Electric Drives and Controls | Hydraulics | Linear Motion and Assembly Technologies | Pneumatics | Service

Rexroth
Bosch Group

DBR
Automation

Rexroth IndraWorks 12VRS IndraLogic 2G PLC Programming System

R911334390
Edition 01

Application Manual

Title	Rexroth IndraWorks 12VRS IndraLogic 2G PLC Programming System
Type of Documentation	Application Manual
Document Typecode	DOK-IWORKS-IL2GPRO*V12-AP01-EN-P
Internal File Reference	RS-1f979330ef67403c0a6846a00053a898-1-en-US-7
Purpose of Documentation	This documentation describes the PLC programming tool IndraLogic 2G and its usage. It is also the basis for the online help.

Record of Revision

Edition	Release Date	Notes
Edition 01	09.2011	First edition for IndraWorks 12VRS

Validity

The specified data is for product description purposes only and may not be deemed to be guaranteed unless expressly confirmed in the contract. All rights are reserved with respect to the content of this documentation and the availability of the product.

Published by

Bosch Rexroth AG

Bgm.-Dr.-Nebel-Str. 2 ■ 97816 Lohr a. Main, Germany

Phone +49 (0)93 52/ 40-0 ■ Fax +49 (0)93 52/ 40-48 85

http://www.boschrexroth.com/

Components & Software BaKo, HBu (vha/MePe)

Note

This document has been printed on chlorine-free bleached paper.

Table of Contents

Table of Contents

Page

Table of Contents

Page

Table of Contents

Page

Table of Contents

Page

Page

Table of Contents

Page

Page

Table of Contents

Page

Page

Table of Contents

Page

Table of Contents

Table of Contents

Page

Table of Contents

Page

1 About this Documentation

1.1 Validity of the Documentation

Target group

This documentation is intended for users programming the PLC component of a control of the type IndraLogic XLC / IndraMotion MLC, IndraMotion MLD or IndraMotion MTX, commissioning the respective control or supporting it with an internally used visualization.

This documentation describes the PLC programming tool IndraLogic 2G and its usage.

Described are:

- Concepts and basic components
- Menu items of the PLC programming tool IndraLogic 2G
- Editors of the PLC programming tool IndraLogic 2G
- Programming references for further information
- Creating visualizations
- Working with libraries

This documentation supports the user during the following phases:

- Programming the PLC program
- Debugging and commissioning
- Visualization

1.2 Required and Supplementing Documentations

Documentation titles with type designation codes and parts numbers

IndraWorks		MLC	XLC
/36/	**Rexroth IndraWorks 12VRS Software Installation** DOK-IWORKS-SOFTINS*V12-COxx-EN-P, R911334396 This documentation describes the IndraWorks installation.	X	X
/5/	**Rexroth IndraWorks 12VRS Engineering** DOK-IWORKS-ENGINEE*V12-APxx-EN-P, R911334388 This documentation describes the application of IndraWorks in which the Rexroth Engineering tools are integrated. It includes instructions on how to work with IndraWorks and how to operate the oscilloscope function.	X	X
/20/	**Rexroth IndraMotion MLC 12VRS Functional Description** DOK-MLC***-FUNC****V12-APxx-EN-P, R911333848 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraMotion MLC.	X	
/20/	**Rexroth IndraLogic XLC 12VRS Functional Description** DOK-XLC***-FUNC****V12-APxx-EN-P, R911333878 This documentation describes wizards, context menus, dialogs, control commissioning, device configuration and functionalities of the IndraLogic XLC.		X
/7/	**Rexroth IndraWorks 12VRS CamBuilder** DOK-IWORKS-CAMBUIL*V12-APxx-EN-P, R911333842 This documentation describes the basic principles and operation of the CamBuilder, the cam editing tool.	X	X

/37/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Automation Interface** DOK-XLCMLC-AUT*INT*V12-APxx-EN-P, R911334178 This documentation describes the script-based access to IndraWorks project data via the interface of the Automation Interface.	X	X
/38/	**Rexroth IndraWorks 12VRS FDT Container** DOK-IWORKS-FDT*CON*V12-APxx-EN-P, R911334398 This documentation describes the IndraWorks FDT Container functionality. It includes the activation of the functionality in the project and working with DTMs.	X	X
/29/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Project Conversion** DOK-XLCMLC-PROCONV*V12-APxx-EN-P, R911334187 This documentation described the project conversion of IndraLogic 04VRS and IndraMotion MLC04VRS on IndraWorks Version 12 with IndraLogic 2G. It especially focuses on changes in the field of Motion and PLC.	X	X
/28/	**Rexroth IndraMotion MLC 12VRS Commissioning** DOK-MLC***-STARTUP*V12-COxx-EN-P, R911333858 This documentation describes the steps required for commissioning and performing service on the IndraMotion MLC system. It includes checklists for tasks to be frequently performed and a detailed description of the steps.	X	

Motion		MLC	XLC
/23/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS PLCopen Libraries** DOK-XLCMLC-FUNLIB**V12-LIxx-EN-P, R911334182 This documentation describes the function blocks, functions and data types of the RIL_CommonTypes, ML_Base and ML_PLCopen libraries for the IndraLogic XLC/IndraMotion MLC. It also includes the error reactions of function blocks.	X	X
/27/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Generic Application Template** DOK-XLCMLC-TF*GAT**V12-APxx-EN-P, R911334191 This documentation provides a structured template to the IndraLogic PLC programmer. This template can be used to add and edit the PLC programming code. It includes the template, the template wizard and example applications.	X	X
/31/	**Rexroth IndraMotion MLC 12VRS RCL Programming Instruction** DOK-MLC***-RCL*PRO*V12-APxx-EN-P, R911333852 This documentation provides information on the RobotControl. It is given most importance to the programming language RCL (RobotControl Language). The program structure, variables, functions, motion statements and the required system parameters are described.	X	
/21/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Parameters** DOK-XLCMLC-PARAM***V12-RExx-EN-P, R911334176 This documentation describes the parameters of the XLC/MLC systems as well as the interaction between parameterization and programming. It includes the axis parameters, control parameters, kinematic parameters, touch probe parameters and programmable limit switch parameters.	X	X
/10/	**Rexroth IndraDrive; Firmware for Drive Controllers MPH, MPB, MPD, MPC-07** DOK-INDRV*-MP*-07VRS**-FKxx-EN-P, R911328670		
/11/	**Rexroth IndraDrive MPx-16 Functions** DOK-INDRV*-MP*-16VRS**-APxx-EN-P, R911326767		

Field buses		MLC	XLC
/39/	**Rexroth IndraMotion MLC 11VRS PLCopen Field Bus** DOK-IM*ML*-PLCFBUS*V11-APxx-EN-P, R911333896 This documentation describes the creation of field bus drives in an IndraWorks project, function blocks, functions and data types of the libraries RIL_CommonTypes.library (excerpt for field bus drives), RMB_PLCopenFieldBus.library, RIL_Utilities.library (excerpt for field bus drives). It also includes the error reactions of function blocks.	X	X
/4/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Field Buses** DOK-XLCMLC-FB******V12-APxx-EN-P, R911334394 This documentation describes the supported field buses and their diagnostic function blocks.	X	X

HMI		MLC	XLC
/8/	**Rexroth IndraWorks 12VRS HMI** DOK-IWORKS-HMI*****V12-APxx-EN-P, R911334392 This documentation describes the functions, configuration and operation of the user interfaces IndraWorks HMI Engineering and IndraWorks HMI Operation.	X	X
/6/	**Rexroth IndraWorks 12VRS WinStudio** DOK-IWORKS-WINSTUD*V12-APxx-EN-P, R911333844 This documentation describes the installation of the software, working with WinStudio and the creation and operation of applications.	X	X
/50/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS HMI Connection** DOK-XLCMLC-HMI*****V12-APxx-EN-P, R911334184 This documentation describes the visualization systems supported by the IndraLogic XLC and IndraMotion MLC and their connection.	X	X

PLC		MLC	XLC
/3/	**Rexroth IndraWorks 12VRS IndraLogic 2G Programming Instruction** DOK-IWORKS-IL2GPRO*V12-APxx-EN-P, R911334390 This documentation describes the PLC programming tool IndraLogic 2G and its usage. It includes the basic usage, first steps, visualization, menu items and editors.	X	X
/33/	**Rexroth IndraWorks 12VRS, Basic Libraries, IndraLogic 2G** DOK-IL*2G*-BASLIB**V12-LIxx-EN-P, R911333835 This documentation describes the system-comprehensive PLC libraries.	X	X

Technology			
/30/	**Rexroth IndraMotion MLC 12VRS Technology Libraries** DOK-MLC***-TF*LIB**V12-LIxx-EN-P, R911333868 This documentation describes the function blocks, functions and data types of the libraries "ML_TechInterface.library", "ML_TechMotion.library", "RMB_TechCam.library" and "ML_TechBase.library". It also includes libraries for the winder functionality, register controller functionality and CrossCutter functionality.	X	
/60/	**Rexroth IndraMotion MLC 12VRS RegisterControl (Library)** DOK-MLC***-REGI*CO*V12-LIxx-EN-P, R911333856 This documentation describes the inputs and outputs of the individual function blocks and provides notes on their usage.	X	

/62/	**Rexroth IndraMotion MLC 12VRS RegisterControl (Application Manual)** DOK-MLC***-REGI*CO*V12-APxx-EN-P, R911333854 This documentation describes the application of the integrated register control for a rotogravure printing machine. The components of the mark stream sensor, the HMI application and the error recovery options are described. This instruction provides information on how to operate the register control, react on errors and query diagnostics. This documentation is written for machine setters and machine operators.	X	
/49/	**Rexroth IndraMotion MLC 12VRS Application of Winder Functions** DOK-MLC***-TF*WIND*V12-APxx-EN-P, R911333870 This application-related system documentation describes the usage of the winder technology functions.	X	

Hardware		MLC	XLC
/1/	**Rexroth IndraControl L45/L65** DOK-CONTRL-IC*L45*L65*-PRxx-EN-P, R911324661	X	X
/2/	**Rexroth IndraControl L25** DOK-CONTRL-IC*L25*****-PRxx-EN-P, R911328474	X	X
/24/	**Rexroth IndraControl Lxx 12VRS Function Modules** DOK-CONTRL-FM*LXX**V12-APxx-EN-P, R911333830 This documentation describes all function modules of the Lxx controls including engineering and diagnostics.	X	X
/12/	**Rexroth IndraDrive Drive Controllers MPx-02 to MPx-07** DOK-INDRV*-GEN-**VRS**-PAxx-EN-P, R911297317		
/13/	**Rexroth IndraDrive MPx-02 to MPx-07 and HMV** DOK-INDRV*-GEN-**VRS**-WAxx-EN-P, R911297319		
/35/	**Rexroth IndraDrive Drive Controller Control Sections CSB01, CSH01, CDB01** DOK-INDRV*-CSH********-PR08-EN-P, R911295012		

Diagnostics and Service		MLC	XLC
/26/	**Rexroth IndraMotion MLC/XLC 11VRS Service Tool** DOK-IM*ML*-IMST****V11-RExx-EN-P, R911331940	X	X
/22/	**Rexroth IndraLogic XLC IndraMotion MLC 12VRS Diagnostics** DOK-XLCMLC-DIAG****V12-RExx-EN-P, R911334180 This documentation includes all control parameters implemented in the control systems IndraLogic XLC and IndraMotion MLC.	X	X

System Overview		MLC	XLC
/48/	**Rexroth IndraMotion for Printing 12VRS System Overview** DOK-IM*PR*-SYSTEM**V11-PRxx-EN-P, R911333840 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	
/48/	**Rexroth IndraMotion for Packaging 12VRS System Overview** DOK-IM*PA*-SYSTEM**V12-PRxx-EN-P, R911333838 This documentation describes the product IndraMotion for Packaging. It introduces the control systems, drive systems and I/O systems as well as the commissioning and programming.	X	

/9/	**Rexroth IndraMotion MLC 12VRS System Overview** DOK-MLC***-SYSTEM**V12-PRxx-EN-P, R911333860 This documentation provides an overview on the possible hardware/software components of the automation system IndraMotion MLC of the named version. It helps assembling a system.	X	
/9/	**Rexroth IndraLogic XLC 12VRS System Overview** DOK-XLC***-SYSTEM**V12-PRxx-EN-P, R911333880 This documentation provides an overview on the possible hardware/software components of the automation system IndraLogic XLC of the named version. It helps assembling a system.		X

First Steps		MLC	XLC
/25/	**Rexroth IndraMotion MLC 12VRS First Steps** DOK-MLC***-F*STEP**V12-COxx-EN-P, R911333846 This documentation describes the first steps of the IndraMotion MLC and the RobotControl. It includes the hardware and software prerequisites as well as the creation of a project.	X	
/25/	**Rexroth IndraLogic XLC 12VRS First Steps** DOK-XLC***-F*STEP**V12-COxx-EN-P, R911333876 This documentation describes the first steps of the IndraLogic XLC. It includes the hardware and software prerequisites as well as the creation of a project.		X

xx Respective edition

Fig. 1-1: XCL/MLC documentation overview

1.3 Information Representation

1.3.1 Safety Instructions

The safety instructions available in the user documentations contain certain signal words (Danger, Warning, Caution, Notice) and if applicable, signal alert symbols (acc. to ANSI Z535.6-2006).

The signal word should direct the attention to the safety instructions. It indicates the severity of the hazard.

The signal alert symbol (warning triangle with exclamation mark) positioned in front of the signal words Danger, Warning and Caution indicates hazards for individuals.

⚠ DANGER

In case of non-compliance with this safety instruction, death or serious injury **will** occur.

⚠ WARNING

In case of non-compliance with this safety instruction, death or serious injury **could** occur.

⚠ CAUTION

In case of non-compliance with this safety instruction, minor or moderate injury could occur.

NOTICE

In case of non-compliance with this safety instruction, property damages can occur.

1.3.2 Symbols Used

Note

Notes are represented as follows:

☞ This is a note for the user.

Tip

Tips are represented as follows:

💡 This is a tip for the user.

1.3.3 Names and Abbreviations

Term	Explanation
SFC	Structuring medium "Sequential Function Chart"
IL	PLC programming language "Instruction List"
CFC	PLC programming language "Continuous Function Chart"
DUT	Data type
EN/ENO	Boolean input (Enable) EN / Boolean output ENO (ENable Out)
FBD	PLC programming language "Function Block Diagram"
GNVL	Global Network Variable List
GVL	Global Variable List
IEC	International Electrotechnical Commission
IndraWorks Engineering framework	Configuration and commissioning tool by Bosch Rexroth
LD	PLC programming language "Ladder Diagram"
POU	Program organization unit

Fig. 1-2: Terms and abbreviations used

2 Concepts and Basic Components

2.1 Concepts and Basic Components, General Information

IndraLogic 2G is a device-independent control programming system.

In accordance with the IEC 61131-3 standard, it supports all standard programming languages but allows only C-routines to be integrated. In combination with the IndraLogic runtime system, it enables several controls project to be programmed in one project.

Observe the following basic concepts that specify programming with IndraLogic 2G:

- **Object orientation:**

  The idea of object orientation is not only expressed in the availability of the corresponding programming elements and functions, but also in the structure and version management of IndraLogic and in the project organization.

  This way, several controls can be arranged in one IndraWorks project.

  The use of various applications on one control is ### in preparation ###.

  Devices that can be parameterized and programmed can be addressed in the same project.

- **IndraLogic versions:**

  IndraLogic is available as IndraLogic 1.x and IndraLogic 2G. Both can be provided with an installation.

  Customers that choose a control that supports IndraLogic 2G can use other controls supporting 2G in the project.

  The same applies for controls supporting IndraLogic 1.x.

  However, a mix of controls, some of which use IndraLogic 1.x and some of which use IndraLogic 2G, is not possible within one IndraWorks project.

☞ Data can be exchanged between a 1.x and a 2G control via network variables, page 71,.

- **Project organization:**

  This is also shaped by the idea of object orientation:

  An IndraLogic project includes a control program consisting of various programming objects and the definition of resources needed to operate instances of this program - handled as "Application" objects - on a specified target system (device, module, control).

  There are two main types of objects in a project:

  1. **Programming objects (POUs):**

     These are programs, functions, function blocks, methods, interfaces, actions, data type definitions, etc. See What is a POU Object, page 27.

     – Programming objects instantiated project-wide, i.e. for all applications defined in the project, have to be managed in the "General module" folder. Instantiation occurs when a program POU is called from the task, page 67, of an application.

     – Programming objects directly assigned to an application cannot be instantiated by other applications.

2. **Resource objects:**

   These are device objects, applications, task configurations, recipe managers, etc.

   According to the rules that apply when creating device objects in the Project Explorer, the hardware environment has to be mapped. See Devices in the Project Explorer, page 63.

   The validity range of objects such as "libraries" and "global variable lists" are defined hierarchically, e.g. by arranging application and device objects.

- **Code generation:**

  Code generation using integrated compilers and machine code allows short execution times.

- **Data transfer to the control device:**

  Data is transferred between IndraLogic 2G and the control using a gateway (component) and a runtime system. A complete online functionality for monitoring the program is available on the control.

2.2 Differences from IndraLogic 1.x

In principle, projects created with IndraLogic 1.x can be opened and processed.

IndraLogic 2G provides the following extensions and improvements:

Object orientation

Object orientation at programming and in the project structure

- *Extensions for function blocks:*
  - Properties, page 46
  - Interfaces, page 49
  - Methods, page 45
  - Inheritance, page 36,
  - Method call, page 40
- Extendable functions, page 33 ### in preparation ###
- Device-dependent applications, page 66, as instances of independent programming objects

New with regard to data types

- ANY_TYPE, ### In preparation ###,
- UNION, page 564
- LTIME, page 555
- REFERENCE, page 556
- Enumeration, page 564, basic data type can be specified
- `di : DINT := DINT#16#FFFFFFFF;` not permitted.

Detailed information under Data Types, page 552,.

New with regard to operators and variables

- New validity range operators, page 608, extended namespaces
- Pointers, page 557, replace the `INSTANCE_OF` operator
- Init methods, page 524, replace the INI operator
- Exit methods, page 45
- Output variables in function calls, page 31, and method calls, page 40
- VAR_TEMP, page 518, VAR_STAT, page 518

- Any expression, page 510, for variable initialization
- Assignment, page 389, as expression
- Index access, page 557, for pointers and strings
- Extendable functions (variable number of parameters) - ### in preparation ###.

New visualization concept

- A visualization editor is provided. This editor works with a toolbox and with an editor for the element properties. Parts of the visualization functionality are implemented according to IEC 61131 and thus - like the visualization elements - provided in the libraries. An internal runtime system executes the most important visualization functions.
- Text lists and image pools

New concept for user management and security

- User accounts, user groups, group-dependent rights for access to and actions with individual objects

News in editors

- ST editor:

  Parenthesis, breaks, code completion, Inline monitoring, Inline set/reset assignment
- IL editor as table editor
- FBD, LD and IL are mutually convertible and have a common editor
- FBD/LD/IL editor: The main output can be set in function blocks with multiple outputs (### in preparation ###)
- FBD/LD/IL editor: Function block parameters are not automatically updated
- FBD/LD/IL editor: Branching and "networks in networks"
- SFC editor: Only one step type, macros, multiple selection of independent elements, no syntax check while editing

News with regard to library management

- Several versions of one library are possible in the same project. Unique access by specifying the namespace.
- Installation in repositories, automatic updates, debugging

And more...

- Configurable menus, toolbar and keyboard operation
- Control configuration and task configuration integrated in the Project Explorer
- Unicode support
- Single line comments: `// Comment`
- `CONTINUE` in loops
- Multiple selection in the Project Explorer
- Online help integrated in the user interface
- Conditional compilation
- Conditional breakpoints
- Debugging: Execution up to cursor, execution up to return
- Field bus driver according to IEC 61131-3
- Symbol and control configuration in the application
- Free memory assignment for code and data
- Each object can be defined as "internal" or "external" (late linking in the runtime system).
- Connection to external data sources

Compatibility with IndraLogic 1.x projects

- Projects in other formats including projects created with IndraLogic 1.x can be imported. The handling of integrated libraries and devices can be specified here.
- Syntactic and semantic limitations with regard to IndraLogic 1.x projects:
  - `FUNCTIONBLOCK` is no longer a valid keyword; but has been replaced by FUNCTION_BLOCK, page 33,.
  - TYPE, page 563, (structure declaration) has to be followed by a `":"`.
  - ARRAY initialization, page 560, requires square brackets.
  - Local declaration of an enumeration, page 564, is now only possible within `TYPE` / `END_TYPE`.
  - `INI` is no longer supported (replaced by the Init method, page 524,).
  - In function calls, page 31. it is no longer possible to mix explicit and implicit assignments. However, this allows the arrangement of the input parameters to be modified:
- Pragmas, page 526, (import of IndraLogic 1.x pragmas ### in preparation ###)

2.3 Project

A project includes the POU objects, page 27, that compose a control program and the definitions of the resource objects, page 63 needed to execute one or more instances of the program (application, page 66) on a specified target system (controls, devices).

POU objects available project-wide are managed in the "General module" folder. Device-specific resource objects and application-specific POU objects are managed in the respective "device".

A project is saved in an "IndraLogic.project" file.

Project-specific configurations can be made in the dialogs of the **Tools ▸ Settings** and **Tools ▸ Options** menu items.

☞ The appearance and properties of the user interface are saved in the programming system and not with the project.

Library information

Information on the project currently edited, e.g. file data, object statistics, the name of the author name, etc., is found in the "Library Information" dialogs.

The library information, page 371, is provided in the "Library Info" object in the "General module" folder.

Data transfer

Devices and individual device objects can be imported into a project. In IndraLogic V1.x and in IndraLogic 2G projects can be imported.

If libraries or devices are integrated in a V1.x project, decide before converting the project whether they should continue to be used in the project or should be replaced by others or whether references should be removed.

Also see

Data transfer, page 115.

2.4 Supported Programming Languages

All programming languages listed in the IEC standard IEC 61131 are supported with specially adapted editors:

- FBD/LD/IL editor, page 339, for function block diagrams (FBD), ladder diagrams (LD) and instruction lists (IL)
- SFC editor, page 399, for sequential function chart (SFC)
- ST editor, page 386, for structured text
- In addition, an editor is provided that supports programming in the CFC (continuous function chart) language:

  CFC editor, page 309, stands for Continuous Function Chart editor.

  CFC is an extension of the standard IEC languages.

2.5 What is a POU Object

POUs are programming units (objects) composing a control program.

POU	= Program organization unit
POE	= Program organization unit

POUs managed in the "General module" folder are not device-specific. Instead, they can be used project-wide and can be instantiated for usage in a device-specific application. For this purpose, program POUs are called using a task of the respective application.

POUs assigned to a "device", i.e. are added to the Project Explorer, page 63, directly below an application can only be used by applications listed below this application in an indented list ("child" applications). Further information can be found in the description of the project tree, page 65, and the "Application" object, page 66.

Using Add, page 234, in the library and in the context menu, "POU" is also used as the name for a specific Subcategory, page 28, of these objects and in this case it only designates programs, function blocks and functions.

A "Program Organization Unit" object is always a programming unit, an object that can be managed in a non-device-specifically in the "General module" folder or in a device-specifically below an application and is displayed in an editor window and can be edited there. A POU object can be a program, a function, a function block, a method, an action, an interface or a DUT (data unit type).

Note that it is possible to define specific properties, page 238, (such as specifications for the compilation, etc.) separately for each POU.

Types of POUs:

- Action, page 51
- Application, page 66
- Library manager, page 367
- Image pool, page 61
- DUT (data type), page 43
- Property (PROPERTY), page 46
- Function (FUNCTION), page 31
- Function block (FUNCTION_BLOCK), page 33
- Global variable list, page 52
- Method (METHOD), page 45
- Program (PROGRAM), page 29
- Interface (INTERFACE), page 49
- Text list, page 55

- Visualization, page 63

In addition to POU objects, "resources" are needed to execute the program on the target system (application, task configuration, etc.). These are managed below a "device" in the Project Explorer, page 63,.

2.6 Program Organization Units (POU)

2.6.1 POU, General Information

The term "POU" designates a program organization unit that is either of type program, function or function block. For superordinate use of the term "POU" for all program organization units, see "What is a POU Object", page 27,. There is also information on managing project-global and device-specific POUs.

A POU can be added to the project by using **Add ▸ POU** in the context menu.

Alternatively, add a POU object from the "PLC Objects" library by dragging it with the mouse.

The dialog "Add POU" opens in which POUs are configured with regard to name, type and implementation language.

For a function block, EXTENDS and IMPLEMENTS properties can be added.

For a function, a property or a method, a return value has to be specified in most of the cases.

Depending on the type and implementation language used, a function block can be extended by method, properties, actions and transitions.

The hierarchical processing of individual POUs assigned to an application depends on the device-specific configuration (Project Explorer).

Each POU consists of a "declaration part" and an "implementation part". The implementation part is written in one of the following programming languages:

- *Text languages:*
  - Instruction list (IL)
  - Structured text (ST)
- *Graphical languages:*
  - Sequential function chart (SFC), structuring medium
  - Ladder diagram (LD)
  - Function block diagram (FBD)
  - Continuous function chart (CFC)

IndraLogic 2G supports the POUs described in the standard 61131-3. To use these default POUs in your project, page 26,, the standard.library library has to be included.

Calling POUs

POUs can call other POUs. Recursions are not permitted.

If a POU belonging to an application calls another POU only by its name (without namespace, page 86, extension), the following order applies in which it is searched in the project for the POU to be called:

1. Current application,
2. Library manager of the current application,
3. POU window
4. Library manager in the POU window.

If a POU with the name specified in the call exists in a library of the "Application" library manager and as object in the "POUs" window, there is no syntax on how the POU in the "POU s" window can be called by its name only.

In this case, the respective library of the "Application" library manager is to be moved to the library manager in the "POUs" window. The POU object in the "POUs" window can the be called using only its name (and if required, the POU of the library by adding the library namespace).

2.6.2 Program (PROGRAM)

A program is a POU, page 28 that provides one or more values at execution. All values remain from one program execution until the following one.

Adding a program:

A program object can be added to the project via **Add ▸ POU** in the context menu.

- If it is to be directly assigned to an existing application, the "Application" object, page 66, has to be highlighted before in the Project Explorer.
- If it is to be available across the projects, it has to be added to the "General module" folder.

In the "Add Object" dialog, select the POU type "Program", enter a name for the program (<program name>) and select the desired implementation language (programming language). After the settings have been confirmed with "Finish", the new POU object is displayed in the Project Explorer. Open the editor window for the new program by double-clicking the POU object or via the "Open" command in the context menu and then start the implementation:

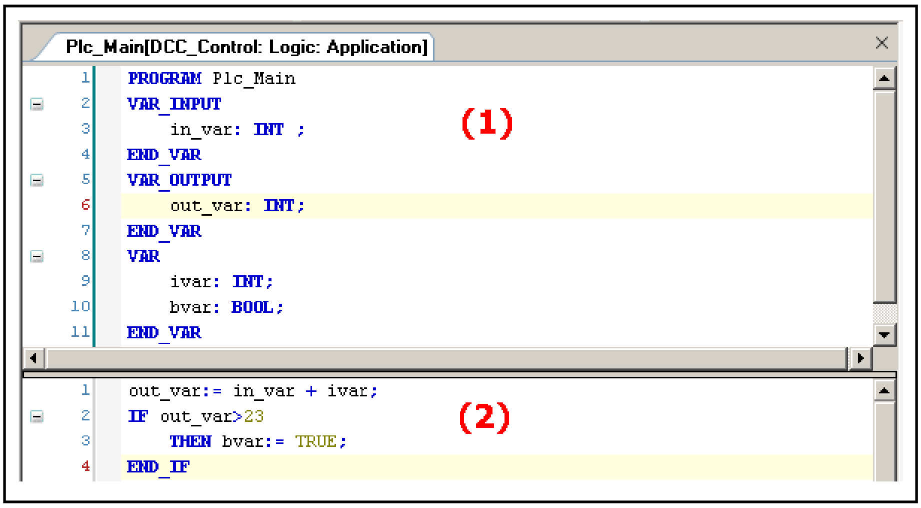
Declaration:

Syntax:

```
PROGRAM <program name>
```

The variable declarations for

- Input Variables (VAR_INPUT), page 517,
- Output Variables (VAR_OUTPUT), page 517,
- Local Variables (VAR), page 516,
- External Variables (VAR_EXTERNAL) , page 519, and
- Access Variables, page 519 (### In preparation ###).



```
PROGRAM Plc_Main
VAR_INPUT
    in_var: INT ;
END_VAR
VAR_OUTPUT
    out_var: INT;
END_VAR
VAR
    ivar: INT;
    bvar: BOOL;
END_VAR
```

```
out_var:= in_var + ivar;
IF out_var>23
    THEN bvar:= TRUE;
END_IF
```

(1) Declaration
(2) Implementation
Fig.2-1: Program example

Program calls: A program can be called by another program or function block instance.

But

- a program call in a function, page 31, is not permitted.
- There are no instances of programs.

If a program was called and that caused changes in the program values, these changes remain until the program is called again. This is also the case if the new call is made by another program or another function block instance.

This differs from calling a function block where only the values in the respective instance of the function block change and the changes are only to be noted if the same instance is called again.

To set input and output parameters at the program call, use a parenthesis right after the program name.

For input parameters, the assignment is given with ":=" as for the initialization, page 509, of variables in the declaration.

For output parameters "=>" is used; see the example below.

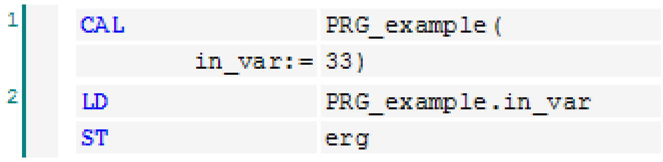
If a program call is added by using the "Input assistance" and the option "Add with arguments" in the implementation part of a text editor, it is automatically displayed with all parameters according to the syntax described in the previous section.

Then, add the corresponding value assignments.

To enable the option "Add with arguments", right-click in the editor workspace and select **Input assistance** in the context menu. In the dialog, set the option "Add with arguments".

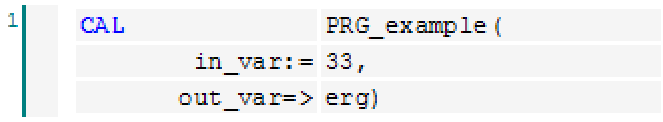
Program call examples:

In IL (instruction list):



```
CAL                PRG_example(
         in_var:=  33)
LD                 PRG_example.in_var
ST                 erg
```

or with parameter assignment (input assistance "Add with arguments", see above):



```
CAL                PRG_example(
         in_var:=  33,
        out_var=>  erg)
```

In ST (structured text):

Note that - in contrast to IndraLogic 2.x - parentheses are required here!

Program:

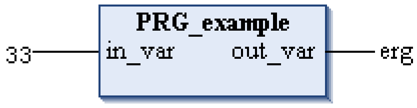
```
PRG_example.in_var:= 33;
PRG_example();
erg:= PRG_example.out_var;
```

or better, with parameter assignment (input assistance "Add with arguments", see above):

Program:

```
PRG_example(in_var:= 33, out_var=> erg);
```

In FBD (function block diagram):



2.6.3 Function (FUNCTION)

A function is a POU, page 28 that can be exactly one data element (can also consist of multiple elements as array or structure for example) at execution. In addition to the return value, output variables can also be provided.

The call, page 32, can be an operator in expressions in text languages.

Add function: A function object can be added to the project via the context menu items **Add ▸ POU**.

- If it is to be directly assigned to an existing application, the "Application" object, page 66, has to be highlighted before in the Project Explorer.
- If it is to be available across the projects, it has to be added to the "General module" folder.

In the "Add Object" dialog, select the POU type "Function", enter a name for the function (<function name>), select a return type (<data type>) and select the desired implementation language (programming language).

Use the ... button to open the input assistance, page 98, to select the return type.

After the settings have been confirmed with "Finish", the new POU object is displayed in the Project Explorer. Open the editor window for the function object by double-clicking the POU object or via "Open" in the context menu and then start the implementation:

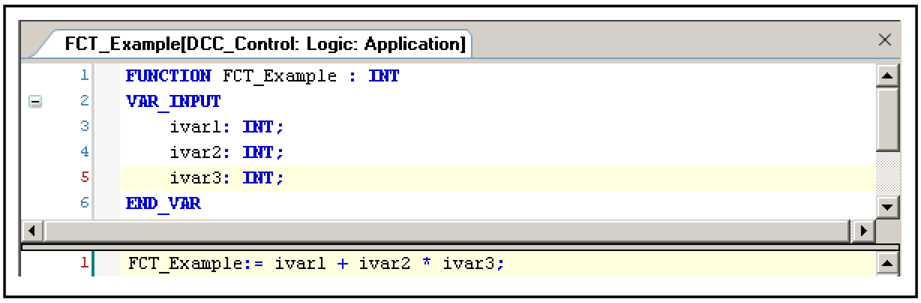
Declaration: *Syntax:*

```
FUNCTION <function name>: <data type>
```

The variable declarations for

- Input Variables (VAR_INPUT), page 517,
- Local Variables (VAR), page 516, and if required
- Output Variables (VAR_OUTPUT), page 517.

A result has to be assigned to each function (return value with type and name of the function).



```
FUNCTION FCT_Example : INT
VAR_INPUT
    ivar1: INT;
    ivar2: INT;
    ivar3: INT;
END_VAR
```

```
FCT_Example:= ivar1 + ivar2 * ivar3;
```

Fig.2-2: Example: Function in ST (The function has three input variables and returns the product of the last two added to the first.)

If a local variable is declared in a function as RETAIN, it has no effect!

The variable is not written in the retain memory!

The "sequential function chart" (SFC) structuring tool is not intended for functions.

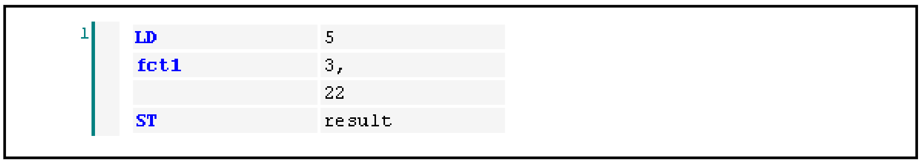
Function call:

In "ST", page 389, (structured text), a function call can be used as an operand in expressions.

In "SFC" (sequential function chart), a function call can only occur in step actions or transitions.

In contrast to programs or function blocks, function variables are reassigned for each call. This means that function calls with the same arguments (input parameters) always return the same value (return value/output parameter). For this reason, functions may not use global variables and addresses!

Examples of function calls:



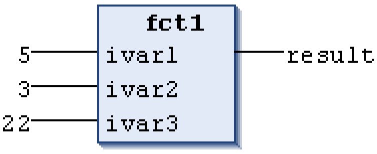
```
LD      5
fct1    3,
        22
ST      result
```

Fig.2-3: In IL (instruction list):

In ST (structured text):

```
result:= fct1(5, 3, 22);
```

In FBD (function block diagram):



In contrast to IndraLogic 1.x, explicit and implicit parameter assignments can no longer be mixed in function calls. Thus, the sequence of parameter assignments at the call is no longer specified.

Example:

```
fun(formal1:= actual1, actual2); // -> Error message
fun(formal2:= actual2, formal1:= actual1);
// Same semantics as the following:
fun(formal1:= actual1, formal2:= actual2);
```

This difference in handling the parameter assignments has to be considered when editing V1.x projects!

VAR_OUTPUT in functions

According to the IEC 61131-3 standard, functions can have additional outputs.

Syntax:

```
out1=><Ausgabevariable1>|,out2=><Ausgabevariable2>|,...
```

Example:

The function "fun" is defined with two input variables "in1" and "in2" (VAR_INPUT) and two output values "out1" and "out2" (VAR_OUTPUT).

Function call

```
fun(in1:=1, in2:=2, out1=>loc1, out2=>loc2);
```

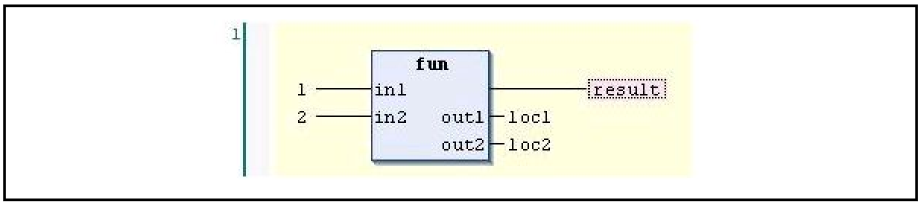


Fig.2-4: Function with return value and two output variables

Extendable functions

As an extension of the IEC 61131-3 standard, functions and methods can be provided with a variable number of input parameters of the same type. Further information can be found in Extendable functions, page 524,.

2.6.4 Function Block (FUNCTION_BLOCK)

Function Block, General Information

A function block is a POU, page 28 providing one or multiple values at execution.

In contrast to a function, the values of the output variables and the local variables used are retained from one execution to the next. This way, when a function block with the same input parameters instance is called more than once, the same output values are not necessarily provided.

Function blocks can be defined as extensions, page 36, of other function blocks and can contain interface definitions, page 38, for method calls, page 40,.

This means that the principles of object-oriented programming (instance generation) of inheritance can be applied when programming with function blocks.

A function block is always called using an instance, page 34.

Adding a function block:

A function block can be added to the project via **Add ▸ POU** in the context menu.

- If it is to be directly assigned to an existing application, the "Application" object, page 66, has to be highlighted before in the Project Explorer.
- To be available project-wide, add it to the "General module" folder.

In the "Add Object" dialog, select the POU type "Function block", enter a name for the function block (<functionblockname>) and select the desired implementation language (programming language).

The following options can also be enabled:

- **EXTENDS (extended):** Enter the name of another function block from the project that is to be used as basis for the present function block.

  A detailed description can be found in the following, seeExtending a Function Block, page 36.
- **IMPLEMENTS (implemented):** Enter the name or names of the interfaces, page 49, defined in the project to be implemented in the present function block.

  Separate multiple interface names with commas.

A detailed description can be found in Implementing Interfaces, page 38.

In the "Method implementation language" field, select the desired programming language for all method objects created by the interface implementation. This is not related to the set function block programming language.

After the settings have been confirmed with "Finish", the editor window for the new function block opens. Start the implementation.

Declaration

Syntax:

```
FUNCTION_BLOCK <function_block type name>
               | EXTENDS <function_block type name>
               | IMPLEMENTS <comma-separated list of interface names>
```

The variable declarations for

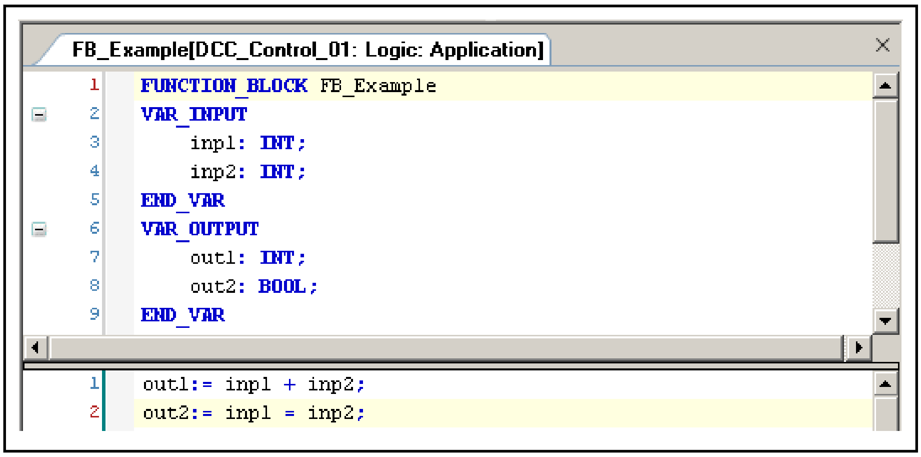
- Input Variables (VAR_INPUT), page 517,
- Output Variables (VAR_OUTPUT), page 517,
- Local Variables (VAR), page 516,
- External Variables (VAR_EXTERNAL) , page 519, and
- Access Variables, page 519 (### In preparation ###).

Example:

The function block example in the figure below has two input variables "inp1" and "inp2" and two output variables "out1" and "out2".

"out1" is the sum of both inputs.

"out2" is the result of an equality check.



FB_Example[DCC_Control_01: Logic: Application]

```
FUNCTION_BLOCK FB_Example
VAR_INPUT
    inp1: INT;
    inp2: INT;
END_VAR
VAR_OUTPUT
    out1: INT;
    out2: BOOL;
END_VAR
```

```
out1:= inp1 + inp2;
out2:= inp1 = inp2;
```

Fig.2-5: Example of a function block in ST

☞ In contrast to IndraLogic 1.x, "FUNCTIONBLOCK" is no longer a valid keyword.

It has to be replaced by FUNCTION_BLOCK!

Instance of a Function Block

Function blocks, page 33, are always called using an instance (copy of the function block).

Each instance has an identifier (instance name) and a data structure that contains its input, output and internal variables.

Like variables, instances are declared to be local or global where the name of the function block is specified as type of the identifier.

Syntax:

```
<Instance name>: <function_block name>;
```

Example:

Declaration (e.g. in the declaration part of a program) of an instance "INSTANCE" of the function block "FUB":

```
INSTANCE: FUB;
```

☞ Instance declarations are typically made in the declaration parts of function blocks and programs; range VAR...END_VAR or VAR RETAIN...END_VAR or for the data transfer in the range of VAR_INPUT...END_VAR.

In functions, they are only possible for the data transfer in the range VAR_INPUT...END_VAR.

Calling a Function Block

Function blocks, page 33, are always called using a function block instance. The instance has to be declared as local or global (<InstanceName>).

This declaration is explained in Instance of a function block, page 34.

The desired function block variable (<VariableName>) can be accessed using the following syntax:

Syntax of calling an input variable:

```
<Instance name>.<Variable name>
```

Note the following:

- From outside the function block instance, only the function block input and output variables can be accessed:
  - From the outside, input variables can only be described
  - From the outside, output variables can only be read
  - From the outside, local variables are not visible
- Access to a function block instance is limited to the POU, page 28, in which it is declared unless it is declared as global.
- When calling the instance, the desired values can be assigned to the function block parameters.

  See below "Assigning Parameters at a Call", page 36.
- Input/Output Variables (`VAR_IN_OUT`) of a function block are transferred as pointers.
- In SFC, function block calls can only occur in actions.
- The name of a function block instance can be used as an input parameter for a function or another function block.
- All values of a function block remain until the next function block execution. For this reason, function block calls do not always deliver the same output values, even if the same input values are used!

☞ If at least one of the function block variable is a "remanent" variable, the entire instance is saved in the retain area.

Examples for accesses to function block variables:

Assumption: Function block **fb** has an input variable "in1" of type `INT`.

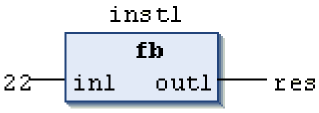
The following shows the calls of this variable from the program "prog".

Declaration and implementation in ST:

```
PROGRAM prog
 VAR
  inst1:fb;
 END_VAR

inst1.in1:= 22;  // the value 22 is assigned to the input variable in1 of the instance inst1
inst1();         // fb is called and edited; this is necessary for the following access
                 // to the output variable
res:=inst1.outl; // the output variable out1 of fb is read
```

Example in FBD (function block language):



Assigning parameters in a call:

In the text languages "IL" and "ST", input and output parameters can be set directly when calling the function block. The values can be assigned to the parameters within parentheses directly following the function block name.

For input parameters, the assignment is given with "`:=`" as for the initialization, page 509, of variables in the declaration.

For output parameters "=>" is used; see the following example.

Example, call with assignments:

In this example, a timer function block (instance "CMD_TMR") with assignments for the "IN" and "PT" parameters is called. Afterwards, the output variable "Q" of the timer is assigned to variable "A".

Syntax of calling an output variable:

```
<Instance name>.<Variable name>
```

Example:

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

If the instance is added to the programming section of a text editor using input assistance and the "Add with arguments" option, it is automatically represented according to the syntax, page 36, described above with all parameters. Then, add the corresponding value assignments.

To enable the option "Add with arguments", right-click in the editor workspace and select **Input assistance** in the context menu. In the "input assistance" dialog, place a checkmark next to the "Add with arguments" option.

For the example described above, the call would then be as follows:

Example, adding with arguments using input assistance:

```
CMD_TMR(IN := %IX5, PT := 300, Q=>A);
```

Extending a Function Block (EXTENDS)

In object-oriented programming, one function block, page 33, can be derived from another function block. That means that one function block can be used to extend another thereby adding the properties of the basic function block to its own.

This extension is carried out in the declaration using the keyword **EXTENDS**.

The "Extended" option can be enabled when a function block is added to the project. To do this, highlight the "Application" node and select the "Add Object" dialog in the **Add ▸ POU** context menu.

Syntax:

```
FUNCTION_BLOCK <function_block type name B> EXTENDS <function_block type name A>
```

See the variable declarations in the following.

Definition of function block fbA:

```
FUNCTION_BLOCK fbA
VAR_INPUT
    ivar_A: int;
.....
```

Definition of function block fbB:

```
FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    ivar_B: int;
.....
```

Extending using EXTENDS means that:

- "fbB" contains all data and methods/properties defined by "fbA"; has"ivar_A" (inherited) and "ivar_B" (itself) as VAR_INPUT.

  An instance of "fbB" can now be used in every context in which a function block of type "fbA" is expected.
- "fbB" may overwrite the methods/properties defined in "fbA".

  That means that "fbB" can define a method/property with the same name, the same inputs and the same return value (as well as outputs if available) as defined by "fbA".

  If it does not overwrite the method/properties, it inherits the original.
- "fbB" may not contain any function block variables with the same name as those used in "fbA". If this is the case, the compiler reports an error.

  The only exception:

  If a variable is declared in "fbA" as **VAR_TEMP**, "fbB" may define a variable of the same name, but can no longer access the variable of the basic function block.
- "fbA" methods and variables can be directly addressed within the valid range of "fbB" by using the keyword **SUPER**

  (`SUPER^.<MethodName> or SUPER^.<MethodName>.<VariableName>`)

☞ Multiple inheritance is not permitted!

Example:

```
FUNCTION_BLOCK FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
    iCnt : INT;
    iRes : INT;
END_VAR
```

```
VAR
END_VAR

THIS^.METH_DoIt();
THIS^.METH_DoAlso();

    METHOD METH_DoIt : BOOL
    VAR
    END_VAR
    iCnt := -1;
    METH_DoIt := TRUE;

    METHOD METH_DoAlso : BOOL
    VAR
    END_VAR
    iRes := -5;
    METH_DoAlso := TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_INPUT
END_VAR
VAR_OUTPUT
    iCnt_Base: INT;
    iCnt_THIS: INT;
    iRes_Base: INT;
    iRes_THIS: INT;
END_VAR
VAR
END_VAR
// Calls the method defined under FB_1
 THIS^.METH_DoIt();
 THIS^.METH_DoAlso();
 iCnt_THIS:= iCnt;
 iRes_THIS:= iRes;
// Calls the method defined under FB_Base
 SUPER^.METH_DoIt();
 SUPER^.METH_DoAlso();
 iCnt_Base:= iCnt;
 iRes_Base:= iRes;

    METHOD METH_DoIt : BOOL
    VAR
    END_VAR
      iCnt := 1111;
      METH_DoIt := TRUE;

PROGRAM PLC_PRG
VAR
    Myfb_1: FB_1;
    iFB: INT;
    iBase: INT;
END_VAR
 Myfb_1();
 iBase := Myfb_1.iCnt_Base;
 iFB := Myfb_1.iCnt_THIS;
```

Implementing Interfaces (IMPLEMENTS)

In object-oriented programming, a function block can implement interfaces, page 49, that enable the use of methods, page 45, and properties, page 46,:

Syntax:

```
FUNCTION_BLOCK <FB type name> IMPLEMENTS <interface name_1>,...,<interface name_n>
```

A function block that implements an interface has to contain all methods/ properties defined in this interface. That means that the name, the inputs and return values (as well as outputs if available) of the methods/properties have to be identical.

In addition, when a new function block that implements an interface is created, all methods and properties defined in the interface are also automatically copied below the new function block.

Currently, changes made later on at the interface, such as adding more methods, are not automatically made in the respective function blocks (interface methods become POU methods in function blocks, interface properties become POU properties in function blocks).

This has still to be carried out explicitly using the Implement interfaces, page 236, at each function block. The implementation language is queried for the method/property.

Example:

INTERFACE ITF_1 contains the method "GetName":

```
METHOD GetName : STRING
```

The function blocks FB_A and FB_B implement the interface ITF_1 each:

```
FUNCTION_BLOCK FB_A IMPLEMENTS ITF_1
END_FUNCTION_BLOCK

FUNCTION_BLOCK FB_B IMPLEMENTS ITF_1
END_FUNCTION_BLOCK
```

This means that the method "GetName" has to be present in both functions and is automatically attached below in the Project Explorer when the function blocks are created. It has to be implemented separately for each function block.

Function blocks FB_A, method "GetName" (example)

```
METHOD GetName : STRING
 GetName:= 'FB_A';
```

Function blocks FB_B, method "GetName" (example)

```
METHOD GetName : STRING
 GetName:= 'FB_B';
```

Look at the "DeliverName" function with its input of a variable of type of the interface ITF_1:

Function "DeliverName": STRING

```
FUNCTION DeliverName : STRING
VAR_INPUT
   I_i: ITF_1;
END_VAR
 DeliverName:= I_i.GetName();  // in the case, it depends on the "actual" type of I_i,
                               // if A.GetName or B.GetName is called.
```

This input variable can receive all function blocks that implement "interface ITF_1".

☞ The interface of a function block has to be assigned to the variable of the type of an interface before it can be used to call a method.

The variable of an interface type is always a reference of the assigned function block instance.

In this way, calling the interface method results in a call of the function block implementation.

In online mode, as soon as a reference is created, the related address is shown.

If a reference has not yet been generated, the value "0" is shown here.

Examples for function calls:

```
PROGRAM PlcProg
VAR
    Name_FB_A : STRING;
    Name_FB_B : STRING;
    FB_A_Inst: FB_A;
    FB_B_Inst: FB_B;
END_VAR
 Name_FB_A:= DeliverName(I_i:= FB_A_Inst); // call with FB_A_instance
 Name_FB_B:= DeliverName(I_i:= FB_B_Inst); // call with FB_B_instance
```

Method Call

Object-oriented programming with function blocks is supported - apart from the option of extension with EXTENDS, page 36 - also by using

Interfaces, page 38, and

Inheritance, page 33,

This requires dynamically resolved method calls, also called "virtual function calls" for further methods.

Virtual function calls require a little more time than normal function calls and are used when:

- a function block is called using a pointer (e.g. "pfub^.method"),
- a method of an interface variable is called (e.g. "interface1.method"),
- a method calls another method of the same function block,
- a function block is called using a reference,
- VAR_IN_OUT of a basic function block type is assigned to an instance of a derived function block type.

Virtual function calls enable the same call in a program source code to call a variety of methods at runtime.

For further information, refer to:

- Method, page 45, about using methods
- THIS pointer, page 42, about using THIS
- SUPER pointer, page 41, about using SUPER.

Calling methods

According to the IEC 61131-3 standard, methods can have additional outputs like functions. These have to be assigned according to the following syntax at method call:

Syntax:

```
<method>(in1:=<value> |, further input assignments, out1 => <output variable 1>
                                               | out2 => <output variable 2> |...further output variables)
```

This causes the method output as defined in the call to be written to the locally declared output variables.

Example:

Assumption:

Function blocks "fub1" and "fub2"

EXTENDS (extend) function block "fubbase"

IMPLEMENT (implement) interface "interface1"

The method "method1" is included.

Possible use of the interfaces and calling the method:

```
// Declaration
VAR_INPUT
```

```
    b : BOOL;
END_VAR
VAR
    pInst : POINTER TO fubbase;
    instBase : fubbase;
    inst1 : fub1;
    inst2 : fub2;
    interfaceRef : interface1;
END_VAR

// Implementation
IF b THEN
    interfaceRef := inst1;    // Interface1 for fub1
    pInst := ADR(instBase);
ELSE
    interfaceRef := inst2;    // Interface1 for fub2
    pInst := ADR(inst1);
END_IF

pInst^.method1();       // If b is true, fubbase.method1 is called, else fub1.method1 is called
interfaceRef.method1(); // If b is true, fub1.method1 is called, else fub2.method1 is called
```

Assuming that "fubbase" contains two methods, "method1" and "method2", and that "fub1" overwrites "method2", but not "method1":

"method1" is called in the following as in the example above:

```
pInst^.method1(); // If b is true fubbase.method1 is called, else fub1.method1 is called
```

Also see the call via THIS pointer, page 42,.

SUPER Pointer

One pointer with the name SUPER is automatically available for each function block. This pointer points to the basic function block instance from which the function block was created by inheritance of the basic function block.

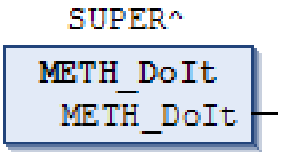
Thus, the following effective problem solution is possible:

- SUPER allows access to the implementation of the basic class methods. Using the keyword SUPER, a method is called that is valid in the instance of the basic or parent class. Thus, no dynamic name linking takes place.

SUPER can only be used in methods and in the respective function block implementations. Since SUPER is a pointer to the basic function block, it has to be unreferenced to keep the address of the function block:

`SUPER^.METH_DoIt.`

Call SUPER in different implementation languages

ST	`SUPER^.METH_DoIt();`
LD/FBD/CFC	
IL	The SUPER functionality for the instruction list (IL) is ### in preparation ###.

Example:

```
FUNCTION_BLOCK FB_Base
VAR_OUTPUT
    iCnt : INT := 5;
END_VAR
    METHOD METH_DoIt : BOOL
    iCnt := -1;
    METH_DoIt := TRUE;
    METHOD METH_DoAlso : BOOL
    METH_DoAlso := TRUE;

FUNCTION_BLOCK FB_1 EXTENDS FB_Base
VAR_OUTPUT
    iSuper: INT;
    iThis: INT;
END_VAR
// Calls the method defined under FB_1
  THIS^.METH_DoIt();
  THIS^.METH_DoAlso();
  iThis := THIS^.iCnt;
// Calls the method defined under FB_Base
  SUPER^.METH_DoIt();
  SUPER^.METH_DoAlso();
  iSuper := SUPER^.iCnt;

    METHOD METH_DoIt : BOOL
    iCnt := 1111;
    METH_DoIt := TRUE;

PROGRAM PLC_PRG
VAR
    myBase: FB_Base;
    myFB_1: FB_1;
    iTHIS: INT;
    iBase: INT;
END_VAR
 myBase();
 iBase := myBase.iCnt;
 myFB_1();
 iTHIS := myFB_1.iCnt;
```

THIS Pointer

A pointer with the name THIS is automatically available for each function block. This pointer points to the function block instance.

Thus, the following effective problem solutions are possible:

- If a locally declared variable shadows a function block variable in the method.
- If the pointer is referenced to the individual function block instance to be used in a function.

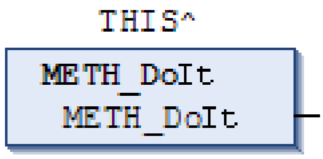
Thus, THIS can only be used in methods and in the respective function block implementations.

THIS has to be written in upper case letters. Other spelling is not permitted.

Since THIS is a pointer to the function block to be inherited, it has to be referenced to keep the address of the overwriting function.

`THIS^.METH_DoIt`.

Call THIS in different implementation languages

ST	`THIS^.METH_DoIt();`
LD/FBD/CFC	
IL	The THIS functionality for the instruction list is (IL) ### in preparation ###.

Example:

The local variable "iVarB" shadows the function block variable "iVarB":

```
FUNCTION_BLOCK  fbA
VAR_INPUT
    iVarA: INT;
END_VAR
 iVarA := 1;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    iVarB: INT := 0;
END_VAR
 iVarA := 11;
 iVarB := 2;

    METHOD DoIt : BOOL
    VAR_INPUT
    END_VAR
    VAR
        iVarB: INT;
    END_VAR
     iVarB := 22;        // Here the local iVarB is set.
     THIS^.iVarB := 222; // Here the function block variable iVarB is set,
                         // although iVarB is overloaded.

PROGRAM PLC_PRG
VAR
    MyfbB: fbB;
END_VAR

 MyfbB(iVarA:=0 , iVarB:= 0);
 MyfbB.DoIt();
```

Example:

A function call requires the reference to the individual instance:

```
FUNCTION funA : BOOL
VAR_INPUT
    pFB: fbA;
END_VAR
...;

FUNCTION_BLOCK  fbA
VAR_INPUT
    iVarA: INT;
END_VAR
...;

FUNCTION_BLOCK fbB EXTENDS fbA
VAR_INPUT
    iVarB: INT := 0;
END_VAR
 iVarA := 11;
 iVarB := 2;

    METHOD DoIt : BOOL
    VAR_INPUT
    END_VAR
    VAR
        iVarB: INT;
    END_VAR
     iVarB := 22;        // Here the local iVarB is set.
     funA(pFB := THIS^); // Here funA is called with THIS^.

PROGRAM PLC_PRG
VAR
    MyfbB: fbB;
END_VAR
 MyfbB(iVarA:=0 , iVarB:= 0);
 MyfbB.DoIt();
```

2.6.5 DUT/Data Type

In addition to the standard data types, users can define their own data types.

- "Arrays", page 560, (ARRAY),
- "Structures", page 563, (STRUCT),
- "Enumeration types", page 564, (ENUM),

- "References", page 556, (REFERENCE TO),
- "Subrange Types", page 566,
- "Unions", page 564, (UNION)

can be created as data type objects (DUT objects) in the DUT editor, page 338,.

A description of the individual standard and user-defined data types can be found in Data types, page 552.

Adding data type:

The "DUT" object can be added to the project via **Add ▸ Data type** in the context menu.

If it is to be directly assigned to an existing application, the application object has to be highlighted before in the Project Explorer.

If the "DUT" object is to be available project-wide, the "General module" folder has to be highlighted. Alternatively, use the mouse to drag the "DUT" object from the "PLC Objects" library to the desired position.

In the "Add Object" dialog, select a name for the new data type from (<DUT name>).

Apply the principle of inheritance for **object-oriented programming** using data types. In the "Add Object" dialog, specify whether the data type is to extend another data type that is already defined in the project. This means that the definitions of extended DUT objects automatically apply here. The extension is enabled via the "Advanced:" option and the name of the "DUT" to be extended is entered.

After the settings have been confirmed with "Finish", the "DUT" object is created in the Project Explorer. Start programming by double-clicking the "DUT" object or via **Open** in the context menu.

Declaration:

☞ The component declaration depends on the type selected, e.g. a structure, page 563, union, page 564, or enumeration, page 564.

Component declaration structure

```
TYPE <DUT name> : <DUT component declaration>
END_TYPE
```

Example:

The following shows two DUT objects defining the structures "struct1" and "struct2"; "struct2" extends "struct1".

This means that the structure "struct2" is provided with four elements (a: INT and b: BOOL, inherited from "struct1"; c and d are self-declared).

TYPE struct1

```
TYPE struct1 :
STRUCT
    a: INT;
    b: BOOL;
END_STRUCT
END_TYPE
```

TYPE struct2 EXTENDS struct1

```
TYPE struct2 EXTENDS struct1 :
STRUCT
    c: DWORD;
    d: STRING;
END_STRUCT
END_TYPE
```

2.6.6 Method (METHOD)

Object-oriented programming is supported by the possible use of methods that contain a sequence of instructions.

A method is not an independent POU, but has to be assigned to a function block, page 33,.

It can be considered as a function in the instance of the respective function block.

Interfaces, page 49, can be used for project-internal method organization for the object-oriented programming.

In this context, an interface is a collection of method prototypes. That means that a method assigned to an interface only contains a declaration part, not a implementation part. The implementation is made in the function block that implements, page 38, the interface and uses the method.

Advantage: The same method call can be used in all function blocks that implement the same interface. That means that the call can be used for a variety of purposes. Calling a method means knowing the purpose to be achieved. That is, the instructions to be actually executed in detail (implementation) to fulfill the purpose depend on the respective function block.

☞ **POU method:** the method is assigned to a function block. Apart from its declaration, it is also provided with an implementation.

Interface method: the method is assigned to an interface. It has only its declaration part. If the interface is implemented in a function block, the interface methods are implemented and become POU methods.

Adding methods: The "Method" object can be assigned to a function block via the context menu items **Add ▸ POU Method**. Alternatively, use the mouse to drag the "POU method" object from the "PLC Objects" library to the desired position.

In the "Add Object" dialog, enter a name for the method (<MethodName>) and a return type (<ReturnDataType>).

Use the … button to open the input assistance, page 98, to select the return type.

For a method assigned to a function block (POU method), select an implementation language (programming language). After the settings have been confirmed with "Finish", the editor window for the method opens.

For a method assigned to an interface (interface method), select an implementation language (programming language). The implementation is carried out when implementing the interface in the function block.

Declaration: *Syntax*

```
METHOD <method name> : <return data type>
VAR_INPUT
    x: INT;
END_VAR
```

In "Interfaces", page 49,, there is a description on the definition of interfaces that handle methods.

Method call: Method calls, page 40, are also called "virtual function calls".

☞ All data for a method is volatile data and only apply at the execution of a method (stack variables).

In the implementation part of a method, **access to the function block instance** variables is allowed.

If required, use the THIS pointer, page 42,.

Note that a locally declared variable might overwrite a function block variable.

VAR_IN_OUT or VAR_TEMP function block variables cannot be accessed in a method!

Like functions, methods can obtain additional outputs. These have to be assigned in the method call, page 40,.

Special function block methods:

- FB_init method (see also page 524):

  A method called "FB_init" is always declared implicitly, but can also be declared explicitly. It contains the initialization code for the function block as defined in the function block declaration part.
- FB_init method (see also page 524):

  If a method called "FB_reinit" is declared, it is called if an instance of the function block is copied. It starts a re-initialization of the new instance module.
- FB_exit method (see also page 526):

  If a method called "FB_exit" is declared, it is called for each instance of the function block before a download or at an online change.

  For further information, see Declaration, page 526.
- Properties (see also page 46): Properties.

Calling a method when the application is in STOP state

In the device description, page 63,, it can be defined that a certain method of a certain function block instance (a library function block) is always to be called task cyclically.

If the method contains the following input parameters, it is also processed if the active application is currently in STOP state:

Calling a method..., declaration

```
VAR_INPUT
    pTaskInfo : POINTER TO DWORD;
    pApplicationInfo: POINTER TO _IMPLICIT_APPLICATION_INFO;
END_VAR
```

The application status can be queried using "pApplicationInfo" and the corresponding instructions can be programmed; see the following example.

Calling a method..., implementation

```
IF pApplicationInfo^.state=RUNNING THEN
    <instructions>
END_IF
```

2.6.7 Property (PROPERTY)

The "Property" object (PROPERTY) can be assigned to a function block. To add the property object to the project, highlight the function block and select the context menu items **Add ▸ POU property** in the context menu. Alternatively, use the mouse to drag the "POU property" object from the "PLC Objects" library to the desired position.

In the "Add Object" dialog, enter a name for the property (<PropertyName>) and a return type (<ReturnDataType>).

Use the ... button to open the input assistance, page 98, to select the return type.

For a POU property, select an implementation language (programming language). After the settings have been confirmed with "Finish", the "Property" object is created in the Project Explorer.

Start programming by double-clicking the "Property" object or via **Open** in the context menu to open the editor.

Such a "Property" contains two special methods, page 45. They are automatically attached below the "Property" object in the object tree:

- The **Set** method is called if the property is to be accessed by writing, i.e. the name of the property is used as an input parameter.
- The **Get** method is called if the property is to be accessed by reading, i.e. the name of the property is used as an output parameter.

Interfaces, page 49, can be used for the project-internal organization of a property in object-oriented programming.

In this context, an interface is a collection of property prototypes. That means that a property that is assigned to an interface only contains a declaration part, not a implementation part. The implementation is made in the function block that implements, page 38, the interface and uses the property.

Advantage: The same property call can be used in all function blocks that implement the same interface. That means that the call can be used for a variety of purposes. Calling a property means knowing the purpose to be achieved. That is, the instructions to be actually executed in detail (implementation) to fulfill the purpose depend on the respective function block.

☞ **POU property:** the property is assigned to a function block. Apart from its declaration, it is also provided with an implementation.

Interface property: the property is assigned to an interface. It has only its declaration part. If the interface is implemented in a function block, the interface properties are implemented and become POU properties.

Example:

Function block "FB1" uses a local variable "milli". This variable is determined by the properties "second" using the methods "Get" and "Set":

Get:

```
seconds := milli / 1000;
```

Set:

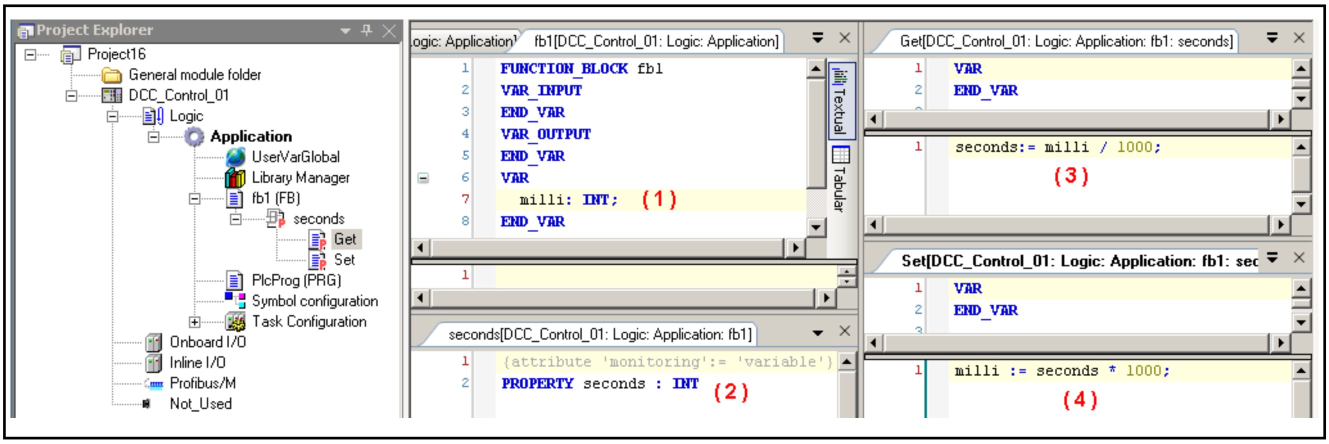
```
milli := seconds * 1000;
```

It can be written on this property (Set method), e.g. with

`fbinst.seconds := 22;` ("fbinst" is the instance of "fb1").

This property can be read (Get method), e.g. with

`testvar := fbinst.seconds;`.



```
FUNCTION_BLOCK fb1
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  milli: INT;
END_VAR
```

```
{attribute 'monitoring':= 'variable'}
PROPERTY seconds : INT
```

```
VAR
END_VAR
```

```
seconds:= milli / 1000;
```

```
VAR
END_VAR
```

```
milli := seconds * 1000;
```

(1) Function block "fb1" with the local variable 'milli'
(2) Property 'seconds' with attribute pragma
(3) "Get" of the 'seconds' property
(4) "Set" of the 'seconds' property

Fig.2-6: Example of the 'seconds' property prepared for monitoring

The figure below shows in the upper part the program with the test variable 'testvar' and the declaration of the function block instance.

The implementation includes in line 1: "Set" method and in line 2: "Get" method.

The figure below shows the monitoring including the display of the 'seconds' property value.

Also refer to

- Attribute monitoring, page 536.

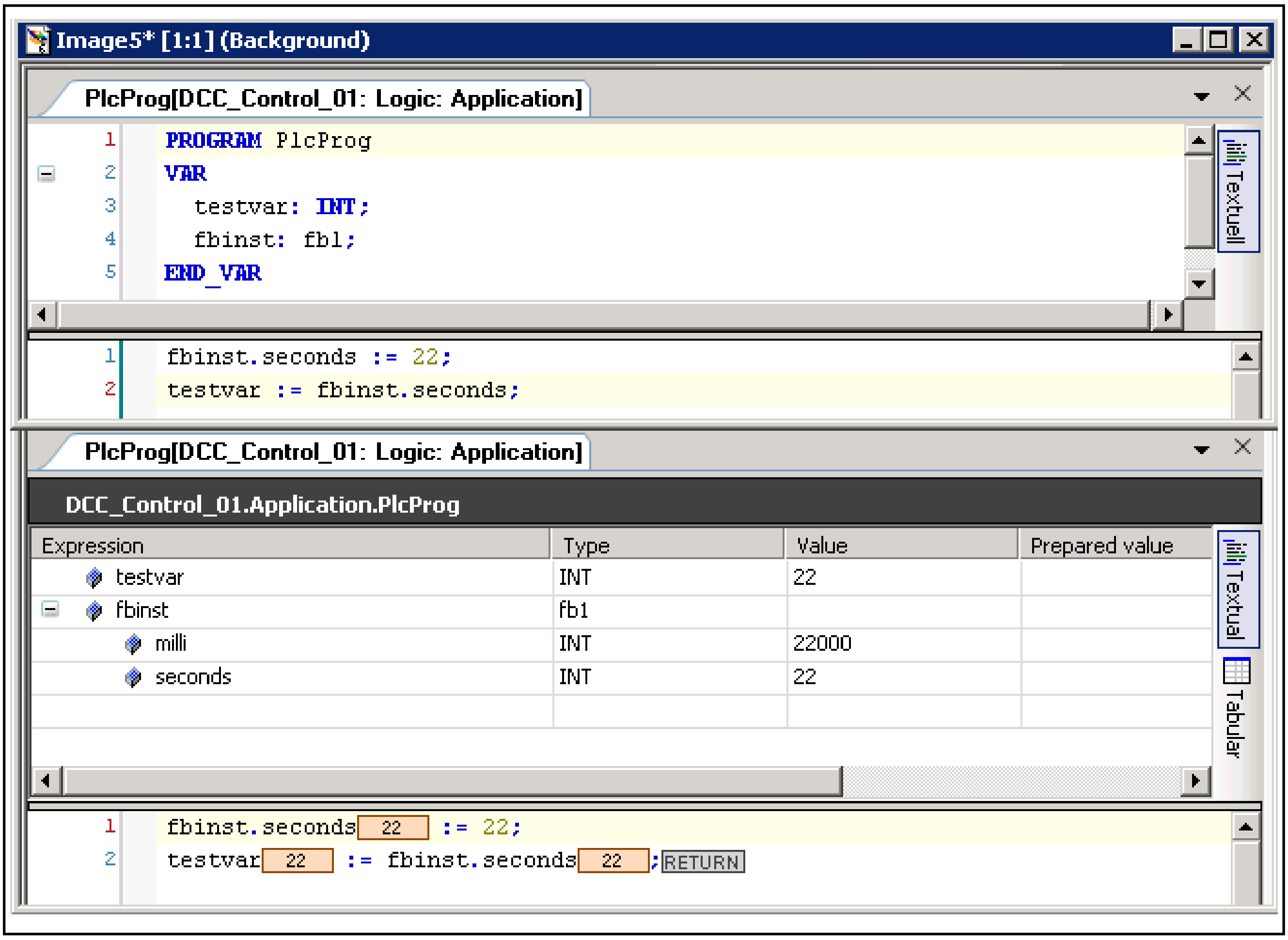


Image5* [1:1] (Background)
PlcProg[DCC_Control_01: Logic: Application]
Textual
PlcProg[DCC_Control_01: Logic: Application]
DCC_Control_01.Application.PlcProg
Textual
Tabular

```
PROGRAM PlcProg
VAR
  testvar: INT;
  fbinst: fb1;
END_VAR
```

```
fbinst.seconds := 22;
testvar := fbinst.seconds;
```

Expression	Type	Value	Prepared value
testvar	INT	22	
fbinst	fb1		
milli	INT	22000	
seconds	INT	22	

```
fbinst.seconds 22 := 22;
testvar 22 := fbinst.seconds 22 ;RETURN
```

Fig.2-7: Example of a monitoring view with the 'seconds' property

A property can also contain local variables, but no additional inputs and - in contrast to a function, page 31, or method, page 45 - no additional outputs.

2.6.8 Interface (INTERFACE)

The use of interfaces is another tool in object-oriented programming.

An "interface" is a POU, page 27 that describes a collection of method prototypes (Interface methods, page 45 / Interface properties, page 46).

Interfaces can be used to manage method prototypes implemented by function blocks, page 33,.

"Prototype" means that only declarations are included, but no implementation.

A function block can implement, page 38, an interface which means that basically all method prototypes specified in an interface have to be available in the function block.

Advantage:

The method calls are specifically defined and consistent across all function blocks that implement the same interface. In a function block, the methods have to be filled with implementation code.

- An interface is a collection of method prototypes (interface methods/interface properties).

  Declaring variables within an interface is not permitted.

  The method prototypes of the interface may only define input, output and "in/output" variables. An interface and its method prototypes has neither an implementation part nor actions.
- Variables that are declared with the type of an interface are always treated as references.
- The methods/properties assigned to a function block that implements an interface has to be named exactly as in the interface and has to contain exactly the same variables.

Adding an interface:

To add the "Interface" object to the project, highlight the "Application" node and select **Add ▸ Interface...** in the context menu.

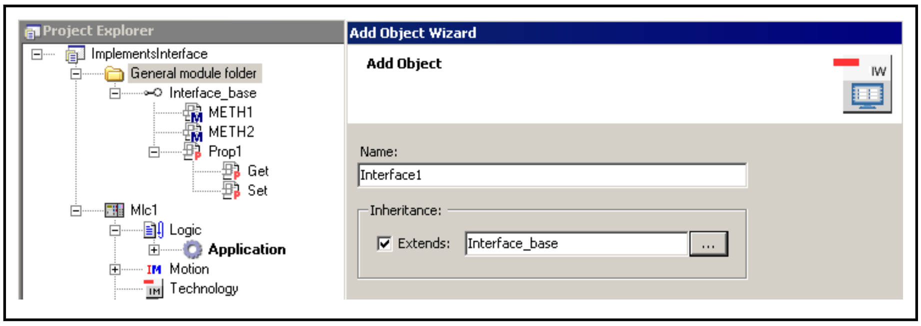
To provide the object project-wide, add it to the "General module" folder.

Alternatively, use the mouse to drag the "Interface" object from the "PLC Objects" library to the desired position.

Enter a name (<InterfaceName>) into the "Add Object" dialog of the interface.

The "Extends:" option can also be selected if the interface is supposed to extend, page 36, another one, which means that its methods definitions apply automatically in addition to those defined locally.

If the interface "Interface1" extends the interface "Interface_base", all methods/properties described by "Interface_base" are also available in "Interface1":



Name: Interface1

Inheritance:

☑ Extends: Interface_base

Fig.2-8: Example for extending an interface

After the settings have been confirmed with "Finish", the "Interface" object is created in the Project Explorer. Start programming by double-clicking the "Interface" object or via **Open** in the context menu to open the editor.

Declaration:

```
INTERFACE <interface name>
```

Example for an interface that extends another one:

```
INTERFACE <interface name B> EXTENDS <interface name A>
```

Example:

```
INTERFACE Interface1 EXTENDS Interface_base
```

Adding methods/properties

To complete the definition of the interface, the desired methods/properties have to be added. To do this, the interface object is selected in the Project Explorer and opened via **Add** in the context menu in the "Add Object" dialog. Enter the name of the return type into the "Add Object" dialog of the interface.

Use the … button to open the "input assistance", page 98, to select the return type.

Add all desired methods/properties and note that these here only contain the declarations of input, output and input/output variables, but no implementations.

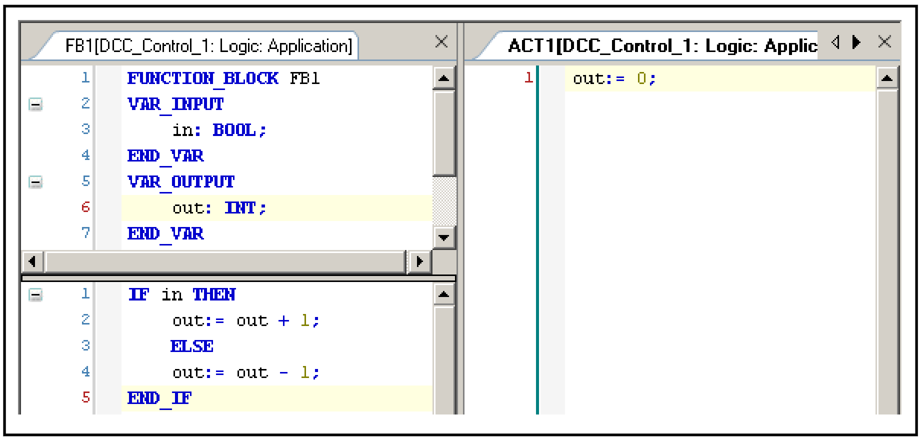
2.6.9 Action (ACTION)

Actions can be assigned to function blocks, page 33, and programs, page 29,.

An action contains further implementation code that can be written in a different language than the "basic" implementation.

Each action receives a name.

An action has no individual declarations. It operates with the data of the function block or the program to which it is assigned. It uses its input, output and local variables.


FB1[DCC_Control_1: Logic: Application]
ACT1[DCC_Control_1: Logic: Applic

```
FUNCTION_BLOCK FB1
VAR_INPUT
    in: BOOL;
END_VAR
VAR_OUTPUT
    out: INT;
END_VAR
```

```
IF in THEN
    out:= out + 1;
    ELSE
    out:= out - 1;
END_IF
```

```
out:= 0;
```

Fig.2-9: Example of a function block action

In this example, each call of the function block "FB1" increases the output variable "out" based on the value of the input variable "in".

Calling the function block action "ACT1" resets the output variable "out" zero.

In both cases, the same variable "out" is written, i.e. the variable declaration for "FB1" also applies to its action "ACT1".

Adding an action:

To add an action, highlight a function block or a program object and select **Add ▸ Action** in the context menu. Alternatively, use the mouse to drag the "action" object from the "PLC Objects" library to the desired position.

In the "Add Object" dialog, enter a name for the action and select the implementation language (programming language).

Action call:

Syntax:

`<program name>.<action name>`

or

`<FB instance name>.<action name>`

If an action is to be called in the function block to which it is assigned, specifying the action name is sufficient.

Examples for calling an action from another POU:

Declaration for all examples:

```
PROGRAM Plc_main
VAR
```

```
    Inst: Counter;
END_VAR
```

Calling the "Reset" action in another function block programmed in "IL" (instruction list):

Example in IL:

```
CAL Inst.Reset(In := FALSE)
LD Inst.out
ST ERG
```

Calling the "Reset" action in another function block programmed in "ST" (structured text):

Example in ST:

```
Inst.Reset(In:= FALSE);
Erg:= Inst.out;
```

Calling the "Reset" action in another function block programmed in "FBD" (function block diagram):

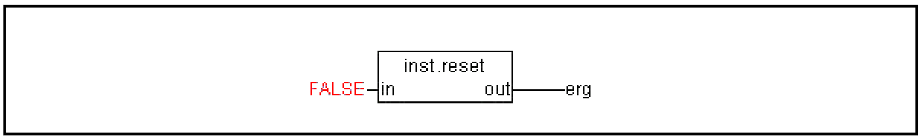


Fig.2-10: Calling the action from another function block

2.6.10 External POUs (Functions, Function Blocks, Methods)

No code is generated by the programming system for external functions, function blocks or methods.

To create an external function block, carry out the following steps:

1. Add a POU to the global "General module" folder. To do so, highlight the "General module" folder and select **Add ▸ POU** in the context menu. Use the "Finish" button to confirm your entries.

   Use **Open** in the context menu to open the editor and define the corresponding input and output variables.

☞ Local variables have to be declared in "external" function blocks that may not be declared in external functions or methods!

Likewise, note that "VAR_STAT" variables cannot be used in the runtime system!

2. Define the POU as "external" POU:

   To do this, highlight the POU object in the Project Explorer and open the "Properties" dialog in the context menu via **Properties**.

   There, open the "Build" tab and enable the "External implementation (late linking in the runtime system)" option.

An equivalent function, function block, etc has to be implemented in the runtime system.

At program download, the equivalent function block in the runtime system is browsed through for each "external" POU and integrated if found.

2.6.11 Global Variable List - GVL

Icon:

A global variable list (GVL) is used to declare global variables, page 518,.

If a GVL is present in the "General module" folder, the variables contained there are available across the entire project.

If a GVL is assigned to a certain application, the variables apply in that application.

To add a GVL, highlight the "General module" folder or the "Application" node and use **Add ▸ Global variable list** in the context menu. Alternatively, use the mouse to drag the "global variable list" object from the "PLC Objects" library to the desired position. Specify a name for the GVL in the "Add Object" dialog.

Double-click on the GVL object or use **Open** in the context menu to work in the GVL editor, page 367,.

If the target system supports the network functionality, the variables of a GVL can become network variables, page 71, and used in data exchanges with other devices in the network. To do this, the corresponding network properties have to be defined for the GVL.

2.6.12 Global Network Variable List - GNVL

Icon:

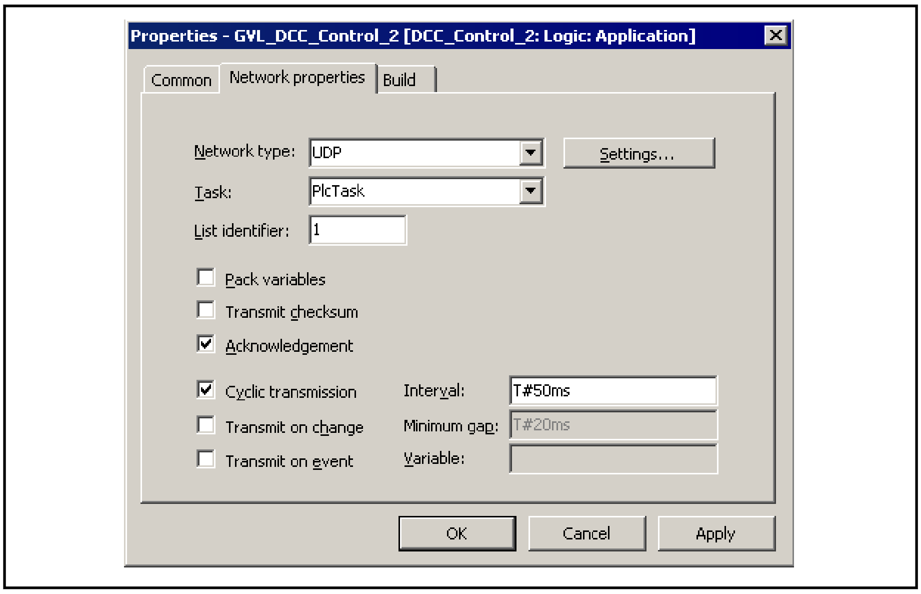
A global network variable list (global NVL, GNVL) includes variables that are defined in another network device as network variables.

☞ The data volume that can be exchanged using a global network variable list is limited (max. 255 bytes).

It is only used below an application in the Project Explorer, page 63,.

☞ A GNVL object can be added to an application if at least one GVL, page 52, with special network properties is present in another device.

Click on **Properties** in the context menu of the GVL and open the "Network properties" tab to assign network properties.


Properties - GVL_DCC_Control_2 [DCC_Control_2: Logic: Application]

Network type: UDP
Task: PlcTask
List identifier: 1

☐ Pack variables
☐ Transmit checksum
☑ Acknowledgement

☑ Cyclic transmission Interval: T#50ms
☐ Transmit on change Minimum gap: T#20ms
☐ Transmit on event Variable:

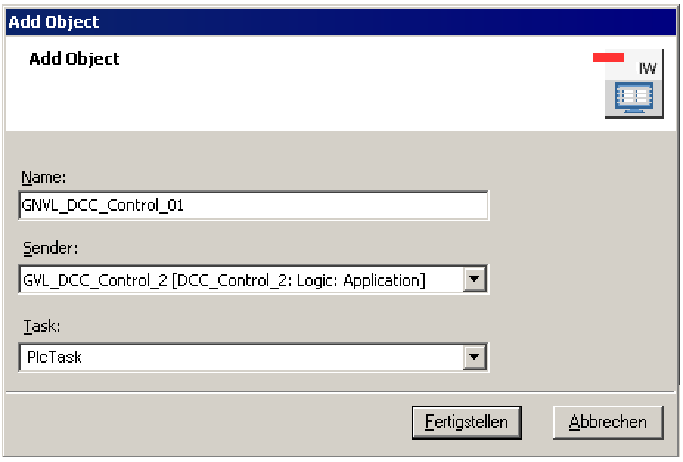
Fig.2-11: "GVL Properties" dialog, network variables

A detailed description of the setting options can be found in "Network variables", page 71.

To add a GNVL, highlight the "Application" node and use the context menu items **Add ▸ Global network variable list** in the context menu. Alternatively, use the mouse to drag the "Global Network Variable List" object from the "PLC Objects" library to the desired position.

If there are several GVLs in the network one can be chosen from the "Sender" selection list, when adding the GNVL in the Add Object dialog, page 234,. A GNVL in one device does not always correspond exactly to a GVL in another device. When creating a GNVL, a task responsible for handling the network variables, page 71, has to be defined as well.

The settings for a global network variable list can always be edited later on in the object properties.



Name: GNVL_DCC_Control_01

Sender: GVL_DCC_Control_2 [DCC_Control_2: Logic: Application]

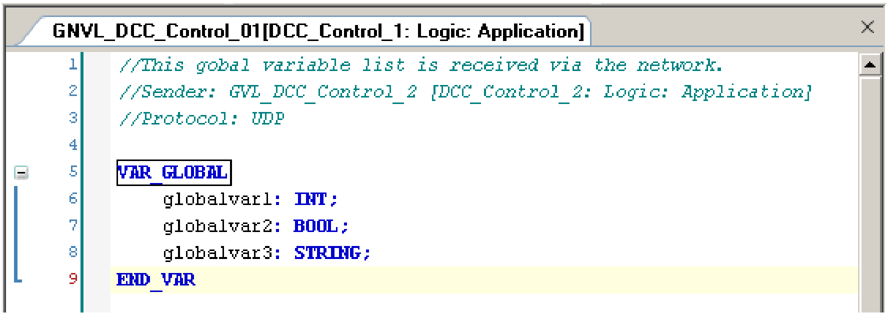
Task: PlcTask

Fig.2-12: "Add Object" dialog, global NVL

A GNVL is displayed in an editor window (NVL editor, page 382), but users cannot edit the content.

The list shows the same variable declarations as the respective GVL. If the GVL is changed, the GNVL is updated respectively.

Above the declarations, a comment is automatically added to the GNVL which contains information on the sender (device path for the device where the GVL is located), the GVL name and the protocol type.



```
//This gobal variable list is received via the network.
//Sender: GVL_DCC_Control_2 [DCC_Control_2: Logic: Application]
//Protocol: UDP

VAR_GLOBAL
    globalvar1: INT;
    globalvar2: BOOL;
    globalvar3: STRING;
END_VAR
```

Fig.2-13: Example of a global network variable list

For general information on using network variables, see "Network variables", page 71.

2.6.13 Persistent Variables (VAR PERSISTENT)

Icon:

Persistent variables are only re-initialized at control reboot or reset. In particular, they maintain their value after a download.

See also "Remanent Variables", page 519.

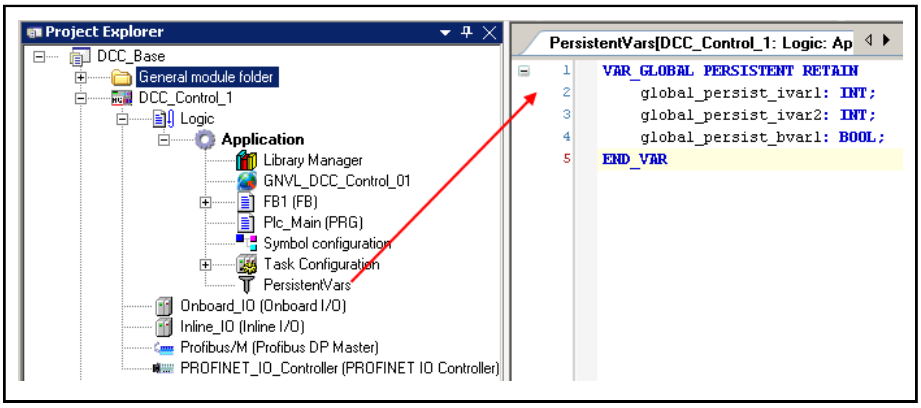
This object is a global variable list, although it only contains the persistent variables of an application. That means that the object has to be assigned to an application.

To add the "PersistentVars" object, highlight the "Application" node and use **Add ▸ PersistentVars** in the context menu. Alternatively, use the mouse to drag the "PersistentVars" object from the "PLC Objects" library to the desired "Application" node.

In the Add Object dialog, page 234,, specify the "PersistentVars" object a name.

Double-click on the "PersistentVars" object or use **Open** in the context menu to work in the editor.

A persistent variable list is created in the GVL editor, page 367,. `VAR_GLOBAL PERSISTENT` is automatically specified in the first line.



```
VAR_GLOBAL PERSISTENT RETAIN
    global_persist_ivar1: INT;
    global_persist_ivar2: INT;
    global_persist_bvar1: BOOL;
END_VAR
```

Fig.2-14: Example of a persistent variable list

Persistent variables are only re-initialized when the control is rebooted or reset.

2.6.14 Text List

Text lists are used to manage texts that can be displayed in the visualization. These can be error messages that output a defined text from the text list when an error occurs for example.

The "Text List" object is assigned and managed globally in the "General module" folder or in an application. It is the basis for

1. **Multilingualism (multilanguage support)** for "static" and "dynamic" texts and tooltips in visualizations and in handling alarms and
2. **Dynamic text change**.

Text lists can be exported and imported, page 60,.

Export is required if a language file, page 628, has to be provided in XML format; see Export and Import Text Lists, page 60.

Text list formats include text format and XML. Support of "Unicode" can be activated, page 59,.

Each text list is uniquely defined using its namespace. It contains character strings that are uniquely referenced within the list by an ID (identifier, index) and a language abbreviation (any respective character string). The text list to use is specified in the configuration of a visualization element.

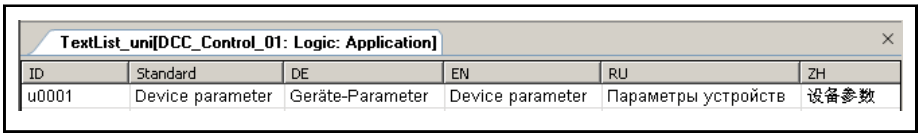
The respective text is then displayed in online mode based on the language just set in the programming system. Each text in the list is at least available in the "standard/default" language. If there is no entry in the text list that matches the language currently set in IndraLogic, the entry defined as default is used. Each text can contain formatting specifications, page 60,.

Structure of a text list:

ID (Index)	Standard	<Language 1>	<Language 2>	.... <Language n>
<unique character string>	<Text abc in the default language>	<Text abc in language 1>	<Text abc in language 2>	...
<unique character string>	<Text xyz in the default language>	<Text xyz in language 1>	<Text xyz in language 2>	...

☞ A "Text Lists" object can include the characters of any language.

With regard to further processing, decide whether the respective characters can be processed as plain text or Unicode.



ID	Standard	DE	EN	RU	ZH
u0001	Device parameter	Geräte-Parameter	Device parameter	Параметры устройств	设备参数

Fig.2-15: Example for a Unicode "Text Lists" object

Text list types

There are two types of text that can be used in visualization elements and there are two types of text lists respectively:

1. GlobalTextList for static texts:

   In contrast to dynamic texts, static texts in a visualization cannot be exchanged in online mode using a variable. Only the local country code can be switched as described above.

   A static text is assigned to a visualization element via Property, page 451, "Text" or "Tooltip" of the "Texts" category.

   As soon as the first static text is defined in a project, a text list with the name "GlobalTextList" **is automatically created** as object in the "General module" folder. First, the list contains the defined text in the "Default" column and an automatically assigned integer (beginning with 0) as text "ID". Other static texts are then added as soon as they are defined in the properties of a visualization element. The ID number is then each time incremented by 1.

   If a static text is entered within a visualization element (e.g. if the text "Example" is entered below the "Texts" "Text" square), it is searched for this text in the GlobalTextList.

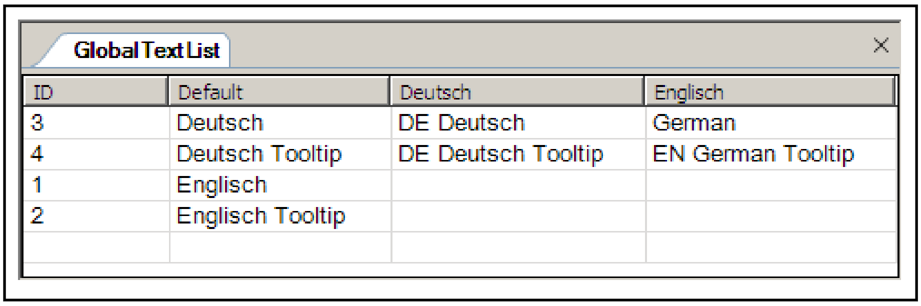
   - If the text is found (e.g. ID "4711", text "Example"), the value 4711 is entered in the element in an internal "TextId" variable. This creates a connection between the element and the line within the GlobalTextList.
   - If the text is not found, a new line is entered in the GlobalTextList (e.g. ID "4712", text "Example"). The value 4712 is applied to the element in the internal "TextId" variable.

This means that for each modification of a static text within the visualization there might also be a modification within the GlobalTextList.

A global text list can also be created explicitly in the visualization editor context menu via **Create Global Text List**.

Alternatively, a text list can also be created in the main menu via **VI Logic Visualization ▸ Create Global Text List**.

"GlobalTextList" is a special text list, in which the identifiers (IDs) for the individual text entries are handled implicitly and users cannot edit them in IndraLogic. This list cannot be deleted. But it can be exported, page 60, edited externally and then re-imported, page 60,. In this case, it is checked during the reimport whether the identifiers still match those specified in the configuration of the respective visualization element. If necessary, an implicit update of the IDs is made in the element configuration.



Global Text List

ID	Default	Deutsch	Englisch
3	Deutsch	DE Deutsch	German
4	Deutsch Tooltip	DE Deutsch Tooltip	EN German Tooltip
1	Englisch		
2	Englisch Tooltip		

Fig.2-16: Example of a GlobalTextList

2. **Text list for dynamic texts:**

   Dynamic texts can be exchanged dynamically in online mode (see above). The text index (ID), a character string, has to be unique within the text list used and, in contrast to a "GlobalTextList" has to be specified by the user. Another difference from "GlobalTextList" is that text lists for dynamic texts have to be created explicitly. To add a text list, highlight the "Application" node and select **Add ▸ Text List** from the context menu. Alternatively, use the mouse to drag the "text list" object from the "PLC Objects" library to the "Application" node.

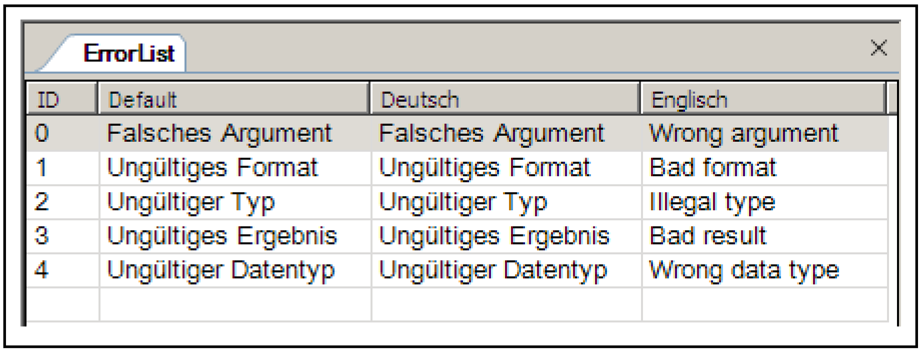
   In the Add Object dialog, page 234,, specify the text list a name and confirm with "Finish".

   All dynamic text lists available in the project are provided when configuring the property, page 451 , "Dynamic Texts" / "Text List" of a visualization element.

   To enter the name of a text list and a text index (ID) from the list, the corresponding text is displayed in online mode. If the ID is not entered as absolute, but using a project variable instead, the text can be switched dynamically using this variable.

☞ In contrast to "GlobalTextList", the IDs are not automatically checked and updated when dynamic text lists are re-imported!

Ensure that the index IDs are not changed when the exported lists are edited!



ID	Default	Deutsch	Englisch
0	Falsches Argument	Falsches Argument	Wrong argument
1	Ungültiges Format	Ungültiges Format	Bad format
2	Ungültiger Typ	Ungültiger Typ	Illegal type
3	Ungültiges Ergebnis	Ungültiges Ergebnis	Bad result
4	Ungültiger Datentyp	Ungültiger Datentyp	Wrong data type

Fig.2-17: Example of a dynamic text list called "ErrorList"

The following is a description of an example.

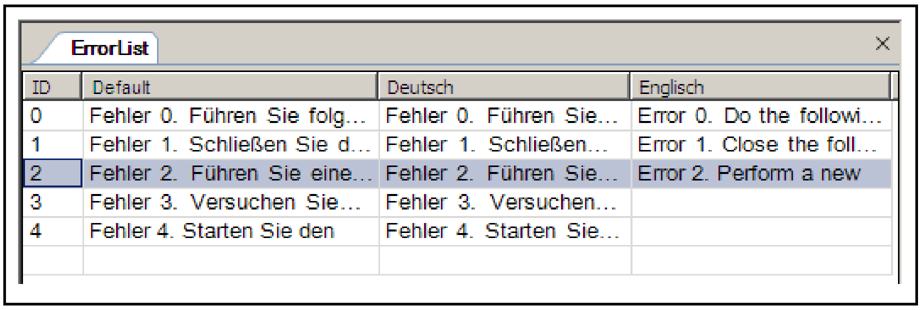
Example:

Dynamic text list

Configure a visualization element that is supposed to output the corresponding error messages when an error occurs. The application processes errors that are identified via numerical IDs - assigned to an integer variable "ivar_err".

Proceed with the following steps:

1. Provide a dynamic text list called "ErrorList" where the error message texts for the error IDs "0" to "4" are defined in "German", "English" and "Default language". See the following figure.

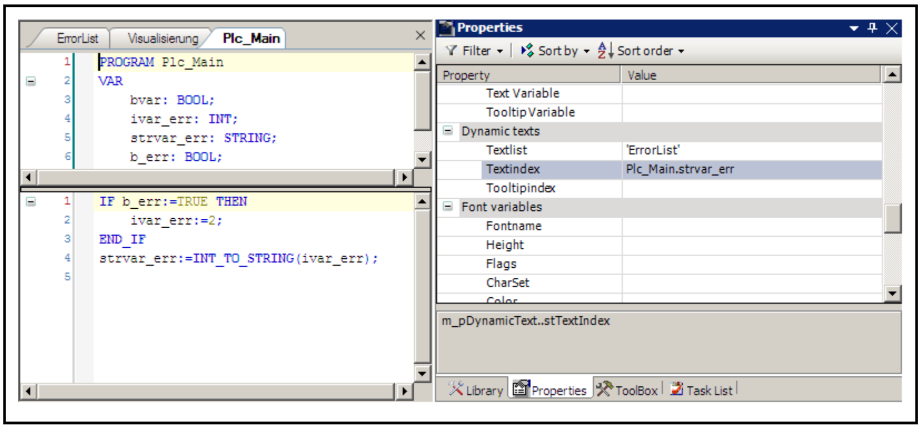


ID	Default	Deutsch	Englisch
0	Fehler 0. Führen Sie folg...	Fehler 0. Führen Sie...	Error 0. Do the followi...
1	Fehler 1. Schließen Sie d...	Fehler 1. Schließen...	Error 1. Close the foll...
2	Fehler 2. Führen Sie eine...	Fehler 2. Führen Sie...	Error 2. Perform a new
3	Fehler 3. Versuchen Sie...	Fehler 3. Versuchen...	
4	Fehler 4. Starten Sie den	Fehler 4. Starten Sie...	

Fig.2-18: "ErrorList" example

2. Declare a STRING variable, e.g. `strvar_err` to use error IDs in a visualization configuration.
3. To assign the value of `ivar_err` to the variable `strvar_err`, use `strvar_err:=INT_TO_STRING(ivar_err);`

Now, "strvar_err" can be used as a text index parameter in the configuration of the "dynamic texts" properties of a visualization element. The element displays the corresponding error message in online mode.



```
PROGRAM Plc_Main
VAR
    bvar: BOOL;
    ivar_err: INT;
    strvar_err: STRING;
    b_err: BOOL;
```

```
IF b_err:=TRUE THEN
    ivar_err:=2;
END_IF
strvar_err:=INT_TO_STRING(ivar_err);
```

Property	Value
Text Variable	
Tooltip Variable	
Dynamic texts	
Textlist	'ErrorList'
Textindex	Plc_Main.strvar_err
Tooltipindex	
Font variables	
Fontname	
Height	
Flags	
CharSet	
Color	

m_pDynamicText..stTextIndex

Fig.2-19: Example: Project variables processing the error ID. Configuration of a visualization element ("Properties" dialog) to output the error messages.

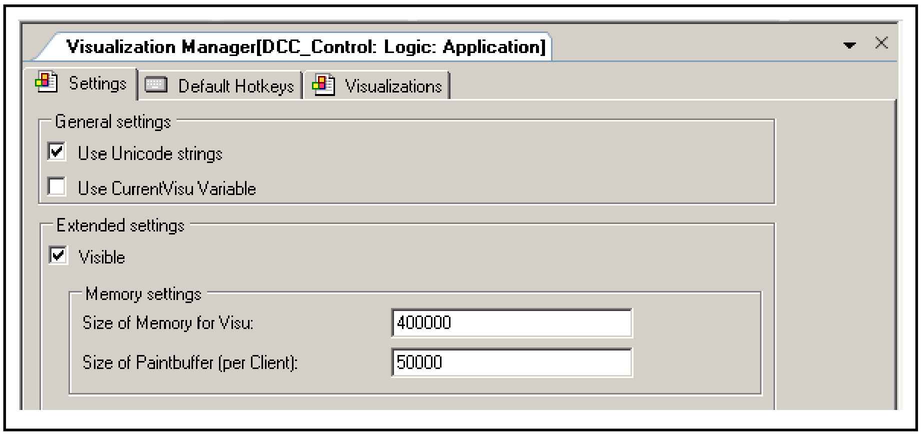
Creating a text list

- If the "Text List" object is to be assigned to an application, highlight the "Application" node in the Project Explorer. If the "Text List" object is to be available project-wide, add it to the "General module" folder. To do this, highlight the "Application" node or the "General module" folder and select **Add ▸ Text List** in the context menu. Alternatively, use the mouse to drag the "Text List" object from the "PLC Objects" library to the respective position.
- To create the text list "GlobalTextList" for static texts, enter any text into the "Properties" category "Texts" at the property "Text" when configuring a visualization element. This procedure automatically generates the list. Alternatively, generate a text list for static texts via **Create global text list** in the context menu. The "Create global text list" command is also available via **VI Logic Visualization** in the main menu.
- To open an existing text list for editing, use **Open** in the context menu or double-click on the object entry. See also Structure of a Text List, page 56 to see how a text list is structured.
- To edit a field in the text list, proceed as follows:
  1. Click on the field to select it.
  2. Click it again or press the <space bar> to open an input field.
  3. Enter any character string and close the field with <Enter>.
  4. Use the arrow keys to move to the next or previous field.

Support for Unicode format

To support the Unicode format, proceed as follows:

1. Open the visualization manager editor window (double-click the entry in the Project Explorer or use the context menu).
2. Enable the "Use Unicode strings" option.


Visualization Manager[DCC_Control: Logic: Application]

General settings

☑ Use Unicode strings

□ Use CurrentVisu Variable

Extended settings

☑ Visible

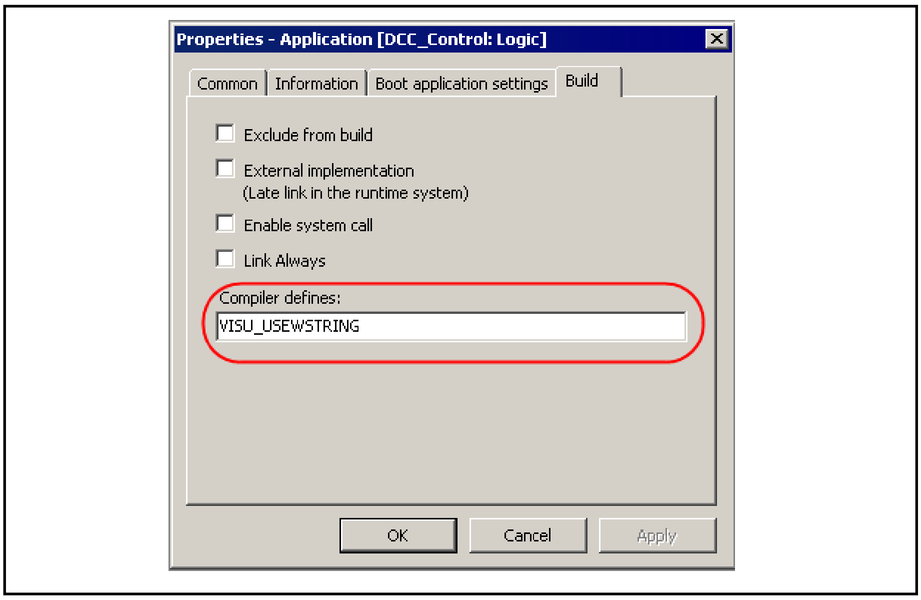
Memory settings

Size of Memory for Visu: 400000

Size of Paintbuffer (per Client): 50000

Fig.2-20: "Visualization Manager" editor window

3. Use "OK" to confirm your entries.
4. Enter a special compiler instruction for the application. To do this, highlight the application in the Project Explorer and select **Properties** in the context menu. Click on the "Build" tab and - in the "Compiler defines:" field - enter "VISU_USEWSTRING":


Properties - Application [DCC_Control: Logic]

□ Exclude from build

□ External implementation
(Late link in the runtime system)

□ Enable system call

□ Link Always

Compiler defines:
VISU_USEWSTRING

Fig.2-21: "Application" properties, entering compiler definition

5. Use "OK" to confirm your entries.

Exporting and importing text lists

Static and dynamic text lists can be exported in text or XML format.

Exported files can also be used to add external texts, e.g. from a compiler. Note that only files in text format (*.txt, *.csv) can be imported again.

Detailed descriptions of the corresponding commands can be found in:

Menu items:

- Import/Export text lists, page 288

Formatting texts

The texts can contain formatting specifications (%s,%d,...) that enable to return current variable values in the text for example. The allowable formatting definitions can be found in "Visualization," page 628,

The text definition is evaluated in the following sequence in order to display the respective current text:

1. The text to be evaluated is determined by list name and text ID.
2. If the text contains formatting specifications, they are replaced by the value of the corresponding variable.

Subsequent delivery of compilations

Adding the "GlobalTextList.csv" (subsequent file) to the directory used for loading text lists allows compilations to be delivered subsequently. When the boot project is started, it is determined if a subsequent file exists and the compilations are compared with the text list files. Both new and modified compilations are applied to the text list files.

Afterwards, the "GlobalTextList.csv" file is marked as loaded. This way, subsequent delivery of texts only affects the start-up time for the boot project once.

Intellisense for text input

A text template file can be specified using the visualization options. All texts in the "Default" column of this file are included in a list that is used for "TextIntellisense". A file that was previously generated with the export command can be used as a text template file.

Multiple user mode

By using source code management in IndraLogic 2G, it is possible for several users to work on a project simultaneously. Note the following points:

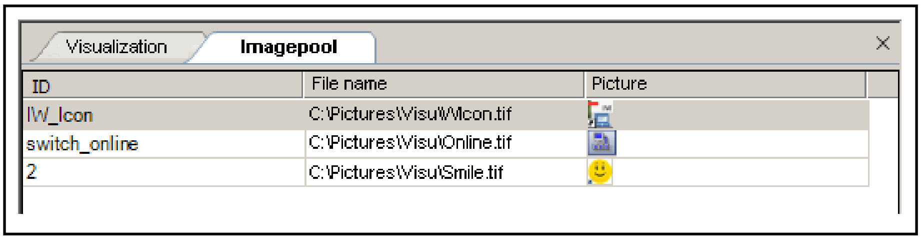
- If a static text is modified within a visualization element, the visualization has to have write access and perhaps the GlobalTextList as well (see GlobalTextList). If the GlobalTextList does not have write access, none of the texts in the visualizations should be modified. But if they are modified, the text IDs might no longer match the texts in a visualization element.
- The Check Visualization Text IDs, page 293, command can determine these kinds of errors in all visualizations.
- The Update Visualization Text IDs, page 293, command can automatically correct these error cases. To do this, all visualizations with error cases and the GlobalTextList have to have write access.

A delivery that with error cases can lead to the wrong texts appearing in the visualization if the language is switched. If no cases of error are reported for a project, the language file can be compiled and delivered subsequently.

2.6.15 Image Pool

Image pools are tables that define the path, a preview and an identifier (ID, character string) for each image file. By entering the ID and - for unique addressing - the name of the image pool, an image file can be referenced if it is used in a visualization in the project, for example (in the configuration of the properties of a visualization element, see Using Image Files from Image Pools, page 62).

Structure of an image pool:



ID	File name	Picture
IW_Icon	C:\Pictures\Visu\IWIcon.tif	
switch_online	C:\Pictures\Visu\Online.tif	
2	C:\Pictures\Visu\Smile.tif	

Fig.2-22: Example of an image pool

"ID":

Identifiers as character string, e.g. "IW_Icon", "switch_online", "2"); unique referencing of an image file is achieved by combining the name of the image pool with the image file ID (e.g. "List1.basic_logo").

"File name":

Image file path (e.g. "C:\Pictures\Visu\Online.tif")

"Image":

Image preview

Creating and editing an image pool

One project can include several image pools.

If the project does not yet include an image pool, as soon as the first "image" element is added to the visualization and an ID (static ID) is entered in the element properties, an image pool is automatically created with the default name "GlobalImagePool" along with an entry for the selected image file. "GlobalImagePool" is a global pool that is always searched first when an image file is is to be used. Individually named pools can also be used.

Image pools can be created manually as follows:

- via the main menu using **VI Logic Visualization ▸ Generate Global Image Pool**
- via the context menu via **Generate Global Image Pool** if the mouse is in the visualization editor
- via the context menu below an application or the "General module" folder via **Add ▸ Image Pools...**.
- with the mouse by dragging and dropping from the "PLC Objects" library.

To add an image file manually to an image pool, place the focus in the **ID** field of the first empty line in the pool table, press the <space> bar to open an input field and enter any character string as ID. If the ID entered is already used in the table, a numerical digit is automatically added, beginning with 0 and incremented by 1 each time the ID is copied. Then place the cursor in the "File Name" field. Here, use the <space> bar and the ... key to open the "Image Selection" dialog to enter the path of the desired image file.

Using image files from image pools

Note the following if the ID of an image file is present in several image pools:

- Search order:

  When an image is selected that is managed in the "GlobalImagePool", the name of the image pool does not have to be specified. The search order for image files corresponds to that for global variables:

  1. "GlobalImagePool" in the "General module" folder
  2. Image pools that are assigned to currently active applications
  3. Image pools that, in addition to "GlobalImagePool", are located in the "General module" folder
  4. Image pools in libraries

- Unique access:

  Address the desired image file directly and uniquely by using the ID of the name of the image pool as prefix.

  Syntax:

  `<name of image pool>.<image id>` (e.g. for the example shown in the figure above: "imagepool.IWIcon").

1. Using an image file in a visualization element, page 446, of the type "Image":

   If an "Image" element is added to a visualization, either a static or dynamic element can be specified, where the dynamic element can be exchanged based on a variable in online mode:

- Static images:

  Enter the image file ID or the name of the image pool + ID in the configuration of the element ("Static ID" property). Note here the information on Search Order and Unique Access, page 62 (see above).

- Dynamic images:

  Enter the variable that defines the image file, e.g. "Plc_Main.imagevar" into the configuration of the element ("Bitmap ID variable" property).

2. Using an image file for the visualization background, page 304:

   An image can be entered in the background definition, page 304, for a visualization. As described above for a visualization element, the image file can be entered using the name of the image pool and the file name.

2.6.16 Visualization

Information on the visualization in IndraLogic 2G and on the visualization editor can be found in Visualization, page 625,.

2.6.17 POUs for Implicit Checks

These special POUs can be added to an application to equip it with available implicit monitoring functionalities. At runtime, they check for array limits or subsection types, the validity of pointer addresses or division by 0.

In the "Add Object" dialog in the "POUs for Implicit Checks" category, the following functions are available:

- CheckBounds, page 560
- CheckDivInt, page 571
- CheckDivLInt, page 571
- CheckDivReal, page 571
- CheckDivLreal, page 571
- CheckRange, page 566
- CheckRangeUnsigned, page 566
- CheckPointer, page 557

After adding a POU for monitoring purposes, it opens in the editor that corresponds to the selected implementation language. A suggestion for the implementation is made in the ST editor and can be adapted as desired.

To prevent multiple linking, a monitoring function that has already been added can no longer be selected in the "Add Object" dialog. If all types of monitoring functions have already been added, the entire "POUs for Implicit Checks" category is removed from the dialog.

☞ To maintain the functionality of monitoring functions, the declaration part may not be modified.

Also refer to

- Floating point Exceptions in the PLC Program, page 335

2.7 Devices in the Project Explorer

All objects required for executing an application, page 66, (a control program) on a device (control, PLC) are managed in the Project Explorer in a tree structure.

These objects are also referred to as "Resource" objects. Device objects, application objects, task configuration and task are "Resource" objects.

Programming objects such as individual POUs, global variable lists and library managers can be managed below a control in the Project Explorer and can then be used only for your application.

☞ Globally applicable programming objects are managed in the "General module" folder in the Project Explorer.

To convert device references when opening projects created in another format, see

Data Transfer, page 115.

General information on the device object tree

- The root node in the Project Explorer is the name of the project given when a new project is created. <project name>.
- The configuration trees for the "control configuration " and "task configuration" that were handled in separate windows in IndraLogic 1.x are integrated into the device object tree in IndraLogic 2G. The configuration of the devices and task parameters is carried out in separate editor dialogs.

  See Task Configuration, page 67, Control Configuration, page 68.

  This way, the structure of the hardware environment to be controlled is illustrated in the device object tree with the corresponding arrangement of objects and it is possible to superimpose a heterogeneous system of controls with multiple networking and underlying field buses.

See the rule for arranging objects below the device node in the following.

- A "Devices" object represents a certain hardware (target system).

  Examples: control device, drive, I/O module, monitor.
- An entry in the Project Explorer shows the icon, the symbolic device name (which can be edited in the tree) and the device type behind it (= device name as defined in the device description).
- There are "programmable" devices and devices that can be "parameterized". The device type determines the possible insertion position below the device node and the selection of objects that can be added below the device.

  Programmable devices automatically obtain an additional "Logic" node below the device object. The objects required for programming the device (visualizations, GVLs, text lists, etc.) can then be added below this node:

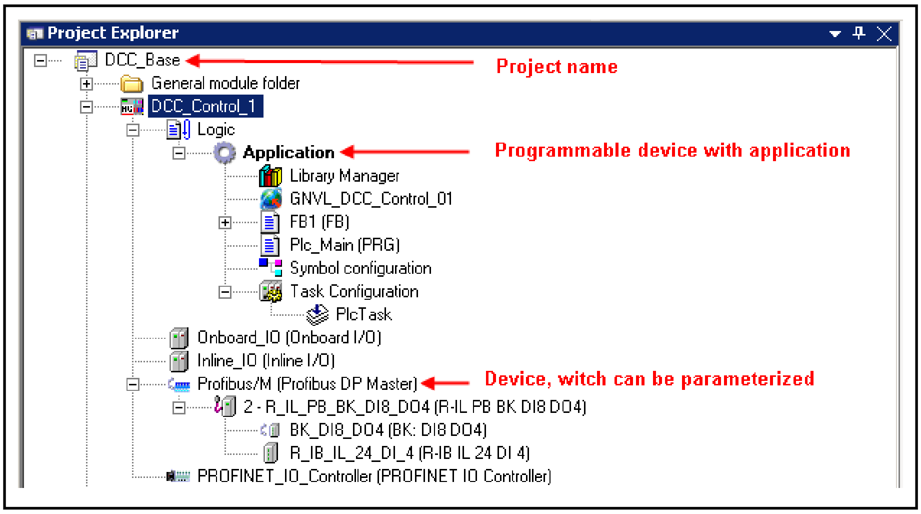


Fig.2-23: Devices in the Project Explorer

- In a project, page 26,, one or more programmable devices can be configured irrespective of the manufacturer or type.
- The configuration of a device with respect to communication, parameters or I/O mapping is carried out in the "Device Editor dialog" that can be opened by double-clicking on the device entry (a detailed description can be found in Device Editor, page 331,).
- In "online mode", an icon in front of a device entry indicates whether the device is currently connected 🔄 or not connected ⚠. Additional diagnostic information can be found in the respective logbook, page 332, in the "Status" category.

See the following notes and rules for arranging, page 65, objects in the device.

Arranging and configuring objects in the device:

- The object types that can be added depend on the currently selected position in the tree.

  Example:

  Modules for a DP PROFIBUS slave cannot be inserted without inserting the respective slave object before.

  Applications cannot be added below non-programmable devices.

  Moreover, the only devices that can be selected for insertion are those that have been correctly installed in the local system and are suitable for use in the currently selected position in the tree.

  To add an object, highlight the position in the tree where the object is to be inserted and use **Add ▸ <ObjectType>** in the context menu. Alternatively, insert objects using the mouse to drag them from the library to the corresponding position.

☞ Programmable devices can only be added from the "Drive and Control" library. Highlight the desired device (e.g. IndraLogic XLC L65) and use the mouse to drag it to the root node.

- The arrangement of objects below an application is sorted alphabetically and by object type. It is not possible to change to any position. On the other hand, the objects below (e.g. actions, transitions) can be positioned as desired by using the mouse to drag them to the corresponding position.

- A device already added to the Project Explorer can be replaced by another version of the same device or by a device of another type. The configuration tree indented below the device can be retained as much as possible. To do this, highlight the device node and select **Update Device** in the context menu.
- Devices can be installed and uninstalled in the Device Database... dialog. The installation is based on "device description files" in XML format. The default extension for a valid description file is ***.devdesc.xml**. However, bus-specific description files, e.g. *.gsd files (PROFIBUS) can also be installed using the "Device Repository" dialog.
- A device is added as node in the Project Explorer. If these are defined in the device description, the subnodes are automatically added as well. In turn, a subnode can also be a programmable device.
- Further devices that are installed in the local system and are provided in the library can be added below a "Device" object. For each level, the programmable devices are arranged first (PLC logic), then the rest of the types.
- An application, page 66, is automatically added below the "Logic" node (symbolic node for programmable devices). Only one application is permitted for each device. Then, the other objects required for programming, e.g. data types (DUT), global variable lists (GVL), visualizations, etc. can be attached below an application. A task configuration is automatically added below every application and the corresponding program calls are defined there.

2.8 Application

Icon: ⚙

- An "application" includes several objects required for executing a certain "instance of the control program" on a certain device, page 63, (control, PLC). To do this, the global objects that are managed in the "General module" folder can be instantiated and assigned to a device. This corresponds to the object-oriented programming.

  However, POUs that are purely application-specific can also be used by the application.
- An application is represented by the "Application" object in the device, page 63, (i.e. programmable device) that is automatically added below a Logic node, page 63, with the programmable device. The objects that make up the "resource set" of the application are automatically added below an application.
- An important object for each application is the task configuration, page 67, to check the execution of a program (POU instances or application-specific POUs).

  In addition, objects such as global variable lists, libraries, etc. can be directly assigned to the application, which, in contrast to the objects managed in the "General module" folder, can only be used by this application and its "child" objects; see Arranging and Configuring Objects in a Device, page 65, for the rules.

☞ Note that several applications cannot be used with the same device at present.

- Note that the application to work with in online mode has to be set as "active application", page 237,.

To do this, highlight the application and select **Set as Active Application** in the context menu. The active application is displayed in the Project Explorer in bold; see the following figure.

- When logging in, page 127, with the application on the control, it is checked which applications are currently on the control and if the application parameters on the control match those in the current project.

  Corresponding messages are output and applications can be deleted on the control.
- Note the "Applications" tab for the Device editor, page 331 to see which applications are currently present on a device and to delete these from the target system.

  Applications can also be displayed that are not represented by a separate object in the device; see "Symbol Configuration", page 306.

2.9 Task Configuration

The task configuration defines one or multiple tasks to control and execute the application program on the control.

It is a required "Resources" object for an application , page 66 , and is automatically added below an application.

A task can call an application-assigned program or a program managed globally in the "General module" folder.

A task configuration can be edited in the task editor, page 428, although the available options depend on the target system.

In online mode, the "task editor" provides a monitoring view with information on cycles, cycle times and task status.

Profiling (monitoring)

☞ The availability of the "profiling (monitoring)" functionality depends on the device type.

Important notes for multitasking systems:

True, pre-emptive multitasking realized on some systems. In this case, observe the following:

As in IndraLogic V1. x, all tasks share a process image.

Reason:

An individual process image for each task would lower performance. The process image can only be consistent with one task. For this reason, when creating a project users are responsible for making sure that in case of conflicts, the input data is copied to a secure area; the same applies to outputs. Possibilities for solving consistency and synchronization problems are provided by the function blocks in the "SysSem" library for example.

Consistency problems can also occur in multitasking systems when other global objects (global variables, function blocks, field buses) are accessed if the objects exceed the data width of the processor (structures or arrays that form a logical unit). A solution is available in the function blocks in the "SysSem" library.

2.10 Communication

2.10.1 Communication, General Information

This section includes information on the following subjects:

- Configuration of a control, page 68
- Data server, page 71

- Network variables, page 71

2.10.2 Control Configuration

The "control configuration" illustrates the target hardware in the programming system in order to make the inputs and outputs and the control parameters and the applications field bus devices accessible. In addition, it enables the available device parameters to be displayed.

The "control configuration" tree that is handled in its own editor in IndraLogic V1. x, is integrated below the "Devices" node in the Project Explorer in IndraLogic 2G where the other objects necessary for running an application on a target system are also arranged. The map of the current hardware configuration below the "Devices" node is simplified in the default device editors by a scan functionality. Information on the device can be found in "Devices in the Project Explorer", page 63,.

The control inputs and outputs for project variables are assigned either with the "AT Declaration", page 512, in the declaration editor or in the "I/O Mapping" dialog of the respective field bus which provides dialogs to configure a device. If a new control is added to the Project Explorer, a preset enters the application automatically added as "mapping application".

2.10.3 Communication in the Control Link via Gateway

The PLC communication from an engineering PC to one or multiple control devices (IndraLogic XLC, IndraMotion MLC or IndraMotion MTX) is always established via an IndraLogic gateway.

the following figure shows the communication relation.

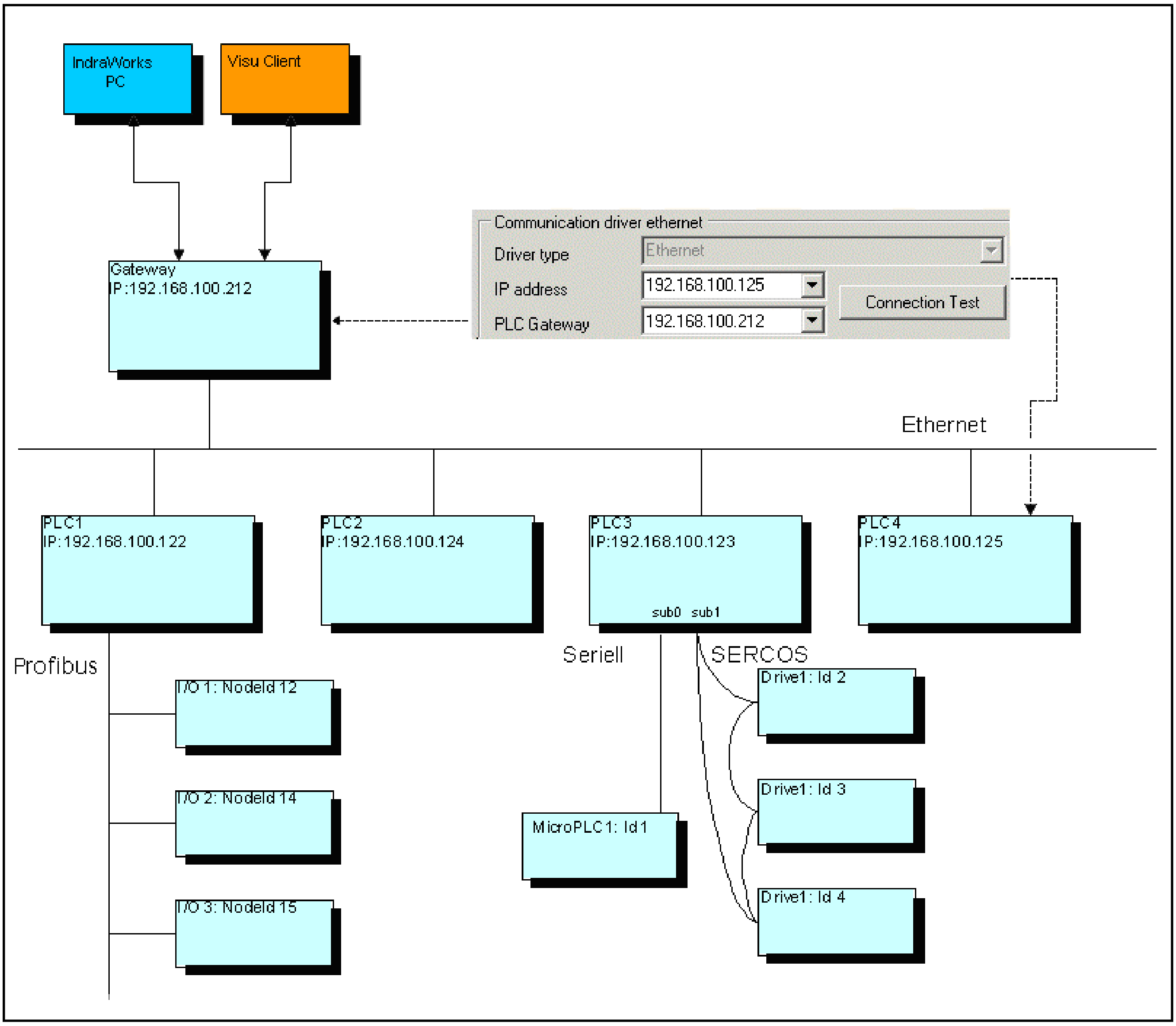
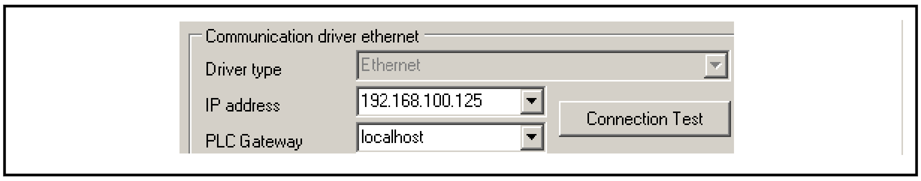


Fig.2-24: Communication relations

The required settings take place in the respective device wizard.

An IndraLogic gateway can either run on the own engineering PC (setting: localhost or 127.0.0.1) or somewhere in the control networks (setting: IP address of the gateway PC).


Communication driver ethernet

Driver type: Ethernet

IP address: 192.168.100.125

PLC Gateway: localhost

Fig.2-25: Setting IP and Gateway address

After a successful connection test, the IP address to the control is displayed (here 192.168.100.125). The TCP address can be directly set in the visualization device and in the OPC server.

If the PLC communication is not successful, investigate the following error causes. The error recovery is exemplarily shown under Windows XP. The solution can be different for other Windows versions.

☞ A PLC communication to the device can only be established via the device-engineering interface!

Checking connection to gateway

If there is no gateway connection, the following message is output after a connection test:

- IndraLogic gateway not found.
- IndraLogic gateway: No communication. Gateway offline?

Ensure that the gateway server is running.

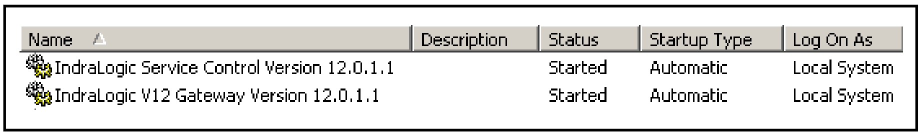
The gateway server is automatically started as service at system start.

Check whether the icon appears in the toolbar at the lower margin of the screen.

If the symbol appears as follows: , the gateway is stopped.

The program icon provides start and stop commands in a context menu opened via the right mouse button. The service can now be started and stopped at any time.

Ensure as well that the services <IndraLogic Service Control> and <IndraLogic V12 Gateway> run in the Windows **Control Panel** under **Administrative Tools - Services**.



Name	Description	Status	Startup Type	Log On As
IndraLogic Service Control Version 12.0.1.1		Started	Automatic	Local System
IndraLogic V12 Gateway Version 12.0.1.1		Started	Automatic	Local System

Fig.2-26: List of services (excerpt)

Only one <IndraLogic Service Control> service and <IndraLogic V12 Gateway> service may run at a time.

Close possible further services and set the AutoStart type of these services to "Disabled".

If the connection test keeps on failing, change the settings for the IndraLogic gateway from <localhost> to <127.0.0.1> and subsequently set your own IP address.

Alternatively, the gateway of another engineering PC can be used by entering the IP address below the IndraLogic gateway in the device wizard.

Checking connection to device

If there is no device connection, the following messages are output after a connection test:

- No connection to the device. Device offline?
- No connection to the device. Device offline? Error: <>

Check the control first. Did the control crash? Reboot the control.

Is a firewall enabled? Especially if the control is running on a PC. Disable the firewall.

Were changes made at the Gateway.cfg or GWClient.cfg file? Use the original files without modifications.

Was CoDeSys installed parallely to IndraWorks? Close the iCoDeSys simulation.

Incorrect response from the device

If the communication with the device is possible at all, device type or device version might not match. In this case, the following messages are output at a connection test:

Different device types:

- The selected target system does not match the connected device.

  ID mismatch:

  requested=1001 0003, online=1001 0103

You are on a third-party device type, e.g. in the device wizard of an MLC L65 device. But the device is an MTX, XLC or MLC with another hardware design (L25, L45).

Different device versions:

- The selected target system does not match the connected device.

  Version mismatching:

  requested=12.6.0.0, online=12.5.2.0

Update the device firmware via the IndraWorks firmware management.

2.10.4 Data Server

A "data server" can be added to an application in order to use remote data sources. In this context, "remote data sources" means that variables ("data items") defined and used in other devices or in the local application can be used.

In contrast to data exchange across network variables (broadcasting), the data server establishes point-to-point connections. Depending on the access flag of the data connections to be exchanged, they are updated in the data source and in the current application each time the respective value changes on the other side.

Using a data server is a faster alternative to provide data via a symbol configuration.

☞ At present, data provided by an OPC server cannot yet be accessed using the data server. In this case, implementing a symbol configuration is still the suitable procedure.

For a description of how a data server is set up and how remote data sources can be used, see Data source editor, page 318,.

2.10.5 Network Variables

Network Variables; Data Exchange between IndraLogic 2G Controls

Network variables have to be defined in fixed variable lists in both the sender and the receiver. Their values are sent via "broadcasting".

Note that this differs fundamentally from the data exchange with a data server, page 71, which uses defined point-to-point connections between the local application and remote data sources.

💡 Based on network variables, 1.x and 2G controls can communicate with each other.

Network variables are handled in...

- Global Variable Lists, page 52, (GVL) in the transmitting device (sender) and in one or more
- Global Network Variable List(s), page 53, (GNVL) in one or more receiving device(s) (receivers)

  GNVLs are displayed in the"network variable list editor", page382,.

The "GVL" and "GNVL" objects that belong to each other have to contain the same variable declarations.

A GVL that is supposed to define network variables has to have special network properties, page 246,.

These are protocol and transfer parameters according to which the variable values are set within the network and can be received by all devices with a matching GNVL.

☞ Note that the transfer of network variables is always in one direction: from the sender (GVL) to the receiver (GNVL)!

However, each device can act as sender or receiver, since each device can handle GVL and GNVL objects.

A prerequisite to exchange network variables is that the suitable "network libraries" are installed. This can be done automatically for the default network functionalities, e.g. for UDP as soon as the network properties for a GVL are set.

The structure of a simple network variable exchange is described in the following example. A GVL is created in the transmitting device and a GNVL in the receiver:

Example:

The following is the preparatory work in a project in which a transmitting device "Dev_Sender" and a receiving device "Dev_Receiver" are created in the Project Explorer:

- Create a POU (program) "prog_sender" below the application in the control Dev_Sender.
- In the task configuration of this application, add the task "Task_S" which calls "prog_sender".
- Create a POU (program) "prog_receiver" below the application in the control "Dev_Receiver".
- In the task configuration of this application, add the task "Task_R" which calls "prog_receiver".

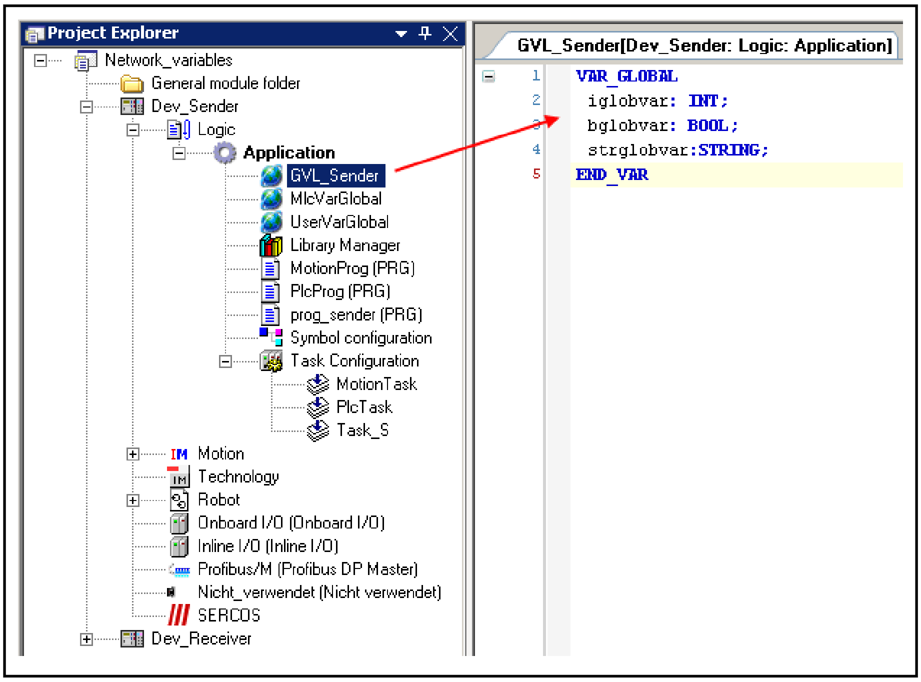
1. Define the global variable list in the transmitting device:

   Highlight the "Application" node in the "Dev_Sender" control. In the context menu, select **Add** ▸ **Global Variable List** and enter the name "GVL_Sender" in the "Add Object" dialog.

   Use "Finish" to confirm your entries. Double-click on "GVL_Sender" to open the GVL editor and enter the following lines.

GVL_Sender

```
VAR_GLOBAL
 iglobvar: INT;
 bglobvar: BOOL;
 strglobvar:STRING;
END_VAR
```

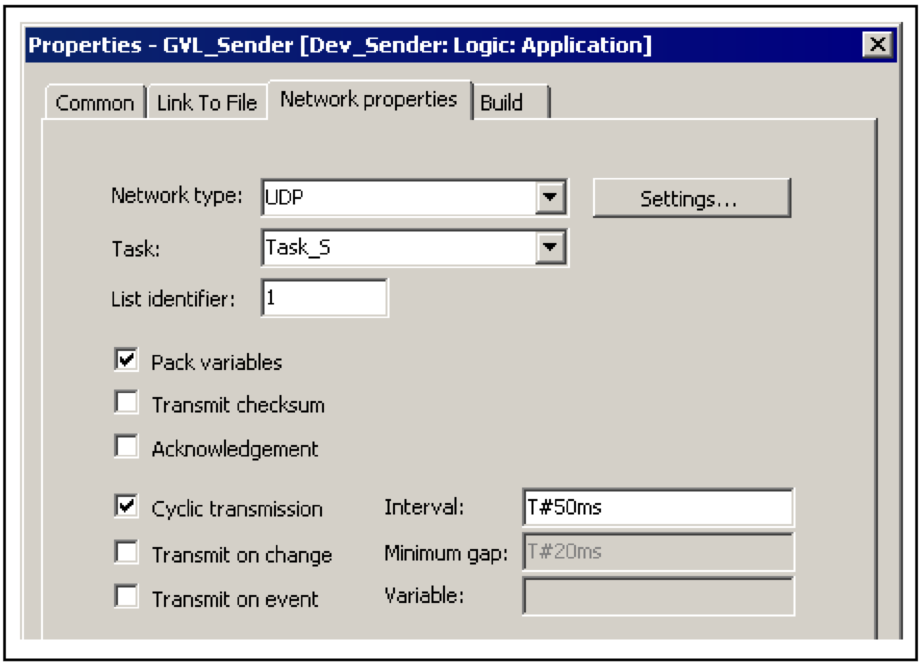


```
VAR_GLOBAL
 iglobvar: INT;
 bglobvar: BOOL;
 strglobvar:STRING;
END_VAR
```

Fig.2-27: Adding a GVL in the transmitting device

2. Define the network properties of the sender GVL:

   Highlight "GVL_Sender" in the Project Explorer and select **Properties** in the context menu. Open the "Network variables" tab. Set the network properties as follows; see the following figure.



Network type: UDP

Task: Task_S

List identifier: 1

☑ Pack variables

☐ Transmit checksum

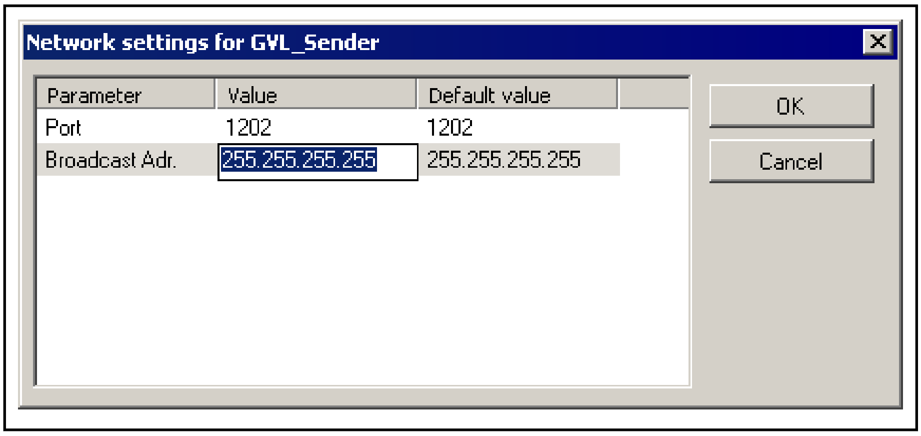
☐ Acknowledgement

☑ Cyclic transmission Interval: T#50ms

☐ Transmit on change Minimum gap: T#20ms

☐ Transmit on event Variable:

Fig.2-28: Setting GVL network properties



Network settings for GVL_Sender

Parameter	Value	Default value
Port	1202	1202
Broadcast Adr.	255.255.255.255	255.255.255.255

Fig.2-29: Setting GVL network properties, settings

☞

- Enter the IP address of the receiver control as broadcast address!

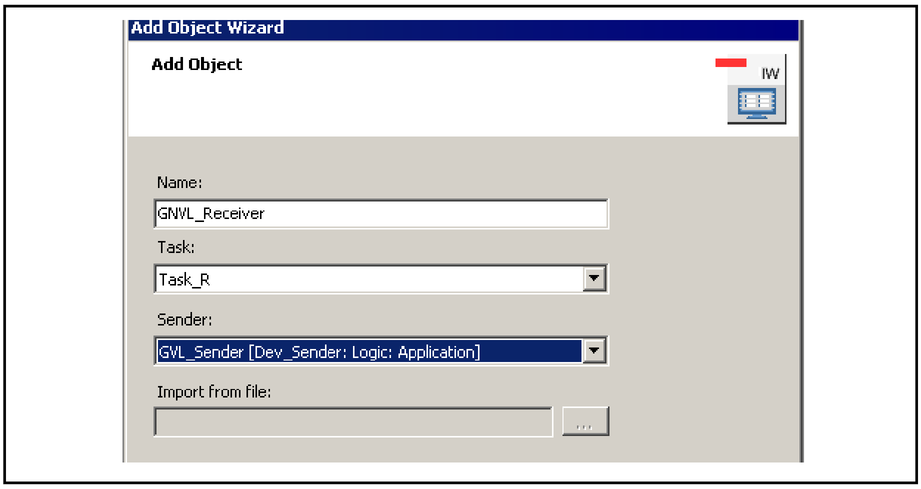
  This way, a point-to-point transmission can be executed between sender and receiver.
- The transmission option from the sender to **several** receivers is ### in preparation ###.

  Here, 255.255.255.255 has to be entered as broadcast address.

- For details, see Network Properties, page 246,

3. Creating a global network variable list in the receiver:

   Highlight the "Application" node in the "Dev_Receiver" control. In the context menu, select **Add** ▸ **Global Network Variable List** to open the "Add Object" dialog.



Add Object Wizard

Add Object

Name: GNVL_Receiver

Task: Task_R

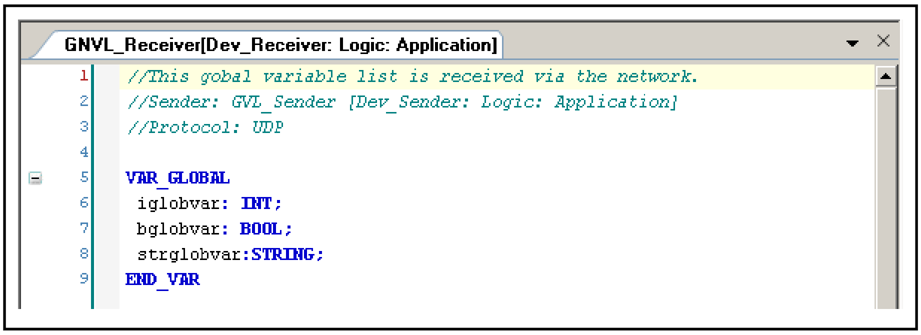
Sender: GVL_Sender [Dev_Sender: Logic: Application]

Import from file:

Fig.2-30: Creating a GNVL in the receiving device

Enter the name "GNVL_Receiver". In the "Sender" field there is a selection list of all of the GVL objects currently available in the project with network properties. This example only includes "GVL_Sender". In the "Task" selection list choose "Task_R" in the "Dev_Receiver" control as defined above. Click on "Finish" to confirm your entries.

Double-click on "GNVL_Receiver" to open the "GNVL" editor. This GNVL automatically contains the same variable declarations as "GVL_Sender"; see the following figure.



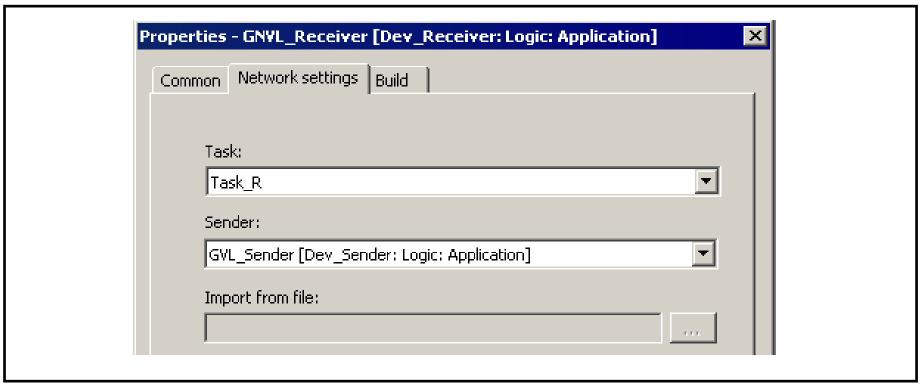
```
//This gobal variable list is received via the network.
//Sender: GVL_Sender [Dev_Sender: Logic: Application]
//Protocol: UDP

VAR_GLOBAL
 iglobvar: INT;
 bglobvar: BOOL;
 strglobvar:STRING;
END_VAR
```

Fig.2-31: GNVL_Receiver contains the variable declarations of GVL_Sender

4. Check or change the network settings of the global network variable list:

   Highlight "GNVL_Receiver" in the Project Explorer and use **Properties...** in the context menu to open the "Properties" dialog. Open the "Network variables" tab and check your entries; see the following figure.


Properties - GNVL_Receiver [Dev_Receiver: Logic: Application]
Common
Network settings
Build
...

Task: Task_R

Sender: GVL_Sender [Dev_Sender: Logic: Application]

Import from file:

Fig.2-32: GNVL network settings

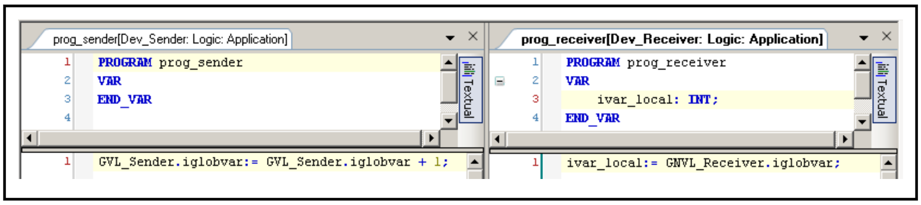
   If necessary, change your entries using the respective selection list and confirm them with "OK".

   - For details, see Network Properties, page 248,

5. Test the network variable exchange:

   In order to test a network variable online, perform the following steps:

   - In "prog_sender" of the sender application use the variable "iglobvar" directly.
   - In "prog_rec" of the receiver application use the local copy of the network variable "iglobvar":
   - Connect sender and receiver applications to the network and start the applications. In the online views of the function blocks, observe whether the values of "iglobvar" in the receiver match with those in the sender.


prog_sender[Dev_Sender: Logic: Application]
prog_receiver[Dev_Receiver: Logic: Application]
Textual
Textual

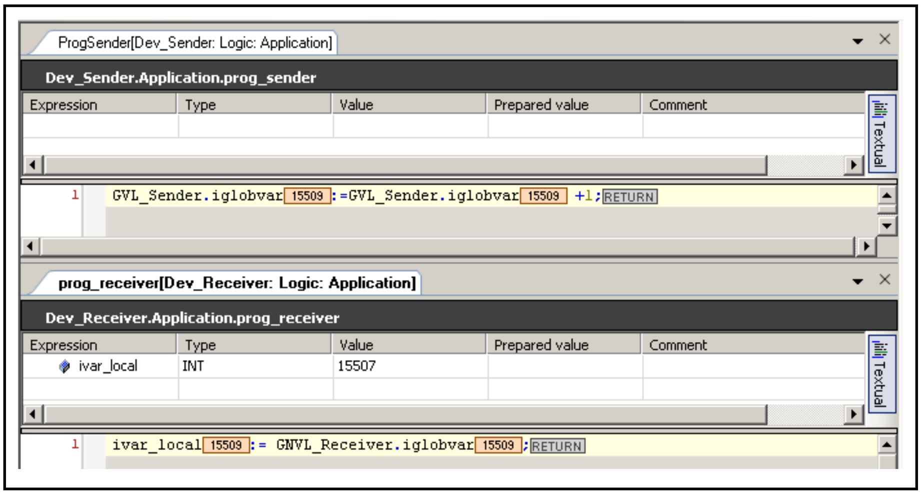
```
PROGRAM prog_sender
VAR
END_VAR

GVL_Sender.iglobvar:= GVL_Sender.iglobvar + 1;
```

```
PROGRAM prog_receiver
VAR
    ivar_local: INT;
END_VAR

ivar_local:= GNVL_Receiver.iglobvar;
```

Fig.2-33: Programming examples, sender end and receiver end



ProgSender[Dev_Sender: Logic: Application]

Dev_Sender.Application.prog_sender

Expression	Type	Value	Prepared value	Comment

```
GVL_Sender.iglobvar 15509 :=GVL_Sender.iglobvar 15509 +1;RETURN
```

prog_receiver[Dev_Receiver: Logic: Application]

Dev_Receiver.Application.prog_receiver

Expression	Type	Value	Prepared value	Comment
ivar_local	INT	15507		

```
ivar_local 15509 := GNVL_Receiver.iglobvar 15509 ;RETURN
```

Fig.2-34: Programming examples, sender end and receiver end, transmission running

Network Variables; Data Exchange between IndraLogic 1.x and 2G Controls

It can also be communicated using network variables if the participating controls operate with applications from different versions of the programming system (1.x ↔ 2G).

In this case, the export/import mechanism to create the exactly matching variable lists required in the sender receiver project cannot be used.

This is caused by the differing information in the 1.x and 2G variable export files (*.exp ↔ *.gvl).

If a reading GNVL is set up in 2G, the respective network parameter configuration has to be present as a *.gvl file that was previously exported from the 2G sender. This information is not present in an *.exp file exported from a 1.x sender.

Possible solution for a network variable exchange between 1.x and 2G applications:

1. Reproduce the 1.x NVL in 2G (Add a GVL with network properties containing the same variable declarations as the 1.x NVL).
2. Export the new GVL to an *.exp file ("Linking with file" properties)

☞ Enable the option "Exclude from build" and you can keep the GVL in the project without getting precompile errors and ambiguous names.

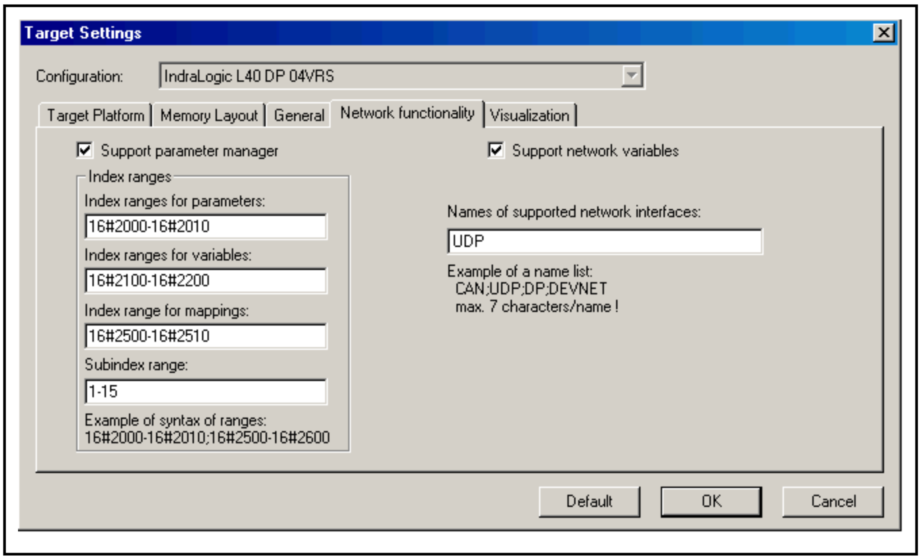
Disable the option if the .exp file has to be created again after the required changes in the GVL.

3. Re-import the list. That means creating a new GNVL using the previously created *.exp file to get a correctly matching variable list in the receiver.

Example:

A prerequisite is that the support of network variables is enabled by a specified control type.

Resources ▸ **Target Settings** ▸ **Project Database** ▸ **Checkout**, then enable "Support network variables".



Target Settings

Configuration: IndraLogic L40 DP 04VRS

☑ Support parameter manager

Index ranges

Index ranges for parameters: 16#2000-16#2010

Index ranges for variables: 16#2100-16#2200

Index range for mappings: 16#2500-16#2510

Subindex range: 1-15

Example of syntax of ranges:
16#2000-16#2010;16#2500-16#2600

☑ Support network variables

Names of supported network interfaces: UDP

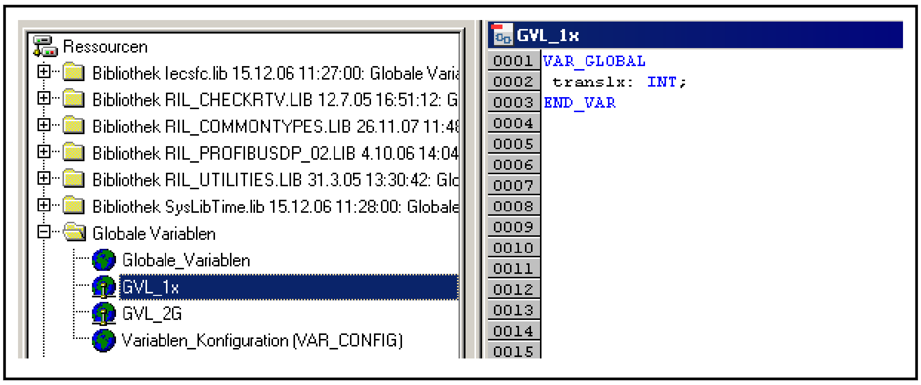
Example of a name list:
CAN;UDP;DP;DEVNET
max. 7 characters/name !

Fig.2-35: Target system setting "Support network variables"

There is one project 1x.pro with one global variable list GVL_1x containing the following declarations:

`VAR_GLOBAL trans1x: INT; END_VAR.`

Variable "trans1x" should be possible to be read from a 2G application.



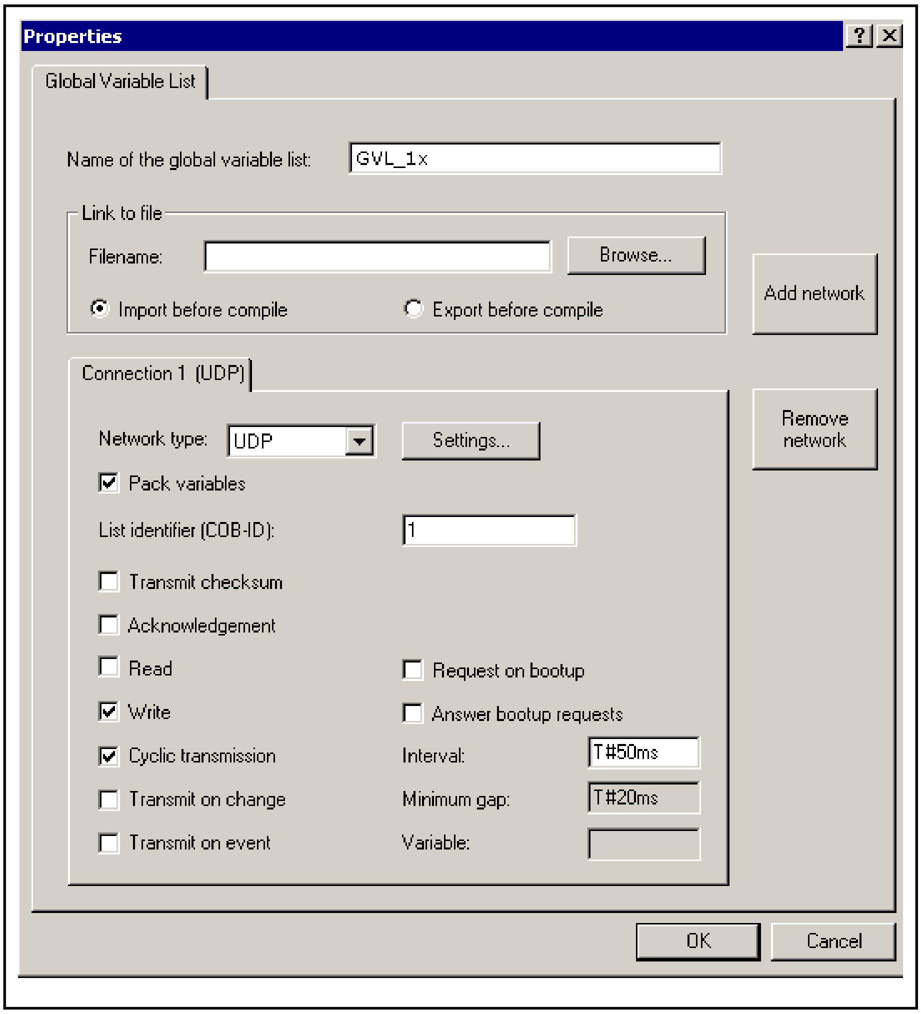
```
Ressourcen
⊞ Bibliothek Iecsfc.lib 15.12.06 11:27:00: Globale Varia
⊞ Bibliothek RIL_CHECKRTV.LIB 12.7.05 16:51:12: G
⊞ Bibliothek RIL_COMMONTYPES.LIB 26.11.07 11:48
⊞ Bibliothek RIL_PROFIBUSDP_02.LIB 4.10.06 14:04
⊞ Bibliothek RIL_UTILITIES.LIB 31.3.05 13:30:42: Glo
⊞ Bibliothek SysLibTime.lib 15.12.06 11:28:00: Globale
⊟ Globale Variablen
    Globale_Variablen
    GVL_1x
    GVL_2G
    Variablen_Konfiguration (VAR_CONFIG)
```

GVL_1x

```
VAR_GLOBAL
 trans1x: INT;
END_VAR
```

Fig.2-36: GVL in the 1x project

The network settings of GVL_1x are configured as follows:



Name of the global variable list: GVL_1x

Link to file

Filename:

● Import before compile ○ Export before compile

Connection 1 (UDP)

Network type: UDP

☑ Pack variables

List identifier (COB-ID): 1

☐ Transmit checksum

☐ Acknowledgement

☐ Read ☐ Request on bootup

☑ Write ☐ Answer bootup requests

☑ Cyclic transmission Interval: T#50ms

☐ Transmit on change Minimum gap: T#20ms

☐ Transmit on event Variable:

Fig.2-37: Properties of GVL_ 1x

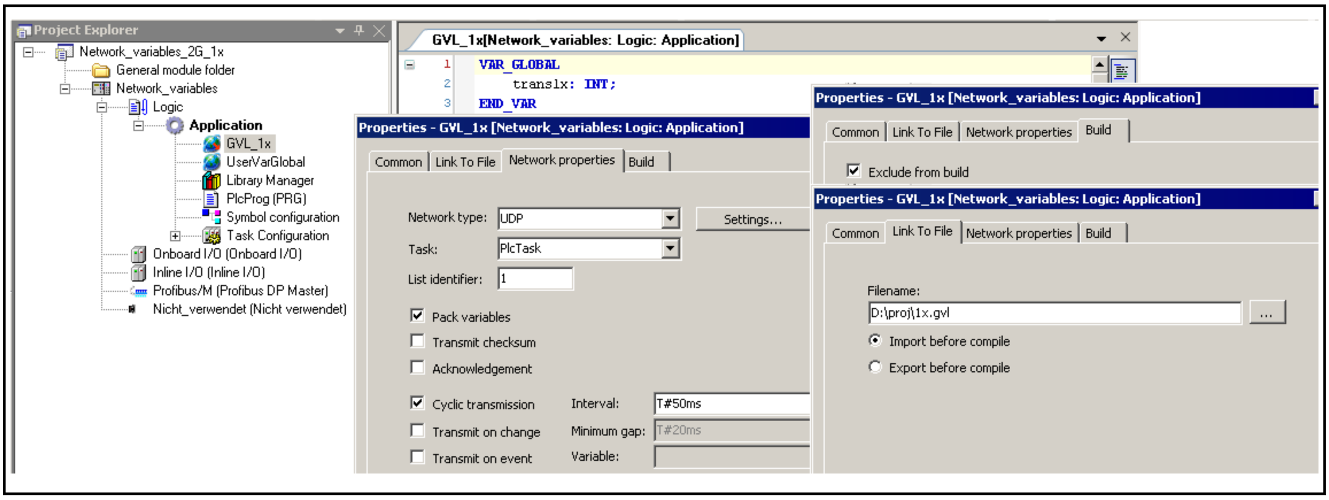
If GVL_1x is exported to an *.exp file, this file contains only the declaration

```
VAR_GLOBAL trans1x: INT; END_VAR
```

Thus, reproduce GVL_1x in 2G first (see GVL_1x in the figure above):

Add a GVL object with the name "GVL_1x" below an application in a 2G project and proceed with the following steps:

- Set the network properties as defined in 1x.pro
- Specify an export file "1x.gvl" in the "Link to file" properties
- Recommendation: Set the option "Exclude from build" (for details refer to Network properties, page 246)
- Compile the 2G project to generate a 1x.gvl file (contains then variable definitions + configuration data!)



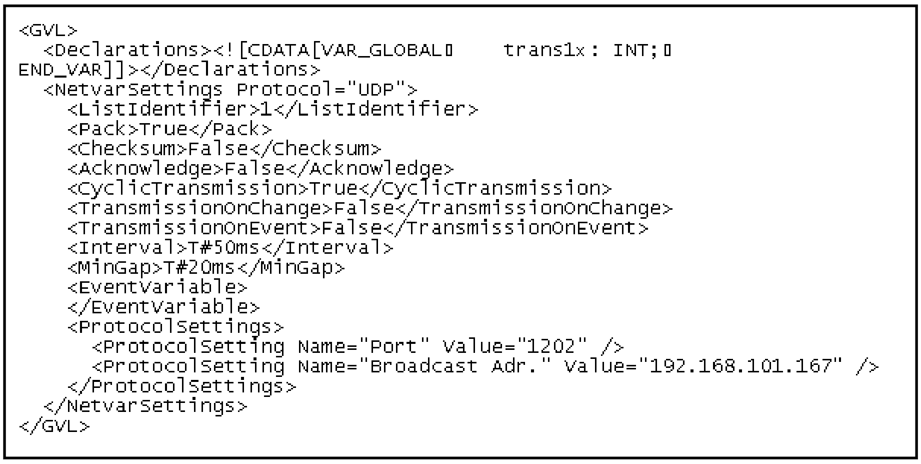
```
VAR_GLOBAL
    trans1x: INT;
END_VAR
```

Network type: UDP
Task: PlcTask
List identifier: 1
☑ Pack variables
☐ Transmit checksum
☐ Acknowledgement
☑ Cyclic transmission Interval: T#50ms
☐ Transmit on change Minimum gap: T#20ms
☐ Transmit on event Variable:

☑ Exclude from build

Filename: D:\proj\1x.gvl
● Import before compile
○ Export before compile

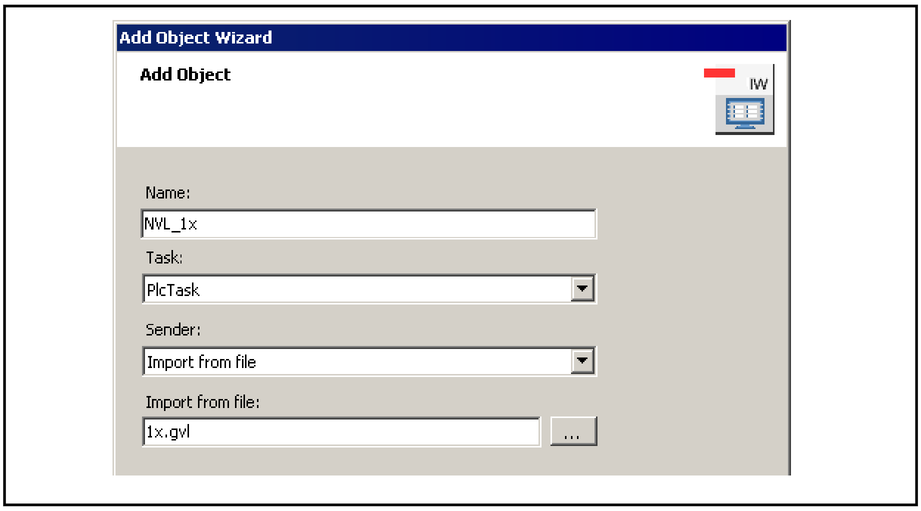
Fig.2-38: Reproduce the GVL in 2G



```
<GVL>
  <Declarations><![CDATA[VAR_GLOBAL▯    trans1x : INT;▯
END_VAR]]></Declarations>
  <NetvarSettings Protocol="UDP">
    <ListIdentifier>1</ListIdentifier>
    <Pack>True</Pack>
    <Checksum>False</Checksum>
    <Acknowledge>False</Acknowledge>
    <CyclicTransmission>True</CyclicTransmission>
    <TransmissionOnChange>False</TransmissionOnChange>
    <TransmissionOnEvent>False</TransmissionOnEvent>
    <Interval>T#50ms</Interval>
    <MinGap>T#20ms</MinGap>
    <EventVariable>
    </EventVariable>
    <ProtocolSettings>
      <ProtocolSetting Name="Port" Value="1202" />
      <ProtocolSetting Name="Broadcast Adr." Value="192.168.101.167" />
    </ProtocolSettings>
  </NetvarSettings>
</GVL>
```

Fig.2-39: Resulting export file "1x.gvl" opened in the text editor

Add a GNVL object (option "Import from file) using the 1x.gvl file. This allows to read the variable "trans1x" from the 1.x control.



Name: NVL_1x
Task: PlcTask
Sender: Import from file
Import from file: 1x.gvl

Fig.2-40: GNVL in 2G project

If the 1.x project as well as the 2G application run in the same network, the 2G application can read the variable "trans1x" from the project 1x.pro.

2.11 Code Generation and Online Change

Machine code is first generated when the application, page 66, is loaded to the control (PLC).

At each download, the compilation log containing the code and identification of each loaded application is saved as file "IndraLogic.<DeviceName>.<application ID>.compileinfo" in the same directory as the project. The compilation log is deleted when executing the command Clear, page 125, or Clear All, page 125,.

☞ Note that no machine code is generated if the project is compiled with Create commands, page 124,.

This compilation process checks for syntax errors in the project. These are output in the message box.

⚠ CAUTION **Th online change modifies the running application program and causes a restart.**

Ensure that the new application code causes the desired behavior of the controlled system. Depending on the system controlled, damages at the system and workpieces can result or the health and life of people can be put at risk.

☞ *Additional notes:*

1. If an online change is made, the program code may not be as it was before the complete initialization, since the machine keeps its status.
2. Pointer variables retain their value from the last cycle. If a pointer points to a variable that changed its size due to the online change, the value is not provided correctly anymore. Ensure that pointer variables are re-assigned in every cycle.

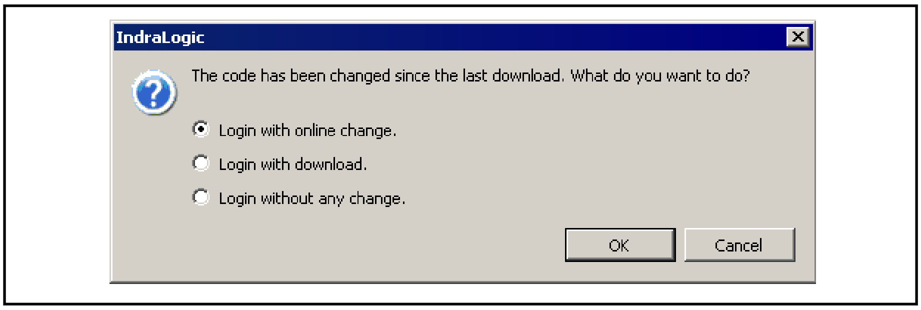
Online change

If the application that is currently running on the control was changed since the last download in the programming system, only the modified project objects are loaded to the control during online change while the program continues running there.

There are two ways to perform an online change:

1. As soon as you try to log in again with a changed application program, a dialog appears prompting what you would like to do. Select from the following three options:
   - Login with online change.
   - Login with download.
   - Login without any change.

   Select the option "Login with online change.".



IndraLogic

The code has been changed since the last download. What do you want to do?

● Login with online change.

○ Login with download.

○ Login without any change.

Fig.2-41: Selection dialog for code download

Confirm with "OK". All changed objects are now loaded and displayed immediately in the online view (monitoring) of the respective object.

Select the "Login with download" menu item and the entire project is loaded to the control.

Select the "Login without any change" menu item and the program on the control continues to run and the new changes are not loaded. Afterwards, download (<application>) explicitly. That download either reloads the entire project or the dialog described above appears again at next login.

2. If you are already logged in and the project on the control is not updated, you can explicitly perform an online change. In the main menu, select **Debug** ▸ **Online Change** to perform an online change.

   A dialog appears prompting if you really want to perform an online change.

   To carry out the online change, click "Yes".

☞ Information on the online change can also be found in "Online Change", page 133,.

☞ Note that an online change in a changed project for an application is no longer possible after a cleanup(commands: Clear All, page 125, "Clear", page 125).

In this case, information on the objects changed since the most recent download is deleted. This means that only the entire project can be reloaded.

☞ **Note the following before performing online change:**

- Is the modified code free of errors?
- Application-specific initializations (reference motion, etc.) are not executed, since the machine retains its status. Can the new program code really work without re-initialization?
- Pointer variables retain their value from the last cycle. If it is pointed to a variable that changed in size, the value is no longer correct. For this reason, ensure that pointer variables are re-assigned in every cycle.
- If the active step in an SFC chart is removed, the chart remains inactive.

Boot application (boot project)

At each download, the active application is automatically saved as a file called <Application>.app in the target system directory. Click on **De-**

bug ▸ Generate Boot Application to save the boot application in a file even in offline mode.

A boot application is started automatically when the control is started. To do this, the application project on the control has to be available in a file <ProjectName>.app. To create the file, go to **Debug ▸ Generate Boot Application**.

2.12 Monitoring

In online mode, there is a variety of possibilities to display the current values of the variables of an object on the control:

- "Inline monitoring" in the implementation editor of an object.

  Details can be found in the description of the respective editor.
- "Online view of the declaration editor" of an object.

  For details, refer to the declaration editor, page 326.
- "Object-independent monitoring lists"

  For more details, refer to Monitoring Window, page 499.
- "Trace curves"

  Recording and display of variable values on the control. For details, refer to Trace Functionality, page 431.
- "Recipes"

  User-defined variable set to set and monitor these variables on the control. See Recipe Management, page 382.

2.13 Debugging/Troubleshooting

To investigate programming errors, the debugging functions in IndraLogic 2G can be used.

Note the option of an application in the simulation mode, page 183, that is without necessary connection to a real target device.

Breakpoints can be set at certain positions in the program to force an execution stop. Certain conditions, specifically which task(s) are affected and in which cycle intervals the breakpoint is to be effective, can be defined for each breakpoint.

Single step processing enables the program to run in controlled steps.

At each stop, defined by the step marks and breakpoints, the respective variables can be investigated.

Breakpoints

A breakpoint set in an application program causes a stop in the execution of the program. The possible breakpoint positions depend on the respective program editor. There is always a breakpoint position at the end of the POU.

A description of the command for handling breakpoints can be found in "Breakpoints", page 126,. An important tool is the "Breakpoints" dialog, page 136 in which all defined breakpoints are listed and in which breakpoints can be added, deleted or modified.

Conditional breakpoints. The stop at the breakpoint can depend on the task currently executed or the number of the cycle currently running.

Breakpoint icons

● Breakpoint activated

○ Breakpoint deactivated

➲ Stop at the breakpoint in online mode

Single step processing

Single step processing (stepping) enables a controlled execution of the application program, e.g. for the purpose of troubleshooting. Repeated pressing of <Alt>+<F12> allows to jump from instruction to instruction. However, called function blocks can also be skipped.

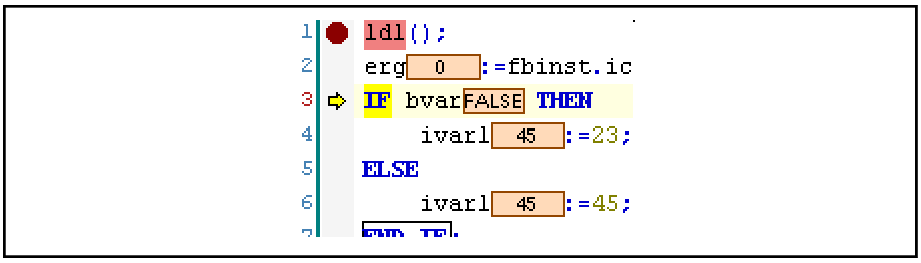
What's new compared to IndraLogic 1.x

- The instruction to be executed as next instruction can be explicitly defined. To do this, click on **Debug** ▸ **Specify Next Instruction** in the main menu.
- The next execution stop can be determined by placing the mouse pointer at the desired position. To do this, click on **Debug** ▸ **Execute to cursor** in the main menu.
- "Execute to Return" causes a backward step to the last call. To do this, click on **Debug** ▸ **Execute to Return** in the main menu.

A description of the stepping commands can be found in "Breakpoints", page 126,.

Icon for single step processing (stepping): ⇨

The current position during stepping is displayed with a yellow arrow in front of the line and yellow shadowing of the related operation.



```
ldl();
erg[ 0 ]:=fbinst.ic
IF bvar[FALSE] THEN
    ivarl[ 45 ]:=23;
ELSE
    ivarl[ 45 ]:=45;
END_IF;
```

Fig.2-42: From the breakpoint, the command "Single step" is used to jump to the next step

2.14 Printing

The view in the currently active editor can be printed using the "Print" function. To do this, click on **File** ▸ **Print** in the main menu. Note the alternative possibility for generating a "Documentation" of selected objects in the project in a defined layout and with a table of contents. A detailed description about "printing" can be found in the IndraWorks documentation.

2.15 Visualization

Information on the visualization in IndraLogic 2G and the visualization editor can be found in "Visualization," page 625, and "Visualization editor", page 445.

2.16 Library Management

2.16.1 Library Management, Overview

Libraries can provide functions, function blocks, data types, global variables and even visualizations which can then be used in the project.

The default extension for a library file is ".library" in contrast to ".lib" which was used for files in IndraLogic V1.x and previous versions. Encrypted libraries have the extension "*.compiled-library".

The management of libraries in a project occurs in the library manager. The previous installation on the system is performed using the "Library Repository" dialog.

The project functions for a global and local "find" and "replace" can also be used for libraries that are not encrypted.

See the following general information on:

- Installation on the System and Integration into a Project, page 84
- Referenced Libraries, page 85
- Library Versions, page 85
- Unique Access to Library Function Blocks (Namespace), page 86
- Creating a Library in IndraWorks, page 191,
- IndraLogic 1.x Libraries, page 86
- External and Internal Libraries or Library Function Blocks, Late Linking, page 87

2.16.2 Installation on the System and Integration (Linking) into a Project

- Libraries can be managed on the local system in various "repositories" (directories, storage locations). Before a library can be integrated into a project, it has to be installed on the local system in a repository.

  This is done in the Library Repository dialog, page 185, in IndraLogic.
- A prerequisite for the installation is that the library information library information, page 371, of a library project has a title, version information and the name of the vendor (company).

  As an option, a category designation can be entered that can later be used in the library manager for sorting.
- If there is no category assignment in the library information, the library automatically belongs in the "Other" category. If other library categories in addition to this default category are to be used in IndraLogic libraries created in 2G, these are defined in one or more external XML file(s) "*.libcat.xml" that can also be extended and created again. Such a file can then be called in the "Library Information" dialog to select a category. For further information on the categories, refer to Creating a "Library" or "Compiled Library" in the IndraWorks Environment, page 191
- Libraries are integrated into a project with the Library manager, page 367,. In a "default project", it is automatically assigned to the default device first. But it can also be added explicitly in Project Explorer, page 63, (below a device or an application) or globally in the "General module" folder. This is done, as for other objects, using the Add Object dialog, page 234. To do this, highlight the application and select **Add ▸ Library manager** in the context menu. Libraries that are integrated into a library are also displayed with a preset in the library manager. However, "hidden libraries" are also possible; see also Referenced Libraries, page 85.
- If the library is not in encrypted and the ".library*" file is present instead, the library POUs listed in the library manager can be opened by double-clicking on the respective entry.
- If a library function in the project is addressed, the libraries and repositories are searched in the sequence in which they are listed in the "Library Repository" dialog; see also Unique Access to Library Function Blocks (Namespace), page 86.

2.16.3 Referenced Libraries

- A library can link other libraries (referenced libraries) where the nesting can be as deep as desired.

  If such a "father" library is linked to the library manager of the global "General module" folder of a project, both the father library and the libraries it references are available in **all** applications of the project.

  If such a "father" library is linked to the library manager of an application, both the father library and the libraries it references are available **in exactly these** applications of the project.

If, for example, the library "RIL_Utilities" references the library "Util" and "Util" contains the function block "BLINK", an instance "bk1" of "BLINK" has to be declared as follows:

```
bk1: RIL_UTILITIES.UTIL.BLINK;
```

- When creating a library project that references others projects, it can be specified in the Properties, page 230, of each linked library how it has to act later when it is linked to a project via the " father" library:
  1. Its visibility in the Library Manager, page 367, indented below the "father" library, can be disabled. This way, "hidden libraries" can be provided in a project.
  2. If a pure "container" library is generated - in other words, a library that does not define any function blocks itself but instead only references other libraries - later access to its function blocks can be simplified.

     When a "container" library is linked to a project, a whole set of libraries is linked along with it.

     In this case, it is possible to simplify the access to the function blocks of these libraries by defining them as "top level" libraries. Then, when accessing the function blocks, the namespace for the libraries can be omitted.

     To do this, use the "Publish..." option in the library properties. However, this option should only be used when creating a container library project!
- See also Library Management, page 83.

2.16.4 Library Versions

- Several versions of a library can be installed simultaneously on the system.
- Several versions of a library can be simultaneously linked to a project. The following are clearly specifies on which version an application accesses in this case:
  – If several versions are located on the same level in the Library manager, it depends on the definition in the Library properties, page 230, which version is to be used (a certain one or always the newest one).

  – If several versions are located on different levels (which can be the case with referenced libraries, page 85), unique access to library function blocks is achieved by entering the corresponding namespace as described in the following.

2.16.5 Unique Access to Library Function Blocks (Namespace)

- Basically, the following applies:

  If several function blocks with the same name are available in the project, access to a function block component has to be unique or compiler errors result. This applies to project-local function blocks and to function blocks available in linked, referenced libraries. In such cases, the unambiguousness is achieved by adding the namespace in front of the function block name.
- The default namespace of a library is defined in the Library Properties, page 230,.

  If it is not explicitly defined, the library name is automatically used. However, when creating a library project, another default namespace can be entered into the "Properties" dialog. Later, when a library is already linked to the library manager of a project, the namespace can also be changed locally - also in the "Properties" dialog.
- In the following examples, the "namespace" of the library "Lib1" is added to the library properties with "Lib1". In the right column, there are the namespaces for unique accesses to the variable "var 1" defined in the function block "module1" and in the function block "POU1".

	Variable "var1" defined in the positions (1) to (5) in the project:	**Unique access to "var1" using the corresponding namespace information...**
(1)	In the library "Lib1" in the global library manager in the "General module" folder	"Lib1.module1.var1"
(2)	In the library "Lib1" in the library manager below an application "App1" of a control "Dev1"	"Dev1.App1.Lib1.module1.var1"
(3)	In the library "Lib1" linked to the library "F_Lib" (referenced) in the global library manager in the "General module" folder	Presets: (Option "Publish..." is disabled in the library properties of Lib1 when "Lib1" is added to "F_Lib"): "F_Lib.Lib1.module1.var1" If the option "Publish..." was activated, "module1" would be treated as a component of a library linked at top level. Then, access without entering the namespace of the "father" library "F_Lib" is normally possible: "Lib1.module1.var1" or "module1.var1"). In the present example, however, this leads to a compiler error because the call is no longer unique; see points (1) and (4).
(4)	In the object "module1" that is defined in the "General module" folder	"module1.var1"
(5)	In the object "POU1" that is defined in the "General module" folder	"POU1.var1"

Fig.2-43: Namespaces

2.16.6 IndraLogic 1.x Libraries

- Libraries that were created with IndraLogic 1.x (*.lib) and earlier versions continue to be supported.
- An old library project (*.lib) can be opened directly in IndraLogic 2G and can be converted into an "IndraLogic 2G library" (*.library).

- If an old project that references old libraries is opened, it can be selected whether these references are to be retained, replaced or deleted. If they are to be retained, the affected libraries are converted into the new format and are automatically installed in the system library repository.

  If they do not contain the necessary library information, page 371,, these can be immediately added.

  The model (mapping) by which an old library was once handled during conversion of an old project can be saved in the project options so that the same library does not need to be explicitly handled each time in future project conversions.
- A description of the procedure for converting projects and libraries can be found in Data Transfer, page 115,.

2.16.7 External and Internal Libraries or Library Function Blocks, Late Linking

- An "external library", in contrast to an internal library (IndraLogic library project), is a library file that is programmed outside of IndraLogic in another programming language, e.g. C. It has to be present on the target system and is only linked if the application is running there.
- As in IndraLogic 1.x, it is also possible to link an IndraLogic library as an external library later on, i.e. when the application is first operated on the runtime system. In addition, it is also possible now to define the late linking individually for every library function block.

  For this purpose, the property in the object properties of one or all function blocks can be enabled ("External Implementation", page 243).

3 Menu Items

3.1 Edit

3.1.1 Edit Menu, Overview

The main menu item "Edit" contains the following components, depending on which editor, dialog or wizard is active or when working in the Project Explorer:

- Undo, page 89
- Redo, page 89
- Cut, page 90
- Copy, page 90
- Paste, page 90
- Delete, page 91
- Rename, page 91
- Select All, page 91
- Find and Replace, page 91, (character strings)
- Bookmarks, page 96, bookmarking functionality in text-based editors
- Insert file as text..., page 97,
- Browse, page 97, information about object position and dependencies
- Input assistance, page 98
- Declare variable, page 100,
- Advanced, page 102, expanded list of commands for working with text-based editors
- "List components" function, page 103

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize ▸ Commands** dialog.

3.1.2 Undo

Icon: ↶

Default shortcut: <Ctrl>+<Z>

Menu: **Edit ▸ Undo**.

The "Undo" command undoes the editing step most recently carried out in the currently active editor.

Repeated execution of the "Undo" command undoes all of the actions carried out since the editor window was opened, one by one. This applies to all actions in the function block editors.

Use "Redo" to restore the result of the editing step just undone.

3.1.3 Redo

Icon: ↷

Default shortcut: <Ctrl>+<Y>

Menu: **Edit ▸ Redo**.

The "Redo" command restores the result of the editing step that was just undone.

Repeated execution of the "Redo" command restores all actions previously undone, one by one.

3.1.4 Cut

Icon:

Default shortcut: <Ctrl>+<X>

Menu: **Edit** ▸ **Cut**.

This command transfers the current selection (object, character string) to the clipboard and **removes it from the previous position** in the Project Explorer or editor. In the Project Explorer, this applies to the currently selected object. Several items can be selected at once.

☞ Note that not all editors support this command and that its use is limited in some editors.

The possibilities for selection differ depending on the editor:

In text editors, for example, the selection can consist of a character string or a single character.

In graphical editors, for example, the selection can include one or more elements inside a selection frame.

To insert the contents of the clipboard use the "Paste" command.

To transfer a selection to the clipboard without removing it from its current position, use the Copy command.

To remove a selection without changing the clipboard contents, use the "Delete" command.

3.1.5 Copy

Icon:

Default shortcut: <Ctrl>+<C>

Menu: **Edit** ▸ **Copy**.

The description for the "Cut" command also applies to this command, except that when copying, the selection is **not removed from its current position**.

To transfer a selection to the clipboard and remove it from its current position, use the "Cut" command.

3.1.6 Paste

Icon:

Default shortcut: <Ctrl>+<V>

Menu: **Edit** ▸ **Insert**.

This command inserts the contents of the clipboard at the current cursor position.

Not all editors support "Paste" and its use can be limited in others. Graphical editors only support the "Paste" command if a correct construct is established when the insertion is made.

In the Project Explorer, this command applies to the currently selected object.

Several items can be selected at once. Depending on the current position, a dialog might open in which to select whether the object on the clipboard should be inserted above or below the position.

To transfer a selection to the clipboard without removing it from its current position, use the "Copy" command.

To remove a selection without changing the clipboard contents, use the "Delete" command.

3.1.7 Delete

Icon: ✕

Default shortcut: <Del>

Menu: **Edit ▸ Deleting.**

This command deletes the selection from the editor or from the Project Explorer. The contents of the clipboard remain unchanged.

In the Project Explorer, this command applies to the currently selected object. Several items can be selected at once.

The possibilities for selection are the same as for the "Cut" command.

To remove a selection from the editor or object tree and simultaneously transfer it to the clipboard, use the "Cut" command.

3.1.8 Rename

Default shortcut: <F2>

Menu: **Edit ▸ Rename.**

The command allows to "rename" an object. In addition to changing the name of the object itself, the names of its dependent objects can also be changed automatically.

Example:

Renaming an axis causes the names of the address constants stored in the global variable list "MlcVarGlobal" to be changed as well.

3.1.9 Select All

Default shortcut: <Ctrl>+<A>

The command can be used in editors to select the entire contents at once.

3.1.10 Find and Replace

Overview of Find and Replace

The Find/Replace commands are located in the "Edit" category. They can be used to search an entire project for a character string and to replace certain character strings with other character strings.

By default, the commands are located in **Edit ▸ Find and Replace** .

Commands:

- Find, page 92
- Replace, page 95
- Search down, page 95,
- Search down (selection)95
- Search up, page 96,
- Search up (selection), page 96

Find

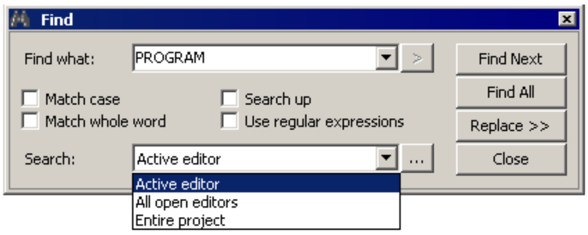
Icon:

Default shortcut: <Ctrl>+<F>

Use this command to search the project for a specific character string. The search is executed in all areas of the objects in the project that can be edited.

To open the dialog "Search", click in the main menu on **Edit ▸ Find and Replace ▸ Find**.

The "Find" command opens the "Find" dialog:



Find what: PROGRAM

☐ Match case

☐ Match whole word

☐ Search up

☐ Use regular expressions

Search: Active editor

Fig.3-1: "Find" dialog

In the "Find" dialog, enter the string to be found, where the search is to take place and whether the positions found are to be displayed all at once or one at a time. Switch to the "Replace" dialog at any time.

Find what: In the "Find what" input field, enter the string to be found (search string). The selection list associated with the ▼ button provides all strings entered since the programming system was started.

Search options Select the desired search options:

- Match case

  This search considers the case of the search string.

- Search up

  The search area entered moves upwards. To search in a downward direction again, disable the "Search up" option.

- Match whole word

  Only strings that exactly match the character string entered are found.

- Use regular expressions

  Use the > button for support when entering regular expressions to search for specific character strings. The most commonly used regular expressions are supported and are sorted into the following submenus:

  – Special characters

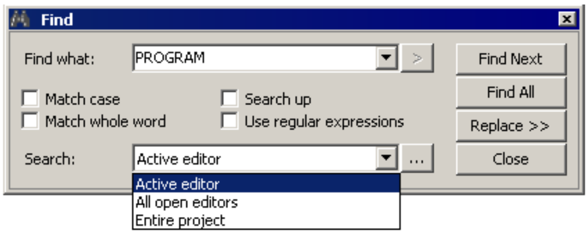
  – Repetitions

  – Alternatives

  – Groups

  – Others

Search in: In the "Search in" selection list enter the objects that are to be searched for and the character string to be found; see the following figure.


Find

Find what: PROGRAM

□ Match case
□ Match whole word
□ Search up
□ Use regular expressions

Search: Active editor

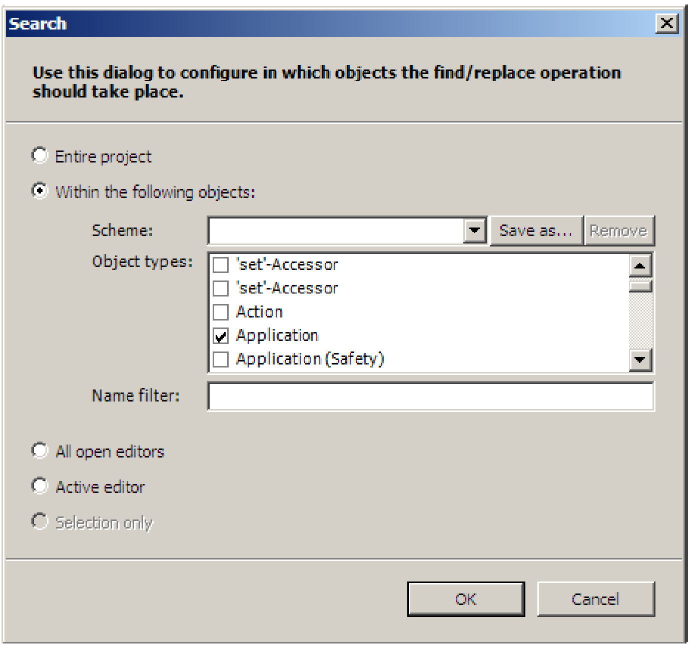
Active editor
All open editors
Entire project

Fig.3-2: "Find" dialog

Use the ▼ button to select one of the options from the list. The following can be selected:

- Active editor,
- All open editors,
- Entire project,
- Only selection.

In addition, the ... button can open the dialog for defining the search area where the search can be specified, see the following figure.


Search

Use this dialog to configure in which objects the find/replace operation should take place.

○ Entire project
● Within the following objects:

Scheme:

Object types:
□ 'set'-Accessor
□ 'set'-Accessor
□ Action
☑ Application
□ Application (Safety)

Name filter:

○ All open editors
○ Active editor
○ Selection only

Fig.3-3: Dialog for entering the search area

- Entire project

  Select the "Entire project" option if the search for the character string should include the entire project.

- Within the following objects

  Select the "Within the following objects" option if the search is to be limited to the selected object types.

  Proceed with the following steps:

  1. **Object types:**In the **Object types** selection field, specify which object types are to be included in the search for the character string

by placing a checkmark next to the object type(s) in the selection list.

2. **Name filter:** Optionally, set a name filter for the objects to be searched whereas placeholders "*" can be used. Example: If all objects with "CAN" in their name are to be searched, enter "*CAN" as filter here.

3. **Scheme:** You can save the current search configuration. Press <Save...> and enter a name for the configuration in the "Save Scheme" dialog.

   All saved schemes are then contained in the selection list (⌄ The currently selected scheme can also be deleted again using the <Remove> button).

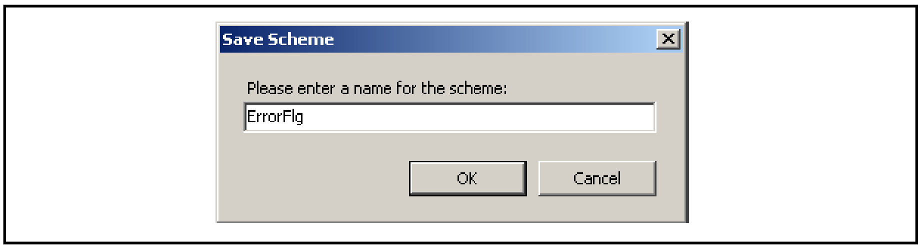


Fig.3-4: "Save Scheme" dialog

- **All open editors:** Select the "All open editors" option if only the open editors are to be searched for the desired character string.
- **Active editor:** Select the "Active editor" option if only the active editors are to be searched for the desired character string. The active editor is the editor for which the working area is displayed in the foreground.
- **Selection only:** The "Selection only" option is ### being prepared ###

Use the "OK" button to confirm the settings.

Once all search options are defined, press the button...

Find next

Click on the "Find Next" button to jump to the next position where the string was found. The respective editors are opened and the string found is highlighted in the display.

Search all

In order to get a list of all positions found in the message window. The progress of the search process is shown in the status bar and can be canceled early by means of the <Cancel> button displayed in the status bar if necessary.

After the search is complete, the following information is indicated regarding each position:

- **Priority:** Message icons.

  Messages can include

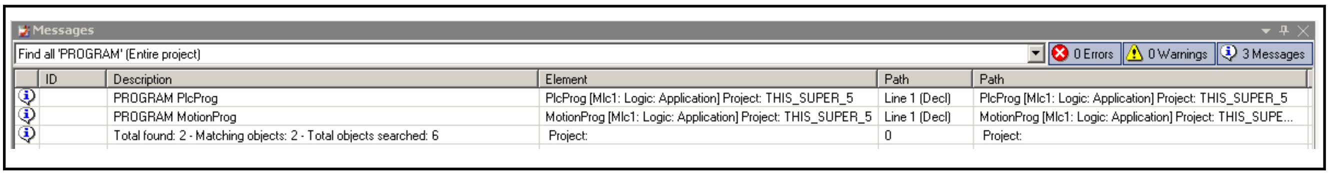
  Errors✖

  Warnings ⚠ or

  Just information ℹ.
- **Description:** Expression that contains the search string.
- **Origin:** Path where the item searched for was found.
- **Position:** Position (e.g. line number) within the objects; "Decl" in brackets stands for the declarations or "Impl" for the implementations of the editor.

The following is displayed below the list:

- Total number of positions found
- Number of objects in which the string was found
- Total number of objects searched



Messages

Find all 'PROGRAM' (Entire project)

0 Errors | 0 Warnings | 3 Messages

	ID	Description	Element	Path	Path
ⓘ		PROGRAM PlcProg	PlcProg [Mlc1: Logic: Application] Project: THIS_SUPER_5	Line 1 (Decl)	PlcProg [Mlc1: Logic: Application] Project: THIS_SUPER_5
ⓘ		PROGRAM MotionProg	MotionProg [Mlc1: Logic: Application] Project: THIS_SUPER_5	Line 1 (Decl)	MotionProg [Mlc1: Logic: Application] Project: THIS_SUPE...
ⓘ		Total found: 2 - Matching objects: 2 - Total objects searched: 6	Project:	0	Project:

Fig.3-5: Message window with positions found

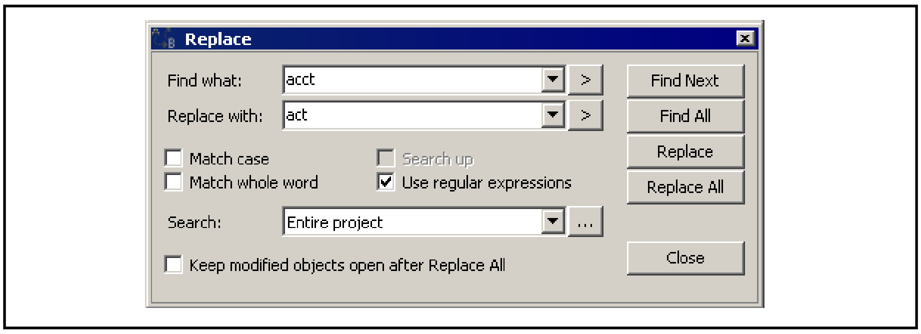
To replace this string with another string, click on the Replace, page 95 button to open the "Replace" dialog.

Replace

Icon:

Default shortcut: <Ctrl>+<H>

The "Replace" command opens the "Replace" dialog, which is an extended "Find" dialog; see the following figure.



Replace

Find what: acct

Replace with: act

□ Match case

□ Match whole word

□ Search up

☑ Use regular expressions

Search: Entire project

□ Keep modified objects open after Replace All

Fig.3-6: "Replace" dialog

As in the "Find" dialog, select the options for finding the character string to be replaced. In addition, enter the new character string in the "Replace with:" field and then click on one of the "Replace" buttons; see the following table:

Button	Function
Replace	The "Replace" button highlights and replaces the next string found in the editor (step by step replacement).
Replace all	The "Replace all" button replaces all of the strings found at once without displaying them in the respective editor(s).

Fig.3-7: "Replace" function commands

Search down

Icon:

Default shortcut: <F3>

Menu: **Edit ▸ Find and Replace ▸ Search Down.**

The "Search down" command can be used to move to the next position found after using the "Find" or "Replace" commands to search for a specific character string in the project.

Search down (Selection)

Default shortcut: <Ctrl>+<F3>

Menu: **Edit** ▸ **Find and Replace** ▸ **Search Down (selection)**.

The "Search down (selection)" command can be used to move to the next position found in the project for a character string that corresponds with the string currently highlighted in the editor.

Search Up

Icon:

Default shortcut: <Shift>+<F3>

Menu: **Edit** ▸ **Find and Replace** ▸ **Search Up**.

The "Search Up" command can be used to move to the position previous to the one found after using the "Find" or "Replace" commands to search for a specific character string in the project.

Search Up (Selection)

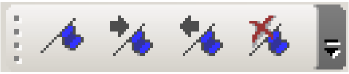
Default shortcut: <Shift>+<Ctrl>+<F3>

Menu: **Edit** ▸ **Find and Replace** ▸ **Search Up (selection)**.

The "Search up (selection)" command can be used to move to the position previous to the one found in the project for a character string that corresponds with the string currently highlighted in the editor.

3.1.11 Bookmarks

Bookmark, Overview

Icons: 

The menu item provides the bookmarking functionality for text editors. It can be accessed in **Edit** ▸ **Bookmark** or via the "Bookmarks" toolbar.

Bookmarks can be added to lines to facilitate navigation in long programs.

☞ Bookmarks remain when closing the editor window.

If IndraWorks Engineering is closed, all bookmarks are removed.

The commands described below allow the setting of bookmarks and the navigation with them.

Commands:

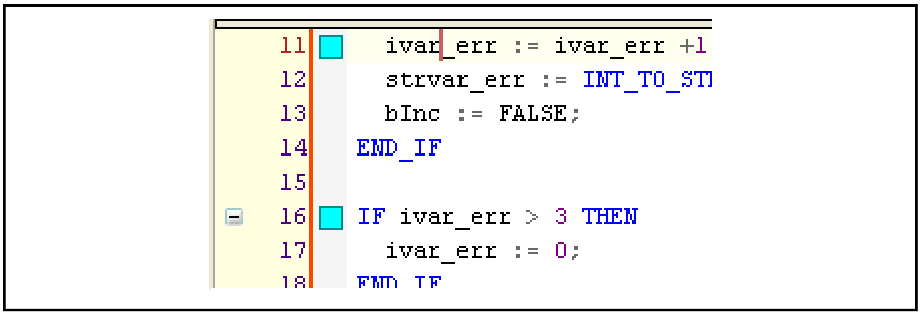
- Switching Bookmarks, page 96,
- Next Bookmark, page 97,
- Previous Bookmark, page 97
- Deleting Bookmark, page 97

Switching Bookmark

Icon:

Menu: **Edit** ▸ **Bookmark** ▸ **Set/Remove Bookmark**

The command can set a bookmark in a text editor to set a bookmark in the current line or remove an existing bookmark. A set bookmark is displayed as cyan blue square in front of the line.



```
  ivar_err := ivar_err +1
  strvar_err := INT_TO_STI
  bInc := FALSE;
END_IF

IF ivar_err > 3 THEN
  ivar_err := 0;
END_IF
```

Fig.3-8: Example of a bookmark in the ST editor

Next Bookmark

Icon:

Menu: **Edit ▸ Bookmark ▸ Next Bookmark**

Use the command to jump to the next bookmark (down) in the text editor.

Previous Bookmark

Icon:

Menu: **Edit ▸ Bookmark ▸ Previous Bookmark**

Use the command to jump to the previous bookmark (up) in the text editor.

Deleting Bookmark

Icon:

Menu: **Edit ▸ Bookmark ▸ Delete bookmark**

Use this command to delete all bookmarks in the active editor window.

3.1.12 Insert File as Text...

Use the "Insert file as text" command to insert a copy of the contents of a text file into the currently active text editor. Open the "Insert file" dialog in order to open a file.

To open the dialog "Insert file", click in the main menu on **Edit ▸ Insert file as text...** .

Find the desired file which has to be available in text format. The file contents are then inserted at the current cursor position.

3.1.13 Browsing

Browsing, Overview

Functions to navigate through the source text are available in the "Edit" category. These are used to search the source text for information on the positions and dependencies (call tree) of a function block or a function currently edited in an editor.

The functionalities for navigating source text are located in the main menu **Edit ▸ Browse**.

Commands:

- Go to Definition, page 97,
- Display Cross References, page 98.

Go to Definition

Icon:

Menu: **Edit** ▸ **Browse** ▸ **Use to definition**

The "Go to definition" command can be used if the cursor is currently pointing to an identifier in an editor window.

It is used to search the project for the position that contains the "definition" of the respective variable or function block. Then, the function block, line or network found is opened in the corresponding editor.

For example, if there are several instances of the selected object in the project, the "Select online status " dialog opens to select whether the function block should be opened in offline or online mode and - for function block instances - whether an instance or the implementation should be displayed.

Alternatively, the command is available in the context menu.

Example

The following function block contains "fbinst", a function block definition, "prog_y", a program call, and "fbinst.out", a function block call.

Example "Go To Definition"

```
VAR
   fbinst:fb1;
   ivar:INT;
END_VAR

prog_y();
ivar:=prog_y.y;
res1:=fbinst.out;
```

Cursor locations after using the "Go to definition" command

- Place the cursor on "prog_y" and the command opens the "prog_y" program in its editor.
- Place the cursor on "fbinst" and the command sets the focus to the declaration window to the line "fbinst:fb1;"
- Place the cursor on "out" and the command opens the "fb1" function block in its editor.

Displaying Cross References

Icon:

The "Display cross references" command displays the cross reference list currently shown in the cross reference list window in the message box. To open the dialog "Display cross references", click in the main menu on **Edit** ▸ **Browse** ▸ **Display Cross References**. Alternatively, "Display Cross References" is available in the context menu.

☞ Cross references in the cross reference list can also be automatically updated.

An automatic update of the cross reference list can be activated in the main menu via **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Smart coding** ▸ **Automatically update cross references when changing selection** (see also "Cross reference list", page 107).

3.1.14 Input Assistance...

Icon:

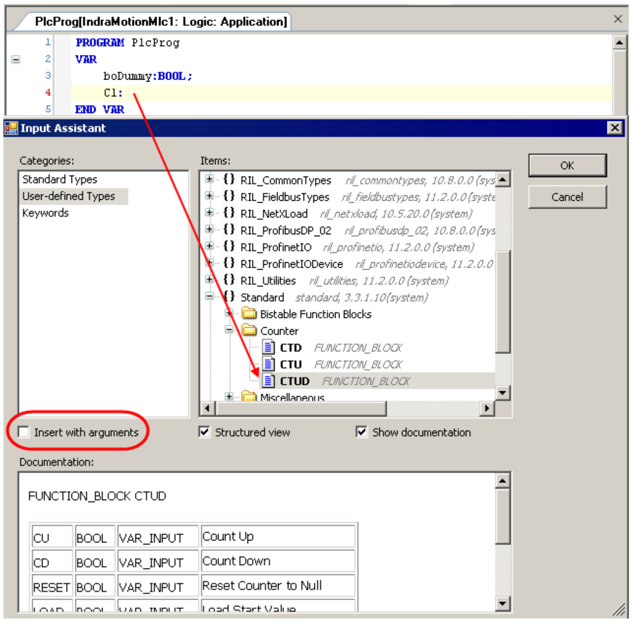
Default shortcut: <F2>

Menu: **Edit** ▸ **Input Assistance...**

The "Input assistance..." command, which opens the input assistance dialog, is only available if the cursor is currently in a programming language editor window.

The dialog provides all programming elements that can be inserted at the current cursor position.

Alternatively, the command is available in the context menu.



PlcProg[IndraMotionMlc1: Logic: Application]
Input Assistant
Categories:
Standard Types
User-defined Types
Keywords
Items:
RIL_CommonTypes ril_commontypes, 10.8.0.0 (sys
RIL_FieldbusTypes ril_fieldbustypes, 11.2.0.0 (syste
RIL_NetXLoad ril_netxload, 10.5.20.0 (system)
RIL_ProfibusDP_02 ril_profibusdp_02, 10.8.0.0 (sys
RIL_ProfinetIO ril_profinetio, 11.2.0.0 (system)
RIL_ProfinetIODevice ril_profinetiodevice, 11.2.0.0
RIL_Utilities ril_utilities, 11.2.0.0 (system)
Standard standard, 3.3.1.10(system)
Bistable Function Blocks
Counter
CTD FUNCTION_BLOCK
CTU FUNCTION_BLOCK
CTUD FUNCTION_BLOCK
Miscellaneous
OK
Cancel
Documentation:

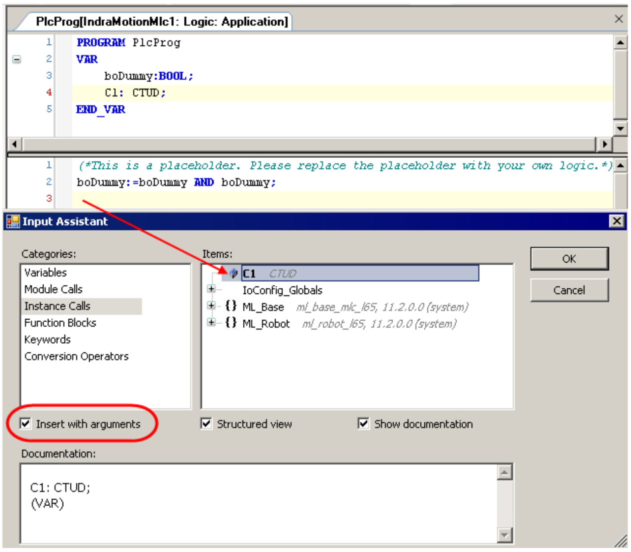
```
PROGRAM PlcProg
VAR
    boDummy:BOOL;
    C1:
END VAR
```

☐ Insert with arguments ☑ Structured view ☑ Show documentation

FUNCTION_BLOCK CTUD

CU	BOOL	VAR_INPUT	Count Up
CD	BOOL	VAR_INPUT	Count Down
RESET	BOOL	VAR_INPUT	Reset Counter to Null
LOAD	BOOL	VAR_INPUT	Load Start Value

Fig.3-9: "Input assistance" dialog: Declaration of an instance (without inserting an argument)



```
PROGRAM PlcProg
VAR
    boDummy:BOOL;
    C1: CTUD;
END_VAR
```

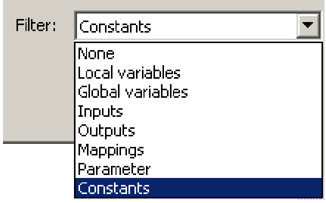
```
(*This is a placeholder. Please replace the placeholder with your own logic.*)
boDummy:=boDummy AND boDummy;
```

☑ Insert with arguments ☑ Structured view ☑ Show documentation

C1: CTUD;
(VAR)

Fig.3-10: "Input assistance" dialog: Implementation of an instance (with inserting an argument)

The elements are sorted according to category. For the "Variables" category, additionally set a **filter** for the variable type, like "Local variables", "Global variables", "Constants", etc.


Filter: Constants

None
Local variables
Global variables
Inputs
Outputs
Mappings
Parameter
Constants

Fig.3-11: "Filter" selection window

Insert with arguments

If the "Insert with arguments" option is selected, elements that have arguments, e.g. functions, are inserted with their arguments at the cursor position.

Implementation, Inserting with Arguments, C1: CTUD;

```
C1(
    CU:= ,
    CD:= ,
    RESET:= ,
    LOAD:= ,
    PV:= ,
    QU=> ,
    QD=> ,
    CV=> );
```

Structured view

If the "Structured view" option is selected, the elements are arranged in a tree structure based on subject and supplemented by icons.

If it is not activated, the elements are listed one after the other in alphabetical order. The function block to which each belongs is also listed (e.g. "GVL1.gvar1").

Show documentation

If the "Show documentation" option is selected, the dialog is expanded to include the "Documentation" field. Here a help text is shown for the currently highlighted element. This help text is automatically generated from a comment that was entered when the element was created.

3.1.15 Declare Variable...

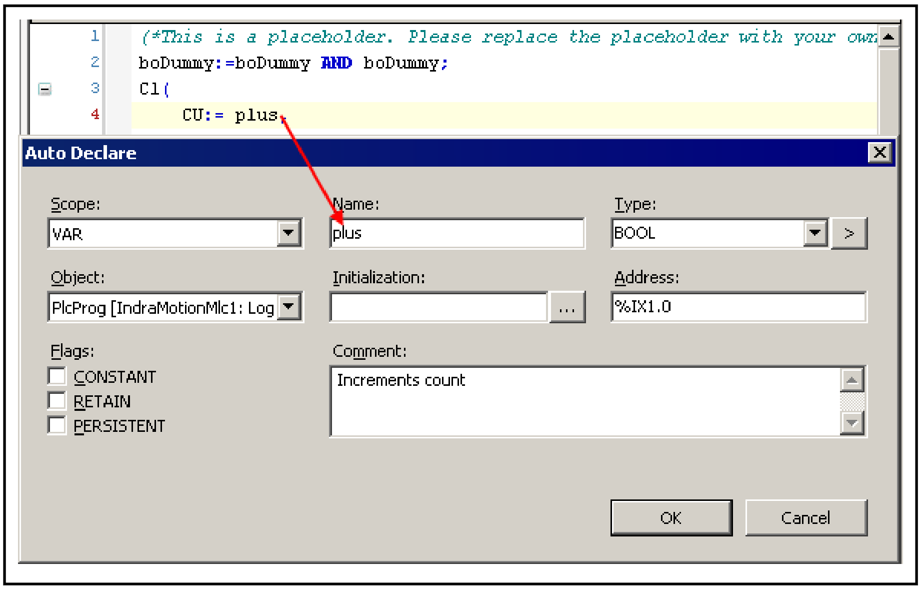
Default shortcut: <Shift>+<F2>

The command opens the "Declare variable" dialog, which supports the Declaration of a variable, page 503,.

To do this, the cursor has to be positioned in a line of the programming section of the editor that contains a variable that has not yet been declared or a variable that has already been declared has to be highlighted.

☞ If the dialog is to be opened automatically as soon as a line that contains a variable not yet declared is exited, the "Auto Declare" function has to be selected in the options for smart code editing. More information on "Auto Declare" or "Smart code editing" is located in "Options, Smart Coding", page205, "Settings".

To open the "Declare variable" dialog, enter the variable name and click in the main menu on **Edit ▸ Declare Variables...** or press <Shift>+<F2>.



```
(*This is a placeholder. Please replace the placeholder with your own
boDummy:=boDummy AND boDummy;
C1(
    CU:= plus,
```

Auto Declare

Scope: VAR

Name: plus

Type: BOOL

Object: PlcProg [IndraMotionMlc1: Log

Initialization:

Address: %IX1.0

Flags:
□ CONSTANT
□ RETAIN
□ PERSISTENT

Comment: Increments count

Fig.3-12: Dialog for variable declaration

A few fields in the dialog are automatically filled with default values, but these can be edited: Name, Data type, Object

Visibility In the "Visibility" selection list, "VAR" is the default entry for "local variables". Alternatively, another validity range can be set in the selection list.

Name The default entry in the "Name" field is the name of the variable that has not yet been declared as it was entered in the editor.

Data type The default entry in the "Data type" selection list is "INT" if this variable is the first one to be declared in the editor line.

However, if there is already a declared variable in the line, the data type for this variable is listed, here "BOOL". To change this setting, use the ... button to access input assistance which provides all data types, see Input assistance..., page 98.

Object The default entry in the "Object" selection list is the name of the Project Object, page 28 that is currently edited. To enter a different object, in which the variable is to be declared, select one from the selection list. If a global variable list should be declared, it contains for example (visibility: VAR_GLOBAL) all available global variable lists of the project.

Initial value The "Initial value" field is empty by default. A valid value, which corresponds to the data type of the variable and which is used to initialize the variable, can be entered here. If the field is left empty, it is initialized with the default value.

The ... button opening the "Input assistance" dialog.

Address The "Address" field is empty by default. In the "Address" field, an IEC address can be directly assigned to the variable (AT Declaration, page 512).

Automatically created declaration line

```
plus AT %IX1.0: BOOL;
```

Flags No flags are enabled by default. "CONSTANT", "RETAIN" or "PERSISTENT" can be selected. Define whether the variable is a constant or a "remanent variable", page 519, by selecting the corresponding "flags". The corresponding attribute keyword is then inserted into the declaration after the keyword defined in the "Visibility" field, e.g. "VAR CONSTANT".

Comment

The "Comment" field is empty by default. Enter a text to be added to the declaration as comment.

Breaks can be inserted with the shortcut <Ctrl>+<Enter>.

The comment is then displayed in the declaration editor in the line above the variable declaration.

Click on the "OK" button to close the "Declare variable" dialog. The variable declaration appears in the declaration editor according to the IEC syntax.

3.1.16 Advanced

Advanced Overview

There are additional commands for editing text in the project under "Edit".

By default, these commands are located in Edit ▸ Advanced .

Commands:

- Go to..., page 102
- Switch to upper case, page 102
- Switch to lower case, page 103
- Corresponding bracket, page 103
- Highlight up to corresponding bracket, page 103

Go to...

Default shortcut: <Ctrl>+<G>

Menu: Edit ▸ Advanced ▸ Go To...

Jump to a specific line within the active text editor using the command. The command opens the "Go To Line" dialog to enter the desired line number (jump target).

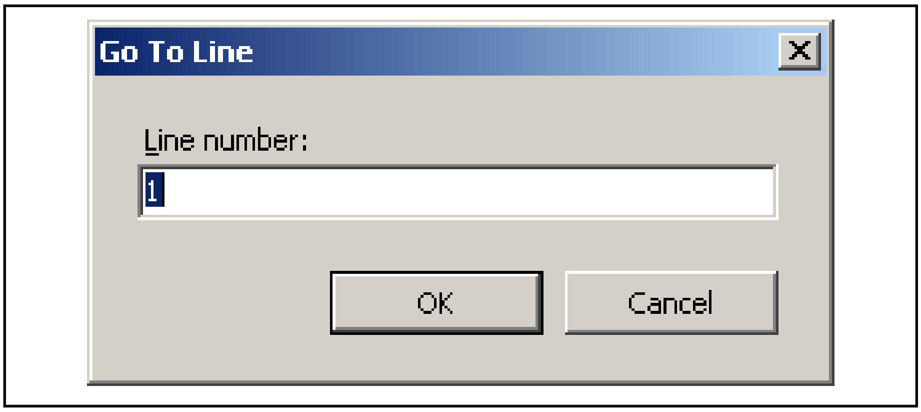


Fig.3-13: Dialog – Go to line

Then close the dialog with the "OK" button. The cursor is placed at the beginning of the corresponding line.

Switch to Upper Case

Default shortcut: <Ctrl>+<Shift>+<U>

Menu: Edit ▸ Advanced ▸ Switch to upper case

Use the command to display the currently highlighted text area in upper case letters.

See also Switch to lower case, page 103.

Switch to Lower Case

Default shortcut: <Ctrl>+<U>

Menu: Edit ▸ Advanced ▸ Switch to lower case

Use the command to display the currently highlighted text area in lower case letters.

See also Switch to upper case, page 102.

Corresponding Bracket

Menu: Edit ▸ Advanced ▸ Corresponding bracket

When the cursor is positioned at a bracket, the command can be used to jump to the next related opening or closing bracket. This applies to brackets within program lines and to "parenthesis enclosures", page212 that extend across several lines.

Highlight up to Corresponding Bracket

Menu: Edit ▸ Advanced ▸ Highlight up to corresponding bracket

When the cursor is positioned at a bracket, the command is used to highlight all of the code lines up to the corresponding closing or opening bracket. The highlighted area can extend upward or downward. This applies to brackets within program lines and to "parenthesis", page 212 across several lines.

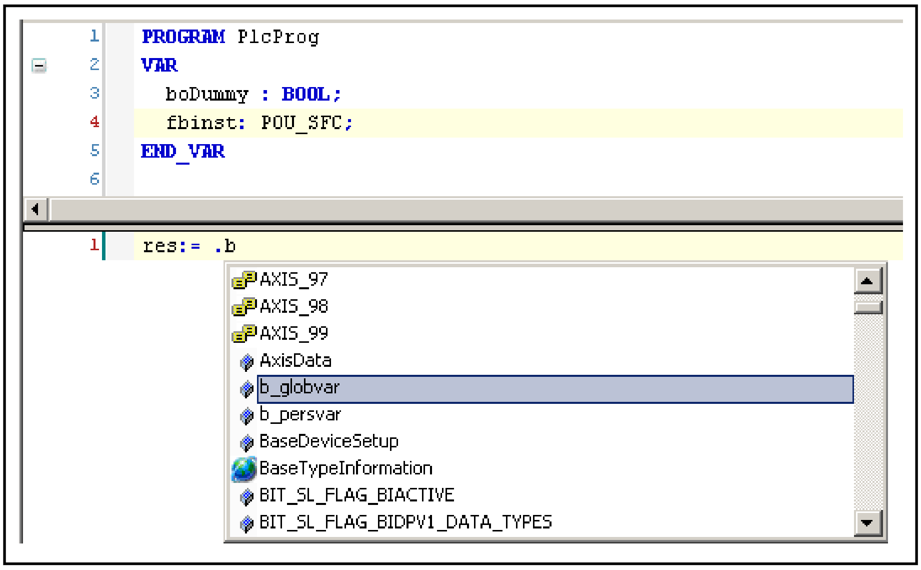
3.1.17 "List Components" Function

Entering text is supported in the context of the IEC 61131-3 standard.

The IEC 61131-3-text editor also provides a type of "Intellisense" functionality that can be enabled or disabled using the dialog in IndraWorks ▸ Tools ▸ Options ▸ IndraLogic 2G ▸ Smart coding.

"List components" makes it easier to enter valid identifiers:

- If only a point "." is entered in a position where a global identifier can be entered, a selection list of all available global variables appears. A variable can be selected and inserted after the period.

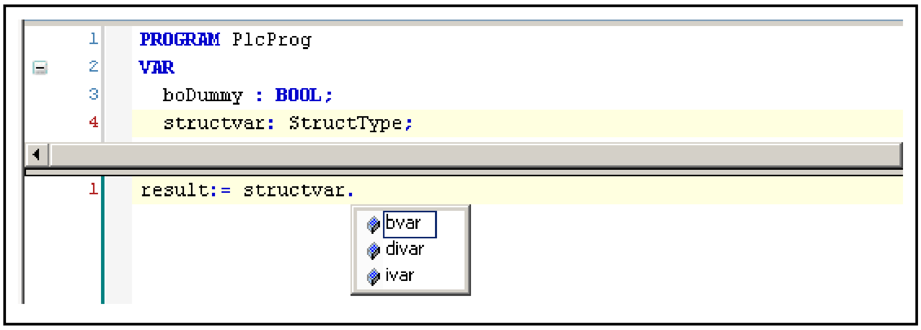


```
PROGRAM PlcProg
VAR
  boDummy : BOOL;
  fbinst: POU_SFC;
END_VAR

res:= .b
```

Fig.3-14: "List components" provides global variables (starting with 'b')

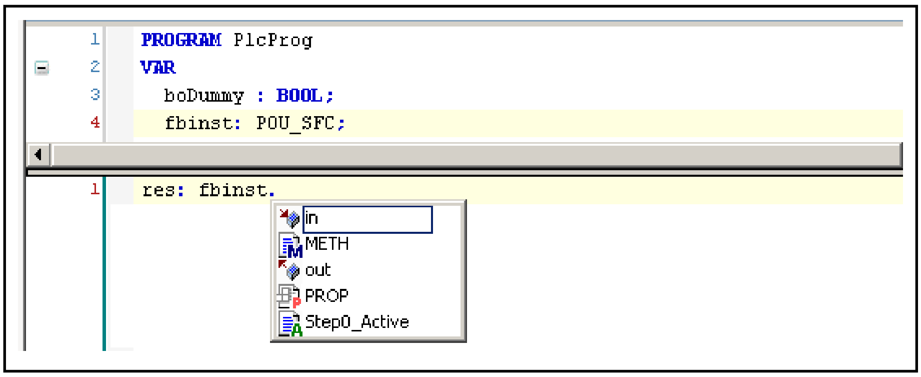
- If a period is entered after a function block instance or structure variable, a selection list provides all of the input and output variables of the function block or all structure components. With a double-click or <Enter>, insert the variable selected in this list after the period.



```
PROGRAM PlcProg
VAR
  boDummy : BOOL;
  structvar: StructType;
```

```
result:= structvar.
```

Fig.3-15: "List components" provides structure components



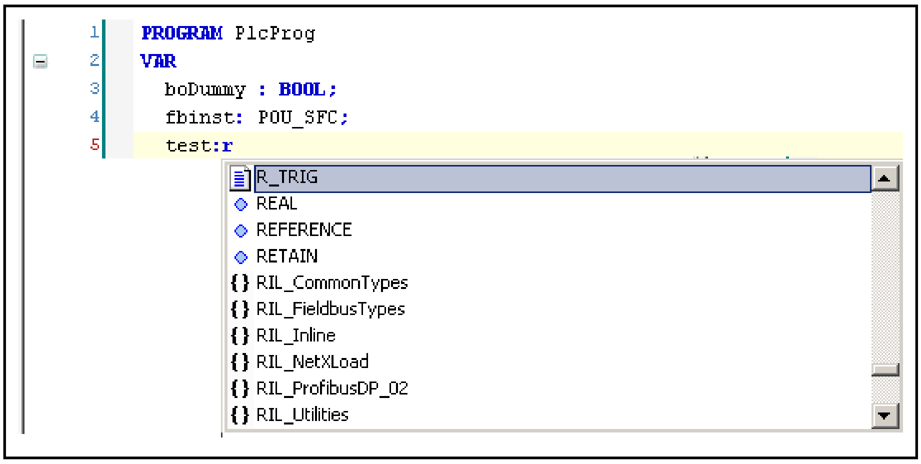
```
PROGRAM PlcProg
VAR
  boDummy : BOOL;
  fbinst: POU_SFC;
```

```
res: fbinst.
```

Fig.3-16: "List components" provides function block variables (components)

- If any character string is entered and <Ctrl>+<space bar> pressed, a selection list of all available POUs and global variables appears.

  The first element in this list that begins with the previously entered character string is automatically selected and can be inserted in the editor with a double-click or by using the <Enter> key.



```
PROGRAM PlcProg
VAR
  boDummy : BOOL;
  fbinst: POU_SFC;
  test:r
```

Fig.3-17: "List components" provides list selection (starting with 'r')

3.2 View

3.2.1 View, General Information

The menu item "View" allows the windows and toolbars within IndraWorks Engineering to be activated and deactivated.

- Project Explorer, main window to the left of the workspace
- Device library, main window to the right of the workspace
- *Other Windows*
  - Output
  - Messages, page 105
  - Search results
  - Tools, page 106
  - Cross reference list, page 107
  - Element properties..., page 108
- Toolbars, (IndraLogic selection), page 109
  - Bookmarks, page 109
  - IndraLogic, page 110, important commands for menu items "Create" and "Debug"
- Object properties in the Project Explorer, page 238.

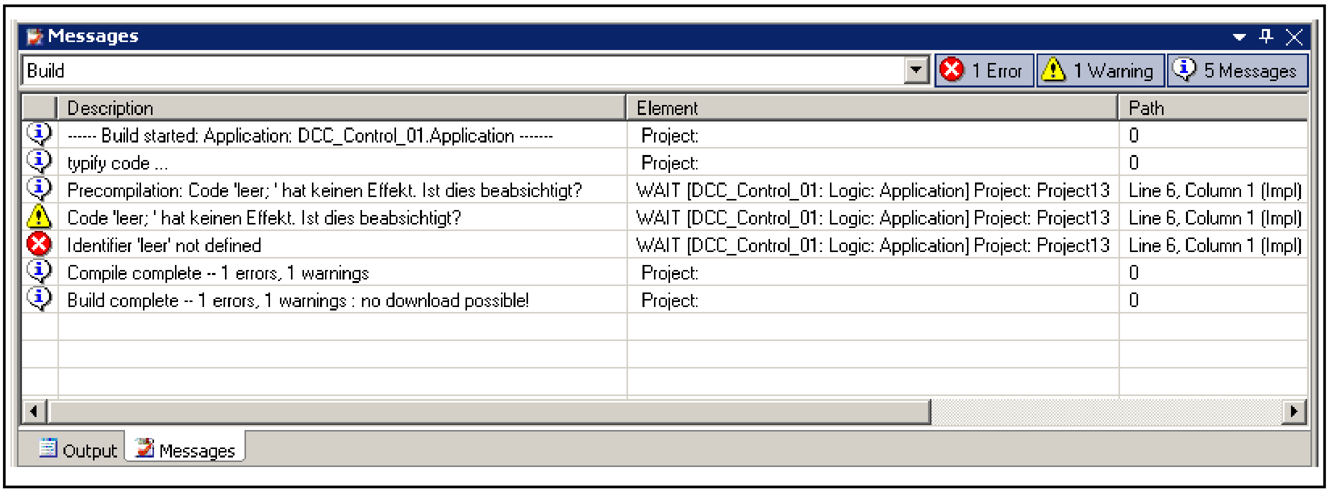
If required, the menu structure can be changed using the dialog in **IndraWorks ▸ Tools ▸ Customize.**

3.2.2 View - Other Windows

Messages

Menu: **View ▸ Other Windows ▸ Messages**

This command opens a window in which messages with regard to the project are displayed.



	Description	Element	Path
ⓘ	------ Build started: Application: DCC_Control_01.Application ------	Project:	0
ⓘ	typify code ...	Project:	0
ⓘ	Precompilation: Code 'leer; ' hat keinen Effekt. Ist dies beabsichtigt?	WAIT [DCC_Control_01: Logic: Application] Project: Project13	Line 6, Column 1 (Impl)
⚠	Code 'leer; ' hat keinen Effekt. Ist dies beabsichtigt?	WAIT [DCC_Control_01: Logic: Application] Project: Project13	Line 6, Column 1 (Impl)
⊗	Identifier 'leer' not defined	WAIT [DCC_Control_01: Logic: Application] Project: Project13	Line 6, Column 1 (Impl)
ⓘ	Compile complete -- 1 errors, 1 warnings	Project:	0
ⓘ	Build complete -- 1 errors, 1 warnings : no download possible!	Project:	0

Fig.3-18: Window: Messages, example after "Generate code"

Messages can include

- Errors⊗
- Warnings ⚠ or
- Information ⓘ

.

Messages are moreover categorized by component or functionality.

Messages regarding project syntax checks e.g. belong to the 'Precompile' category; messages for compiling the project to the 'Compile' category (e.g.

compilation errors, code size). Messages of the "Import", "Library Manager", etc. categories are possible as well.

A filter for the desired **message category** can be set in the selection list at the top of the message window (refer to the preceding figure) except for the **Precompile** messages which are always displayed in the area below the actual message table.

The messages in the selected category are output in table form and include the following information:

- Description (message text)
- Project (project name)
- Object (name of the associated project object)
- Position (position in the object that triggered the message, e.g. line number, network number, etc.).

If display of a certain message type is to be hidden or shown, use the buttons in the upper right-hand corner of the window. **Errors**, **"arnings**, **Information**, **Messages**. The buttons also show the currently available number of messages of the type.

Switch between the messages in the message window or jump from the currently selected message to the respective **position in the object**. For this purpose, the 'Next message', 'Previous message' and 'Go to source text position' **commands** are available.

Switch between the messages in the message window or jump from the currently selected message to the respective position in the object.

To do this, use these commands:

- "Next message" and
- "Previous message"

Double-click on a message entry in the table to move to the source text position.

The task list can also be sorted according to a variety of criteria using the "Sort by" command (context menu).

Choose from the following variants:

- "Time sequence"
- "Priority"
- "Description"
- "Origin" and
- "Position"

Double-click on a message entry in the table or click on the precompile message underlined in blue to move to the source text position.

Tools

Menu: **View** ▸ **Other Windows** ▸ **ToolBox**

This command opens a window with "tools" ("Toolbox") for the **currently active editor.**

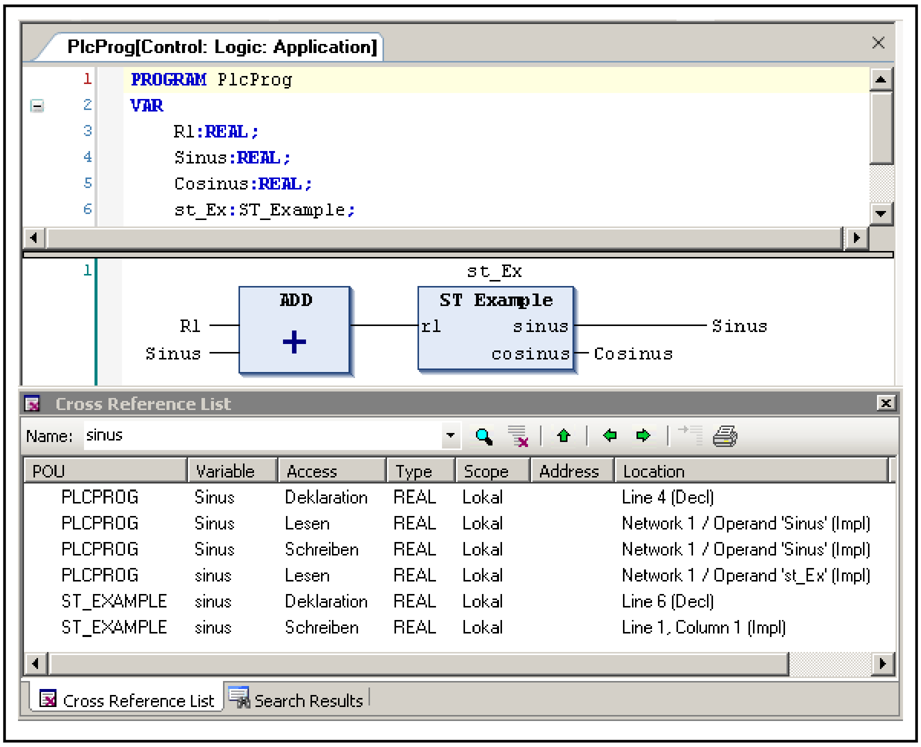
By default, this ToolBox is available for graphical editors or for the visualization editor. It contains graphical programming elements that can be dragged into the editor window using "drag&drop".

Examples:

- ToolBox for CFC editor, page 311
- ToolBox for FBD/LD/IL editor, page 355
- Toolbox for the Visualization editor, page 446.

Cross Reference List

This command opens a window in which the cross references for a specific project variable are listed. Cross references include the positions in the project where the variable is used.



PlcProg[Control: Logic: Application]

```
PROGRAM PlcProg
VAR
    R1:REAL;
    Sinus:REAL;
    Cosinus:REAL;
    st_Ex:ST_Example;
```

Cross Reference List

Name: sinus

POU	Variable	Access	Type	Scope	Address	Location
PLCPROG	Sinus	Deklaration	REAL	Lokal		Line 4 (Decl)
PLCPROG	Sinus	Lesen	REAL	Lokal		Network 1 / Operand 'Sinus' (Impl)
PLCPROG	Sinus	Schreiben	REAL	Lokal		Network 1 / Operand 'Sinus' (Impl)
PLCPROG	sinus	Lesen	REAL	Lokal		Network 1 / Operand 'st_Ex' (Impl)
ST_EXAMPLE	sinus	Deklaration	REAL	Lokal		Line 6 (Decl)
ST_EXAMPLE	sinus	Schreiben	REAL	Lokal		Line 1, Column 1 (Impl)

Fig.3-19: Example of a cross reference list

If **all cross references within the entire project** are to be listed: Jog the variable identifier manually or copy it from the editor window in to the **Name** field by means of copy&paste and confirm by means of the <Enter> key or click on the 🔍 button

If **only the cross references within the same POU** are to be listed: Mark either the variable identifier in an editor window and drag it into the cross reference window using the mouse or activate the Automatically update cross references when changing selection, page 205 option and place the mouse cursor in the identifier name in the editor window or select it. In this case, the identifier is automatically applied to the **Name** field and the cross references are listed below.

An **automatic update** of the cross reference list can be activated in **Tools ▸ Options ▸ IndraLogic 2G ▸ Smart coding** .

If a valid identifier is entered, the use positions found are displayed in table form with the following information about the variable:

POU	Name of the function block in which the variable is used.
Variable	Variable name

Access	Use of the variable at the given position: Declaration/read/write/call
Type	Data type
Range	Validity range: global/local
Address	IEC address, if defined
Position	Position of the variable used within the editor (e.g. line number, network number)
Comment	Declaration comment for the variable

Fig.3-20: Cross reference list, table header explanation

The list can be **sorted** alphabetically by column. By clicking on the column title, switch the sorting arrangement between ascending and descending order.

Double-clicking a line in the cross reference list **opens the respective POU** and selects the position identified. This corresponds with the use of the

⇧button (**Show position**) if an entry in the list is selected.

Use ⇦ (**Show previous position**, shortcut: <Shift> + <F4>) to jump to the previous entry in the list.

Use ⇨ (**Show next position**, shortcut: <F4>) to jump to the next entry in the list.

Use ☰ (**Go to definition**, shortcut: <F2>) to jump to the position where the respective variable is declared. To do this, the corresponding declaration editor opens and the variable is highlighted there.

The ☰ button corresponds to the command Display cross references, page 98, which outputs a **display of the current cross reference list in the message window**.

This can be useful if the current list is to be kept available even though the automatic update is active and the list in the cross reference window can change.

View - Element Properties

View - Element Properties, General Information

Icon: ☰

Menu: **View** ▸ **Other Windows** ▸ **Element properties**

When this command is used, the main window "Properties" opens to the right of the workspace.

If an appropriate object is highlighted in the workspace, its properties appear in this window.

- Element properties of sequential function chart objects, page 108
- Element properties of visualization elements, page 109

Using the command again closes this window.

Element Properties - Sequential Function Chart Object (SFC)

Icon: ☰

Menu: **View** ▸ **Other Windows** ▸ **Element properties**

This command opens the Properties window for the currently selected SFC element.

Element properties such as name, comment, step times and associated actions are displayed in a subdivided table.

These can be edited in the 'Value' column:

Go to input mode by clicking the mouse and for steps place or remove a checkmark in the checkbox to enable or disable the "Initial step" property.

See also the detailed information for the individual

- Element properties, page 404.

Element Properties - Visualization Element

Icon:

Menu: **View** ▸ **Other Windows** ▸ **Element properties**

This command opens the Properties window for the currently selected visualization element.

The element properties are displayed in a table, sorted in groups. Click on the plus or minus sign in front of the group name to display or hide the related parameters.

Which properties can be configured depends on the visualization element.

A description of the individual parameters is located in the visualization editor, page 451,.

3.2.3 View - Toolbars

View - Toolbars, General Information

The IndraLogic PLC programming system provides the following toolbars that are activated if editors associated with the programming system support it and are active:

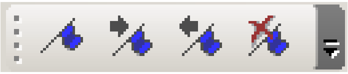
- Bookmarks, page 96, bookmarking functionality in text-based editors
- IndraLogic, page 110, frequently used functionality of the menu items **Create** and **Debug**.

Toolbars can be changed based on user-defined needs in **Tools** ▸ **Customize** .

The IndraLogic PLC programming system uses the following toolbars in connection with the comparison of PLC objects:

- Toolbar in the Comparison dialog, page 110.
- Toolbar in the Comparison results dialog, page 111.

Bookmark, Overview

Icons: 

The "bookmarks" toolbar provides bookmarking functionality for text editors. It can be activated using **View** ▸ **Toolbars...**.

Bookmarks can be added to lines to facilitate navigation in long programs.

Bookmarks remain when closing the editor window.

If IndraWorks Engineering is closed, all bookmarks are removed.

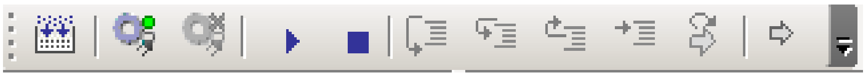
The commands described below allow the setting of bookmarks and the navigation with them.

Commands:

- Switching Bookmarks, page 96,
- Next Bookmark, page 97,
- Previous Bookmark, page 97
- Deleting Bookmark, page 97

IndraLogic Overview

Icons:



The "IndraLogic" toolbar contains frequently used commands for the menu items **Create** and **Debug**.

- , **Create** ▸ Compile, page 124; the command starts the compilation for the application that is currently active.
- , **Debug** ▸ Login, page 127; this command connects the programming system, i.e. the active application with the control, thus establishing the **online mode**.
- , **Debug** ▸ Logout, page 128; this command ends the connection between the development system and the control, causing a return to **offline mode**.
- , **Debug** ▸ Start, page 133; this command starts the application on the control.
- , **Debug** ▸ Stop, page 133; this command stops the application on the control.
- , **Debug** ▸ Procedure Step, page 142; this command carries out a sequence of defined steps in the program in online mode.
- , **Debug** ▸ Stop, page 133; this command carries out a single step in the program in online mode.
- , **Debug** ▸ Execute to Return , page 143; this command starts the processing of a program loop up to the return in online mode.
- , **Debug** ▸ Execute to Cursor, page 143; this command starts the processing of program lines up to the cursor in online mode.
- , **Debug** ▸ Determine Next Instruction , page 143; this command specifies the instruction that is to be processed next.
- , **Debug** ▸ Display Next Instruction , page 143, this command shows the instruction that is to be processed next.

Comparison, Toolbar

The toolbar in the upper part of the dialog serves the navigation and provides the following functions for comparison/merger:

Symbol	Description
←	**Show previous element:** Clicking this symbol navigates to the previous element in the history of the already displayed elements.
→	**Show next element:** Clicking this symbol navigates to the next element in the history of the already displayed elements.
↓	**Select next difference:** Clicking this symbol selects the next difference.
↑	**Select previous difference:** Clicking this symbol selects the previous difference.
✓	**Preselect for the merger:** Clicking this symbol preselects the selected line for the merger. Note: If no merger is possible for the selected line, no preselection can be made.
☒	**Reset preselection for merger:** Clicking this symbol no longer preselects the selected line for the merger.
↖	**Start merger:** Clicking this symbol carries out the merger for all preselected lines.

Fig.3-21: Toolbar

Comparison Results, Toolbar

The toolbar in the upper part of the "Comparison results" dialog serves as filter and provides the following functions:

Symbol	Description
✱	**Show all elements:** Clicking this symbol shows all differing and complying elements.
≠	**Show only differing elements:** Clicking this symbol shows only the differing elements.
=	**Show only complying elements:** Clicking this symbol shows only the complying elements.

Fig.3-22: Toolbar

3.3 Project

3.3.1 Archiving and Restoring

Archiving IndraLogic Projects

A detailed description of how to archive projects is located in the IndraWorks help under Archiving Projects.

When archiving IndraLogic projects, a check determines which libraries and device descriptions are required for the project.

These are saved locally or on the FTP server in the archive.

Restoring IndraLogic Projects

A detailed description of how to restore projects is located in the IndraWorks help under Restoring Projects.

When restoring IndraLogic projects, the libraries and device descriptions required for the project to be restored are determined from the archive.

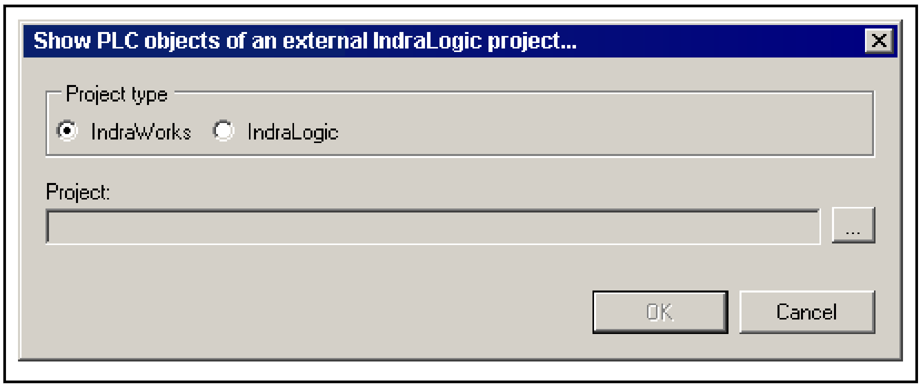
If these files are not in the Library Repository..., page 185, or in the device database, the respective repository is supplemented from the archive when the project is opened for the first time after restoration.

3.3.2 Displaying PLC Objects of an External IndraWorks Project

Icon:

Menu: **Project** ▸ **Display PLC objects of an external IndraLogic project**

The command allows to open an external project and to copy PLC components from its content.



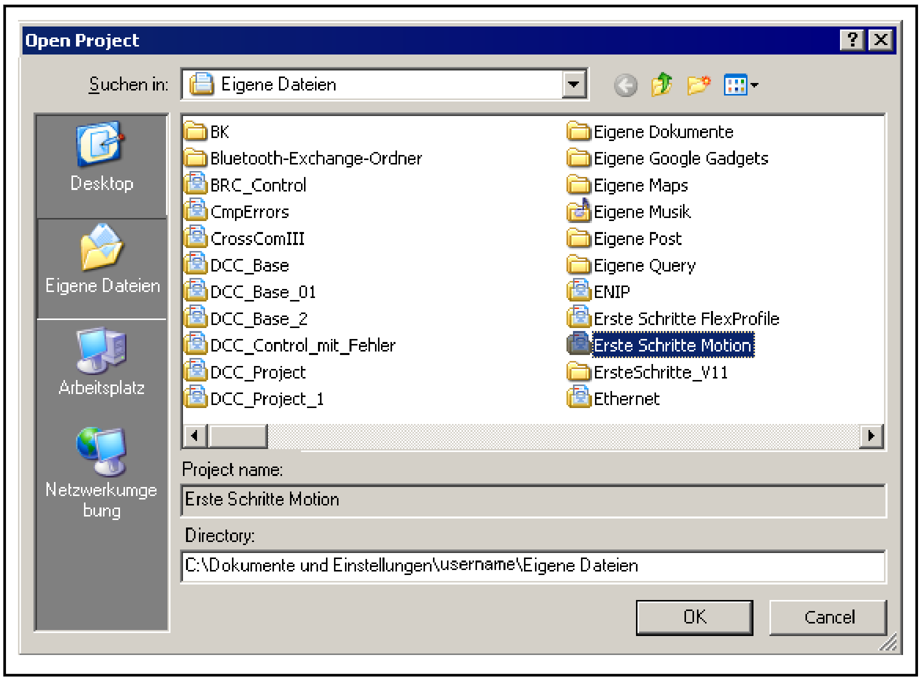
Project type: ● IndraWorks ○ IndraLogic

Project:

Fig.3-23: Dialog: Selecting the source

Only the "IndraWorks" and "IndraLogic" project types are permitted.

With the "IndraWorks" project type, the standard dialog for selection of an IndraWorks project opens after clicking on ....



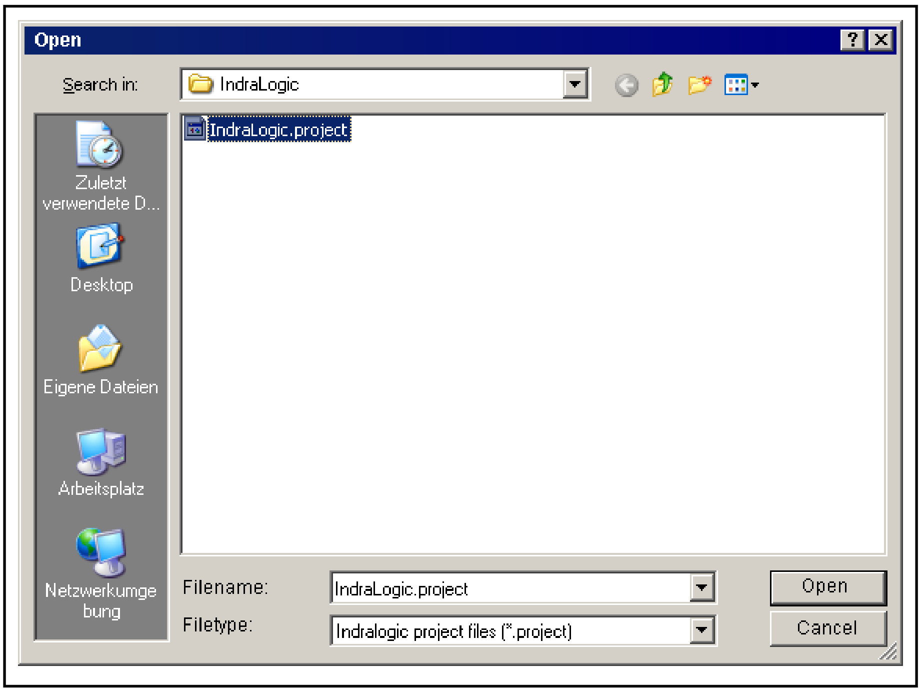
Suchen in: Eigene Dateien

Project name: Erste Schritte Motion

Directory: C:\Dokumente und Einstellungen\username\Eigene Dateien

Fig.3-24: Dialog: Selecting the IndraWorks project

With the "IndraLogic" project type, you have to navigate to the desired IndraLogic project after operation of ..., e.g. **First steps Motion ▸ IndraLogic ▸ IndraLogic.project.**



Search in: IndraLogic

IndraLogic.project

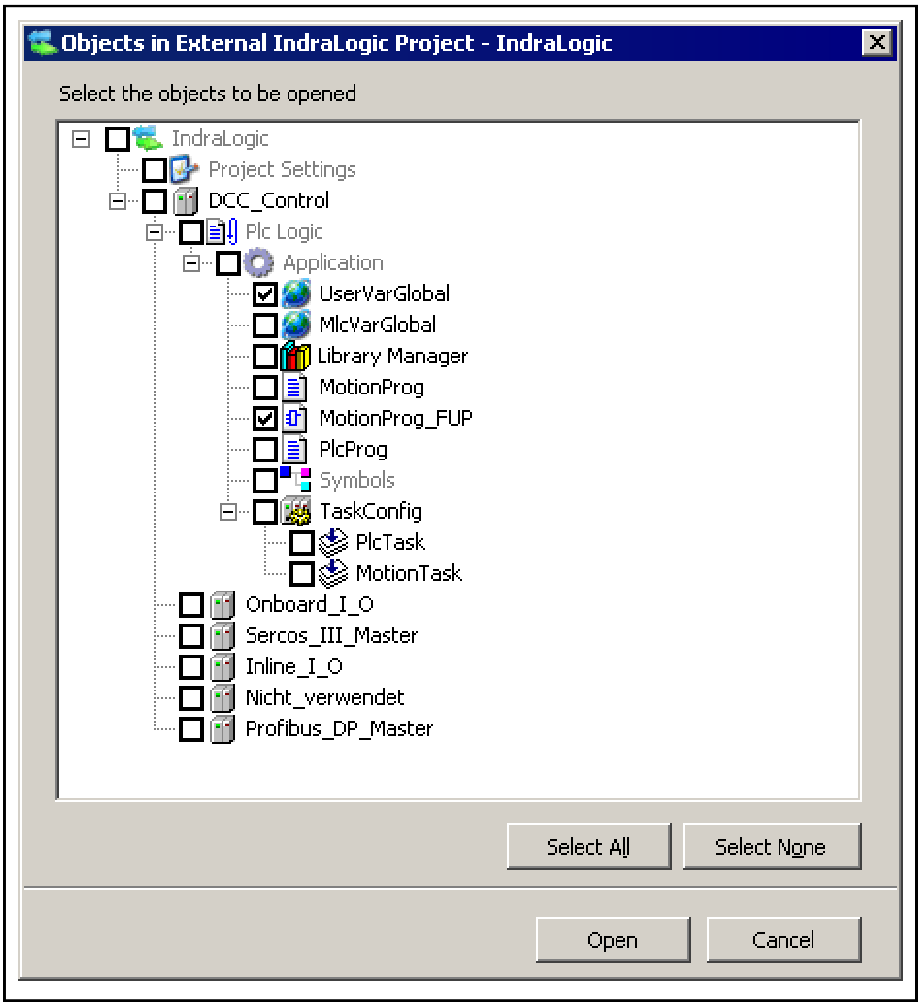
Filename: IndraLogic.project

Filetype: Indralogic project files (*.project)

Fig.3-25: Dialog: Selecting the IndraLogic project

For both project types, a dialog appears to select the objects to be opened:

Menu Items


Objects in External IndraLogic Project - IndraLogic

Select the objects to be opened

- □ IndraLogic
  - □ Project Settings
  - □ DCC_Control
    - □ Plc Logic
      - □ Application
        - ☑ UserVarGlobal
        - □ MlcVarGlobal
        - □ Library Manager
        - □ MotionProg
        - ☑ MotionProg_FUP
        - □ PlcProg
        - □ Symbols
        - □ TaskConfig
          - □ PlcTask
          - □ MotionTask
    - □ Onboard_I_O
    - □ Sercos_III_Master
    - □ Inline_I_O
    - □ Nicht_verwendet
    - □ Profibus_DP_Master

Fig.3-26: Dialog: PLC object selection

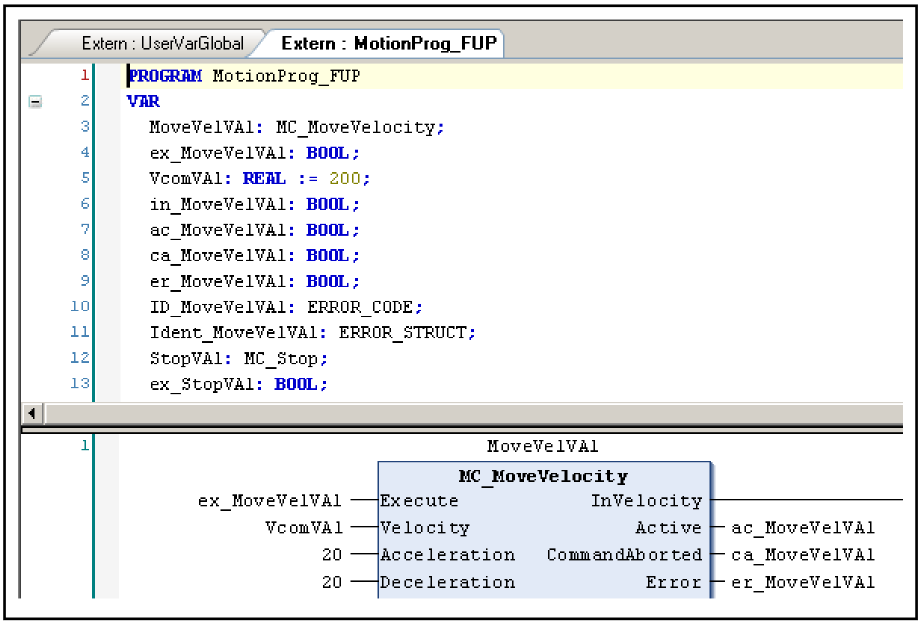
Objects that can be selected (i.e. displayed) have black font, all others are shown in gray font.

Using the "Select All" or "Select None" button, the user can mark all objects for opening or remove their marking.

Clicking the button in front of a selectable object with the mouse marks individual objects for opening.

If the user presses the "Open" button, the selected objects are opened for read access. The user can now e.g. mark text passages in these objects and copy them to objects of the open project.

The "External:" entry in the title of the document window shows the user that the opened object comes from an external project.



```
PROGRAM MotionProg_FUP
VAR
  MoveVelVAl: MC_MoveVelocity;
  ex_MoveVelVAl: BOOL;
  VcomVAl: REAL := 200;
  in_MoveVelVAl: BOOL;
  ac_MoveVelVAl: BOOL;
  ca_MoveVelVAl: BOOL;
  er_MoveVelVAl: BOOL;
  ID_MoveVelVAl: ERROR_CODE;
  Ident_MoveVelVAl: ERROR_STRUCT;
  StopVAl: MC_Stop;
  ex_StopVAl: BOOL;
```

Fig.3-27: Opened external PLC object

3.3.3 Data Transfer

Data Transfer, Overview

IndraWorks provides a way to export **complete projects** and then to import them again.

These projects contain components associated with the related system.

An export is performed from the respective object where the menu items "Export" and "Import" can be called in its context menu.

See also Exporting and importing project data, Rexroth IndraWorks 12VRS, Engineering, DOK-IWORKS-ENGINEE*V12-APxx-EN-P.

IndraLogic itself provides a way to import **IndraLogic projects** created using different platforms. Blocks, data types, visualizations and resources (global variables, variable configuration, trace recording, control configuration, task configuration, etc.) can be imported.

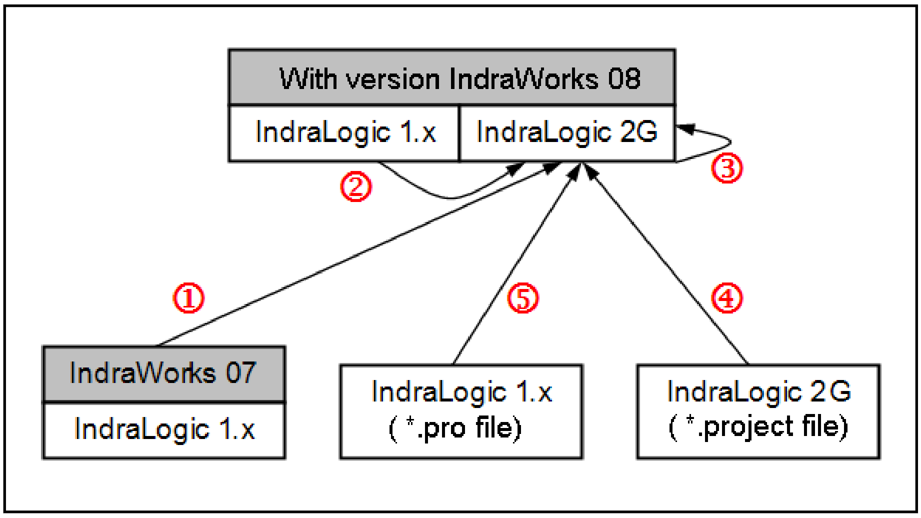
In order for projects to be imported, the following prerequisites have to be met:

1. IndraWorks Version 8 or later
2. Open IndraLogic 2G project
3. IndraLogic 2G libraries
4. Device description files

The "Application" node context menu can be used to import existing IndraLogic 1.x projects (both stand-alone projects and projects embedded in IndraWorks) and IndraLogic 2G projects into IndraLogic 2G projects.

The following figure clarifies the relationships.

Menu Items



① IndraLogic 1.x projects embedded in IndraWorks (Version 7) are automatically converted and then imported.

② IndraLogic 1.x projects embedded in IndraWorks (Version 8) are automatically converted and then imported.

③ IndraLogic 2G projects embedded in IndraWorks (Version 8) are imported. In this case the project is not converted, since both file types are identical.

④ Stand-alone IndraLogic 2G project files (*.project files) are imported.

⑤ Stand-alone IndraLogic 1.x project files (*.pro files) are automatically converted and then imported.

Fig.3-28: Schematic representation of various imports

Importing IndraLogic 1.x Projects

Importing projects

To import an IndraLogic 1.x project, highlight the "Application node" in the Project Explorer and select "Data Import..." from the context menu.

Alternatively, also import projects using the main menu **Application** ▸ **Data Import...** .

☞ Note that you can have several applications in one IndraWorks project. The data transfer applies only to the highlighted "Application" nodes.

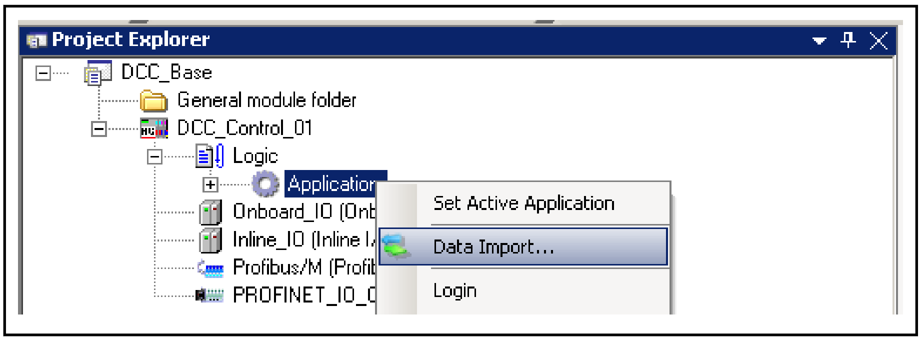


Fig.3-29: Importing IndraLogic projects

The "Data Import..." dialog appears.

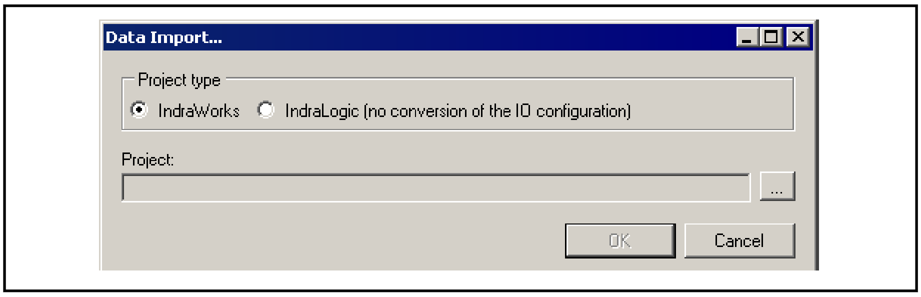

Data Import...
Project type
● IndraWorks ○ IndraLogic (no conversion of the IO configuration)
Project:
...
OK
Cancel

Fig.3-30: Selecting project type and project path

Under "Project type" select the type of your source project.

- IndraWorks:

  In this case, an IndraLogic project (*.project) is embedded in an IndraWorks project.

- IndraLogic (IO configuration is not converted)

  In this case, a stand-alone IndraLogic project file (*.pro or .project) is the source project.

☞ Note that the control configuration is not imported for stand-alone IndraLogic 1.x projects (*.pro).

Use the ... button to open the "Open Project" dialog and move to the project to be imported.

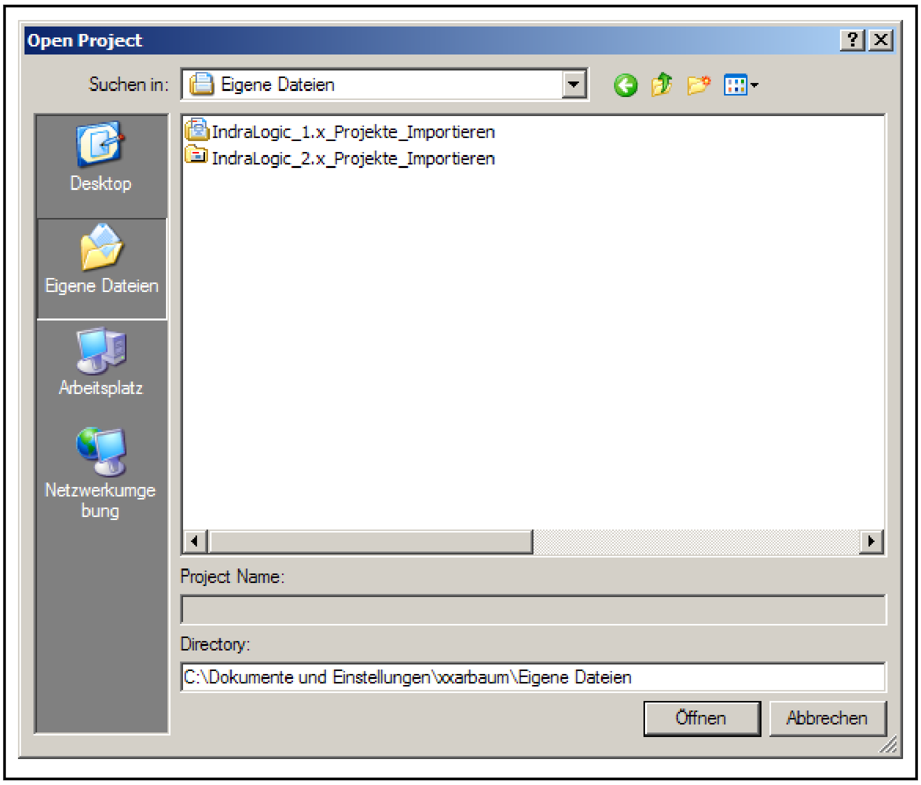
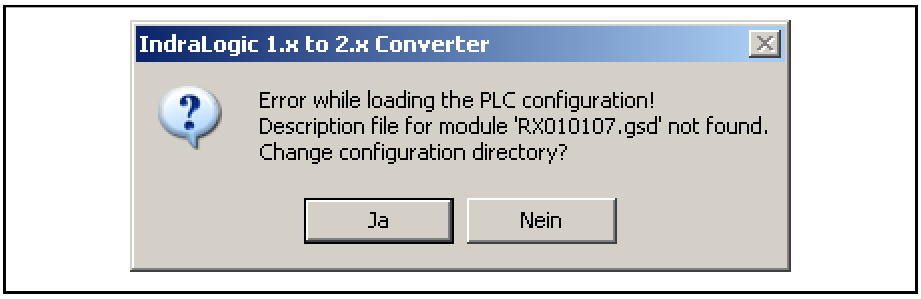

Open Project
Suchen in: Eigene Dateien
Desktop
Eigene Dateien
Arbeitsplatz
Netzwerkumgebung
IndraLogic_1.x_Projekte_Importieren
IndraLogic_2.x_Projekte_Importieren
Project Name:
Directory:
C:\Dokumente und Einstellungen\xxarbaum\Eigene Dateien
Öffnen
Abbrechen

Fig.3-31: File selection

Confirm your selection with "Open".

☞ Before the import takes place, the project is recompiled. Compilation errors might occur (e.g. if a library is missing), which can be ignored in terms of the project import.

The download for the project starts. In a later step, device descriptions can be installed to be able to ignore the following or similar error messages with "No".

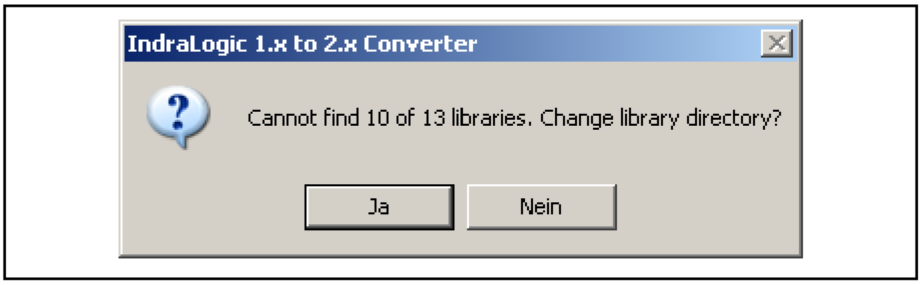


IndraLogic 1.x to 2.x Converter

Error while loading the PLC configuration!
Description file for module 'RX010107.gsd' not found.
Change configuration directory?

Fig.3-32: Error message that might appear while the PLC configuration is loading

The download includes a search for libraries required for the project. If the required libraries are not found, the following or a similar dialog appears.



IndraLogic 1.x to 2.x Converter

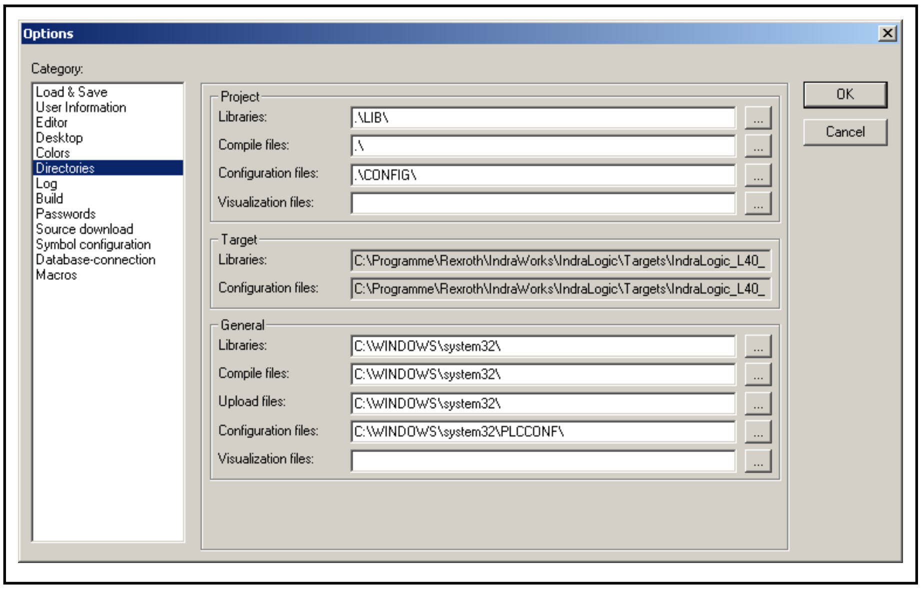
Cannot find 10 of 13 libraries. Change library directory?

Fig.3-33: Dialog for libraries that are not found

To enter the library path for the respective library, click "Yes". The "Options" dialog opens automatically in which the paths for the requested libraries can be entered.

Library path

Enter the correct path in the "Options" dialog.



Options

Category:

Project
Libraries: .\LIB\
Compile files: .\
Configuration files: .\CONFIG\
Visualization files:

Target
Libraries: C:\Programme\Rexroth\IndraWorks\IndraLogic\Targets\IndraLogic_L40_
Configuration files: C:\Programme\Rexroth\IndraWorks\IndraLogic\Targets\IndraLogic_L40_

General
Libraries: C:\WINDOWS\system32\
Compile files: C:\WINDOWS\system32\
Upload files: C:\WINDOWS\system32\
Configuration files: C:\WINDOWS\system32\PLCCONF\
Visualization files:

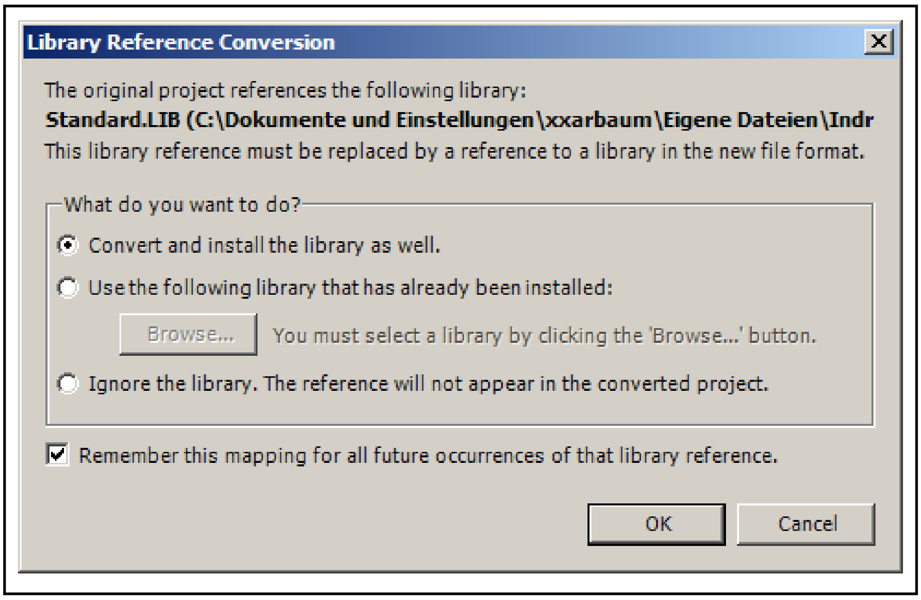
Fig.3-34: Specifying library paths

- In the "Project" and "General" areas, in the "Libraries" selection field, select the directories that IndraLogic should search and use. The entries in the "Project" area are saved with the project and those in the "General " area apply to all projects.
- The "Target" area includes a listing of the directories for libraries and configuration files that are set in the target system, e.g. by input into the target file. These fields cannot be edited.

Use "OK" to confirm your entries.

Converting library references

If there are libraries included in the project that do not yet have conversion assignment saved in the "Library options", page 222,, the "Library Reference Conversion" dialog appears for defining how these references are to be converted:


Library Reference Conversion

The original project references the following library:
Standard.LIB (C:\Dokumente und Einstellungen\xxarbaum\Eigene Dateien\Indr
This library reference must be replaced by a reference to a library in the new file format.

What do you want to do?

● Convert and install the library as well.

○ Use the following library that has already been installed:

You must select a library by clicking the 'Browse...' button.

○ Ignore the library. The reference will not appear in the converted project.

☑ Remember this mapping for all future occurrences of that library reference.

Fig.3-35: "Library Reference Conversion" dialog

The following options are possible:

- Convert and install the library.

  If this option is selected, the integrated library is converted to the new format and remains referenced within the project. It is automatically installed in the library repository in the "Other" category and remains used. If the project information required for an installation (title, version) does not come with the library, you are requested to enter this information in the "Enter project information" dialog.
- Use the following library that has already been installed.

  Select this option if a library that has already been installed is to be used for this reference. Use the "Browse" button to open the "Select Library" dialog. Select the desired version of one of the installed libraries here.

  This corresponds with the configuration of the version handling in the "Library Properties" dialog (see "Library manager properties", page 230).

  "*" means that the latest version of the library available in the system is always used in the project. The list of available libraries is structured as described for the "Library repository dialog", page 185,.

  It can be sorted according to company and category.

- Ignore the library. The reference does not appear in the converted project.

  If this option is selected, the library reference is removed and the library is no longer included in the converted project.

- Remember this mapping for all future occurrences of that library reference.

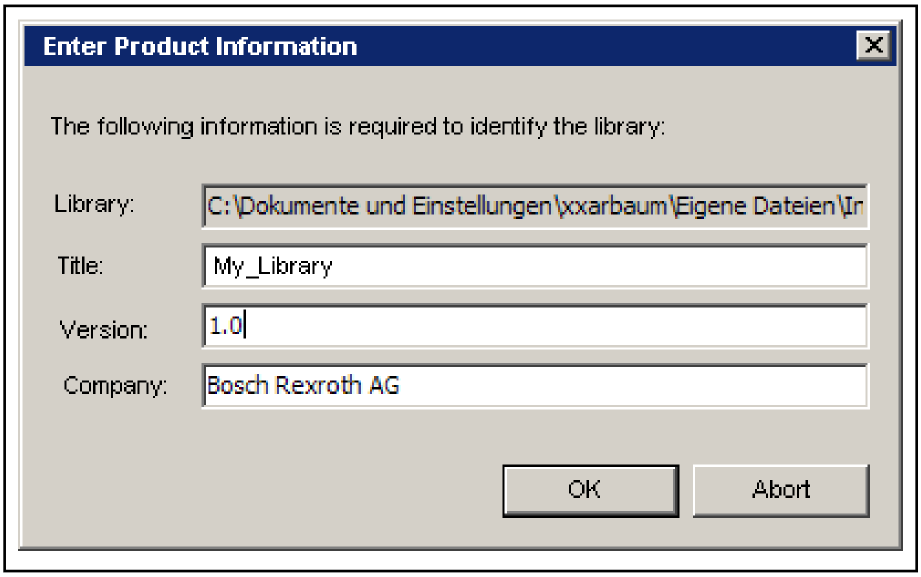
  If this option is selected, the settings made in this dialog are also used for future project conversions as soon as the respective library is referenced.

  For standard libraries, an assignment to the new libraries is already specified.

Select the desired option and confirm with "OK".

Enter project information (product information)

In the "Enter project information" dialog, the imported library is uniquely identified. Fill in the "Title", "Version" and "Company" fields here.



Enter Product Information

The following information is required to identify the library:

Library: C:\Dokumente und Einstellungen\xxarbaum\Eigene Dateien\Ir

Title: My_Library

Version: 1.0

Company: Bosch Rexroth AG

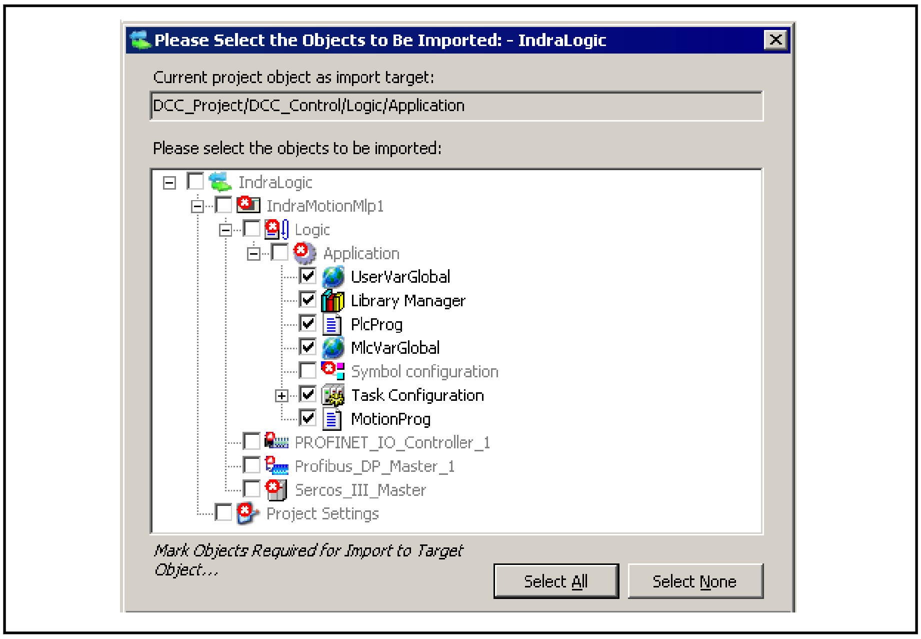
Fig.3-36: Enter project information (product information)

Title	Specify the desired name for the library.
Version	Input a unique version of the library.
Company	Input the company name.

Use "OK" to confirm your entries.

Import objects

In the "Current project object as import target:" window, please select the objects to be imported.



Please Select the Objects to Be Imported: - IndraLogic

Current project object as import target:

DCC_Project/DCC_Control/Logic/Application

Please select the objects to be imported:

- ☐ IndraLogic
  - ☐ IndraMotionMlp1
    - ☐ Logic
      - ☐ Application
        - ☑ UserVarGlobal
        - ☑ Library Manager
        - ☑ PlcProg
        - ☑ MlcVarGlobal
        - ☐ Symbol configuration
        - ☑ Task Configuration
        - ☑ MotionProg
    - ☐ PROFINET_IO_Controller_1
    - ☐ Profibus_DP_Master_1
    - ☐ Sercos_III_Master
  - ☐ Project Settings

Mark Objects Required for Import to Target Object...

Fig.3-37: Selecting the objects to be imported

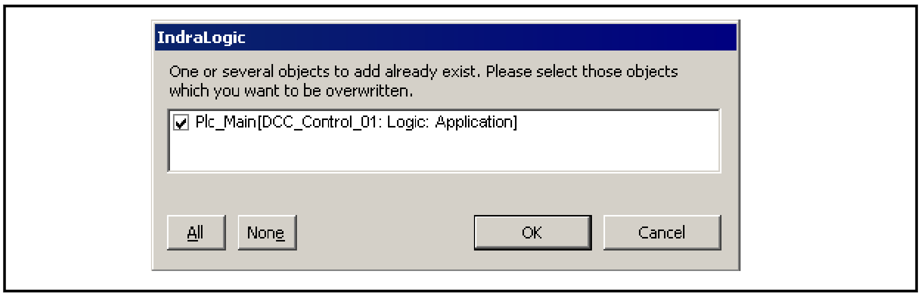
- Use the "Select All" button to highlight and import all of the objects.
- Use the "Select None" button to deselect objects that have been highlighted.

☞ For stand-alone IndraLogic 1.x projects (*.pro), the I/O configuration is not imported into the target system.

☞ If you select objects that already exist in the target project before the import, a selection dialog prompts whether to overwrite these objects; see "Overwriting objects", page 121.

Overwriting objects

Objects that already exist in the target project can be overwritten by placing a checkmark in the following selection dialog.



IndraLogic

One or several objects to add already exist. Please select those objects which you want to be overwritten.

☑ Plc_Main[DCC_Control_01: Logic: Application]

Fig.3-38: Overwriting objects

- Use the "All" button to highlight all of the objects available in the dialog. These objects are overwritten in the target project.
- Use the "None" button if you do not wish to overwrite any of the objects in this selection dialog.

Click on "OK" to complete the object import. The imported objects are added to the project. Finally, a message appears that indicates that the objects have been imported.

Importing IndraLogic 2G Projects

Importing projects

To import an IndraLogic 2G project, highlight the "Application" node in the Project Explorer and select "Data Import..." from the context menu.

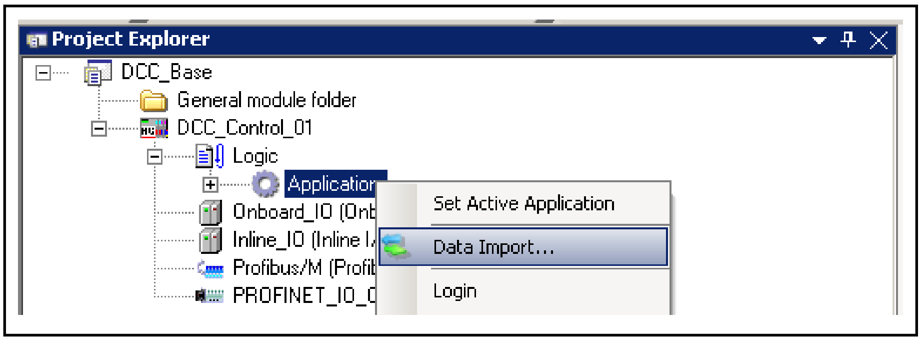
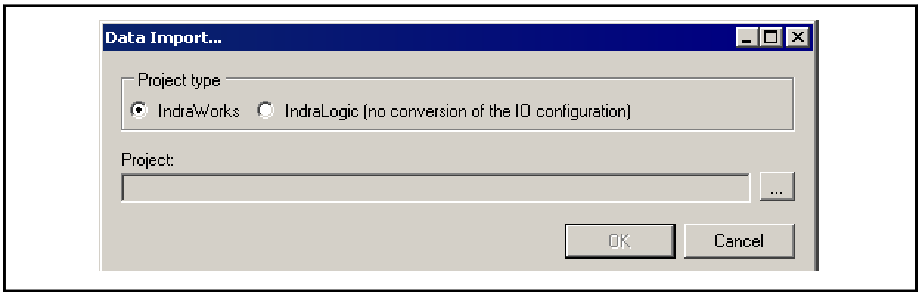


Fig.3-39: Importing IndraLogic projects

The "Data Import..." dialog appears.


Data Import...

Project type

● IndraWorks ○ IndraLogic (no conversion of the IO configuration)

Project:

Fig.3-40: Selection dialog: Project import...

Under "Project type" select the type of your source project.

- IndraWorks:

  In this case, an IndraLogic project is embedded in an IndraWorks project (*.project).

- IndraLogic (IO configuration is not converted)

  In this case, a stand-alone IndraLogic project file (*.pro or .project) is the source project.

Use the ... button to open the "Open Project" dialog and move to the project to be imported.

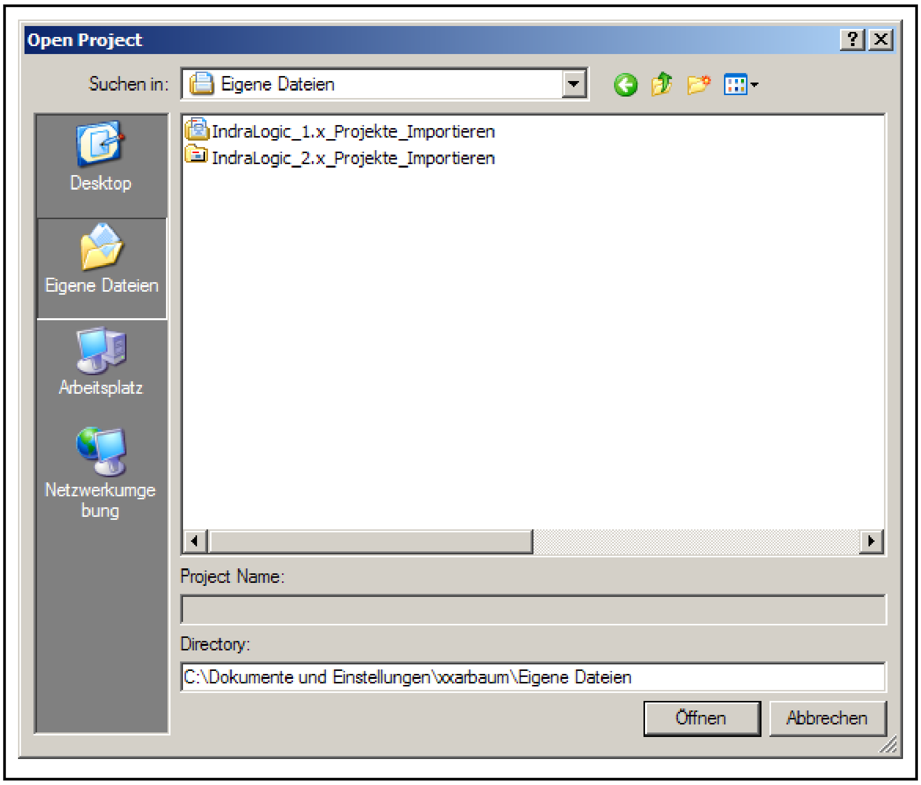
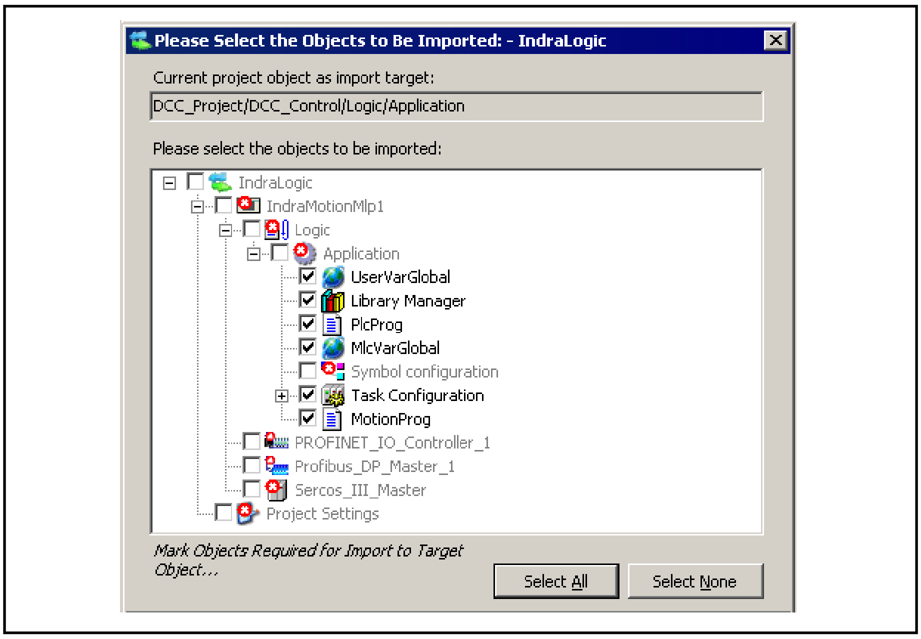

Open Project
Suchen in:
Eigene Dateien
Desktop
Eigene Dateien
Arbeitsplatz
Netzwerkumgebung
IndraLogic_1.x_Projekte_Importieren
IndraLogic_2.x_Projekte_Importieren
Project Name:
Directory:
C:\Dokumente und Einstellungen\xxarbaum\Eigene Dateien
Öffnen
Abbrechen

Fig.3-41: Project selection

Select objects

In the "Current project object as import target:" window, please select the objects to be imported.



Current project object as import target:
DCC_Project/DCC_Control/Logic/Application

Please select the objects to be imported:

- □ IndraLogic
  - □ IndraMotionMlp1
    - □ Logic
      - □ Application
        - ☑ UserVarGlobal
        - ☑ Library Manager
        - ☑ PlcProg
        - ☑ MlcVarGlobal
        - □ Symbol configuration
        - ☑ Task Configuration
        - ☑ MotionProg
    - □ PROFINET_IO_Controller_1
    - □ Profibus_DP_Master_1
    - □ Sercos_III_Master
  - □ Project Settings

Mark Objects Required for Import to Target Object...

Fig.3-42: Selecting the objects to be imported

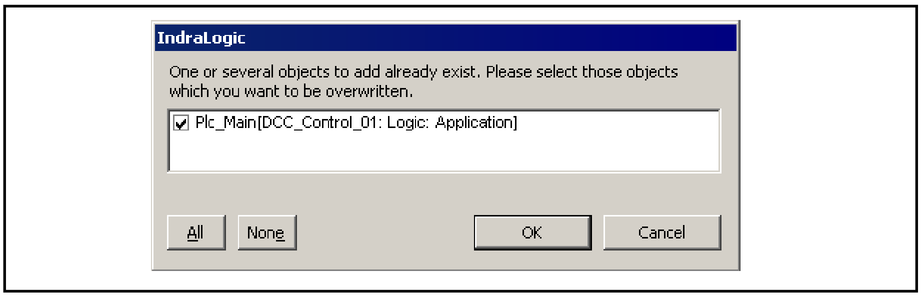
- Use the "Select All" button to highlight and import all of the objects.
- Use the "Select None" button to deselect objects that have been highlighted.

To start the import, click "OK".

☞ If you select objects that already exist in the target project before the import, a selection dialog prompts whether to overwrite these objects; see "Overwriting objects", page 124.

Overwriting objects

Objects that already exist in the target project can be overwritten by placing a checkmark in the following selection dialog.


IndraLogic

One or several objects to add already exist. Please select those objects which you want to be overwritten.

☑ Plc_Main[DCC_Control_01: Logic: Application]

Fig.3-43: Overwriting objects

- Use the "All" button to highlight all of the objects available in the dialog. These objects are overwritten in the target project.
- Use the "None" button if you do not wish to overwrite any of the objects in this selection dialog.

To complete the import, click "OK". The imported objects are added to the project, which is confirmed by a dialog.

3.4 Create

3.4.1 Create, Overview

The "Create" menu item provides functionalities to build a program, i.e. for compiling an application program.

The "Create" commands are used for code generation and for syntactic checks in all objects or in modified objects in the active application. Offline code generation is possible in order to be able to check for compilation errors before the code is loaded to the device.

The results are output in the message box, message category: Compilation

The "Clean" commands are used to delete the compilation information from the target system that was saved during the most recent download and during code generation. This affects Online Change, page 80, for example.

If required, the menu structure can be reconfigured via the **IndraWorks** ▸ **Tools** ▸ **Customize** ▸ **Commands** dialog.

Commands:

- Compile, page 124,
- Recompile, page 125,
- Generate code, page 125
- Clean, page 125
- Clean all, page 125

3.4.2 Compile

Icon:

Default shortcut: <Shift>+<F11>

Menu: **Create** ▸ **Compilation**

This command starts the compilation for the currently active application. This means that all objects for this application are subject to a syntactic check.

> ☞ Note that **no** code is generated as it is the case when logging into the target system or downloading the application!

The compilation is carried out automatically before every login with changed program.

After the syntactic check is complete, error messages or warnings might be displayed within the message window in the "Compile" category.

If the number of errors/warnings exceeds 500, all other messages are skipped and a special message provides information about the situation.

Commands of the Message window, page 105, category are available in order to navigate between the messages and between messages and source code.

If the program has not been modified since the most recent error-free compilation, it is not recompiled and the message "The application is current" is output in the message window.

However, if the syntactic check should nevertheless be repeated, use the Recompile, page 125 command.

3.4.3 Recompile

Menu: **Create ▸ Recompile**.

The command starts the compilation for the currently active application again, even if it was recently compiled without errors.

3.4.4 Generate Code

Menu: **Create ▸ Generate Code**

Generate code for test purposes with this command.

As with the login, compilation code is generated with the application in the control, but the code is not loaded into the control. In this way, the code can still be checked for compilation errors and corrected before it is used in online operation.

3.4.5 Clean

Menu: **Create ▸ Clean**

The command deletes the compilation information for the currently active application. This information was created and saved during the most recent download, page 132, of the application to the target system.

After the cleaning is complete, an online change, page 80, is no longer possible for the respective application.

First, the program has to be completely reloaded to the control.

3.4.6 Clean All

Menu: **Create ▸ Clean All**

The command deletes the compilation information for all applications,

After cleaning is complete, an online change, page 80, is no longer possible for the respective applications.

First, the program has to be completely reloaded to the control.

See also Clean, page 125.

3.5 Debug

3.5.1 Debugging, Overview

These commands influence the application program on the control in online operation.

By default they are available in the main menu "Debug".

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize ▸ Commands** dialog.

For security reasons, when calling the commands that are marked with an asterisk * below, users are always requested to confirm the selection.

An additional request for confirmation for the commands marked with two asterisks ** appears if the option Secure online operation, page 239, is enabled in the communication settings for the respective device.

⚠ CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

Evaluate possible risks before writing or forcing variable values and take corresponding precautions.

⚠ CAUTION

The online change modifies the running application program and causes a restart.

Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

Ensure that the new application code still results in the desired behavior of the controlled system.

The commands:

- Login, page 127
- Logout, page 128
- Generate boot application, page 129
- Logout current online user, page 129,,
- Edit object (offline), page 130,
- Load, page 132, *
- Online Change, page 133, *
- Start, page 133, **,
- Stop, page 133, **
- Single cycle, page 134, **
- Multiple download..., page 134**
- Reset warm, page 135*,
- Reset cold, page 135*,
- Reset origin, page 136*,

- Breakpoints, page 136,
- New breakpoint..., page 138
- Toggle breakpoint, page 140
- Breakpoint positions, page 138
- Call stack, page 141
- Monitoring, page 141
- Procedure step, page 142
- Single step, page 142
- Execute to return, page 143
- Execute to cursor, page 143
- Determine next instruction, page 143
- Display next instruction, page 143
- Write values for all online applications, page 144, **
- Force values on all online applications, prepare value, page 145, **,
- Cancel forcing for all value on all online applications, page 147**
- Add all forces to monitoring window, page 148,
- Sequence control, page 148,
- Display mode, page 151, **

3.5.2 Login

Icon:

Default shortcut: <Alt> + <F8>

This command connects the programming system, i.e. the "active application", page 66, with the target system (control or simulated device), creating the **online mode**. To achieve this, the communication settings, page67, for the device have to be configured correctly.

If the "Login" command is called from the online mode, the currently active application is concerned.

If the command is called from the context menu if an application is selected in the Project Explorer, login is carried out with this selected application even if it is not set as "active application".

The following situations are possible in case of login with the currently active application (without errors, communication settings correct):

- **The application is not available on the control yet:** Confirmation of the download (Load, page 132) is requested. A dialog with the following text appears: "<Application name> application does not exist on the control. Do you want to create and load the application?" The "Details" button in this dialog leads to information on applications already available on the control.
- **The application is already available on the control** and has not been changed since the last loading. Login is carried out without further interaction with the user.
- **The application is already available on the control,** has, however, been **changed** since the last loading. You are asked whether you want to complete an Online change, page 133, or load the entire application or login without changing the currently running application.

  A dialog with the following text appears:

"The code has changed since the last download. What do you want to do?

- Login with online change.
- Login with download.
- Login without any change."

The "Details" button in this dialog leads to information on the application changed in the programming system as compared to its previous version which is currently available on the control.

- Another version of the application is already available on the control, is however, currently **not** running. You are asked whether it is to be replaced. A dialog with the following text appears: "On the control, there is an unknown version of the< Application name> application. Do you want to replace it by the application in the project?".

  The "Details" button in this dialog leads to information on the application in the IDE (integrated development environment = programming system) as compared to the one on the control.

- **A version of the application is already available on the control and is running.** You are asked whether you want to login and overwrite the currently running application nevertheless. A dialog with the following text appears:

  "Warning: An unknown version of the <Application name> application is currently running on the control. Do you still want to load the current code and replace the existing application?"

  The "Details" button in this dialog leads to information on the application in the IDE (integrated development environment = programming system) as compared to the one on the control.

Compilation before login

Before login and if the current application project has not yet been compiled since opening or since its last change, it is compiled.

This means that it is compiled in the logged off condition as in a compilation run, page 124, and a compilation code is generated for the control, as well.

If errors occur during compilation, a message box appears with the following text: "Errors occurred during compilation. Would you like to log in without loading the program?" This way, choose to correct the errors first or to log in despite the errors in the most recent version of the application that might be on the control. The errors are output in the message box in the "Compile" category.

See also:

- Compile, page 124.

Information on loading

If the project is loaded to the control completely at login or partially at online change, information appears in the message window on the generated code size, on the size of the global data, on the resulting memory requirement on the control and in case of online change also on a list of the respective function blocks.

3.5.3 Logout

Icon:

Default shortcut: <Ctrl>+<F8>

This command ends the connection between the development system and the target system (control or simulated device) causing a return to **offline mode**.

3.5.4 Generating a Boot Application

This command is available in online mode for generating a boot application. A boot application (boot project) is started automatically when the control is started. It is named <ApplicationName>.app

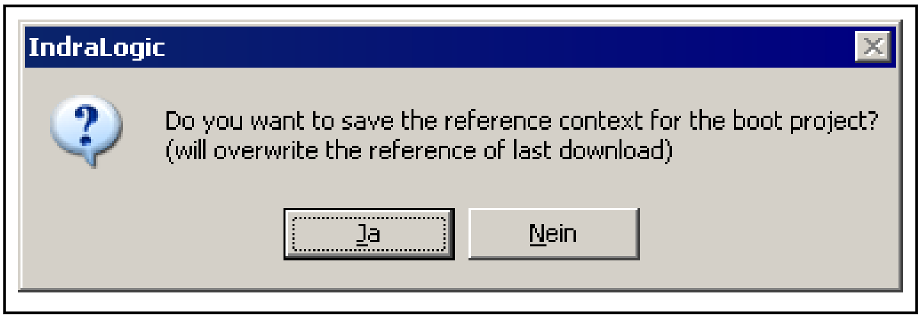
Use in online mode:

The boot application is saved on the target device with the name <ApplicationName>.app.

Use in offline mode:

The standard dialog for saving a file opens.

Select a directory for saving the current application as a boot application in file format to load it to a target system at a later time. The file is a "boot application" type file and the extension ".app" is automatically added to its name. After confirming with "Save", another dialog appears, in which you are asked if a "compilation protocol", page 80 that might be present in the project directory should be overwritten:


IndraLogic

Do you want to save the reference context for the boot project?
(will overwrite the reference of last download)

Fig.3-44: Dialog for saving the compilation information for the boot application

Select "Yes" if you plan to transfer the new boot project to the control with an external tool and you still want to log into the application without being forced to download it again.

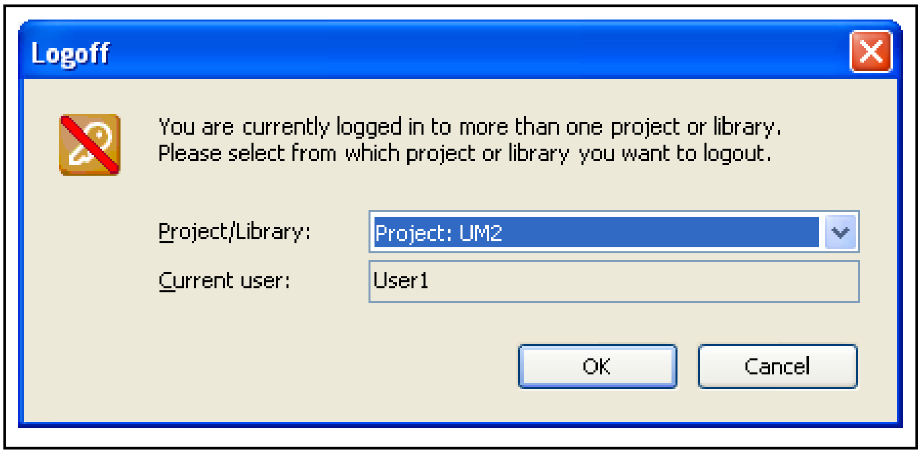
A compilation protocol from a previous download would not match the newly created project, since it contains code and references from the previous application.

3.5.5 Logging out the Current Online User

Icon:

This command (User management) logs of the user currently logged in to the project or the library. A corresponding message is output only if currently no user is logged in. Otherwise, no dialog and no message appears.

If the user is currently logged in to several projects or integrated libraries (not necessarily using the same user account), the following logout dialog is opened.


Logoff

You are currently logged in to more than one project or library.
Please select from which project or library you want to logout.

Project/Library: Project: UM2

Current user: User1

Fig.3-45: Logout dialog

Via the selection list in the **Project/Library** field, you can now select the project or the library from which the **Current user** is to be logged off.

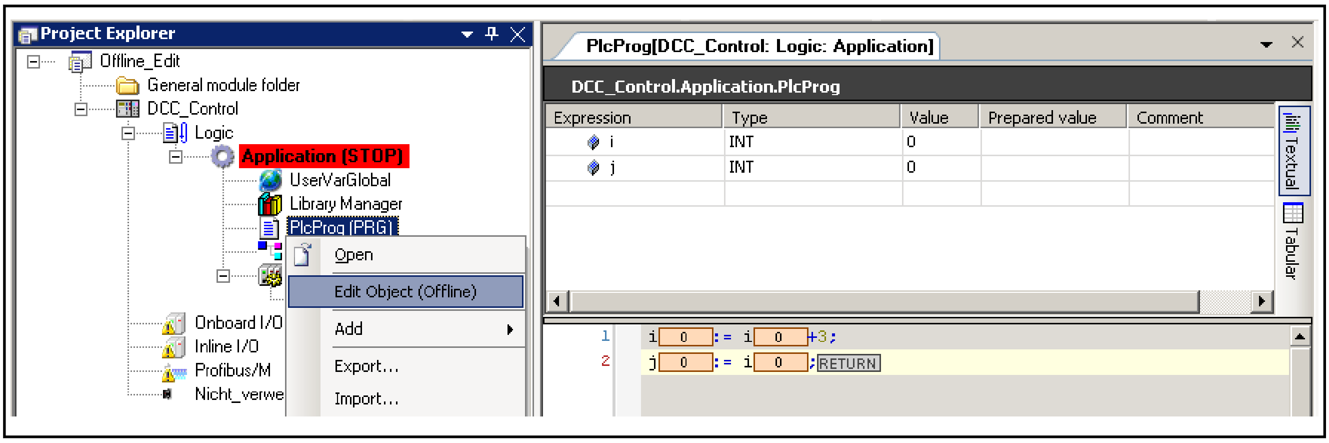
The toolbar always shows the user currently logged into the project.

3.5.6 Editing Objects Offline in the Online Mode (Hijacking)

This command is only available in online mode with running or stopped application.

If the user is logged in with an application, all opened PLC objects belonging to this application are displayed in online mode. If another object is opened, it is also displayed in the online mode.

The user can now in the context menu select the **Edit object (offline)** menu item in order to also open the selected object in the editable mode.



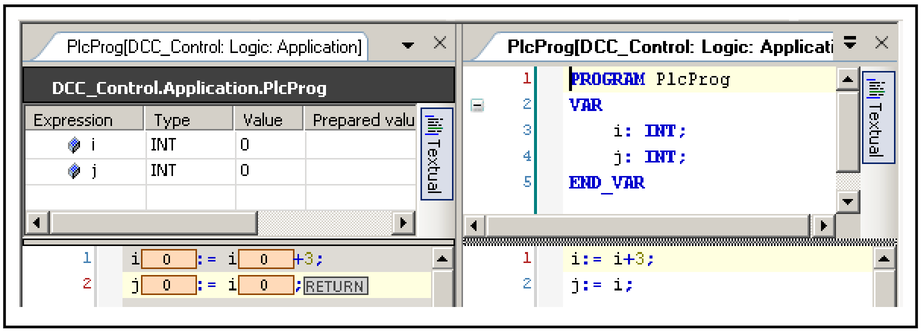
Expression	Type	Value	Prepared value	Comment
i	INT	0		
j	INT	0		

```
i 0 := i 0 +3;
j 0 := i 0 ;RETURN
```

Fig.3-46: Opening the PlcProg object for offline editing

Alternatively, this command is also available in the **Debug ▸ Edit object (offline)** menu. It affects the POU selected in the Project Explorer.

As a result, the user has opened the object online and in the editable mode.



PlcProg[DCC_Control: Logic: Application]

DCC_Control.Application.PlcProg

Expression	Type	Value	Prepared valu
i	INT	0	
j	INT	0	

```
i[ 0 ]:= i[ 0 ]+3;
j[ 0 ]:= i[ 0 ];RETURN
```

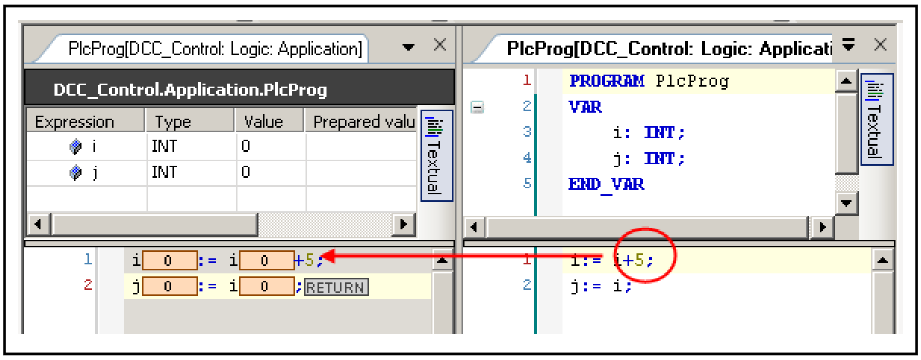
PlcProg[DCC_Control: Logic: Applicati

```
PROGRAM PlcProg
VAR
    i: INT;
    j: INT;
END_VAR
```

```
i:= i+3;
j:= i;
```

Fig.3-47: POU PlcProg online (left) and offline in the edit mode (right)

Any change in the right window is immediately displayed in the left window as well **without** becoming effective in the program code or in the control.



PlcProg[DCC_Control: Logic: Application]

DCC_Control.Application.PlcProg

Expression	Type	Value	Prepared valu
i	INT	0	
j	INT	0	

```
i[ 0 ]:= i[ 0 ]+5;
j[ 0 ]:= i[ 0 ];RETURN
```

PlcProg[DCC_Control: Logic: Applicati

```
PROGRAM PlcProg
VAR
    i: INT;
    j: INT;
END_VAR
```

```
i:= i+5;
j:= i;
```

Fig.3-48: Offline change completed

The changes made in the offline editor can be transferred to the control directly via the

- **Debug ▸ Online change** or
- **Debug ▸ Loading**

menu commands, i.e. without log off and log in.

In the example, only the step width changes at the running application after the **online change**.

After **loading**, the variables are re-initialized and after application (re)start, counting is started at zero with the new step width.

☞ *Note the following points before performing an online change:*

- Is the modified code free of errors?
- Application-specific initializations (reference motion, etc.) are not executed, since the machine retains its status.
- Can the new program code really work without re-initialization?
- Pointer variables retain their value from the last cycle. If it is pointed to a variable that changed in size, the value is no longer correct. For this reason, ensure that pointer variables are re-assigned in every cycle.
- If the active step in an SFC chart is removed, the chart remains inactive.

At saving, the changes made are also retained at the interface in the project without "Online change" or "Loading".

If the project is under **version control** and the object opened "offline" is checked in, it is brought into the "hijacked" state analogously to other objects in the logged in state.

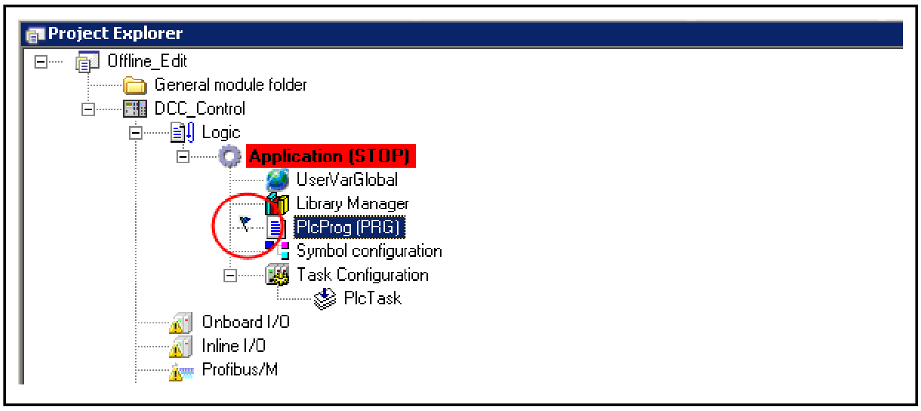


Fig.3-49: Display of the changed object in the Project Explorer

3.5.7 Load

This command is available in online mode. It triggers the compilation, page 124, and compilation of the active application, page 66,. This means that in addition to a syntactic check, application code is generated and loaded to the control.

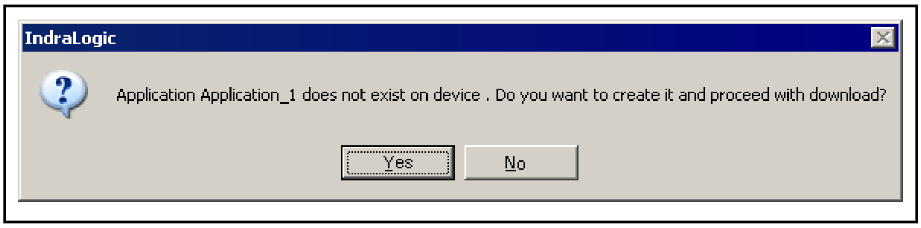
☞ All of the variables, with the exception of persistent variables, are re-initialized.

The following situations are possible:

If there is no application on the control yet, the user is asked if the application is to be loaded to the control.

After the user confirms with "Yes", the "download" is performed.

A matching dialog appears during Login, page 127.

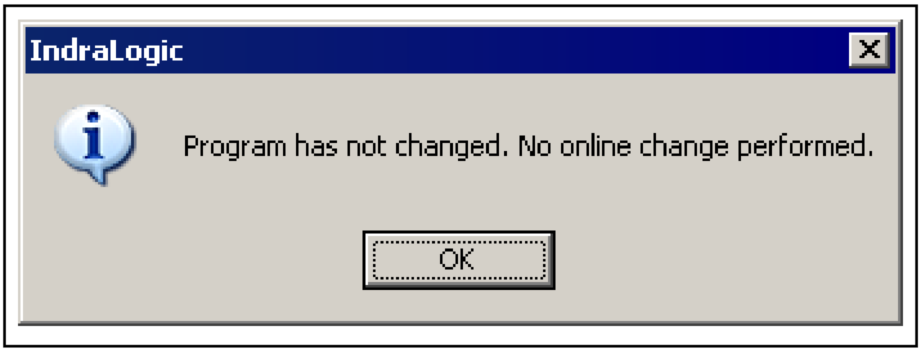

IndraLogic

Application Application_1 does not exist on device . Do you want to create it and proceed with download?

Fig.3-50: Message box, load application?

If other applications are on the control, the user is first asked if they are to be deleted. Then the application is loaded. See the related description for the same situation with the "Login" command.

If a different version of the same application is already on the control, the active application is loaded, overwriting the previous version. If the versions do not differ from each other, a dialog appears that indicates this situation and the application is not reloaded.


IndraLogic

Program has not changed. No online change performed.

Fig.3-51: Message dialog, no download

3.5.8 Online Change

⚠ CAUTION

The online change modifies the running application program and causes a restart.

Depending on the system controlled, damages at the system and workpieces can result or the health and life of people can be put at risk.

Ensure that the new application code still results in the desired behavior of the controlled system.

This command causes an explicit "online change" to be carried out for the "active application", page 66,.

Online change means that only the modified parts of an application already running on the control are loaded to the control. This cannot be done after "Clean all" or "Clean"!

The cleaning deletes the compilation information that is automatically saved during each compilation (build, code generation) and is the basis for an online change.

An online change is automatically provided if login occurs with an application that is already running on the control but has been modified since the most recent download in the programming system.

See further information on "code generation and online change", page 80.

3.5.9 Start

Icon: ▶

This command starts the application on the control.

If the command is called from the online mode, the currently **active application** is concerned.

If the command is called from the context menu if an application is selected in the Project Explorer, login is carried out with this **selected application** even if it is not set as "active application".

3.5.10 Stop

Icon: ■

This command stops the application on the control.

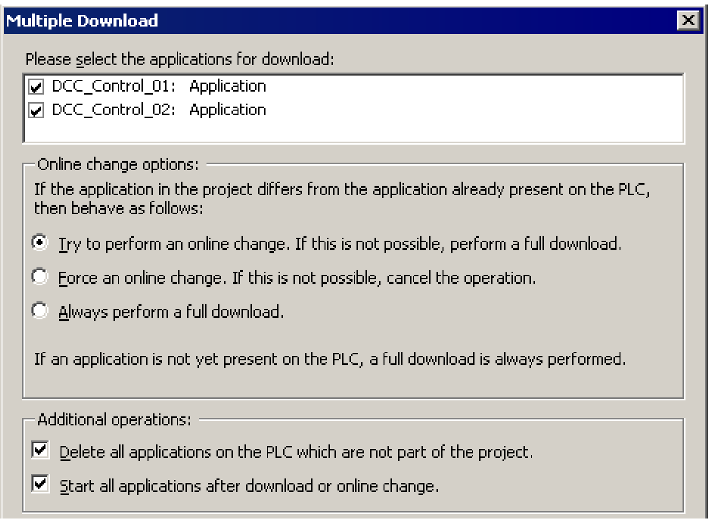
3.5.11 Single Cycle

This command causes the active application to be executed for a single cycle.

3.5.12 Multiple Download...

This command is available in online or offline mode. It triggers the compilation, page 124, of and code generation, page 80, for all applications included in the project intended by the user. This way, in addition to a syntactical check of these applications, the related code is also generated and loaded to the respective control. For each application selected for download, a corresponding compilation protocol, page 80 named <projectname>.<devicename>.<application ID>.compileinfo is generated in the project directory.

After the "Multiple download..." command is enabled, a dialog window opens that displays a list of all of the applications included in the project:



Multiple Download

Please select the applications for download:

☑ DCC_Control_01: Application
☑ DCC_Control_02: Application

Online change options:

If the application in the project differs from the application already present on the PLC, then behave as follows:

● Try to perform an online change. If this is not possible, perform a full download.

○ Force an online change. If this is not possible, cancel the operation.

○ Always perform a full download.

If an application is not yet present on the PLC, a full download is always performed.

Additional operations:

☑ Delete all applications on the PLC which are not part of the project.

☑ Start all applications after download or online change.

Fig.3-52: "Multiple Download" dialog

An application can be selected for download by placing a check in the respective checkbox. Several applications may be selected, even if they are to be loaded to different controls. By default, either all of the checkboxes are selected or only those for the applications selected by the user for the most recent download.

If a previous version of the selected application already exists on the control and it differs from the current version, the following options appear for selection:

- Try to perform an online change. If this is not possible, perform a download. This option is selected by default. If it is selected, the modified parts of the selected applications on the control are changed and only the newly created parts of the respective applications are loaded to the control.
- Force an online change. If this is not possible, cancel the operation. If an online change for (at least) one of the selected applications cannot be completed (e. g. if the command 'Clean all' or 'Clean application' was previously executed), no download is performed.
- Always perform a download. All of the parts of the selected applications are loaded to the control without taking existing versions into account.

Selected applications that do not yet exist on the control are automatically downloaded to the related control.

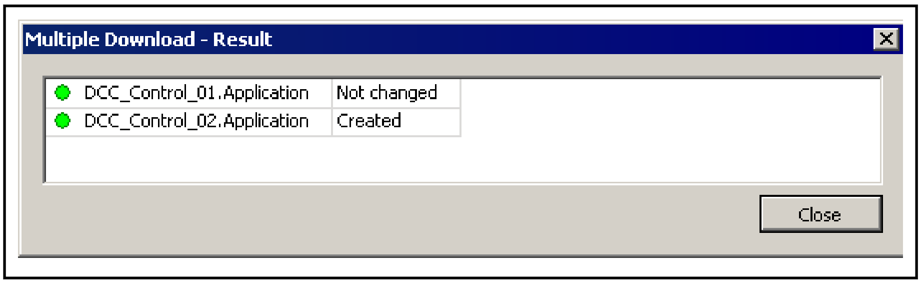
In addition, by selecting the corresponding checkbox, users can specify if

- Old applications that are no longer part of the project are to be deleted from the control
- The selected applications are to be started after the download/online change.

Note that PERSISTENT type variables are not initialized in general. However, if the data layout was changed, the persistent variables are automatically re-initialized.

After having confirmed the settings in the dialog with "OK", a syntactical check of all of the selected applications is carried out. Then, for each application, the communication with the related control is verified before the download is carried out.

After the download is complete, a list of the selected applications appears, which contains detailed information about the operations performed for each application.


Multiple Download - Result

🟢 DCC_Control_01.Application	Not changed
🟢 DCC_Control_02.Application	Created

Fig.3-53: "Multiple download - Result" dialog

3.5.13 Reset Warm

This command is available in online mode. With the exception of the remanent variables (retain, persistent), page 519, it resets all variables to their initialization value.

If variables were initialized with specific values, they are reset to these exact values. All other variables are reset to the default initialization values (for example, integer variables are reset to 0).

As a safety precaution, IndraLogic requires another user confirmation before all variables are overwritten. The situation is comparable to a loss of power or switching the control off and back on while the application is running ("warm start").

A reset disables the breakpoints currently set in the project. If the "Warm" reset command is called just when the program sequence is stopped at a breakpoint, the user is asked if the current cycle should be completed before executing the reset or if the task should be stopped and the reset executed immediately. However, not all runtime systems are able to execute a reset without completing the current cycle first.

To start the application again, use Start, page 133 after the reset.

Also refer to

- Reset origin, page 136,
- Reset cold, page 135.

3.5.14 Reset Cold

This command is available in online mode. It corresponds to the "Reset warm" command, but in addition to the "normal" variables, retain variables, page 519, (!) are also reset to their initialization values. The situation is comparable to starting an application program that was just loaded to the control ("cold start").

A reset disables the breakpoints currently set in the project.

If the "Cold reset" command is called just when the program sequence is stopped at a breakpoint, the user is asked if the current cycle should be completed before executing the reset or if the task should be stopped and the reset executed immediately.

However, not all runtime systems are able to execute a reset without completing the current cycle first.

Also refer to

- Reset warm, page 135,
- Reset origin, page 136.

3.5.15 Reset Origin

This command is available in online mode. It resets the values of all inclusive remanent variables, page 519, back to their initialization values and **deletes the application program from the control**.

A reset disables the breakpoints currently set in the project.

If the "Origin" reset command is called just when the program sequence is stopped at a breakpoint, the user is asked if the current cycle should be completed before executing the reset or if the task should be stopped and the reset executed immediately. However, not all runtime systems are able to execute a reset without completing the current cycle first.

Also refer to

- Reset warm, page 135,
- Reset cold, page 135.

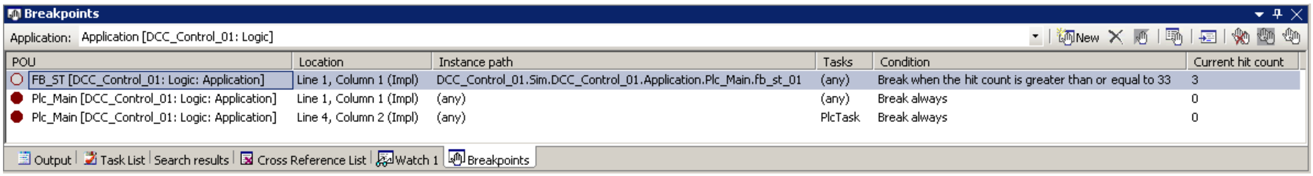
3.5.16 Breakpoints

Icon:

This command opens the **Breakpoints** dialog providing an overview of all breakpoints currently set in the project. The breakpoint parameters are displayed and can be modified. In addition, breakpoints can be added and removed or enabled and disabled.

The first breakpoint parameters shown are those that were set using the commands Toggle breakpoint, page 140, (here default values are set for the breakpoint "conditions") or New breakpoint, page 138, in the "Debug" menu (here the conditions can be defined directly).

☞ If you have decided to use Multitasking, remember that by using the breakpoint you only stop the **one** task in which the breakpoint is located.



Application: Application [DCC_Control_01: Logic]

POU	Location	Instance path	Tasks	Condition	Current hit count
○ FB_ST [DCC_Control_01: Logic: Application]	Line 1, Column 1 (Impl)	DCC_Control_01.Sim.DCC_Control_01.Application.Plc_Main.fb_st_01	(any)	Break when the hit count is greater than or equal to 33	3
● Plc_Main [DCC_Control_01: Logic: Application]	Line 1, Column 1 (Impl)	(any)	(any)	Break always	0
● Plc_Main [DCC_Control_01: Logic: Application]	Line 4, Column 2 (Impl)	(any)	PlcTask	Break always	0

Fig.3-54: "Breakpoints" dialog

	Description	Example
Application	Name of the active application	Application [DCC_Control_01: Logic]
POU	Name of the function block that contains the breakpoint	Plc_Main
Position	Breakpoint position within the POU: line and column number (text editor) or network or element numbers (graphical editor); In the case of function blocks, "(Impl)" indicates that the breakpoint is located in the implementation of the function block, not in an instance.	Line 4, column 2 (Impl)
Instance path	Complete object path of the breakpoint position	DCC_Control_01.Sim.DCC_Control_01.Application.Plc_Main.fb_st_01
Tasks	Names of the tasks in the execution of which the breakpoint becomes effective. If there is a limitation (default), "(all)" is listed here.	PlcTask
Condition	Indicates when the breakpoint becomes effective (dependency on number of hits); for possible values see New breakpoint, page 138, (Default: 1).	Break if the number of hits is greater than or equal to 33
Current Number of hits	Indicates how often the breakpoint has been ("hit") during the execution up to this point.	3

Fig.3-55: "Breakpoints" dialog, description

The following functions are available by using the **buttons** in the upper right part of the dialog. They are used to edit the breakpoint parameters of the breakpoints selected in the list or to add new breakpoints or delete them:

New	New breakpoint	Opens the New breakpoint dialog See also the description of the corresponding New breakpoint, page 138, command.
✕	Delete breakpoint	Removes the breakpoint; Attention: do not confuse this function with "Disable"!
	Enable/disable breakpoint	Switches back and forth between the "enabled" ● and "disabled" ○ status. In case of disabling, the breakpoint is not removed from the list and can be enabled again.
	Properties	Opens the "Breakpoint properties" dialog where the breakpoint parameters can be edited. The dialog is comparable to the "New breakpoint" dialog; see New breakpoint, page 138, for a description.
	Go to source code position	Opens the "Select online status" dialog; from where you can move to the position of the breakpoint in the source code.
	Delete all breakpoints	Deletes all breakpoints in the application The list is emptied. Attention: do not confuse this function with Disable!
	Enable all breakpoints	Enables (●) all breakpoints that are currently disabled
	Disable all breakpoints	Disables (○) all breakpoints that are currently enabled. The breakpoints remain in the list and can be disabled again.

Fig.3-56: Symbols

3.5.17 Breakpoint Positions

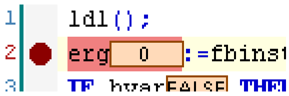
Possible "breakpoint" positions, page 82, depend on the type of editor.

In principle, these include any position in a POU where the values of variables can change or where the program flow branches or another function block is called.

> ☞ A breakpoint is always automatically set in all methods that can be called
>
> that means that if a method managed by an interface is called, the breakpoints are also set in all of the methods of the function blocks that implement the interface and likewise in all of the function blocks derived that use the method.
>
> If a method is called using a pointer to a function block, breakpoints are set in the method of the function block and in all of the function blocks derived that require the method.

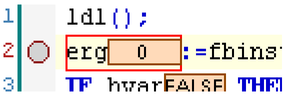
Breakpoint symbols:

- Breakpoint in online mode:



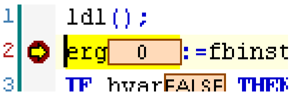
```
ldl();
erg 0 :=fbinst
IF bvarFALSE THEN
```

- Disabled breakpoint:



```
ldl();
erg 0 :=fbins
IF bvarFALSE THEN
```

- Program stop at breakpoint:



```
ldl();
erg 0 :=fbinst
IF bvarFALSE THEN
```

Also refer to

- Breakpoint positions in the IL/LD/FBD editor, page 365
- Breakpoint positions in the CFC editor, page 317
- Breakpoint positions in the ST editor, page388

3.5.18 New Breakpoint

Icon:

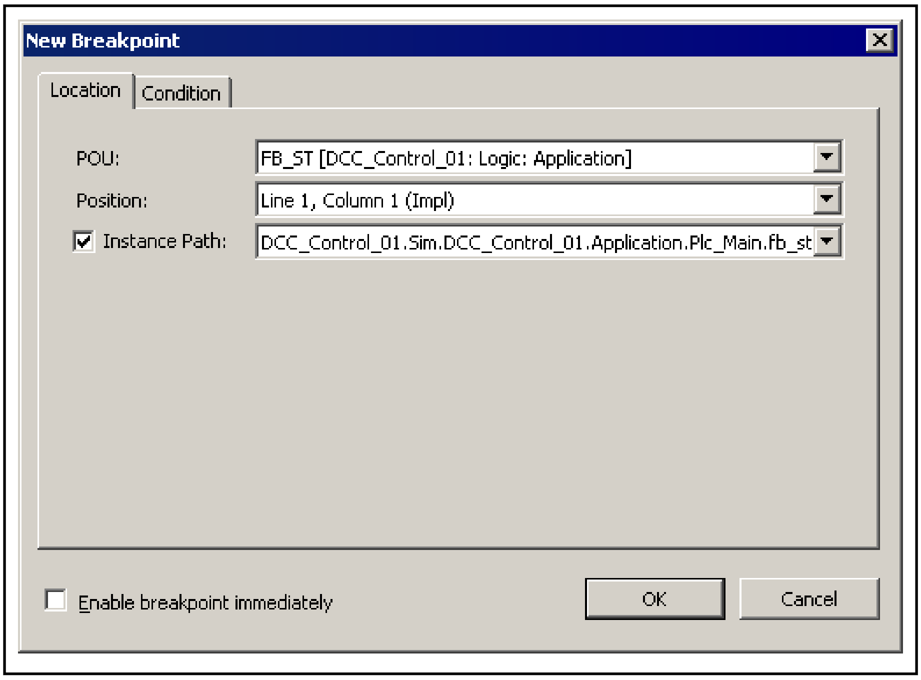
Menu: **Debug** ▸ **New breakpoint...**

This command is used to define a new breakpoint, page 82, in the project.

It is not relevant where the cursor is currently positioned, the 'New breakpoint' dialog opens where in the Position sub-dialog, one of the possible breakpoint positions in the entire project can be selected and in the "Condition" subdialog, certain conditions can be defined for the breakpoint becoming effective.

> ☞ To set a breakpoint at the current position, also use the Toggle breakpoint, page 140, command or the corresponding function in the Breakpoint dialog, page 136,.

Position



POU: FB_ST [DCC_Control_01: Logic: Application]

Position: Line 1, Column 1 (Impl)

☑ Instance Path: DCC_Control_01.Sim.DCC_Control_01.Application.Plc_Main.fb_st

☐ Enable breakpoint immediately

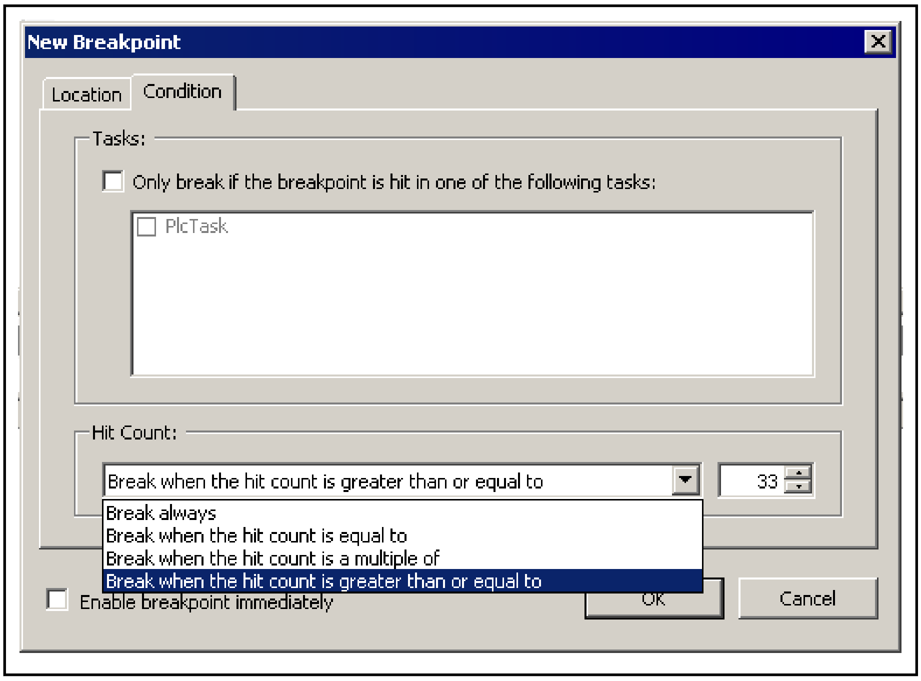
Fig.3-57: Dialog - "New breakpoint", position

- POU: The selection list provides all of the POU objects in the project. Select the POU object in which the breakpoint is to be set.
- Position: The selection list provides all of the possible breakpoint positions at the selected POU in the "POU" selection list; see Breakpoint positions, page 138. Depending on the type of editor, these positions are listed as line and column numbers (text editor) or as network or element numbers.

  For function blocks, "(Impl)" is also displayed.

  The breakpoint can be set in the **implementation** or in an **instance**. If it is to be set in the implementation version, leave the "Instance path" option disabled. If it is to be set in an instance, enable the "Instance path" option to select the instance.
- Instance path: If the POU selected above is a function block and this option is disabled, the new breakpoint is set in the implementation version of the function block. If the breakpoint is to be set in an instance, enable this option and select the desired instance.

Condition


New Breakpoint

Location | Condition

Tasks:

□ Only break if the breakpoint is hit in one of the following tasks:

□ PlcTask

Hit Count:

Break when the hit count is greater than or equal to ▼ 33

Break always
Break when the hit count is equal to
Break when the hit count is a multiple of
Break when the hit count is greater than or equal to

□ Enable breakpoint immediately

Fig.3-58: Dialog - "New Breakpoint", condition

- **Tasks:** If you enable the option "Only break when the breakpoint is hit in one of the following tasks" (by default: disabled), the breakpoint only becomes effective if the POU in which it is set is executed by one of the tasks selected here. All of the tasks defined in the project are listed and the desired tasks can be selected by placing a checkmark in the checkbox.
- *Number of hits:*
  - **Always break:** The program always stops at this breakpoint.

  Alternatively, a number can be specified that indicates how often the breakpoint has to be hit before the program stops in accordance with the following conditions at execution.

  - **Break when the hit count is equal to**
  - **Break when the hit count is a multiple of**
  - **Break when the hit count is greater than or equal to**

  the "number".

3.5.19 Toggle Breakpoint

Default shortcut: <Alt>+<F9>

Menu: **Debug** ▸ **Toggle breakpoint**

In principle, this command is used to switch back and forth between active and inactive status for a breakpoint, page 82.

However, it also can set a new breakpoint if none has already been set at the current breakpoint position, page 138,.

See New breakpoint, page 138.

If an active breakpoint is already present, it is disabled.

If an inactive breakpoint is already present, it is enabled.

☞ Every active breakpoint becomes inactive when a user exits the online mode and then logs in again.

An active breakpoint is identified by a ● symbol; an inactive breakpoint is identified by a ○ symbol.

3.5.20 Call Stack

Icon:

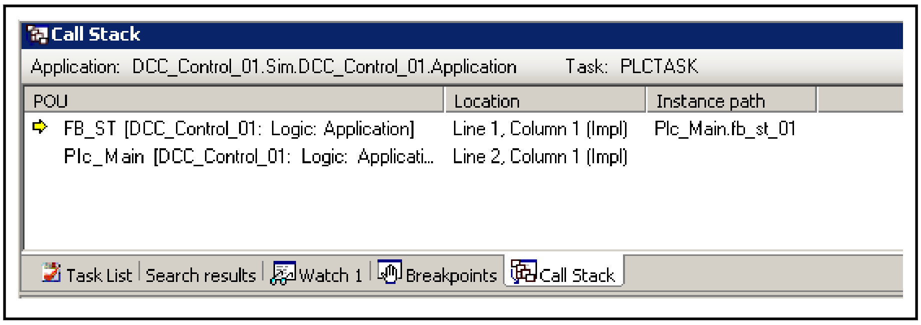
This command opens the window with the **call stack**. When a program is executed step by step, the current position is always displayed here with its complete call path.

Below the title line, the window always displays the name of the active **application** and the name of the **task** which is controlled by the POU that was just reached.

The call stack consists of a list of positions, each described by the **POU** name, the **position** and - in the case of instances - the **instance path**. Depending on the editor, the position is indicated by line and column number (text editor) or as network or element numbers (graphical editor).

The first line in the list, indicated by a yellow arrow, describes the current execution position. If this position is located in a function block that is called by another function block, the position of the call is described in the second line. If the caller is then called by another function block, this call position is described in the third line, etc.

The call stack is also available in offline mode or in normal online operation (without having to use debugging functions). In this case it contains the most recent position indicated during a step by step execution, though it is displayed in a "grayed out" font.



Application: DCC_Control_01.Sim.DCC_Control_01.Application Task: PLCTASK

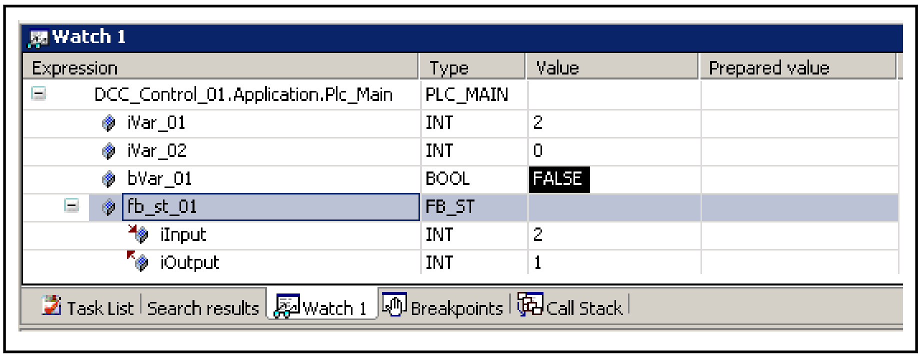
POU	Location	Instance path
⇨ FB_ST [DCC_Control_01: Logic: Application]	Line 1, Column 1 (Impl)	Plc_Main.fb_st_01
Plc_Main [DCC_Control_01: Logic: Applicati...	Line 2, Column 1 (Impl)	

Fig.3-59: Call stack, current position of FB_ST, called by Plc_Main

3.5.21 Monitoring

This command opens a submenu with the five commands described in the following.

Watch 1, **Watch 2**, **Watch 3** and **Watch 4**.

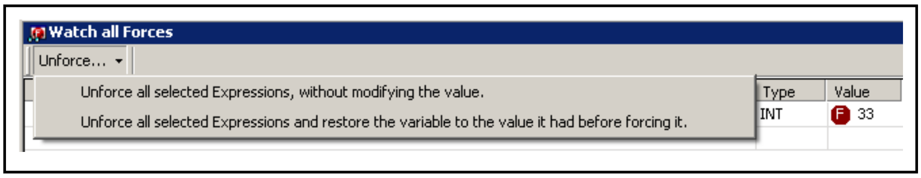


Expression	Type	Value	Prepared value
⊟ DCC_Control_01.Application.Plc_Main	PLC_MAIN		
iVar_01	INT	2	
iVar_02	INT	0	
bVar_01	BOOL	FALSE	
⊟ fb_st_01	FB_ST		
iInput	INT	2	
iOutput	INT	1	

Fig.3-60: Example of a watch list in online mode

In this way, the respective monitoring list can be opened in a window and edited, page 499,.

In addition, **Watch all forces** can be used to get an overview of all of the permanently affected variables.



Unforce all selected Expressions, without modifying the value.

Unforce all selected Expressions and restore the variable to the value it had before forcing it.

Fig.3-61: Example, "Watch all forces"

3.5.22 Procedure Step

Icon:

Default shortcut: <F12>

Menu: **Debug** ▸ **Procedure Step**.

This command can be used to execute a program in defined steps in online mode (Stepping, page 82), e.g. to make troubleshooting easier.

A single step is executed. For single step instructions, this is comparable to "stepping" with "single step"; see Single step, page 142.

If a function block call is reached, "Procedure step" causes the function block called to be completely processed within the current step.

A complete action is processed in one sequence diagram.

If you only want to execute one step to the first instruction of a called function block, you have to use the Single step command, page 142,.

Also refer to

- Execute to return, page 143.

3.5.23 Single Step

Icon:

Default shortcut <Alt>+<F12>

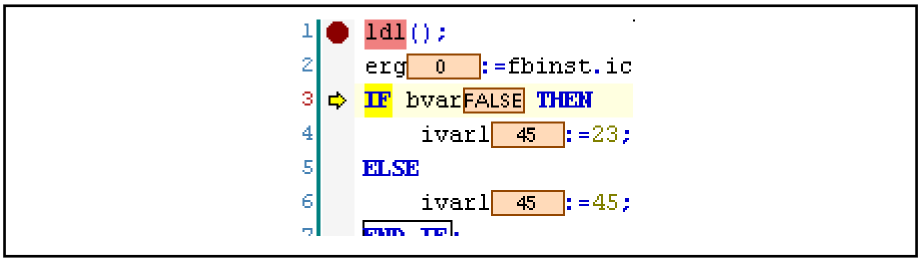
Menu: **Debug** ▸ **Single Step**.

This command can be used to execute a program in single steps in online mode (Stepping, page 82), e.g. to make troubleshooting easier.

A single step is executed. The program stops before the next instruction. If required, it is switched to another POU. If the current position is a function or a function block call, it is stopped before the first instruction of the function block called.

In all other situations the command has the same effect as a procedure step; see "Procedure step", page 142.

Possible stop positions during single step processing depends on the editor type. The current position is displayed with yellow shadowing.



```
ldl();
erg 0 :=fbinst.ic
IF bvar FALSE THEN
    ivar1 45 :=23;
ELSE
    ivar1 45 :=45;
END_IF;
```

Fig.3-62: Example of a single step

3.5.24 Execute to Return

Icon:

Default shortcut: <Shift>+<F10>

Menu: **Debug ▸ Execute to Return**.

This command can be used with a step by step execution of an application program (e.g. for debugging purposes).

If the program does not include any calls, the command 'Execute to return' causes a jump back to the start of the program. If, however, a jump was made previously into a called function block, it triggers a jump back to the calling instruction (note:. This is new compared to IndraLogic 1.x).

This way, in the case of nested calls, 'Execute to return' takes effect step by step backwards in the hierarchy of the caller. This allows to jump back one step in order to jump into another called POU from there, for example.

Also refer to

- Single step, page 142,
- Procedure step, page 142.

3.5.25 Execute to Cursor

Icon:

Default shortcut: <F7>

Menu: **Debug ▸ Execute to Cursor**.

This command can be used to execute the program up to a temporarily defined position (e.g. for debugging purposes).

The effect is the same as when the next breakpoint is defined at this position; see "New breakpoint", page 138.

The instructions located between the current and the newly defined position are executed.

3.5.26 Determine Next Instruction

Icon:

Menu: **Debug ▸ Determine Next Instruction**.

This command can be used with a step by step execution of an application program (e.g. for debugging purposes) in order to define the instruction that is to be carried out next.

To do this, position the cursor in the instruction that is to be executed next and trigger the command.

3.5.27 Display Next Instruction

Icon:

Menu: **Debug** ▸ **Display Next Instruction**.

This command can be used in online mode to move back to the current execution position if the cursor was placed somewhere else in the user interface during the step by step processing of a program. The window of the corresponding function block is brought back into the foreground and the cursor is placed in front of the next instruction to be executed.

3.5.28 Write Values for All Online Applications

Default shortcut: <Ctrl>+<F7>

Menu: **Debug** ▸ **Write values for all online applications**.

⚠ CAUTION

The extraordinary modification of values in an application running on the control might lead to unwanted behavior of the controlled system.

Depending on the controlled system, damage at the system and workpieces might result or the safety of persons might be endangered.

Evaluate possible risks before writing or forcing variable values and take corresponding precautions.

"Writing" a value with this command means that at the beginning of the next cycle, the corresponding variable in the control is set to the value defined for it in the programming system.

☞ For more information, see also the "Force values" command used to set the variable values permanently in the control.

To prepare variables for writing, the desired value has to be defined in online mode at one of the following positions used for monitoring, page 82,:

- In a monitoring window that was defined in the project and that contains a list of the variables to be monitored (monitoring list, watch list, page 499).
- In the online view of an object in the declarations, page 330, of the associated editor

Example:

Write values

Open an object in online mode, e.g. a program written in ST.

The expressions (variables) that can be monitored are displayed in a table in the declarations.

Click on the corresponding field in the **Prepared value** column and enter the desired value.

Then execute the **Write values** command that is in the **Online** menu by default.

The prepared value is then entered in the same line in the **Value** column, which indicates that it was written to the control.

The "Prepared value" field is now empty again.

The same action can be performed in a monitoring list, watch list, page 499,, which contains the corresponding expression or variable.

In this context, also note the Prepare value, page 145 dialog, which can be opened for the variables that were just "forced" and in which a new value can also be defined for "writing".

3.5.29 Force Values on All Online Applications

Menu: **Debug** ▸ **Force values on all online applications.**

⚠ CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

Evaluate possible risks before writing or forcing variable values and take corresponding precautions.

This command is available in online mode. It causes one or more expressions (variables) in the application on the control to be permanently set to defined values.

This setting is then carried out once at the beginning and once at the end of a processing cycle.

Sequence of the process in a cycle:

1. Read inputs
2. Force values
3. Process code
4. Force values
5. Write outputs

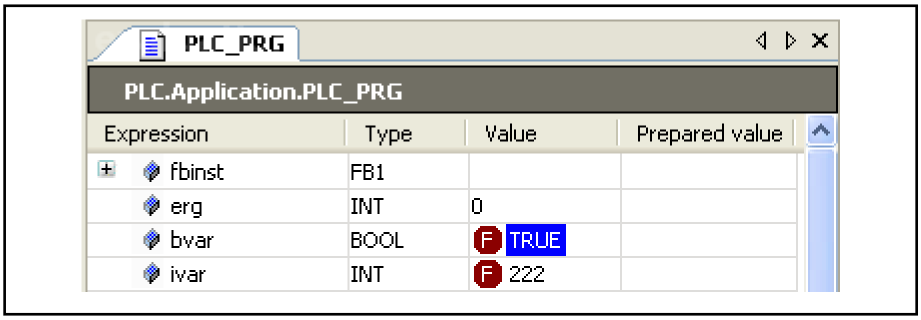
☞ See also the Write values, page 144 command used to make the setting with a defined value only once at the beginning of the cycle.

The forcing is performed until the user cancels it explicitly for a single variable or for all variables or until the user logs out of the application.

To prepare variables for forcing, the desired value has to be defined in online mode at one of the following positions used for monitoring, page 82,:

- In a monitoring window that was defined in the project and that contains a list of the variables to be monitored (monitoring list, watch list, page 499).
- In the online view of an object in the declarations, page 330, of the associated editor
- In the online view of an object within the implementation section of the FBD/LD/IL editor, page 362.

A "forced" value is indicated by an F symbol.



PLC_PRG

PLC.Application.PLC_PRG

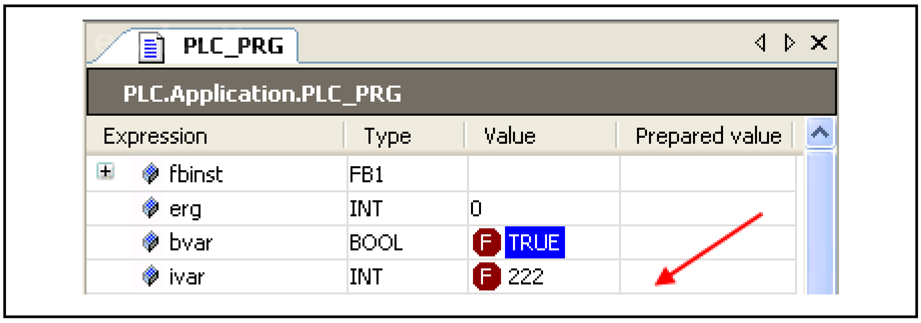
Expression	Type	Value	Prepared value
fbinst	FB1		
erg	INT	0	
bvar	BOOL	F TRUE	
ivar	INT	F 222	

Fig.3-63: Example of forced variables in the declaration editor of a function block (online view)

"Prepare value" dialog

This dialog is used to define a new value for a variable. To do this, either a new value is entered or forcing is canceled and the value returns to the current or previous value.

The dialog opens when clicking the "Prepared value" field of a variable that has just been forced



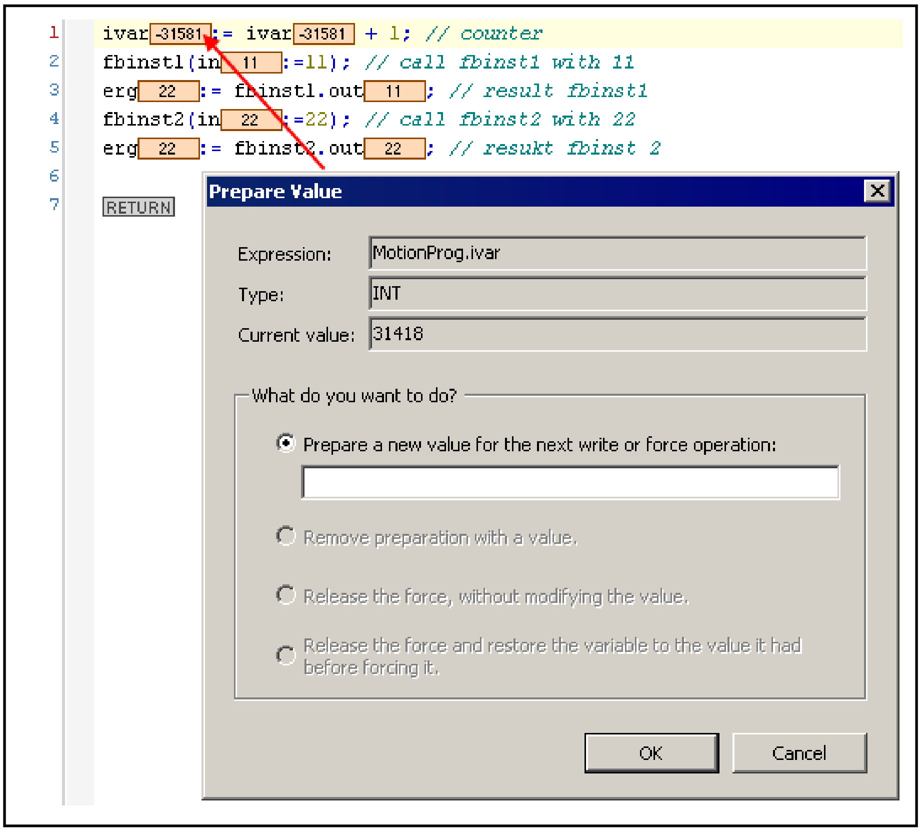
PLC_PRG

PLC.Application.PLC_PRG

Expression	Type	Value	Prepared value
fbinst	FB1		
erg	INT	0	
bvar	BOOL	F TRUE	
ivar	INT	F 222	

Fig.3-64: Example of forced variables in the declaration editor, click "Prepared value"

or in the Inline monitoring field of the variable in the implementation section of the FBD/LD/IL/ST editor.



```
ivar -31581 := ivar -31581 + 1; // counter
fbinst1(in 11 :=11); // call fbinst1 with 11
erg 22 := fbinst1.out 11 ; // result fbinst1
fbinst2(in 22 :=22); // call fbinst2 with 22
erg 22 := fbinst2.out 22 ; // resukt fbinst 2

RETURN
```

Prepare Value

Expression: MotionProg.ivar

Type: INT

Current value: 31418

What do you want to do?

● Prepare a new value for the next write or force operation:

○ Remove preparation with a value.

○ Release the force, without modifying the value.

○ Release the force and restore the variable to the value it had before forcing it.

Fig.3-65: "Prepare value" dialog

The following information on the respective variables is displayed:

- **Expression:** Path: e.g. "PLC.Application.PLC_PRG.ivar"
- **Data type:** e.g. "DWORD"
- **Current value:** e.g. "TRUE" or "23"

Select one of the following options to determine **what should happen with the variable**:

- **Prepare a new value for the next write or force operation:** Depending on the data type of the variable, a new numerical value or string value can be entered to which the variable is to be set.
- **Remove preparation with a value:** The prepared value is deleted.
- **Release the force without modifying the value:** The variable is marked with <Unforce> and thus prepared to receive the current value from the control.
- **Release the force and restore the variable to the value it had before forcing it:** The variable is marked with <Unforce and restore> and thus now prepared to receive the value that it had before forcing.

After closing the dialog with "OK", the variable in the "Prepared value" field in the monitoring view(s) now indicates a newly defined value or "<Unforce>" or "<Unforce and restore>". Then next time the "Force values" or "Write values" command is carried out (for the first option), all of these prepared values are set.

3.5.30 Cancel Forcing for All Values on All Online Applications

Menu: **Debug** ▸ **Cancel forcing for all values on all online applications.**

⚠ CAUTION

The extraordinary modification of variable values in an application running on the control might lead to unwanted behavior of the controlled system.

Depending on the controlled system, damage at the system and workpieces might result or the health and life of persons might be at risk.

Evaluate possible risks before writing or forcing variable values and take corresponding precautions.

This command can be used in online mode to cancel forcing for all of the variables currently being forced for the applications included in the project.

The variables then receive their current value(s) from the control.

This corresponds to the option "Release the force without modifying the value." that can be selected in the Prepare values dialog for a forced value.

3.5.31 Add All Forces to Monitoring Window

This command is available in online mode as soon as one of the four monitoring views, page 141, "Watch 1", "Watch 2", "Watch 3" or "Watch 4" is active.

It adds all of the variables in the current application that are now prepared to be forced or have already been forced to the watch list.

Please note, however, that this only works for docked monitoring views.

Also note the "Watch all forces" view, which automatically and continuously displays all variables that have been prepared for forcing and also provides commands for canceling the forcing.

3.5.32 Sequence Control

Menu: **Debug ▸ Sequence control.**

This command is used to switch the sequence control on and off which is supported for the ST, FBD, LD and IL language editors.

An activated sequence control allows to track the processing of the application program.

1. The current values of the variables and thus also the results of function calls and operations are displayed in the editor windows, at the relevant processing position and at the relevant processing time. For comparison: The standard monitoring only provides the value of a variable between two processing cycles.
2. Exactly the code lines or networks are marked in color that have been run through in the current cycle.

The sequence control can only be activated in online mode and only works in the currently visible part of the currently active editor window. 'Sequence control activated' is displayed in the status line as long as the function is active and sequence control positions (parts of the code run through) are visible in the editor window.

An activated sequence control extends the application runtime!

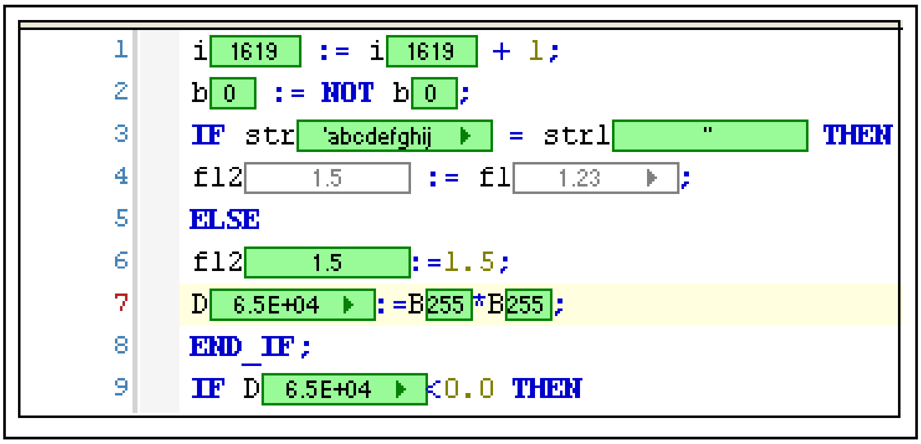
If the "Secure online operation" option is enabled in the "Communication settings" of the control, a message box appears upon sequence control switch-on. You are asked whether you really want to activate the function and are once again provided the possibility to cancel the process.

Illustration of the sequence control in the different language editors:

By default, green is configured as color for marking the sequence control positions. (The color can be changed in the Text editor options, page 218,.)

In all editors, the current values of variables and inputs and outputs concerned are displayed in small rectangular fields as they are also used for the standard monitoring. In code parts run through, these fields are displayed in the color defined for the sequence control, in code parts not run through, they are shown in white with grey font.

Please note that the displayed value at a code position not run through is a "normal" monitoring value, i.e. the value between two task cycles.



```
i 1619 := i 1619 + 1;
b 0 := NOT b 0 ;
IF str 'abcdefghij ▸ = strl '' THEN
fl2 1.5 := fl 1.23 ▸ ;
ELSE
fl2 1.5 :=1.5;
D 6.5E+04 ▸ :=B 255 *B 255 ;
END_IF;
IF D 6.5E+04 ▸ <0.0 THEN
```

Fig.3-66: Example: Sequence control in the ST editor

In **network editors**, the networks run through are marked at the left edge by means of bars in the 'sequence control color'.

In the **LD**, the connection lines currently run through are generally shown in **green** (or the 'sequence control color' currently set), the other ones in **gray**.

The current value of the connection is shown, as well:

- TRUE by means of **thick blue**,
- FALSE by means of **thick black** lines,
- unknown or analog values by means of **thin black** lines.

Due to the combination of the relevant information, this might lead to dashed lines.

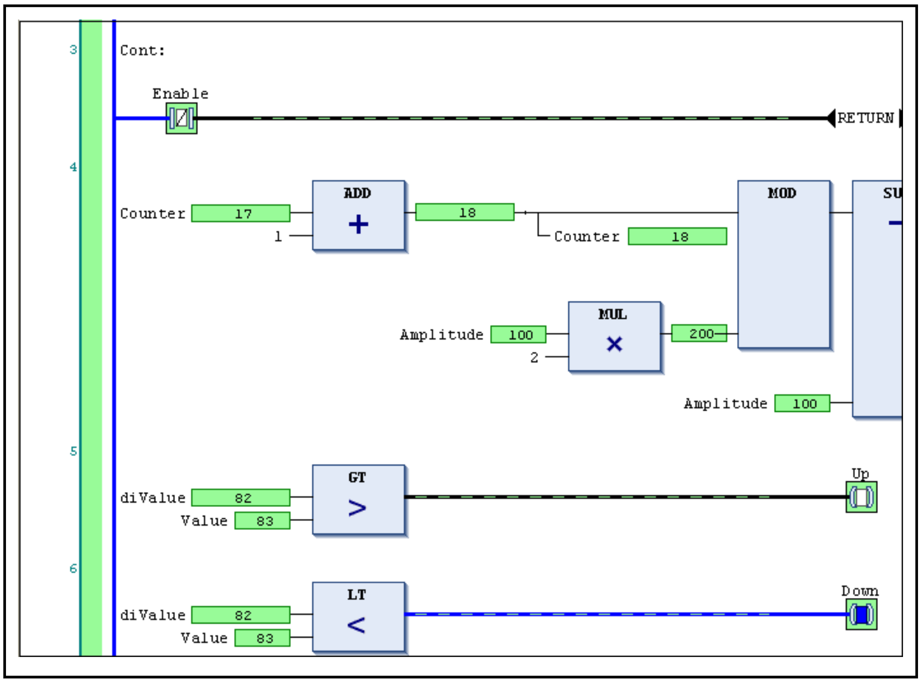
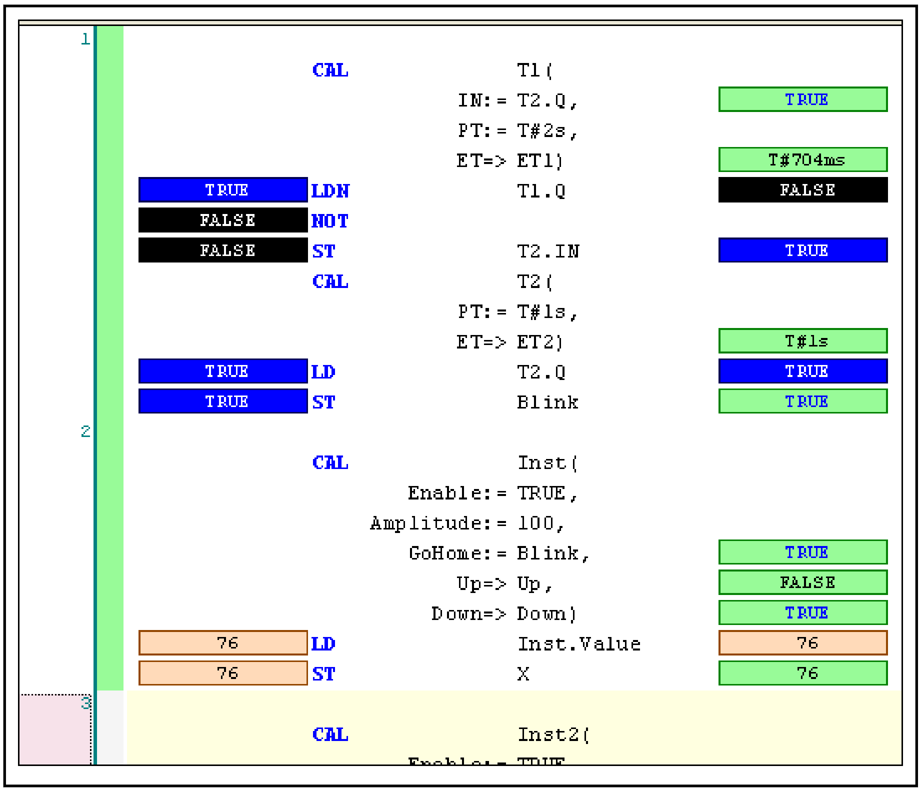


Fig.3-67: Example: Sequence control in the LD editor

In IL, two fields are used for each instruction to display the current values.

The field on the left side of the operator contains the current accumulator value, the field on the right side of the operand the operand value.



```
            CAL                T1(
                          IN:= T2.Q,              TRUE
                          PT:= T#2s,
                          ET=> ET1)               T#704ms
TRUE        LDN                T1.Q               FALSE
FALSE       NOT
FALSE       ST                 T2.IN              TRUE
            CAL                T2(
                          PT:= T#1s,
                          ET=> ET2)               T#1s
TRUE        LD                 T2.Q               TRUE
TRUE        ST                 Blink              TRUE

            CAL                Inst(
                      Enable:= TRUE,
                   Amplitude:= 100,
                      GoHome:= Blink,             TRUE
                          Up=> Up,                FALSE
                        Down=> Down)              TRUE
76          LD                 Inst.Value         76
76          ST                 X                  76

            CAL                Inst2(
                      Enable:= TRUE
```

Fig.3-68: Example: Sequence control in the IL editor

Note that **writing values** is also possible in the sequence control mode.

Forcing is not possible!

Double-clicking the value display field opens the Prepare value, page 145, dialog to enter the desired value.

3.5.33 Display Mode

This command can be used to set the format for displaying the values (Monitoring, page 82) in online mode.

To do this, click on the desired option, which is then displayed with a checkmark.

- Binary
- Decimal
- Hexadecimal

3.6 Tools

3.6.1 Comparison of PLC Objects

Comparison of PLC Objects, General

The Compare... menu item provides the option

1. to compare PLC objects of the **current** project with each other,
2. to compare PLC objects of the **current** project with those of an export file (*.iwx),
3. to compare PLC objects of two export files with each other.

In the first case, the desired objects are to be highlighted in the Project Explorer.

In the second case, the desired object is to be highlighted in the Project Explorer and "Compare..." is to be selected in the context menu of the object. In the appearing window, the desired export file has to be selected.

To compare two export files, the menu item can be accessed in **Tools ▸ Compare...** .

The description is made based on the combination of project/export file. This applies similarly to export file/export file.

If the comparison cannot be made, since the object types differ, an error message appears.

Basics of the comparison in IndraWorks Engineering:

- Rexroth IndraWorks 12VRS Engineering,
  DOK-IWORKS-ENGINEE*V12-APxx-EN-P, R911334388.
- – Toolbar in the "Compare" dialog, page 110.
  – Toolbar in the" Comparison results" dialog, page 111.

Comparison of PLC objects

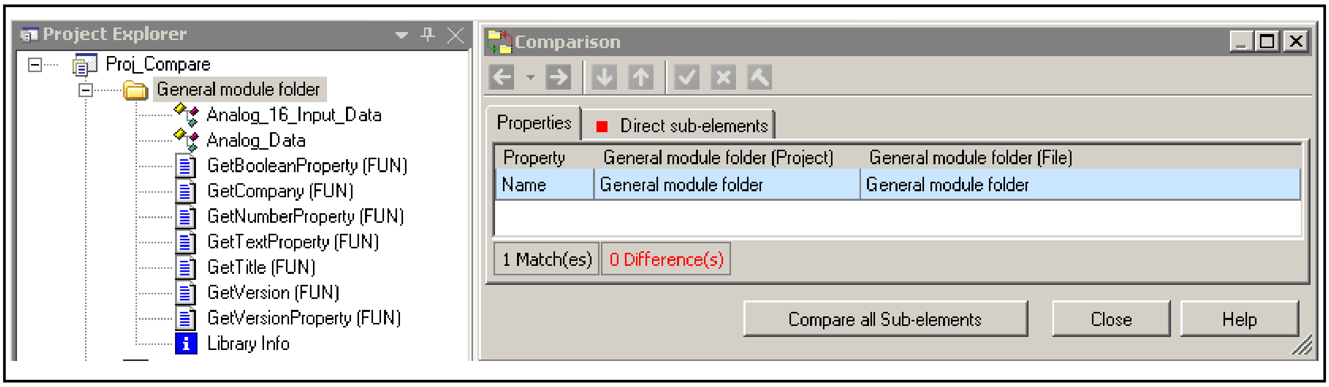
- General module, page 152
- Logic, page 153
- Application, page 154
- Task configuration, page 157
- Task, page 152

- Folder (Difference from IndraWorks folder), page 159
- Data types, page 159
- Global variable list, persistent variables, network variables, page 161
- Network variable lists, receiver end, page 162
- Programs, function blocks, functions..., page 163
- Interfaces, interface methods, interface properties, page 169
- Library manager, page 170
- Library info, page 172
- Data server, page 173,
- Data source (data server), page 174,
- Symbol configuration, page 175,
- Visualization manager, page 175,
- Visualization, page 177
- Text list, page 177
- Image pool, page 177
- Recipe manager, page 177
- Recipe definition, page 179
- Devices (Onboard I/O, Inline I/O, field busses, …), page 179

Comparison: General Module

Here only the name is displayed in the window. "Merging" is not possible, since the element is fixed.

Differences can result based on the name or language.

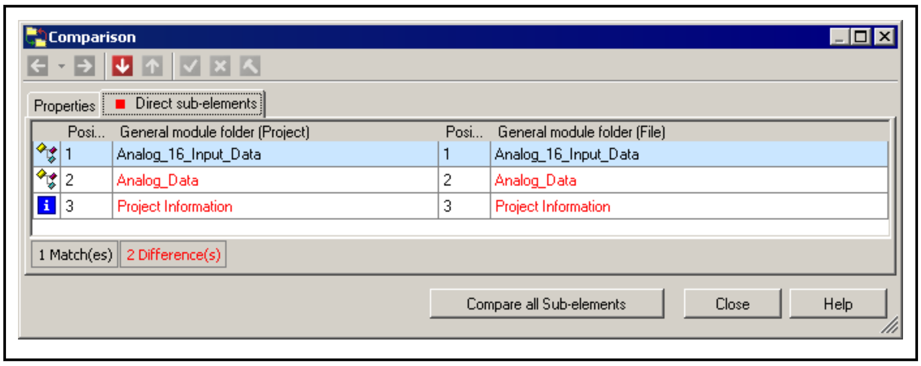


Proj_Compare
General module folder
Analog_16_Input_Data
Analog_Data
GetBooleanProperty (FUN)
GetCompany (FUN)
GetNumberProperty (FUN)
GetTextProperty (FUN)
GetTitle (FUN)
GetVersion (FUN)
GetVersionProperty (FUN)
Library Info

Property	General module folder (Project)	General module folder (File)
Name	General module folder	General module folder

1 Match(es) 0 Difference(s)

Fig.3-69: Comparison: General module folder



Posi...	General module folder (Project)	Posi...	General module folder (File)
1	Analog_16_Input_Data	1	Analog_16_Input_Data
2	Analog_Data	2	Analog_Data
3	Project Information	3	Project Information

1 Match(es) 2 Difference(s)

Fig.3-70: Comparison: Listing of the "Direct subelements", two elements with differences (red)

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

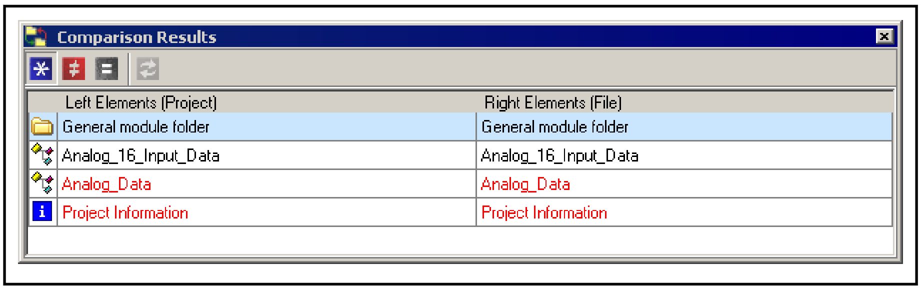
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

☞ The terms "Project Information" and "Library Info" designate the same element.

Comparison: All subelements

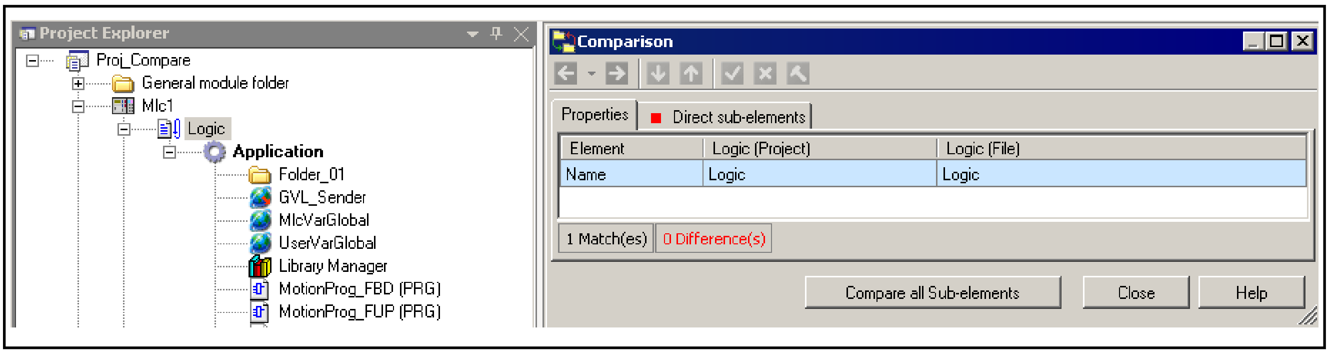


Left Elements (Project)	Right Elements (File)
General module folder	General module folder
Analog_16_Input_Data	Analog_16_Input_Data
Analog_Data	Analog_Data
Project Information	Project Information

Fig.3-71: Comparison: "All subelements", two elements with differences (red) with filter option "All elements", "Different elements", "Identical elements"

Comparison: Logic

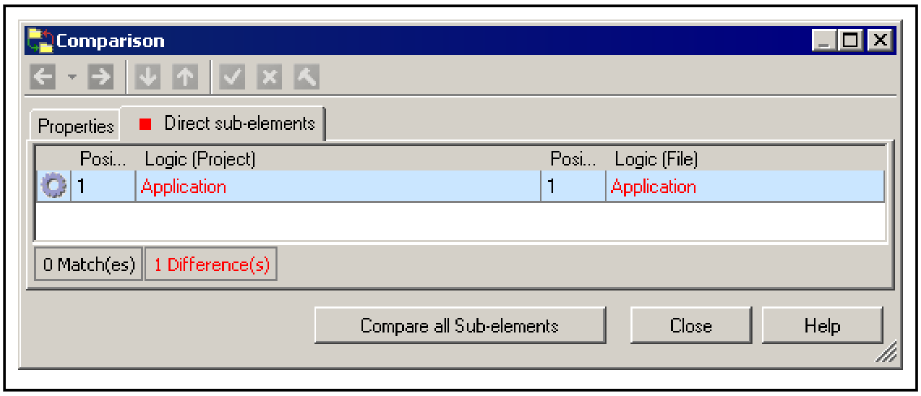
Here only the name is displayed in the window. "Merging" is not possible, since the element is fixed.



Element	Logic (Project)	Logic (File)
Name	Logic	Logic

1 Match(es) 0 Difference(s)

Fig.3-72: Comparison: Logic

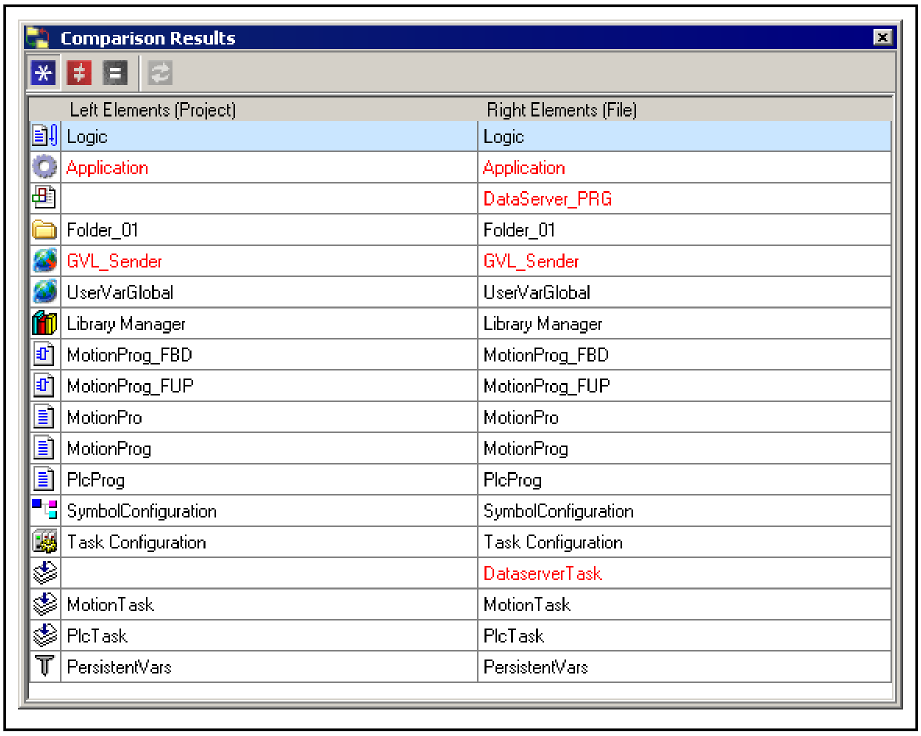


Posi...	Logic (Project)	Posi...	Logic (File)
1	Application	1	Application

0 Match(es) 1 Difference(s)

Fig.3-73: Comparison: Logic with one single subelement "Application"

Comparison: All subelements

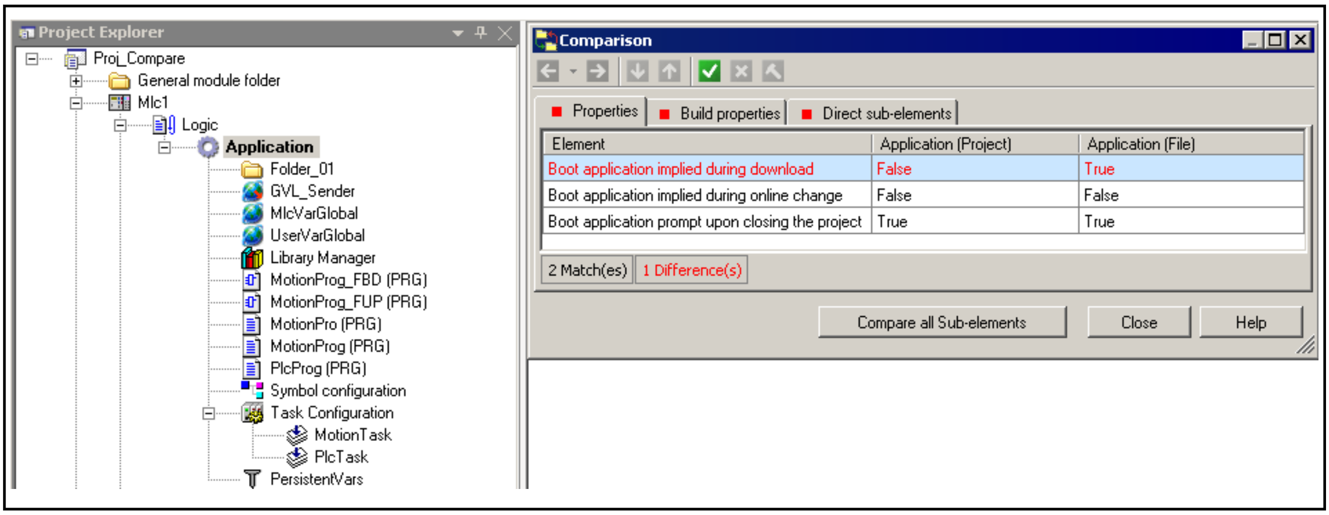


Left Elements (Project)	Right Elements (File)
Logic	Logic
Application	Application
	DataServer_PRG
Folder_01	Folder_01
GVL_Sender	GVL_Sender
UserVarGlobal	UserVarGlobal
Library Manager	Library Manager
MotionProg_FBD	MotionProg_FBD
MotionProg_FUP	MotionProg_FUP
MotionPro	MotionPro
MotionProg	MotionProg
PlcProg	PlcProg
SymbolConfiguration	SymbolConfiguration
Task Configuration	Task Configuration
	DataserverTask
MotionTask	MotionTask
PlcTask	PlcTask
PersistentVars	PersistentVars

Fig.3-74: Comparison: Logic with all subelements with filter option "All elements", "Different elements", "Identical elements"

Comparison: Application

Here, the name is displayed, the options for the boot project handling and the other application properties. These are located under Properties in the "Application" node context menu.



Element	Application (Project)	Application (File)
Boot application implied during download	False	True
Boot application implied during online change	False	False
Boot application prompt upon closing the project	True	True

2 Match(es) 1 Difference(s)

Fig.3-75: Comparison: Application

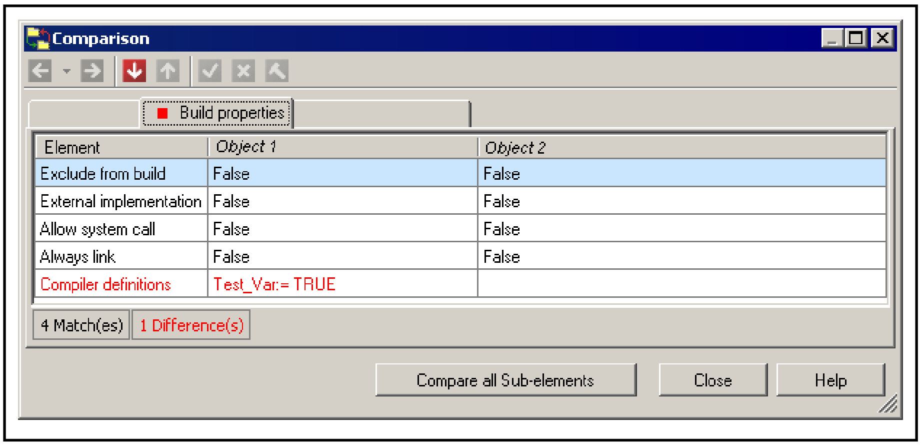
If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-76: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

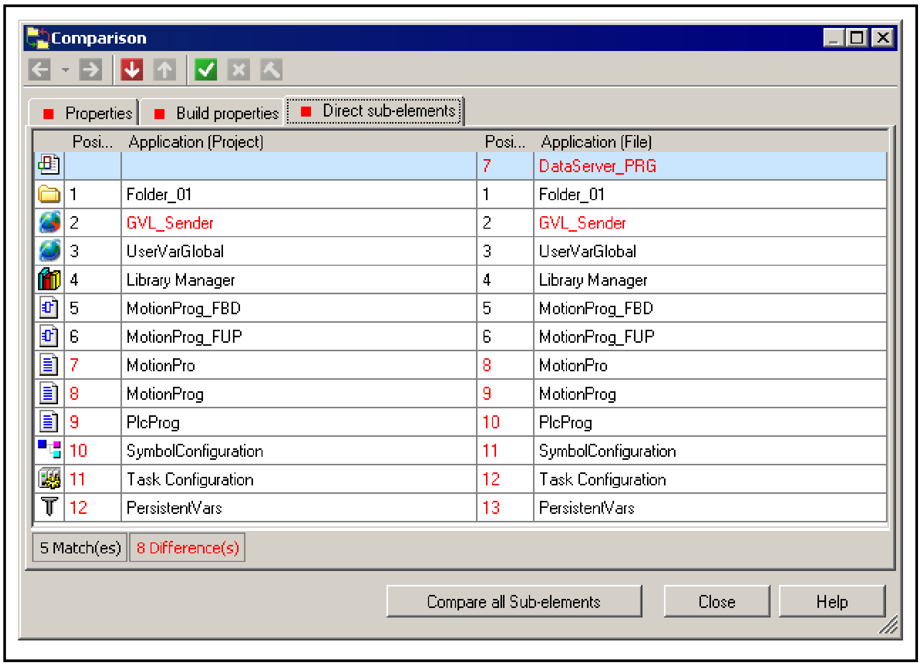
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

The "DataServer_PRG" program (right position 7) is missing in the project and is to be taken from the file.

"GVL_Sender" differs with regard to the content and the order of the objects is different from position 7.



Posi...	Application (Project)	Posi...	Application (File)
		7	DataServer_PRG
1	Folder_01	1	Folder_01
2	GVL_Sender	2	GVL_Sender
3	UserVarGlobal	3	UserVarGlobal
4	Library Manager	4	Library Manager
5	MotionProg_FBD	5	MotionProg_FBD
6	MotionProg_FUP	6	MotionProg_FUP
7	MotionPro	8	MotionPro
8	MotionProg	9	MotionProg
9	PlcProg	10	PlcProg
10	SymbolConfiguration	11	SymbolConfiguration
11	Task Configuration	12	Task Configuration
12	PersistentVars	13	PersistentVars

5 Match(es) 8 Difference(s)

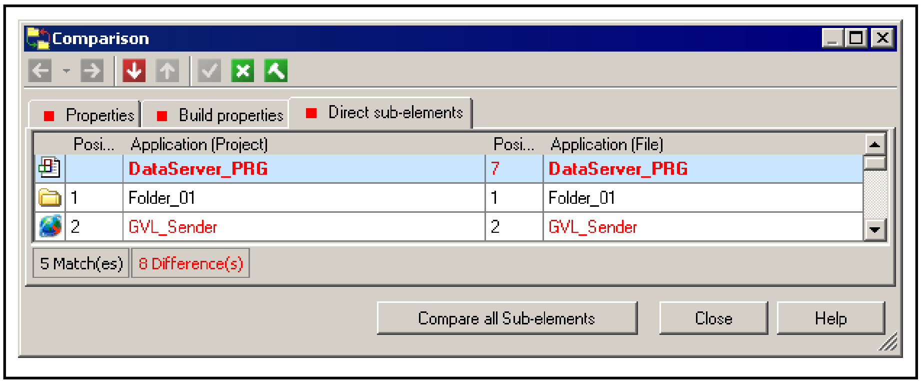
Fig.3-77: Comparison: "Direct subelements"

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

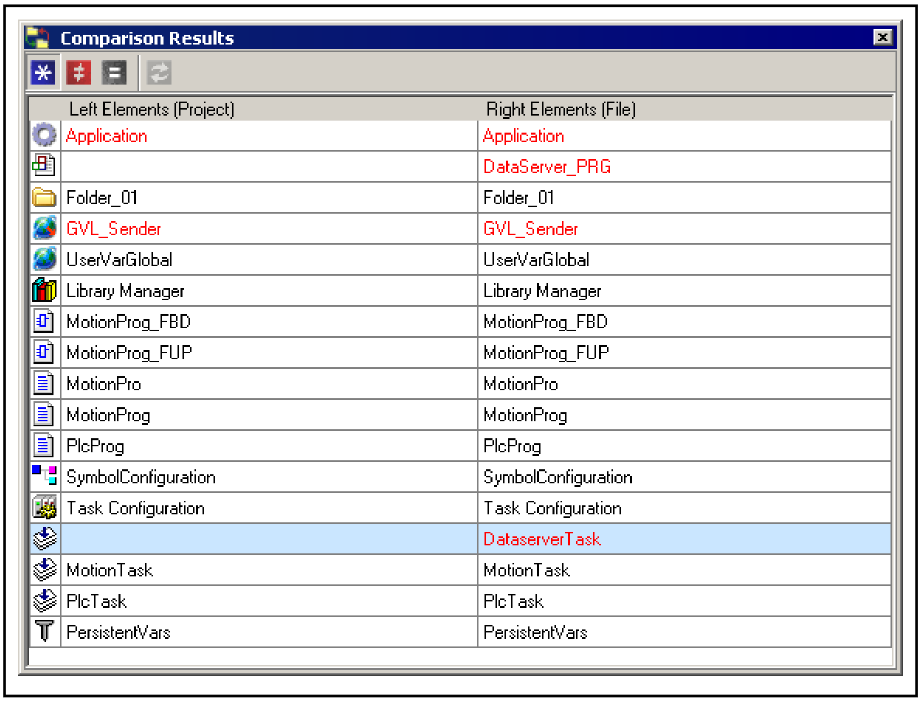


Posi...	Application (Project)	Posi...	Application (File)
	DataServer_PRG	7	**DataServer_PRG**
1	Folder_01	1	Folder_01
2	GVL_Sender	2	GVL_Sender

5 Match(es) 8 Difference(s)

Fig.3-78: Comparison: File is accepted into the project with "Start merge operation"

Comparison: All subelements



Comparison Results

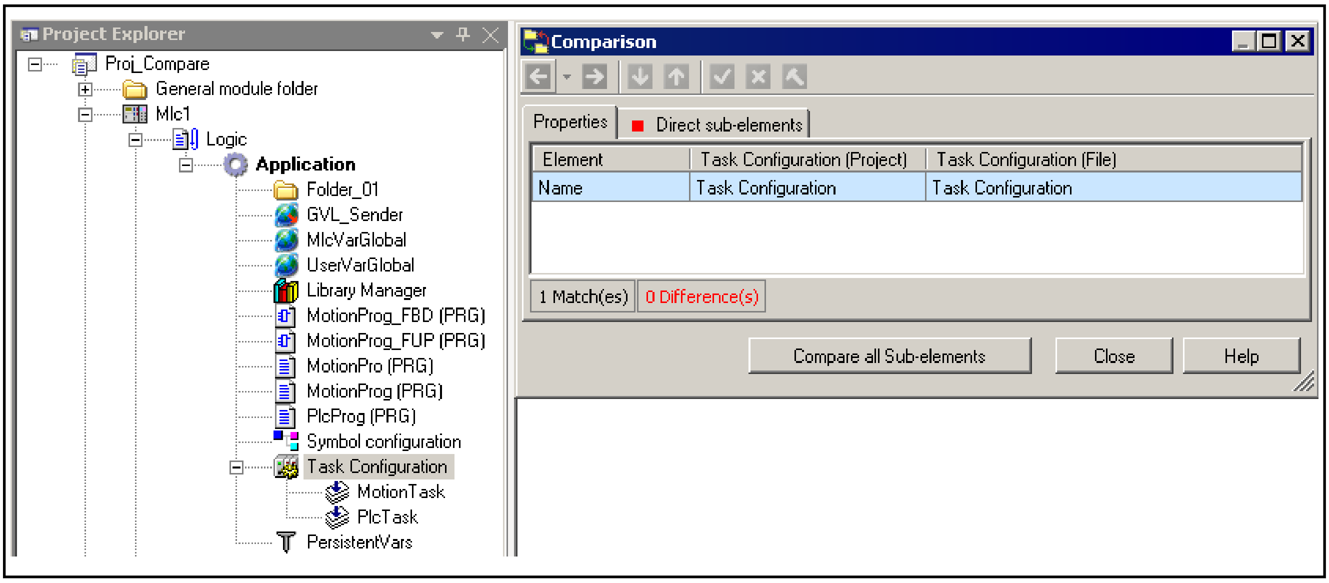
Left Elements (Project)	Right Elements (File)
Application	Application
	DataServer_PRG
Folder_01	Folder_01
GVL_Sender	GVL_Sender
UserVarGlobal	UserVarGlobal
Library Manager	Library Manager
MotionProg_FBD	MotionProg_FBD
MotionProg_FUP	MotionProg_FUP
MotionPro	MotionPro
MotionProg	MotionProg
PlcProg	PlcProg
SymbolConfiguration	SymbolConfiguration
Task Configuration	Task Configuration
	DataserverTask
MotionTask	MotionTask
PlcTask	PlcTask
PersistentVars	PersistentVars

Fig.3-79: Comparison: Application with all subelements with filter option "All elements", "Different elements", "Identical elements"

In addition to the direct subelements, their subelements are displayed, as well. Here, the "sub-subelements" of the task configuration (DataServerTask, only right, MotionTask and PlcTask).

Comparison: Task Configuration

Here only the name Task configurationis displayed in the window. "Merging" is not possible, since the element is fixed.



Project Explorer

- Proj_Compare
  - General module folder
  - Mlc1
    - Logic
      - Application
        - Folder_01
        - GVL_Sender
        - MlcVarGlobal
        - UserVarGlobal
        - Library Manager
        - MotionProg_FBD (PRG)
        - MotionProg_FUP (PRG)
        - MotionPro (PRG)
        - MotionProg (PRG)
        - PlcProg (PRG)
        - Symbol configuration
        - Task Configuration
          - MotionTask
          - PlcTask
        - PersistentVars

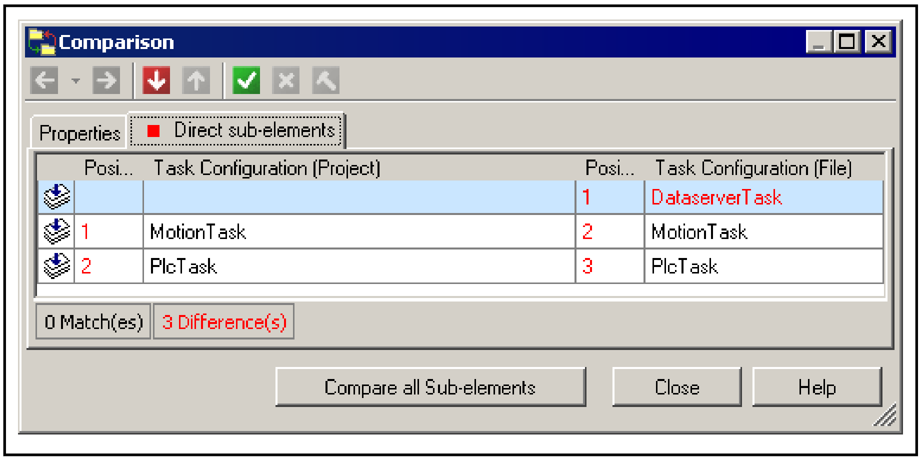
Comparison

Properties | Direct sub-elements

Element	Task Configuration (Project)	Task Configuration (File)
Name	Task Configuration	Task Configuration

1 Match(es) 0 Difference(s)

Fig.3-80: Comparison: Task configuration



Posi...	Task Configuration (Project)	Posi...	Task Configuration (File)
		1	DataserverTask
1	MotionTask	2	MotionTask
2	PlcTask	3	PlcTask

Fig.3-81: Comparison: Task configuration, tasks as "Direct subelements"

The "DataserverTask" task is only contained in the file storage and can be accepted into the project.

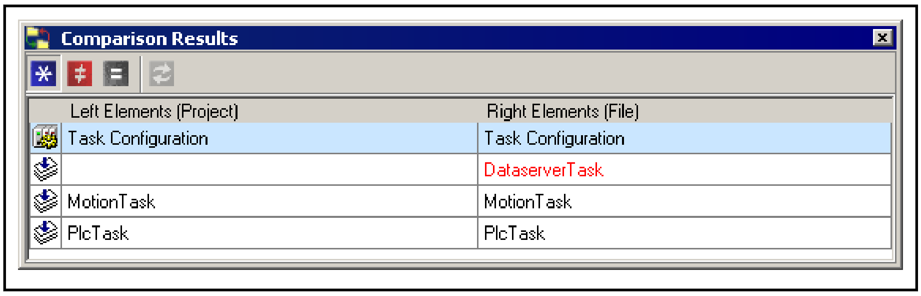
If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

Comparison: All subelements



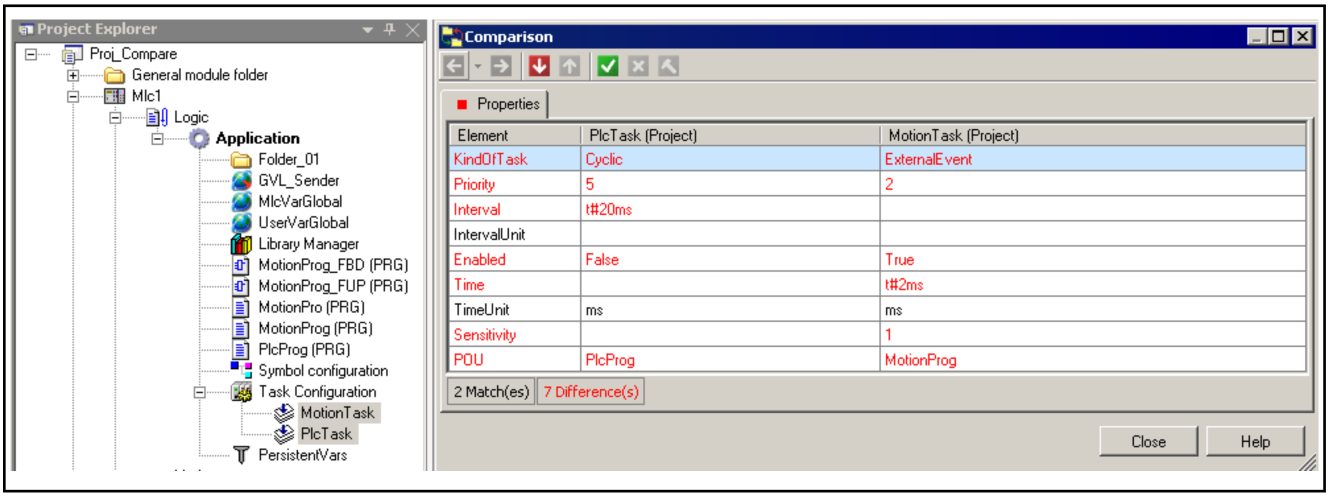
Left Elements (Project)	Right Elements (File)
Task Configuration	Task Configuration
	DataserverTask
MotionTask	MotionTask
PlcTask	PlcTask

Fig.3-82: Comparison: Application with all subelements with filter option "All elements", "Different elements", "Identical elements"

Comparison: Task

Task objects are compared in tabular form. Here, comparison between the "PlcTask" and "MotionTask" tasks.

☞ Merging is not recommended...



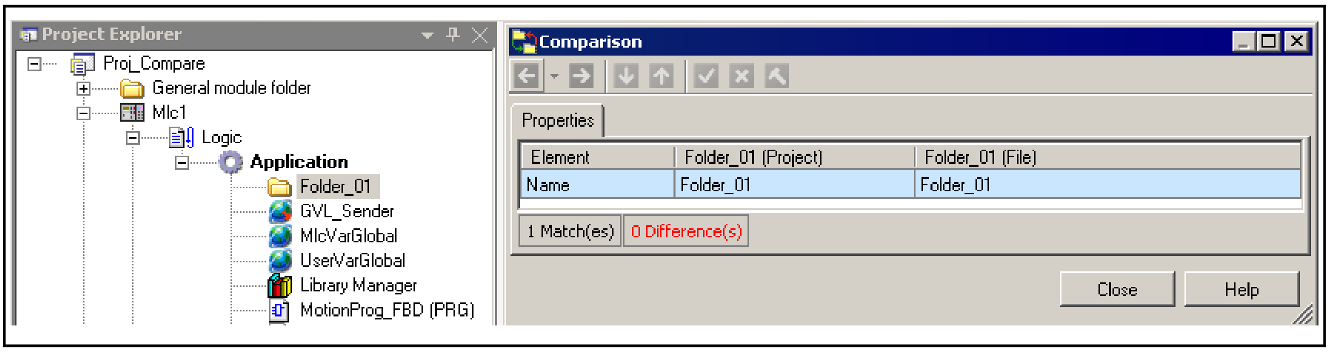
Element	PlcTask (Project)	MotionTask (Project)
KindOfTask	Cyclic	ExternalEvent
Priority	5	2
Interval	t#20ms	
IntervalUnit		
Enabled	False	True
Time		t#2ms
TimeUnit	ms	ms
Sensitivity		1
POU	PlcProg	MotionProg

2 Match(es) 7 Difference(s)

Fig.3-83: Comparison: Task, here comparison between the "PlcTask" and "MotionTask" tasks

Comparison: Folder (Differences with Respect to the IndraWorks Folder)

Only the name is displayed. It can be renamed if the folder in the file (right) has another name.



Element	Folder_01 (Project)	Folder_01 (File)
Name	Folder_01	Folder_01

1 Match(es) 0 Difference(s)

Fig.3-84: Comparison: Folder

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

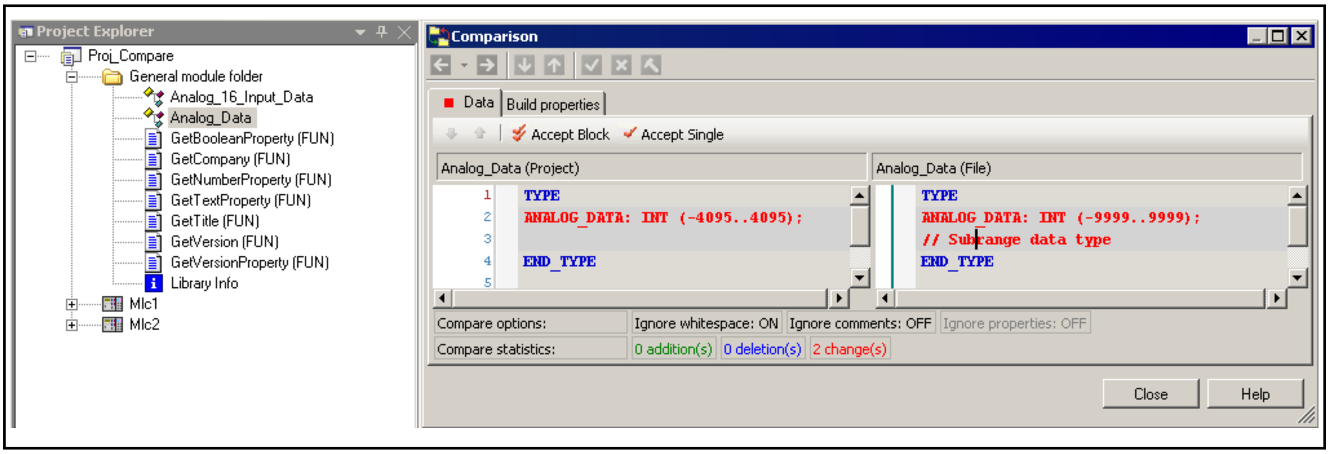
Select the cross ✗ to prevent (undo) merging.

Comparison: Data Types

The comparison applies to all data types that can be created with the DUT editor as independent objects.

Among others, the following are supported: structures (STRUCT), arrays (ARRAY), unions (UNION), enumeration data types (ENUM) and partial range data types (SUBRANGE).

When data types are compared, the content is displayed as text and can be modified.



Analog_Data (Project)

```
TYPE
ANALOG_DATA: INT (-4095..4095);

END_TYPE
```

Analog_Data (File)

```
TYPE
ANALOG_DATA: INT (-9999..9999);
// Subrange data type
END_TYPE
```

Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 2 change(s)

Fig.3-85: Comparison: Data Types

The next/previous difference is selected with ⬇ or ⬆.

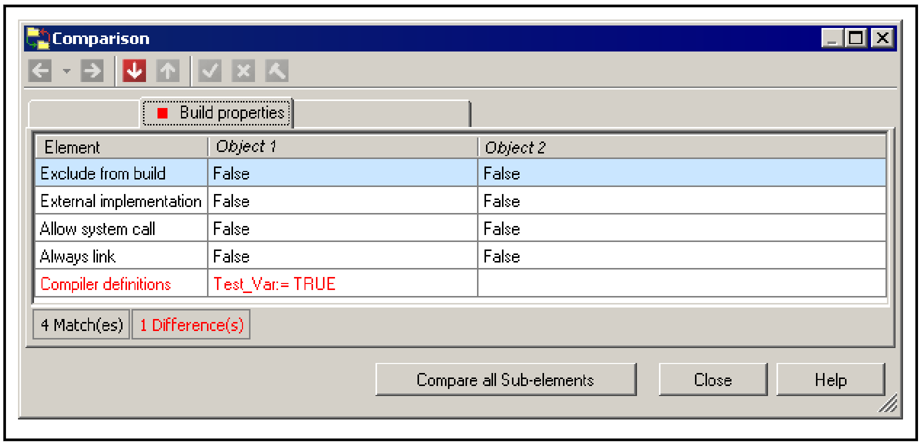
✓ Accept Block can be used to accept the coplete highlighted text block.

✓ Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-86: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

Comparison: Global Variable List, Persistent Variables, Network Variables

The comparison applies to all global variable lists ①, global variable lists used as network variables on the sender end ② and persistent variable lists ③ that can be added as independent objects of an application (or in the general module folder).

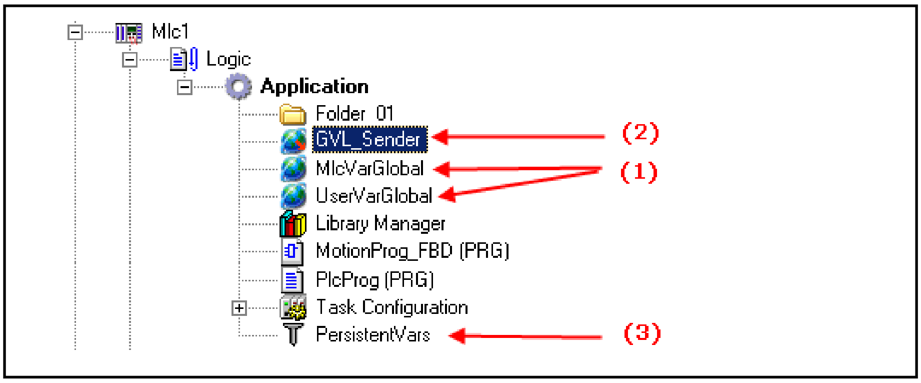
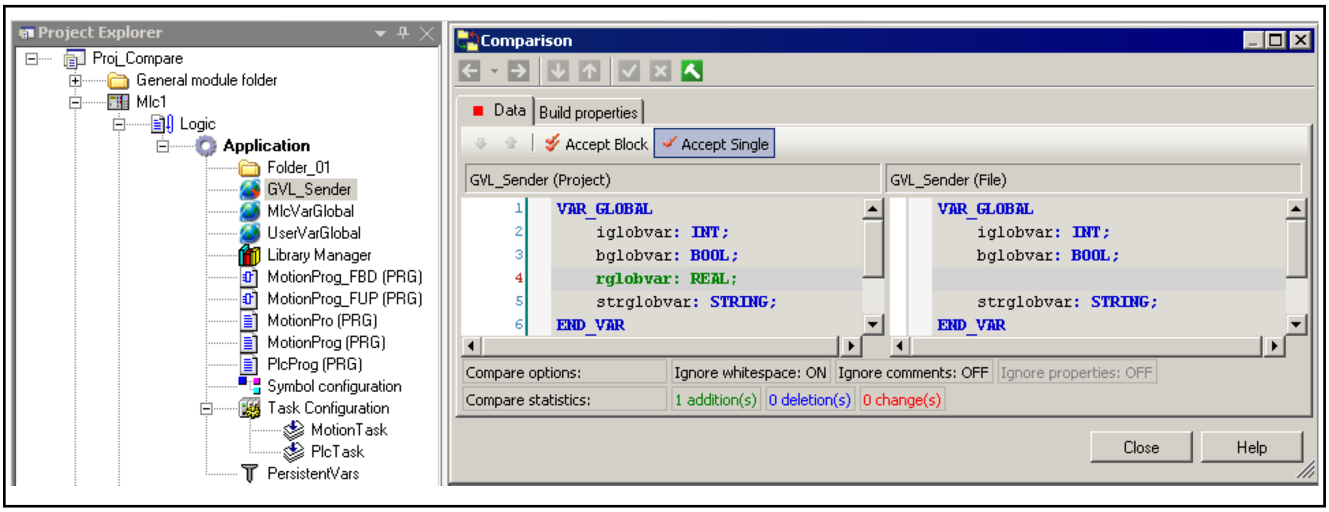


Fig.3-87: Comparison: Possible global variable lists

When data types are compared, the content is displayed as text and can be modified.



GVL_Sender (Project)

```
VAR_GLOBAL
    iglobvar: INT;
    bglobvar: BOOL;
    rglobvar: REAL;
    strglobvar: STRING;
END_VAR
```

GVL_Sender (File)

```
VAR_GLOBAL
    iglobvar: INT;
    bglobvar: BOOL;

    strglobvar: STRING;
END_VAR
```

Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 1 addition(s) 0 deletion(s) 0 change(s)

Fig.3-88: Comparison: Global variable lists

The next/previous difference is selected with ⬇ or ⬆.

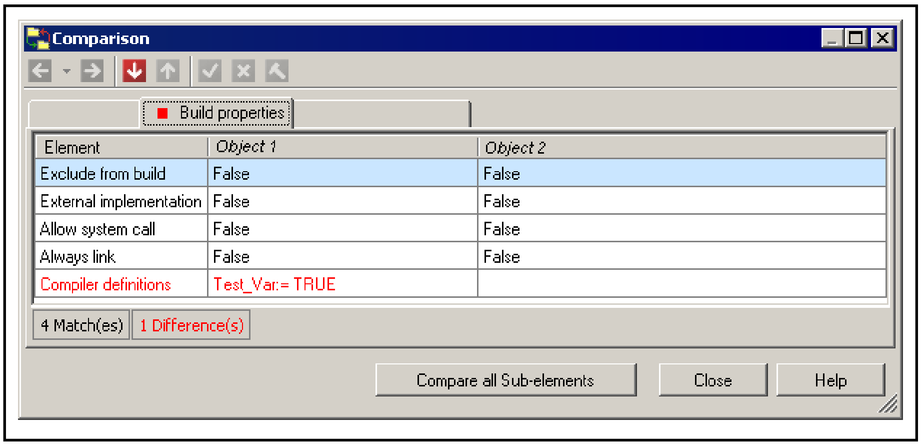
✓ Accept Block can be used to accept the coplete highlighted text block.

✓ Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer to merge.

Select the cross ☒ to prevent (undo) merging.

The next tab shows the build properties.



Comparison

■ Build properties

Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-89: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✖ to prevent (undo) merging.

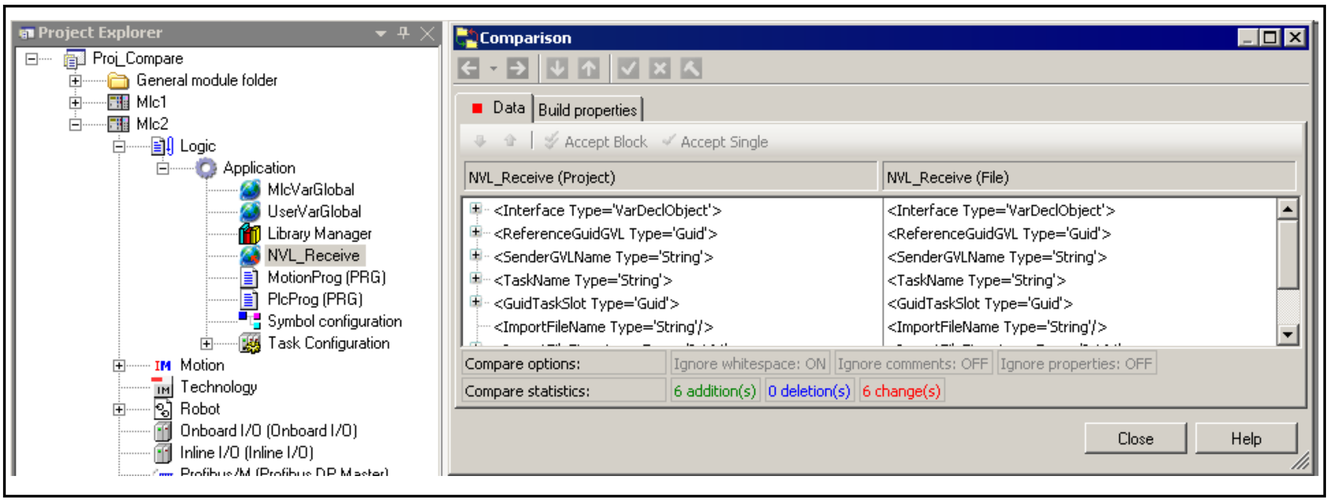
Comparison: Network Variable Lists, Receiver End

The comparison applies to global variable lists used as network variables on the receiver end that were created as independent objects of an application.

☞ These lists are created automatically and remain current within the project.

In this way, the comparison is only executed as a check for differences.

When comparing network variable lists, the content display is XML based. Modification (merging) is not possible.



NVL_Receive (Project)	NVL_Receive (File)
<Interface Type='VarDeclObject'>	<Interface Type='VarDeclObject'>
<ReferenceGuidGVL Type='Guid'>	<ReferenceGuidGVL Type='Guid'>
<SenderGVLName Type='String'>	<SenderGVLName Type='String'>
<TaskName Type='String'>	<TaskName Type='String'>
<GuidTaskSlot Type='Guid'>	<GuidTaskSlot Type='Guid'>
<ImportFileName Type='String'/>	<ImportFileName Type='String'/>

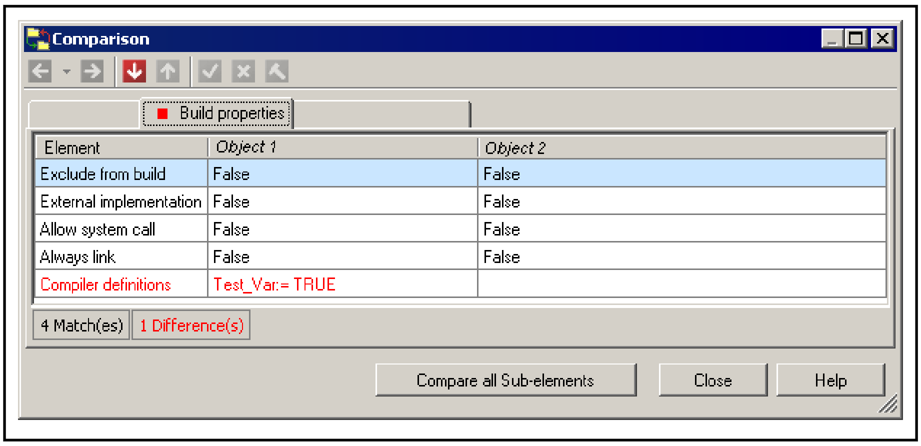
Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 6 addition(s) 0 deletion(s) 6 change(s)

Fig.3-90: Comparison: Network variable lists on the receiver end

The next/previous difference is selected with ⬇ or ⬆.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-91: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

Comparison: Programs, Function Blocks, Functions...

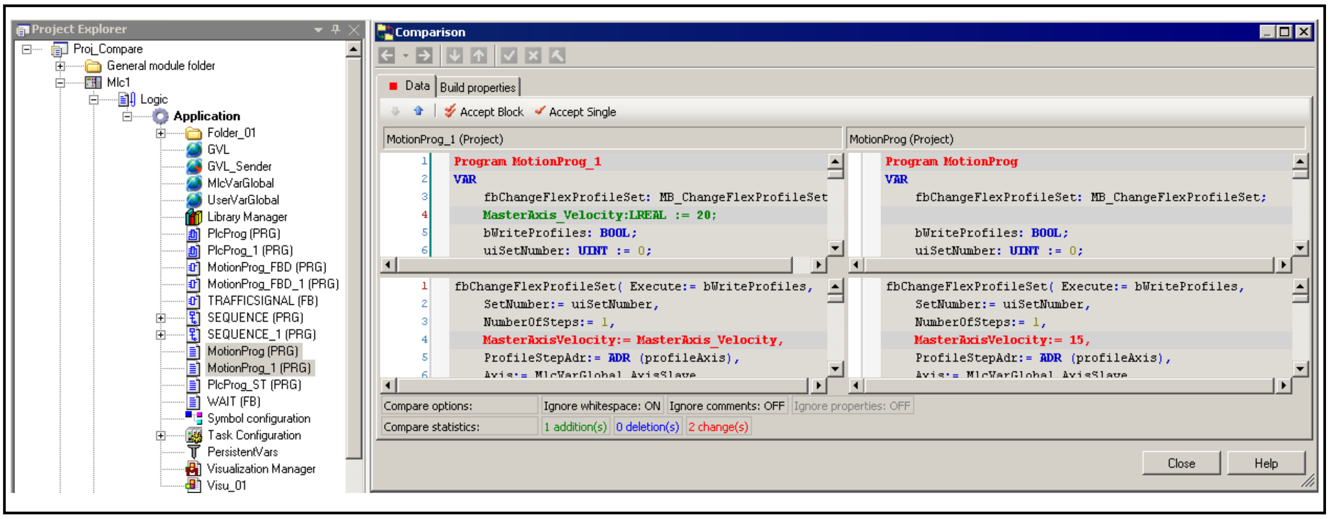
The comparison applies to the following objects:

- PROGRAM, FUNCTION_BLOCK, FUNCTION
- ACTION, TRANSITION
- METHOD, PROPERTY with GET and SET

The objects named can be declared and implemented in the IEC programming language editors available in the system.

Comparison: ST

Comparisons of the declaration and objects which have been implemented in structured text are text based.



MotionProg_1 (Project)

```
Program MotionProg_1
VAR
    fbChangeFlexProfileSet: MB_ChangeFlexProfileSet
    MasterAxis_Velocity:LREAL := 20;
    bWriteProfiles: BOOL;
    uiSetNumber: UINT := 0;
```

```
fbChangeFlexProfileSet( Execute:= bWriteProfiles,
    SetNumber:= uiSetNumber,
    NumberOfSteps:= 1,
    MasterAxisVelocity:= MasterAxis_Velocity,
    ProfileStepAdr:= ADR (profileAxis),
    Axis:= MlcVarGlobal.AxisSlave
```

MotionProg (Project)

```
Program MotionProg
VAR
    fbChangeFlexProfileSet: MB_ChangeFlexProfileSet;

    bWriteProfiles: BOOL;
    uiSetNumber: UINT := 0;
```

```
fbChangeFlexProfileSet( Execute:= bWriteProfiles,
    SetNumber:= uiSetNumber,
    NumberOfSteps:= 1,
    MasterAxisVelocity:= 15,
    ProfileStepAdr:= ADR (profileAxis),
    Axis:= MlcVarGlobal.AxisSlave
```

Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 1 addition(s) 0 deletion(s) 2 change(s)

Fig.3-92: Comparison: PROGRAM in ST, declaration and implementation

Merging is possible in the declaration and in the implementation.

The next/previous difference is selected with ⇩ or ⇧.

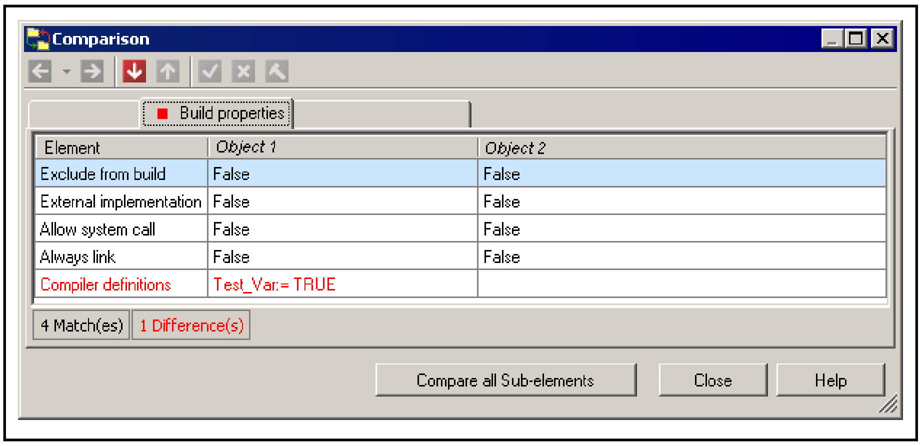
Accept Block can be used to accept the coplete highlighted text block.

Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer ⚒ to merge.

Select the cross ☒ to prevent (undo) merging.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-93: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⇩ can be used to select the next difference (⇧ for the previous difference).

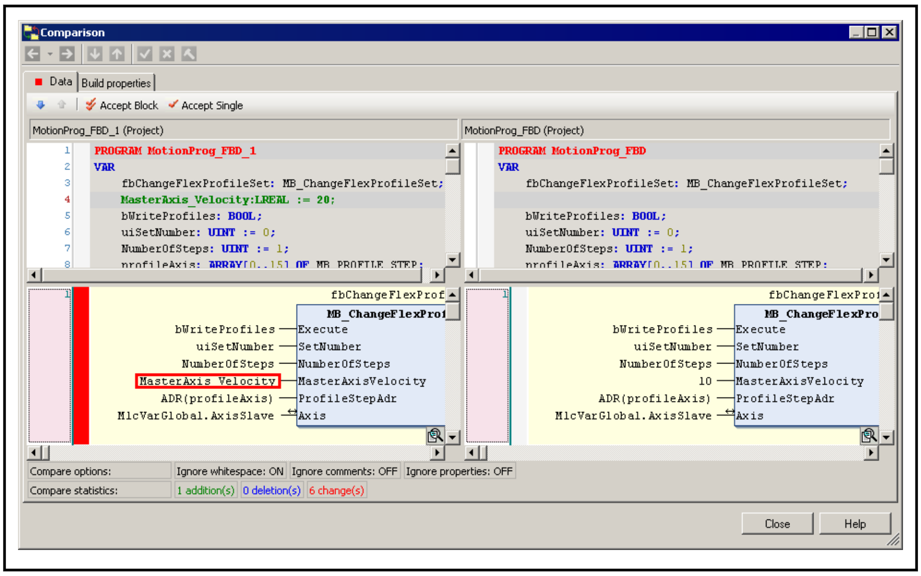
If the checkmark ☑ appears, it can be used to prepare the acceptance (merging):

Select the hammer to merge.

Select the cross ✕ to prevent (undo) merging.

Comparison: FBD

Objects that have been implemented as function block diagram are compared graphically.



MotionProg_FBD_1 (Project)

```
PROGRAM MotionProg_FBD_1
VAR
    fbChangeFlexProfileSet: MB_ChangeFlexProfileSet;
    MasterAxis_Velocity:LREAL := 20;
    bWriteProfiles: BOOL;
    uiSetNumber: UINT := 0;
    NumberOfSteps: UINT := 1;
    profileAxis: ARRAY[0..15] OF MB_PROFILE_STEP;
```

MotionProg_FBD (Project)

```
PROGRAM MotionProg_FBD
VAR
    fbChangeFlexProfileSet: MB_ChangeFlexProfileSet;

    bWriteProfiles: BOOL;
    uiSetNumber: UINT := 0;
    NumberOfSteps: UINT := 1;
    profileAxis: ARRAY[0..15] OF MB_PROFILE_STEP;
```

fbChangeFlexProf
MB_ChangeFlexProf
bWriteProfiles — Execute
uiSetNumber — SetNumber
NumberOfSteps — NumberOfSteps
MasterAxis_Velocity — MasterAxisVelocity
ADR(profileAxis) — ProfileStepAdr
MlcVarGlobal.AxisSlave — Axis
fbChangeFlexProf
MB_ChangeFlexPro
bWriteProfiles — Execute
uiSetNumber — SetNumber
NumberOfSteps — NumberOfSteps
10 — MasterAxisVelocity
ADR(profileAxis) — ProfileStepAdr
MlcVarGlobal.AxisSlave — Axis

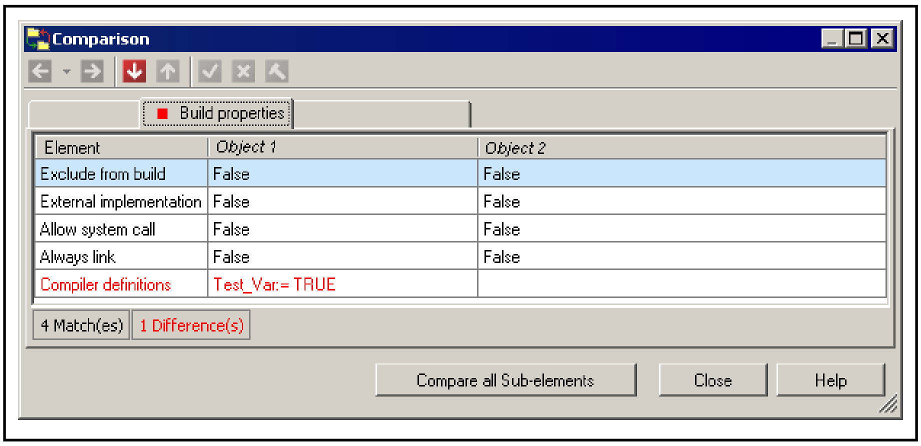
Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 1 addition(s) | 0 deletion(s) | 6 change(s)

Fig.3-94: Comparison: PROGRAM in FBD, declaration and implementation

Merging is possible in the declaration and in the implementation.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

Fig.3-95: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

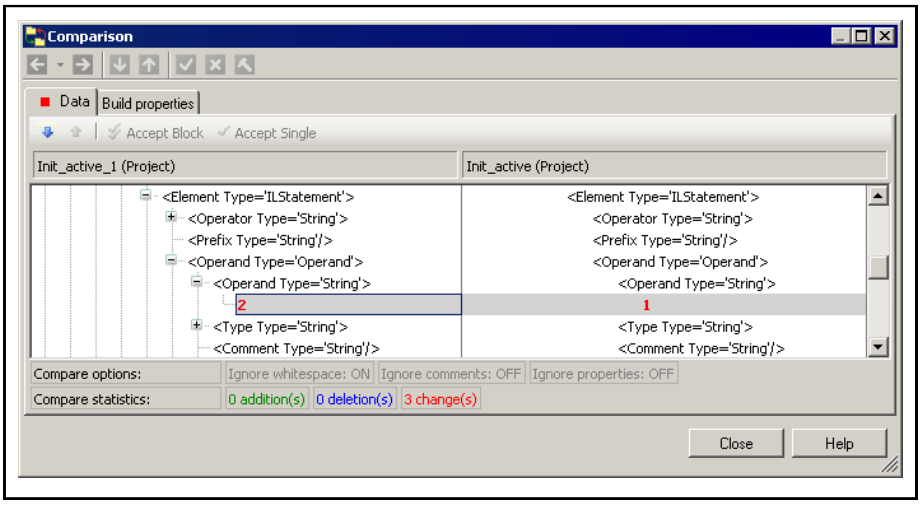
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer to merge.

Select the cross ✕ to prevent (undo) merging.

Comparison: IL

POUs implemented in the instruction list are compared as XML. Merging is not possible here.



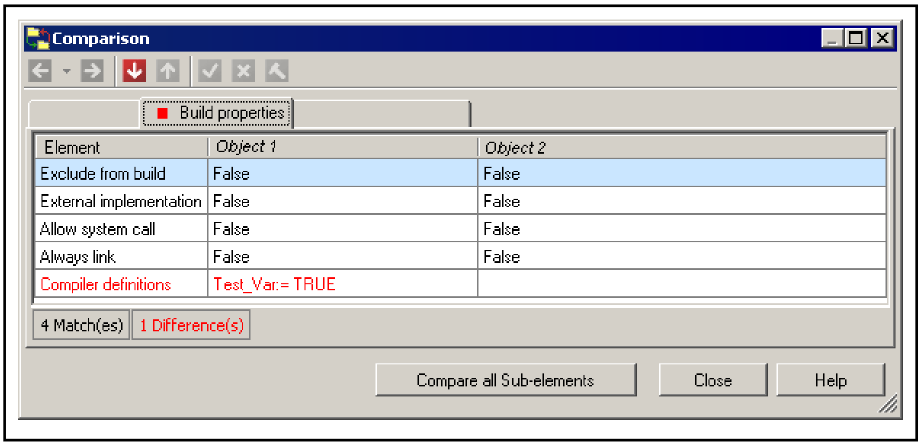
Init_active_1 (Project)	Init_active (Project)
<Element Type='ILStatement'>	<Element Type='ILStatement'>
<Operator Type='String'>	<Operator Type='String'>
<Prefix Type='String'/>	<Prefix Type='String'/>
<Operand Type='Operand'>	<Operand Type='Operand'>
<Operand Type='String'>	<Operand Type='String'>
2	1
<Type Type='String'>	<Type Type='String'>
<Comment Type='String'/>	<Comment Type='String'/>

Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 3 change(s)

Fig.3-96: Comparison: Action in instruction list, implementation

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-97: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

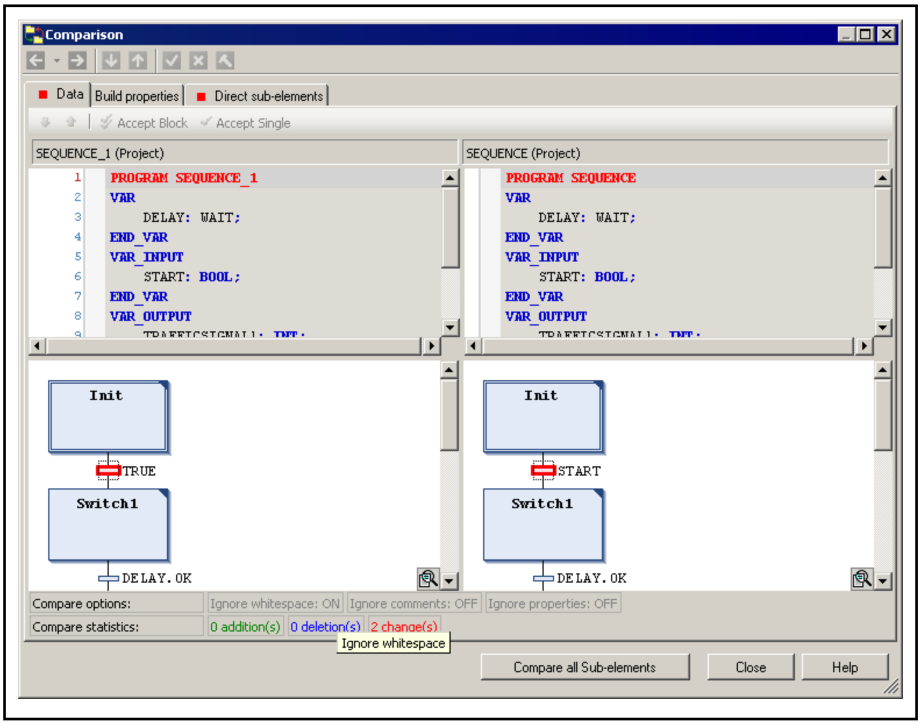
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

Comparison: SFC

Objects that have been structured by means of SFC (sequential function chart) are compared graphically. Merging is only possible in the declaration.



SEQUENCE_1 (Project)

```
PROGRAM SEQUENCE_1
VAR
    DELAY: WAIT;
END_VAR
VAR_INPUT
    START: BOOL;
END_VAR
VAR_OUTPUT
    TRAFFICSIGNAL1: INT;
```

TRUE

DELAY.OK

SEQUENCE (Project)

```
PROGRAM SEQUENCE
VAR
    DELAY: WAIT;
END_VAR
VAR_INPUT
    START: BOOL;
END_VAR
VAR_OUTPUT
    TRAFFICSIGNAL1: INT;
```

START

DELAY.OK

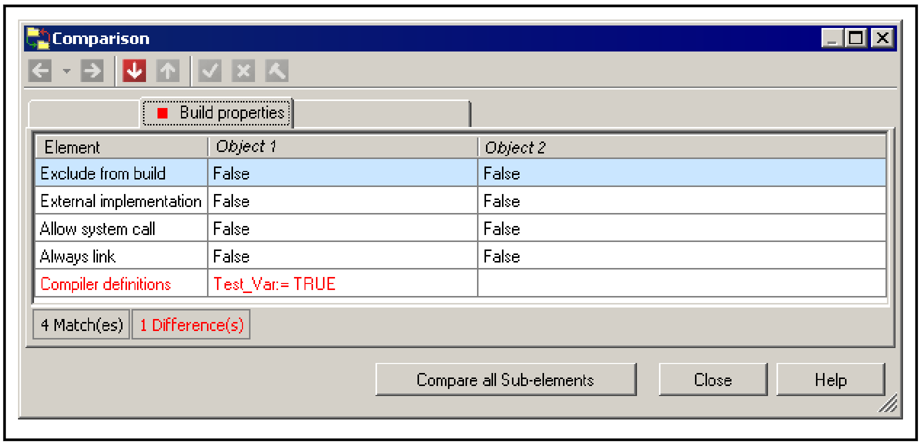
Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 2 change(s)

Ignore whitespace

Fig.3-98: Comparison: FUNCTION_BLOCK with SFC structure, declaration and implementation

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

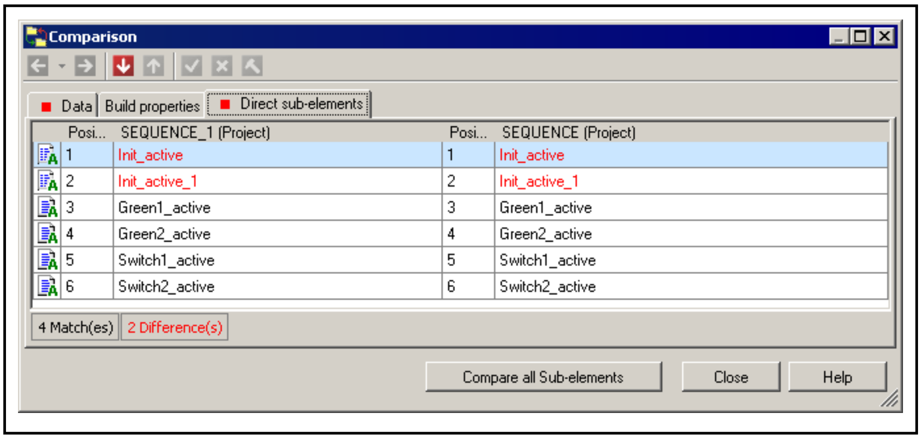
Fig.3-99: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.



Posi...	SEQUENCE_1 (Project)	Posi...	SEQUENCE (Project)
1	Init_active	1	Init_active
2	Init_active_1	2	Init_active_1
3	Green1_active	3	Green1_active
4	Green2_active	4	Green2_active
5	Switch1_active	5	Switch1_active
6	Switch2_active	6	Switch2_active

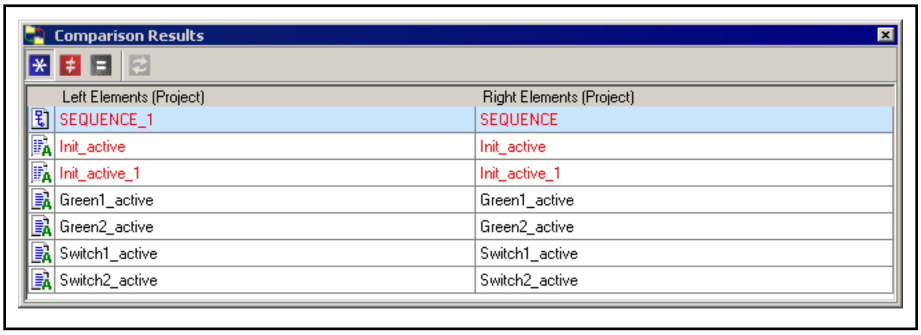
Fig.3-100: Comparison: POU "Direct subelements"

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer ⚒ to merge.

Select the cross ✗ to prevent (undo) merging.

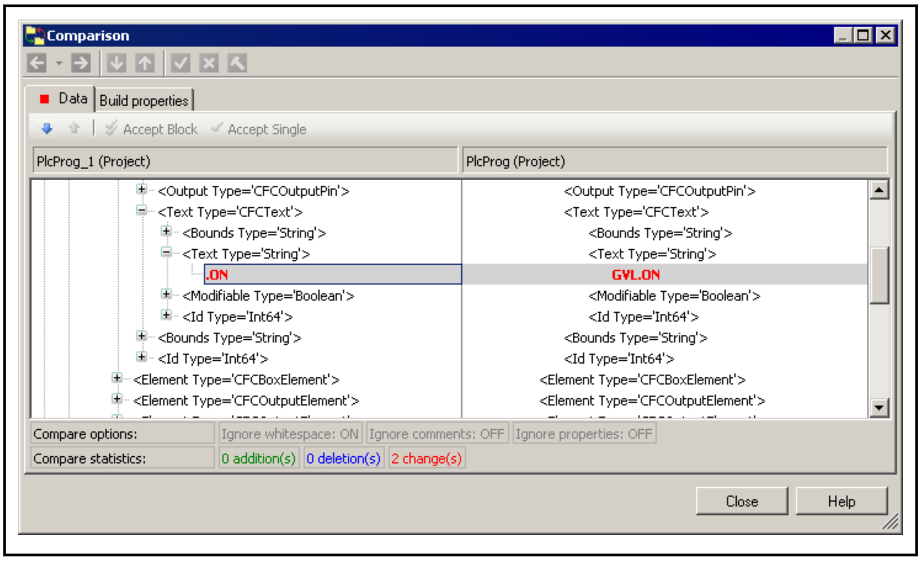


Left Elements (Project)	Right Elements (Project)
SEQUENCE_1	SEQUENCE
Init_active	Init_active
Init_active_1	Init_active_1
Green1_active	Green1_active
Green2_active	Green2_active
Switch1_active	Switch1_active
Switch2_active	Switch2_active

Fig.3-101: Comparison: POU with all subelements with the filter option "All elements", "Different elements", "Identical elements"

Comparison: CFC

POUs implemented in the instruction list are compared as XML. Merging is not possible here.



PlcProg_1 (Project) | PlcProg (Project)

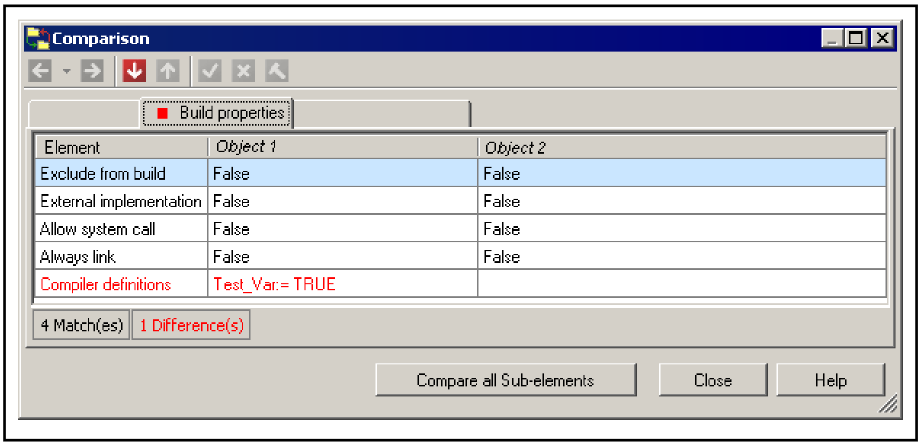
```
<Output Type='CFCOutputPin'>          <Output Type='CFCOutputPin'>
<Text Type='CFCText'>                 <Text Type='CFCText'>
  <Bounds Type='String'>                <Bounds Type='String'>
  <Text Type='String'>                  <Text Type='String'>
    .ON                                   GVL.ON
  <Modifiable Type='Boolean'>           <Modifiable Type='Boolean'>
  <Id Type='Int64'>                     <Id Type='Int64'>
<Bounds Type='String'>                <Bounds Type='String'>
<Id Type='Int64'>                     <Id Type='Int64'>
<Element Type='CFCBoxElement'>        <Element Type='CFCBoxElement'>
<Element Type='CFCOutputElement'>     <Element Type='CFCOutputElement'>
```

Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 0 deletion(s) | 2 change(s)

Fig.3-102: Comparison: PROGRAM in CFC, declaration and implementation

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

Fig.3-103: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

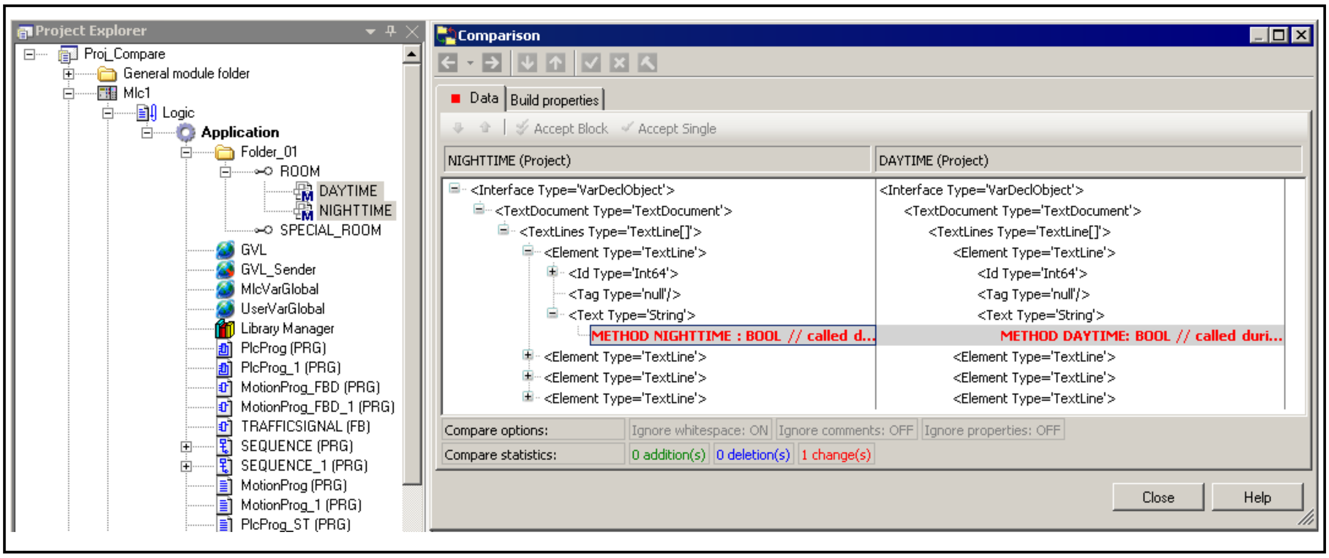
Select the cross ✗ to prevent (undo) merging.

Comparison: Interfaces, Interface Methods, Interface Properties

When comparing interfaces, interface methods or interface properties, the content display is XML based.

Modification (merging) is not possible.

The next/previous difference is selected with ⬇ or ⬆.



Project Explorer

- Proj_Compare
  - General module folder
  - Mlc1
    - Logic
      - Application
        - Folder_01
          - ROOM
            - DAYTIME
            - NIGHTTIME
          - SPECIAL_ROOM
        - GVL
        - GVL_Sender
        - MlcVarGlobal
        - UserVarGlobal
        - Library Manager
        - PlcProg (PRG)
        - PlcProg_1 (PRG)
        - MotionProg_FBD (PRG)
        - MotionProg_FBD_1 (PRG)
        - TRAFFICSIGNAL (FB)
        - SEQUENCE (PRG)
        - SEQUENCE_1 (PRG)
        - MotionProg (PRG)
        - MotionProg_1 (PRG)
        - PlcProg_ST (PRG)

Comparison

Data | Build properties

Accept Block | Accept Single

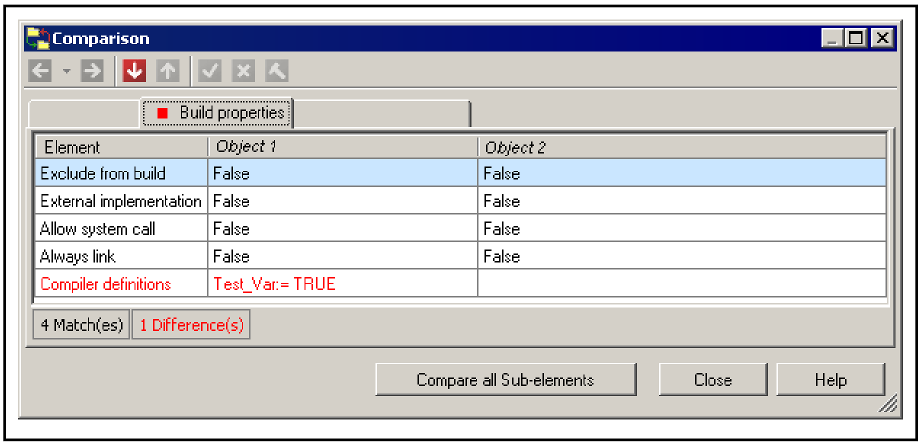
NIGHTTIME (Project)	DAYTIME (Project)
<Interface Type='VarDeclObject'>	<Interface Type='VarDeclObject'>
<TextDocument Type='TextDocument'>	<TextDocument Type='TextDocument'>
<TextLines Type='TextLine[]'>	<TextLines Type='TextLine[]'>
<Element Type='TextLine'>	<Element Type='TextLine'>
<Id Type='Int64'>	<Id Type='Int64'>
<Tag Type='null'/>	<Tag Type='null'/>
<Text Type='String'>	<Text Type='String'>
METHOD NIGHTTIME : BOOL // called d...	METHOD DAYTIME: BOOL // called duri...
<Element Type='TextLine'>	<Element Type='TextLine'>
<Element Type='TextLine'>	<Element Type='TextLine'>
<Element Type='TextLine'>	<Element Type='TextLine'>

Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 0 deletion(s) | 1 change(s)

Fig.3-104: Comparison: Interfaces, interface methods, interface properties

The next tab shows the build properties.



Comparison

Build properties

Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

Fig.3-105: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

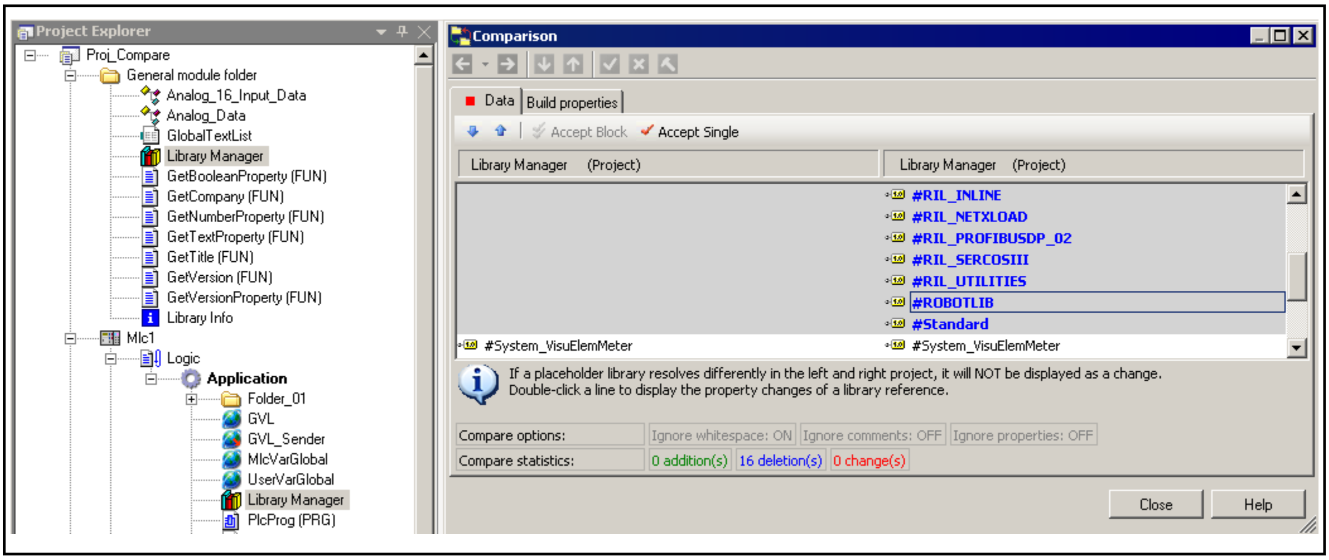
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

Comparison: Library Manager

When comparing Library Managers, the content is listed in tabular form. Acceptance is possible line-by-line.



If a placeholder library resolves differently in the left and right project, it will NOT be displayed as a change.
Double-click a line to display the property changes of a library reference.

Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 16 deletion(s) | 0 change(s)

Fig.3-106: Comparison: Library Manager, here ML_Robot added into the project

The next/previous difference is selected with ⬇ or ⬆.

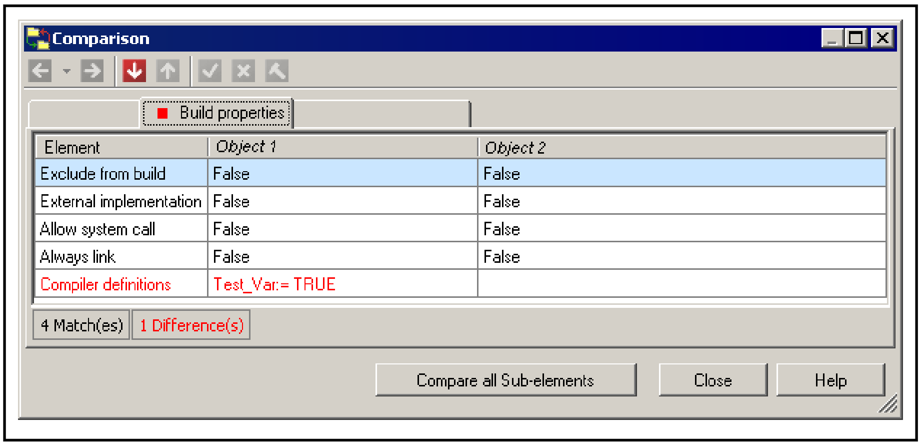
Accept Block can be used to accept the coplete highlighted text block.

Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

Fig.3-107: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

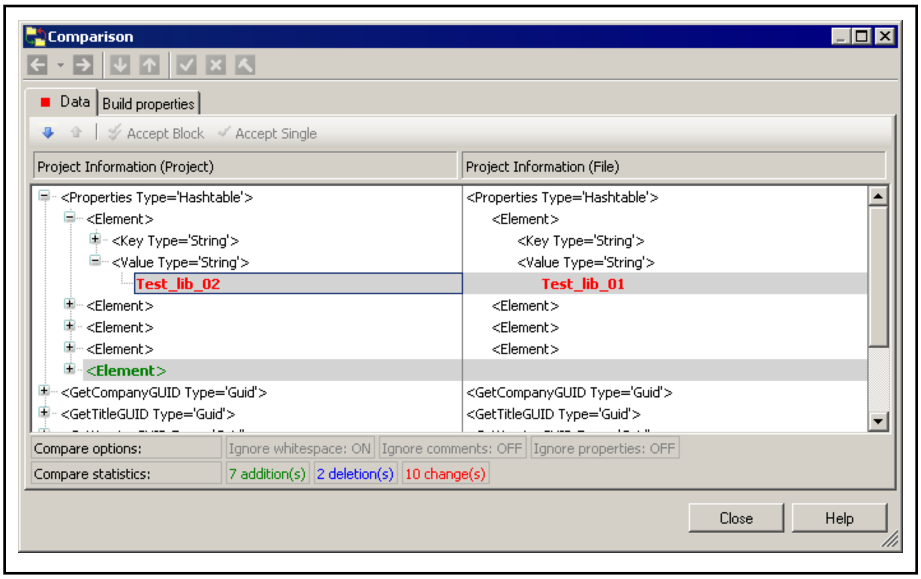
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

Comparison: Library Info

When comparing "library info", the content display is XML based. Modification (merging) is not possible.



Project Information (Project)	Project Information (File)
<Properties Type='Hashtable'>	<Properties Type='Hashtable'>
<Element>	<Element>
<Key Type='String'>	<Key Type='String'>
<Value Type='String'>	<Value Type='String'>
Test_lib_02	Test_lib_01
<Element>	<Element>
<Element>	<Element>
<Element>	<Element>
<Element>	
<GetCompanyGUID Type='Guid'>	<GetCompanyGUID Type='Guid'>
<GetTitleGUID Type='Guid'>	<GetTitleGUID Type='Guid'>

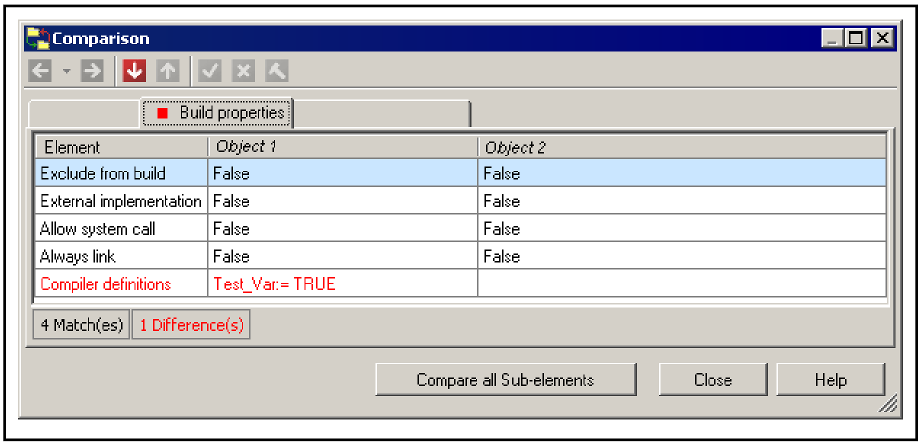
Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 7 addition(s) | 2 deletion(s) | 10 change(s)

Fig.3-108: Comparison: Library Info; library enabled with version number 10.06.0.1

The terms "Project Information" and "Library Info" designate the same element.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

Fig.3-109: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

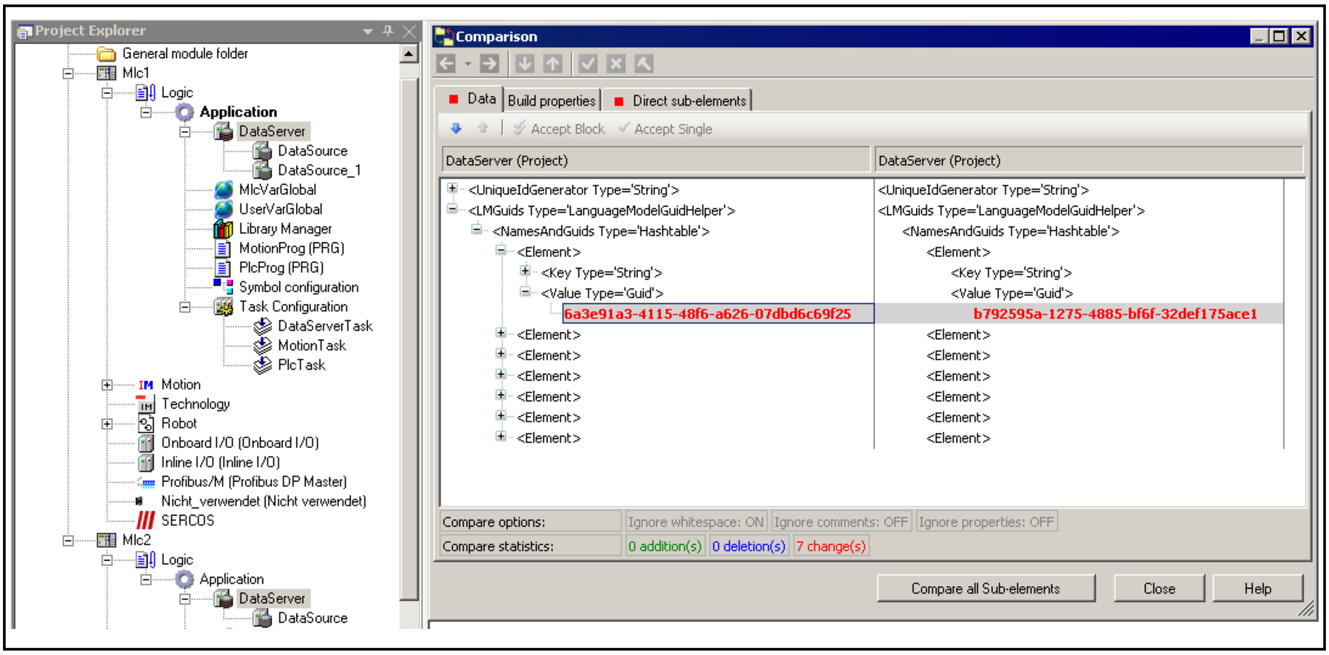
If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

Comparison: Data Server

When comparing data servers, the content display is XML based. Modification (merging) is not possible.



DataServer (Project)	DataServer (Project)
<UniqueIdGenerator Type='String'>	<UniqueIdGenerator Type='String'>
<LMGuids Type='LanguageModelGuidHelper'>	<LMGuids Type='LanguageModelGuidHelper'>
<NamesAndGuids Type='Hashtable'>	<NamesAndGuids Type='Hashtable'>
<Element>	<Element>
<Key Type='String'>	<Key Type='String'>
<Value Type='Guid'>	<Value Type='Guid'>
6a3e91a3-4115-48f6-a626-07dbd6c69f25	b792595a-1275-4885-bf6f-32def175ace1
<Element>	<Element>
<Element>	<Element>
<Element>	<Element>
<Element>	<Element>
<Element>	<Element>
<Element>	<Element>

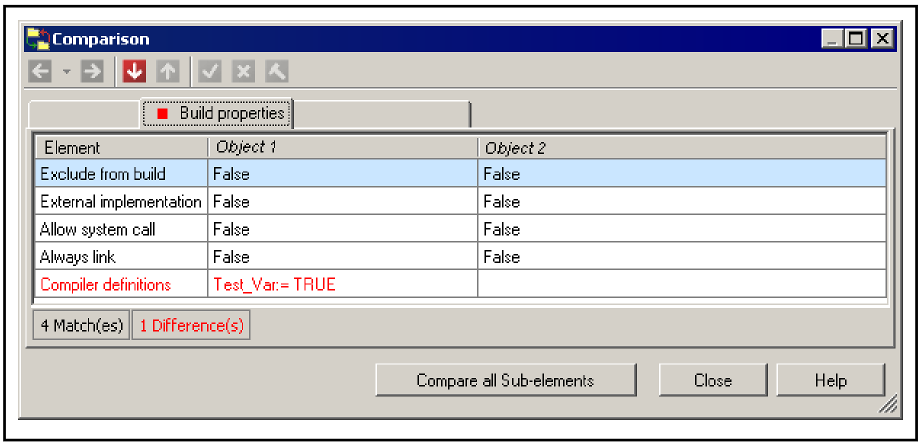
Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 0 deletion(s) | 7 change(s)

Fig.3-110: Comparison: Data server

The next/previous difference is selected with ⬇ or ⬆.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

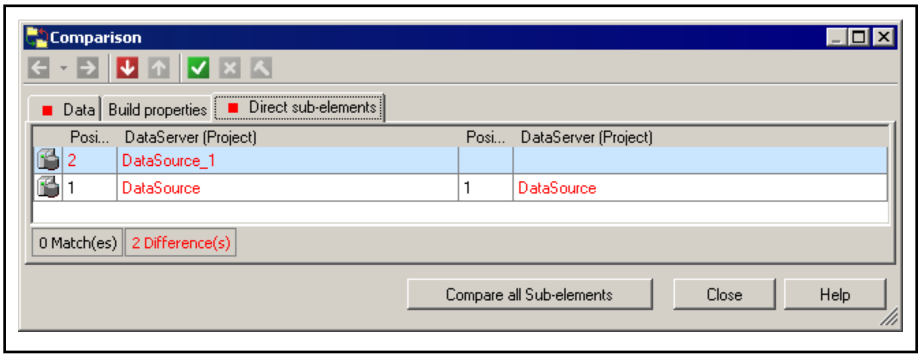
Fig.3-111: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.



Posi...	DataServer (Project)	Posi...	DataServer (Project)
2	DataSource_1		
1	DataSource	1	DataSource

0 Match(es) 2 Difference(s)

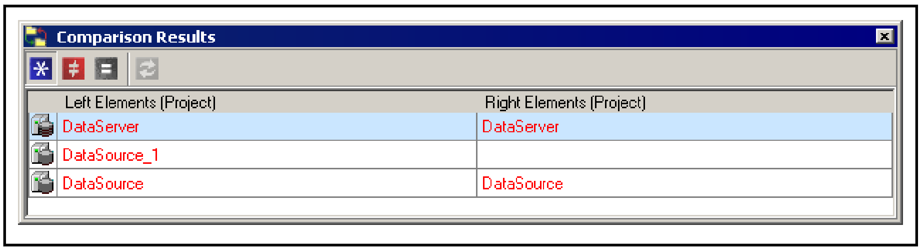
Fig.3-112: Comparison: Data server, Direct subelements

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

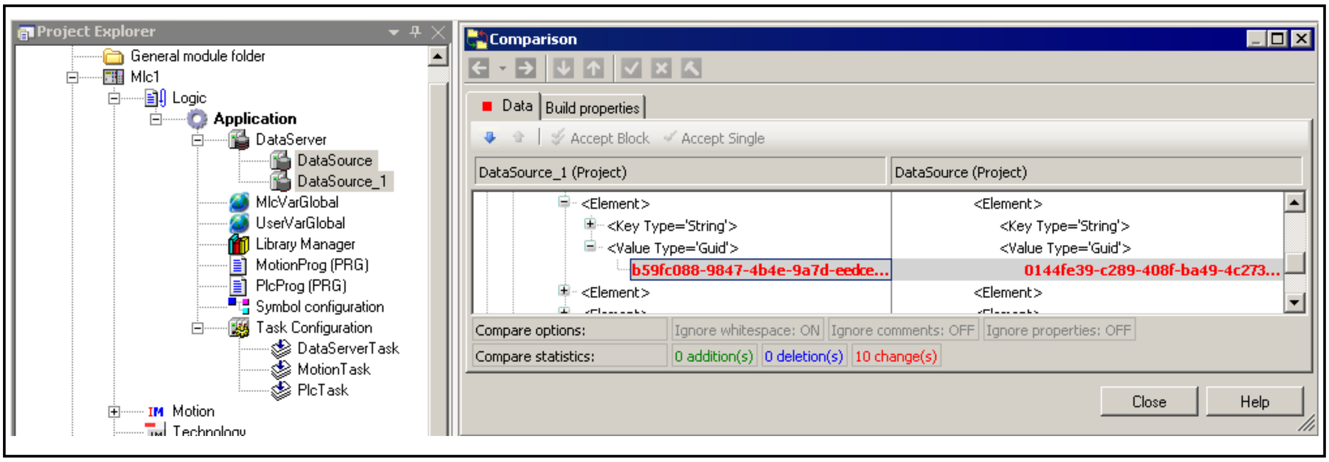


Left Elements (Project)	Right Elements (Project)
DataServer	DataServer
DataSource_1	
DataSource	DataSource

Fig.3-113: Comparison results: Data server

Comparison: Data Source (Data Server)

When comparing data sources, the content display is XML based. Modification (merging) is not possible.



DataSource_1 (Project)	DataSource (Project)
<Element>	<Element>
<Key Type='String'>	<Key Type='String'>
<Value Type='Guid'>	<Value Type='Guid'>
b59fc088-9847-4b4e-9a7d-eedce...	0144fe39-c289-408f-ba49-4c273...
<Element>	<Element>

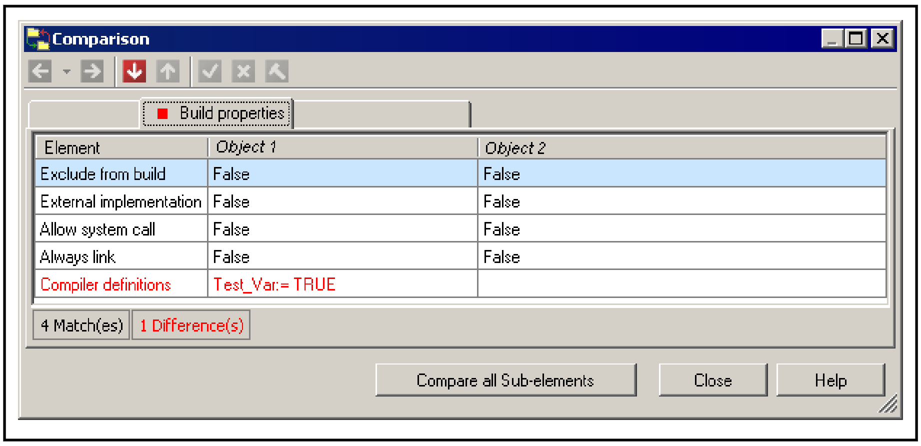
Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 10 change(s)

Fig.3-114: Comparison: DataSources

The next/previous difference is selected with ↓ or ↑.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-115: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

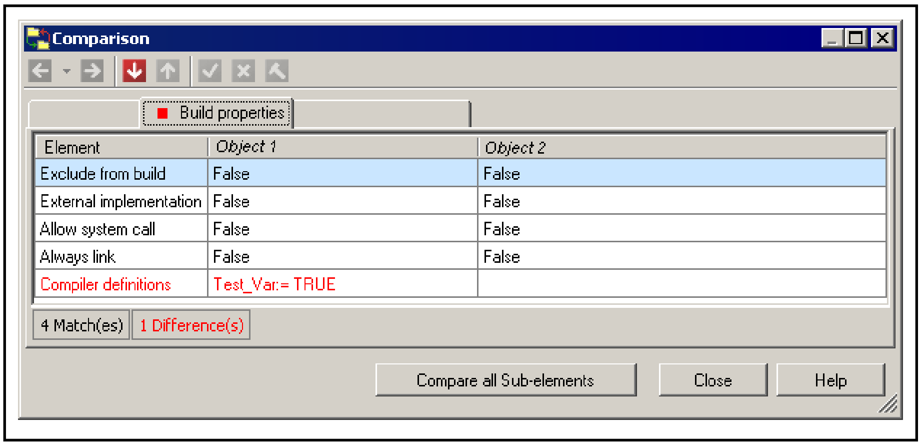
Comparison: Symbol Configuration

When comparing symbol configurations, the content display is XML based. Modification (merging) is not possible.

Comparison: Visualization Manager

When comparing visualization managers, the content display is XML based. Modification (merging) is not possible.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

Fig.3-116: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

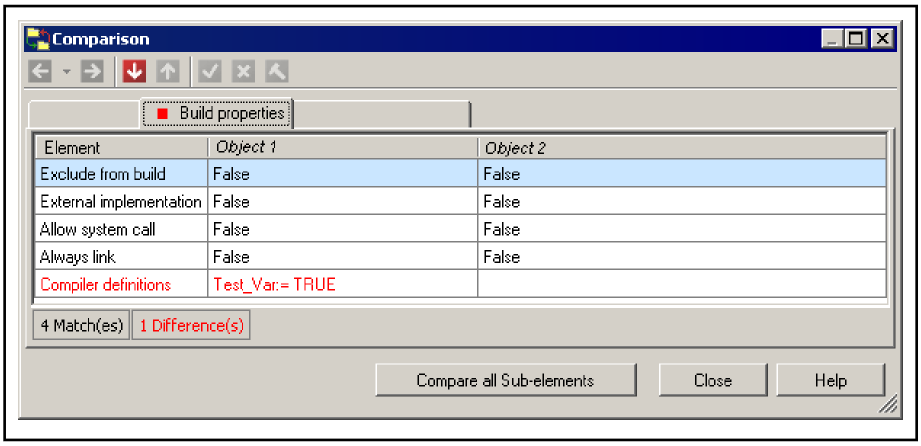
Select the hammer 🔨 to merge.

Select the cross ✖ to prevent (undo) merging.

Comparison: Visualization

When comparing visualizations, the content display is XML based. Modification (merging) is not possible.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

Fig.3-117: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

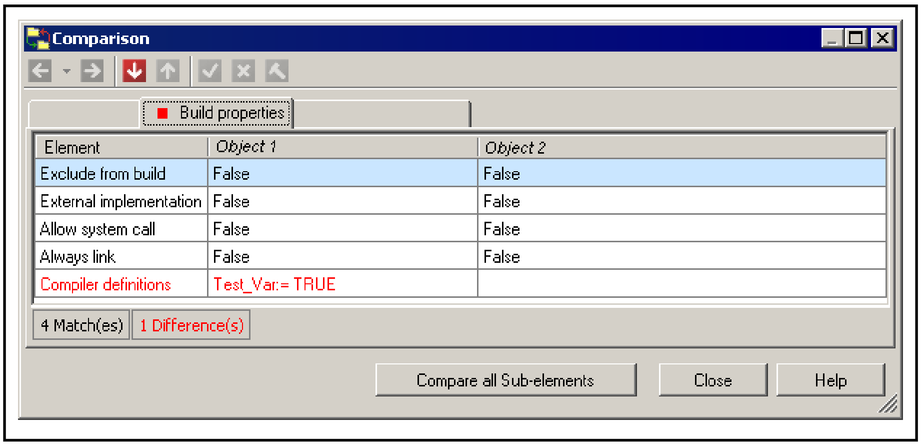
Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

Comparison: Text List

When comparing text lists, the content display is XML based. Modification (merging) is not possible.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-118: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

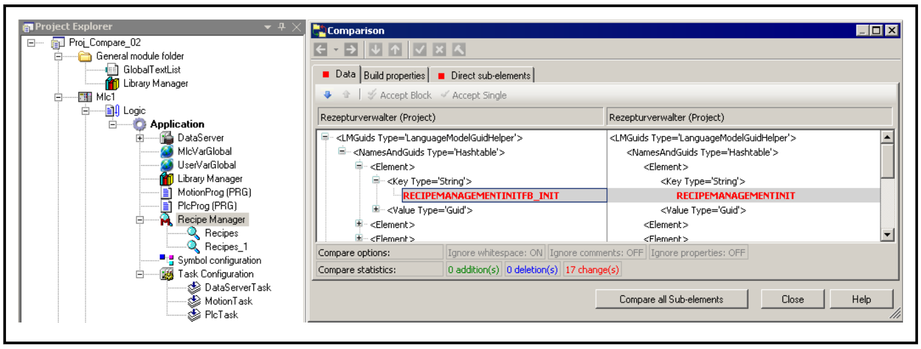
Select the cross ☒ to prevent (undo) merging.

Comparison: Image Pool

When comparing image pools, the content display is XML based. Modification (merging) is not possible.

Comparison: Recipe Manager

When comparing recipe managers, the content display is XML based. Modification (merging) is not possible.



Rezepturverwalter (Project)	Rezepturverwalter (Project)
<LMGuids Type='LanguageModelGuidHelper'>	<LMGuids Type='LanguageModelGuidHelper'>
<NamesAndGuids Type='Hashtable'>	<NamesAndGuids Type='Hashtable'>
<Element>	<Element>
<Key Type='String'>	<Key Type='String'>
RECIPEMANAGEMENTINITFB_INIT	RECIPEMANAGEMENTINIT
<Value Type='Guid'>	<Value Type='Guid'>
<Element>	<Element>
<Element>	<Element>

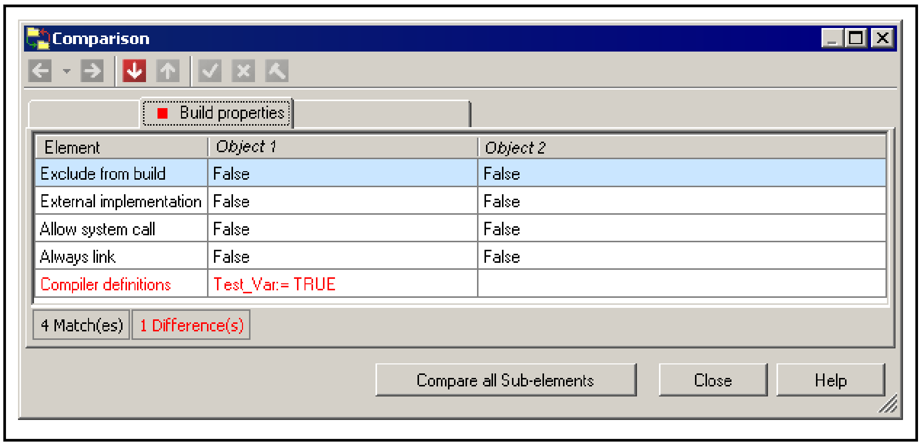
Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 17 change(s)

Fig.3-119: Comparison: Recipe Manager

The next/previous difference is selected with ↓ or ↑.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

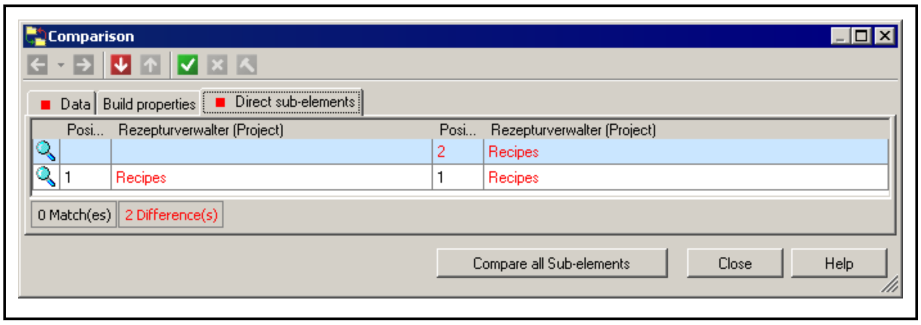
Fig.3-120: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer ⚒ to merge.

Select the cross ✗ to prevent (undo) merging.



	Posi...	Rezepturverwalter (Project)	Posi...	Rezepturverwalter (Project)
🔍			2	Recipes
🔍	1	Recipes	1	Recipes

0 Match(es) 2 Difference(s)

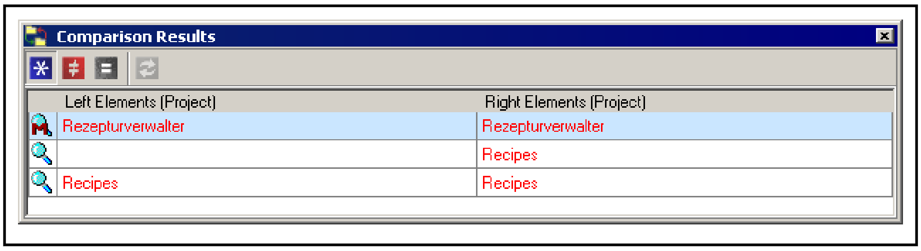
Fig.3-121: Comparison: Recipe manager, Direct subelements

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.



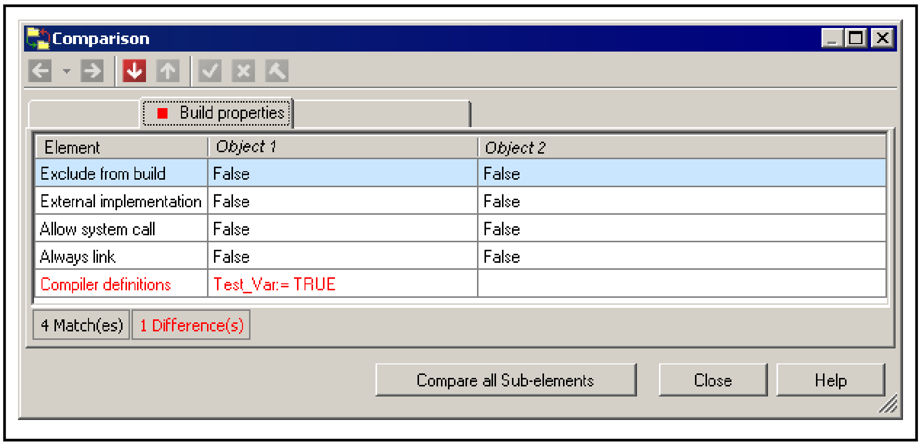
Left Elements (Project)	Right Elements (Project)
Rezepturverwalter	Rezepturverwalter
	Recipes
Recipes	Recipes

Fig.3-122: Comparison results: Recipe Manager

Comparison: Recipe Definition

When comparing recipe managers, the content display is XML based. Modification (merging) is not possible.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

Fig.3-123: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

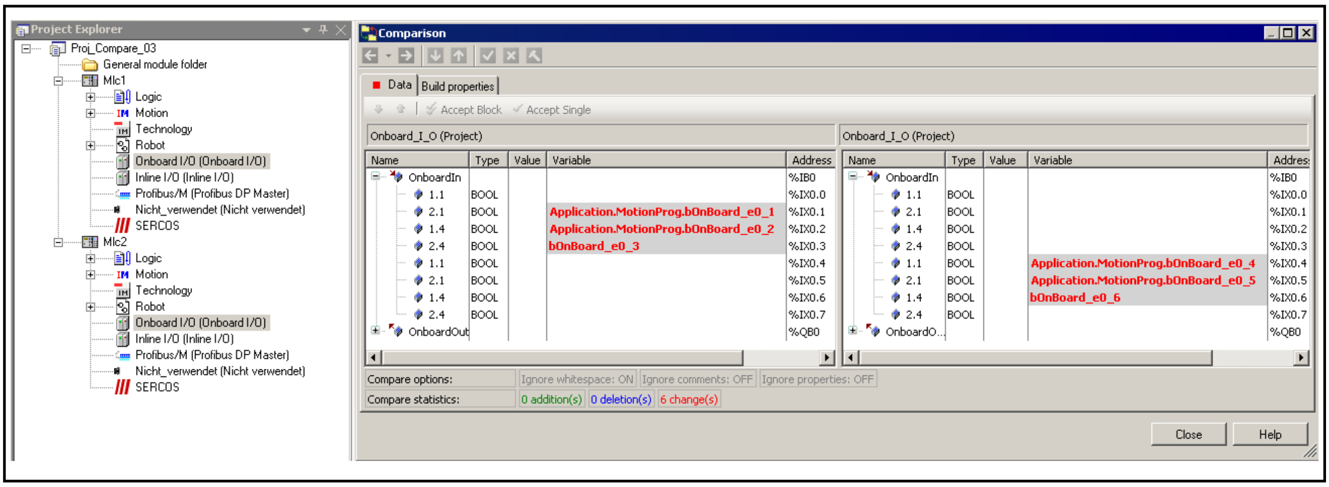
Select the cross ☒ to prevent (undo) merging.

Comparison: Devices (Onboard I/O, Inline I/O, Field Busses, …)

The comparison applies to the devices that are associated with the respective control (onboard I/O, Inline I/O, field busses). Since these differ from control type to control type, the following comparison results have to be seen as examples.

In principle, it is tried to provide a familiar environment at comparison with regard to the input and to enable merging based on the comparison.

Onboard I/O:



Project Explorer

- Proj_Compare_03
  - General module folder
  - Mlc1
    - Logic
    - Motion
    - Technology
    - Robot
    - Onboard I/O (Onboard I/O)
    - Inline I/O (Inline I/O)
    - Profibus/M (Profibus DP Master)
    - Nicht_verwendet (Nicht verwendet)
    - SERCOS
  - Mlc2
    - Logic
    - Motion
    - Technology
    - Robot
    - Onboard I/O (Onboard I/O)
    - Inline I/O (Inline I/O)
    - Profibus/M (Profibus DP Master)
    - Nicht_verwendet (Nicht verwendet)
    - SERCOS

Comparison

Onboard_I_O (Project)

Name	Type	Value	Variable	Address
OnboardIn				%IB0
1.1	BOOL			%IX0.0
2.1	BOOL		Application.MotionProg.bOnBoard_e0_1	%IX0.1
1.4	BOOL		Application.MotionProg.bOnBoard_e0_2	%IX0.2
2.4	BOOL		bOnBoard_e0_3	%IX0.3
1.1	BOOL			%IX0.4
2.1	BOOL			%IX0.5
1.4	BOOL			%IX0.6
2.4	BOOL			%IX0.7
OnboardOut				%QB0

Onboard_I_O (Project)

Name	Type	Value	Variable	Addres:
OnboardIn				%IB0
1.1	BOOL			%IX0.0
2.1	BOOL			%IX0.1
1.4	BOOL			%IX0.2
2.4	BOOL			%IX0.3
1.1	BOOL		Application.MotionProg.bOnBoard_e0_4	%IX0.4
2.1	BOOL		Application.MotionProg.bOnBoard_e0_5	%IX0.5
1.4	BOOL		bOnBoard_e0_6	%IX0.6
2.4	BOOL			%IX0.7
OnboardO...				%QB0

Compare options: Ignore whitespace: ON Ignore comments: OFF Ignore properties: OFF

Compare statistics: 0 addition(s) 0 deletion(s) 6 change(s)

Fig.3-124: Comparison: Onboard I/O:

The next/previous difference is selected with ⬇ or ⬆.

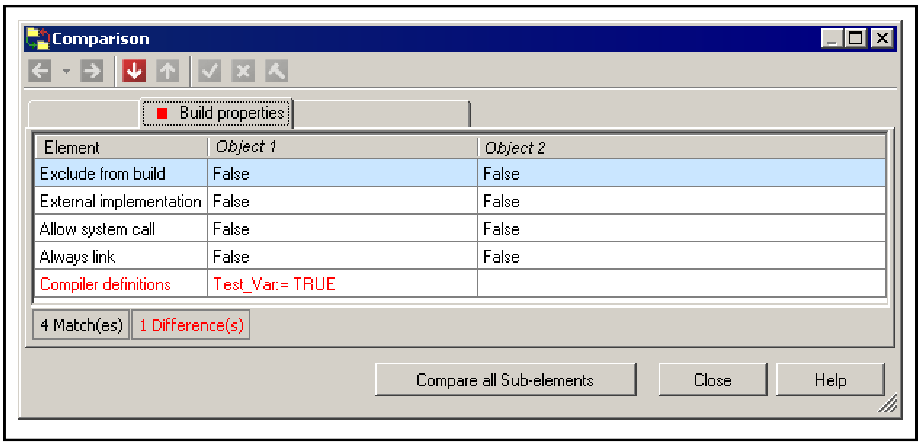
✓ Accept Block can be used to accept the coplete highlighted text block.

✓ Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

The next tab shows the build properties.



Comparison

Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-125: Comparison: Build properties object 1/object 2, one difference identified and marked in red

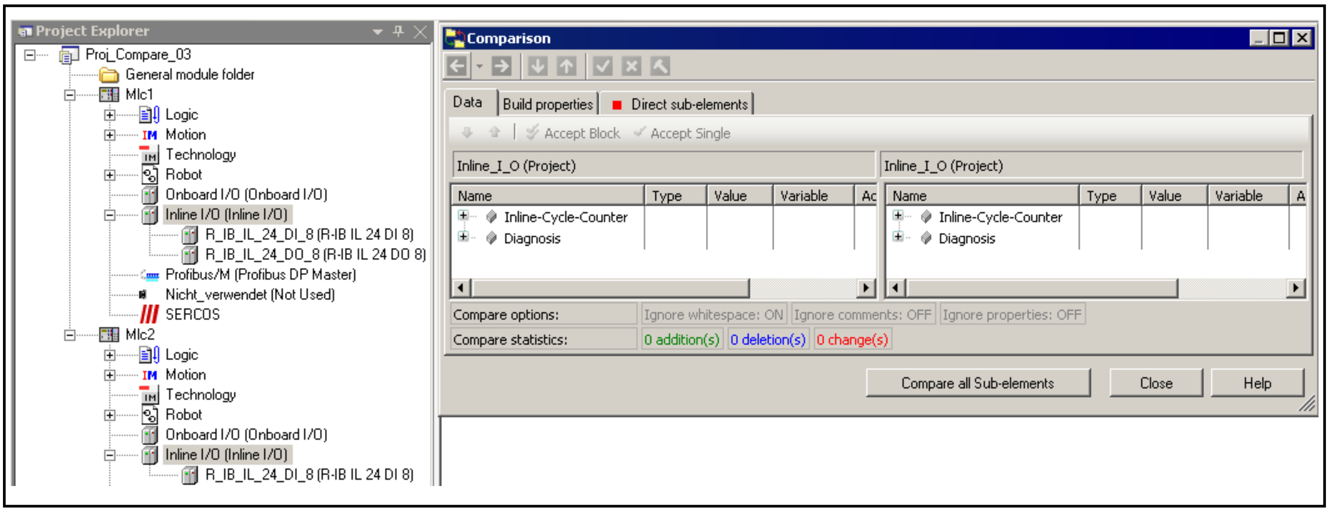
If the assignment in the columns of the two objects differs, ⬇ can be used to select the next difference (⬆ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.

Inline I/O objects: The comparison shows the Inline cycle counters and diagnostic data tabs belonging to the Inline I/O objects. Merging is not intended.

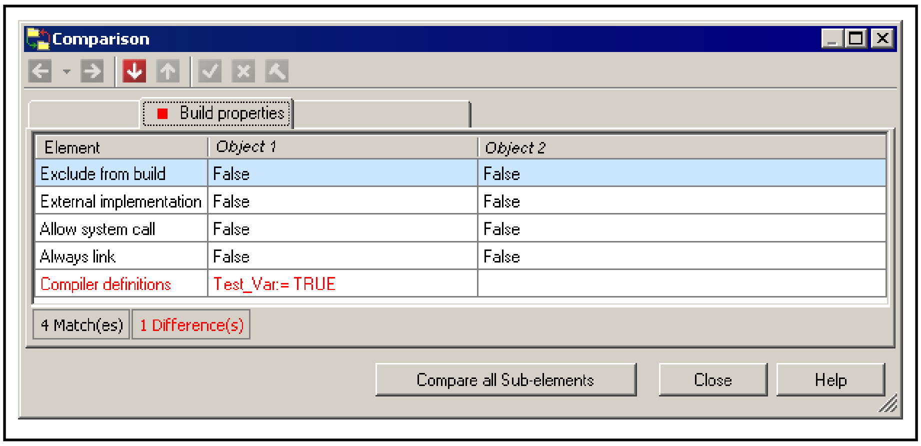


Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 0 deletion(s) | 0 change(s)

Fig.3-126: Comparison: Inline I/O: Inline cycle counters and diagnostic data tabs

The next tab shows the build properties.



Element	*Object 1*	*Object 2*
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) | 1 Difference(s)

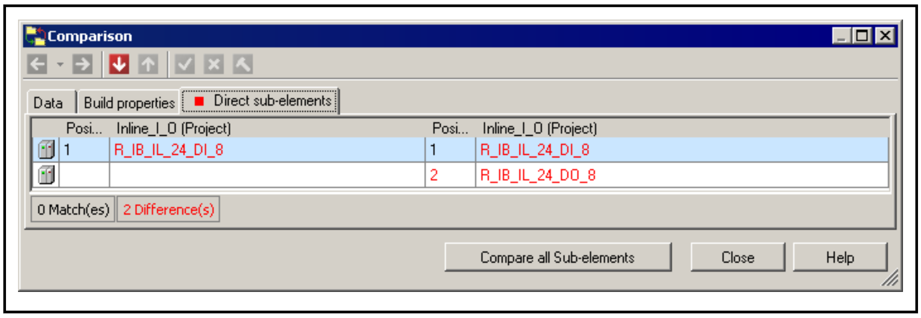
Fig.3-127: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✗ to prevent (undo) merging.



	Posi...	Inline_I_O (Project)	Posi...	Inline_I_O (Project)
	1	R_IB_IL_24_DI_8	1	R_IB_IL_24_DI_8
			2	R_IB_IL_24_DO_8

0 Match(es) | 2 Difference(s)

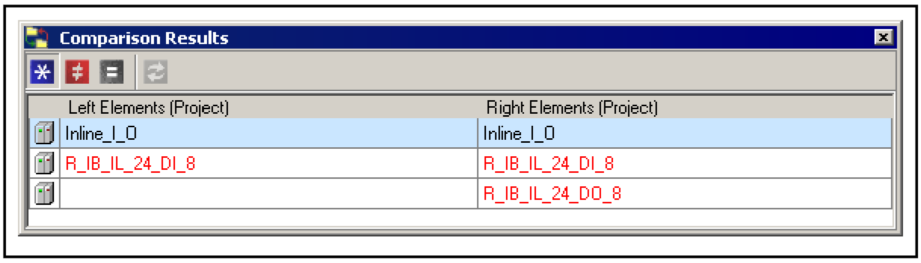
Fig.3-128: Comparison: Inline I/O: Direct subelements

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ☒ to prevent (undo) merging.

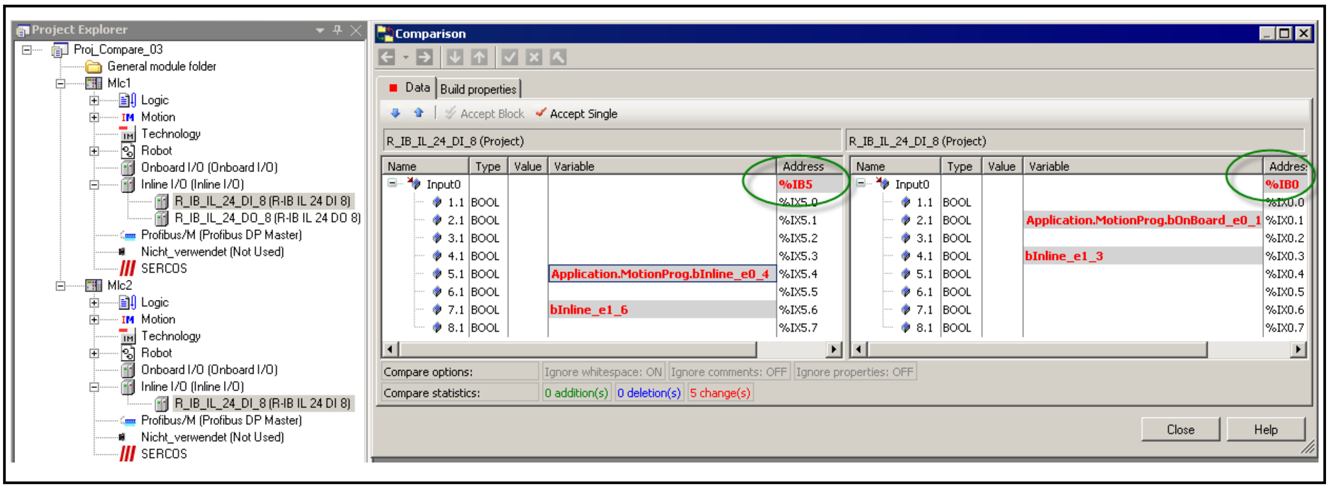


	Left Elements (Project)	Right Elements (Project)
	Inline_I_O	Inline_I_O
	R_IB_IL_24_DI_8	R_IB_IL_24_DI_8
		R_IB_IL_24_DO_8

Fig.3-129: Comparison results: Inline I/O: All subelements

Inline I/O modules

If the Inline I/O modules under "Direct subelements" and "Comparison results" have identical names and are marked in red, this is an indication of different assignments:



- Proj_Compare_03
  - General module folder
  - Mlc1
    - Logic
    - Motion
    - Technology
    - Robot
    - Onboard I/O (Onboard I/O)
    - Inline I/O (Inline I/O)
      - R_IB_IL_24_DI_8 (R-IB IL 24 DI 8)
      - R_IB_IL_24_DO_8 (R-IB IL 24 DO 8)
    - Profibus/M (Profibus DP Master)
    - Nicht_verwendet (Not Used)
    - SERCOS
  - Mlc2
    - Logic
    - Motion
    - Technology
    - Robot
    - Onboard I/O (Onboard I/O)
    - Inline I/O (Inline I/O)
      - R_IB_IL_24_DI_8 (R-IB IL 24 DI 8)
    - Profibus/M (Profibus DP Master)
    - Nicht_verwendet (Not Used)
    - SERCOS

R_IB_IL_24_DI_8 (Project)

Name	Type	Value	Variable	Address
Input0				%IB5
1.1	BOOL			%IX5.0
2.1	BOOL			%IX5.1
3.1	BOOL			%IX5.2
4.1	BOOL			%IX5.3
5.1	BOOL		Application.MotionProg.bInline_e0_4	%IX5.4
6.1	BOOL			%IX5.5
7.1	BOOL		bInline_e1_6	%IX5.6
8.1	BOOL			%IX5.7

R_IB_IL_24_DI_8 (Project)

Name	Type	Value	Variable	Addres:
Input0				%IB0
1.1	BOOL			%IX0.0
2.1	BOOL		Application.MotionProg.bOnBoard_e0_1	%IX0.1
3.1	BOOL			%IX0.2
4.1	BOOL		bInline_e1_3	%IX0.3
5.1	BOOL			%IX0.4
6.1	BOOL			%IX0.5
7.1	BOOL			%IX0.6
8.1	BOOL			%IX0.7

Compare options: Ignore whitespace: ON | Ignore comments: OFF | Ignore properties: OFF

Compare statistics: 0 addition(s) | 0 deletion(s) | 5 change(s)

Fig.3-130: Comparison: Inline I/O modules

The different addresses with identical location of the modules in the Project Explorer indicate an address shift.

The next/previous difference is selected with ⬇ or ⬆.

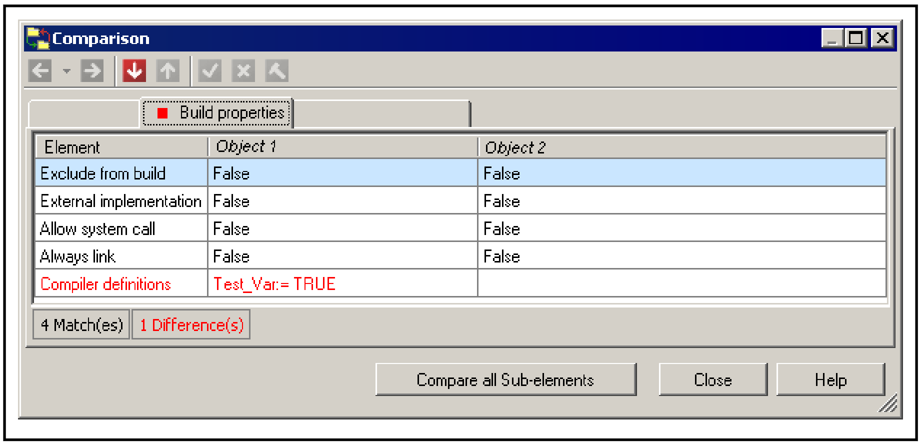
✓ Accept Block can be used to accept the coplete highlighted text block.

✓ Accept Single can be used to accept text line by line. In this case, only the possible result of the merge is displayed first. Clicking the button again resets the display.

Select the hammer 🔨 to merge.

Select the cross ✖ to prevent (undo) merging.

The next tab shows the build properties.



Element	Object 1	Object 2
Exclude from build	False	False
External implementation	False	False
Allow system call	False	False
Always link	False	False
Compiler definitions	Test_Var:= TRUE	

4 Match(es) 1 Difference(s)

Fig.3-131: Comparison: Build properties object 1/object 2, one difference identified and marked in red

If the assignment in the columns of the two objects differs, ↓ can be used to select the next difference (↑ for the previous difference).

If the checkmark ✓ appears, it can be used to prepare the acceptance (merging):

Select the hammer 🔨 to merge.

Select the cross ✖ to prevent (undo) merging.

3.6.2 Simulation

The command is available in the context menu of the relevant control and is used to switch on and off the simulation mode of the programming system.

In the simulation mode, the active application can be started and debugged on a "simulated target device". As such a simulated device is always integrated into the programming system, no real target device is needed to test the online behavior of an application. If the command is called from the context menu if an application is selected in the Project Explorer, login is carried out with this selected application even if it is not set as "active application".

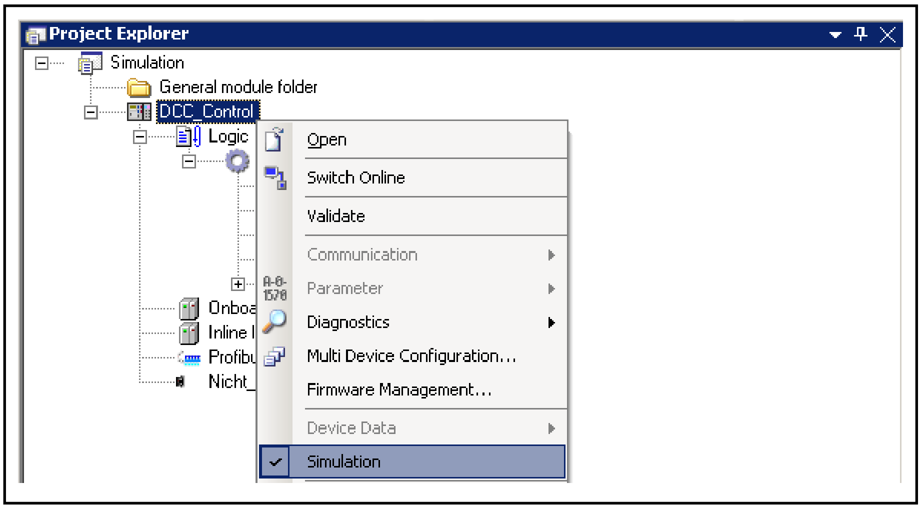
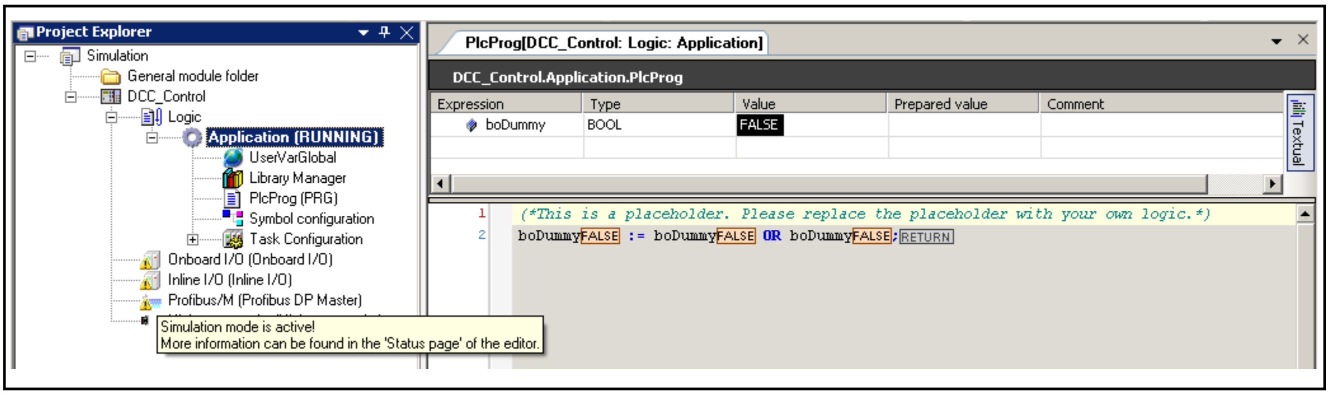


Fig.3-132: Context menu, simulation switch-on

No communication settings need to be entered for the simulated device.

Following a successful login, the corresponding online commands can be used to test the application.



Expression	Type	Value	Prepared value	Comment
boDummy	BOOL	FALSE		

```
(*This is a placeholder. Please replace the placeholder with your own logic.*)
boDummyFALSE := boDummyFALSE OR boDummyFALSE;RETURN
```

Simulation mode is active!
More information can be found in the 'Status page' of the editor.

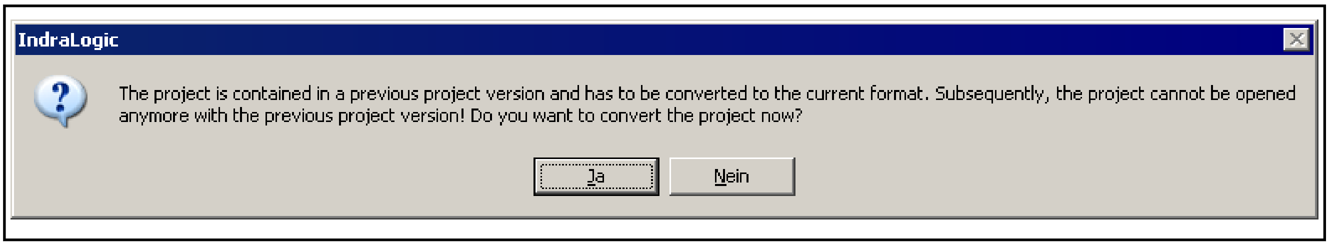
Fig.3-133: Control in the simulation mode

To switch off the simulation mode, first log off from the control and then call the"Simulation" command again. The tick in front of the command disappears, the entry of the target device in the Project Explorer is no longer shown in italics and it can be thus logged into the real target device.

3.6.3 IndraLogic Project Versions

From IndraWorks 12VRS, a PLC project contains a project version. Using this version code, compatible PLC projects can be generated for all known releases (V02, V04, V06, ...) of an IndraWorks version.

If an older project version is identified while a project is opened, the following dialog appears:

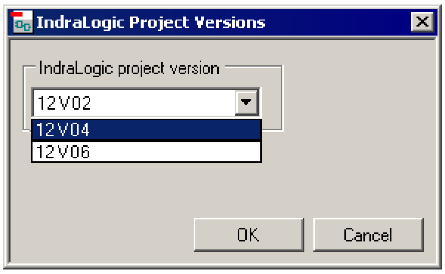


The project is contained in a previous project version and has to be converted to the current format. Subsequently, the project cannot be opened anymore with the previous project version! Do you want to convert the project now?

Fig.3-134: Dialog: Project in old project version detected

Using the dialog, an accidental conversion of the PLC project can be prevented.

If the project is available in a 12VRS release version, it can be edited in all later IndraWorks releases of this version without conversion and saved in a release-compatible form. The project can be explicitly saved in a later release version. For this purpose, the "IndraLogic project version" command is used which is by default executed under **Tools** ▸ **IndraLogic project version** :


IndraLogic Project Versions

IndraLogic project version

12V02

Fig.3-135: Dialog: IndraLogic project version

Using the dialog, the project can be converted in a later release version. Conversion into an older format is not possible as here, data would be lost.

A newly created project is automatically always generated in the latest release version. As long as no device has been inserted into this project, the project can be converted to an older project release using the <IndraLogic Project Version> command described above.

☞ By using new features of the current Indraworks version that are not available in the project version, the project version can be updated automatically.

These automatic changes can be corrected using the "Undo" command.

3.6.4 Library Repository - Commands

Library Repository - Commands, General

"Library management" provides access to the "Library Repository ..." command for opening the dialog with the same name.

By default, the command is located in the "Tools" menu. If required, the menu structure can be changed using the dialog in **IndraWorks** ▸ **Tools** ▸ **Customize** ▸ **Commands** ▸ **Library Manager**.

Further information

- Using libraries, see page 83.

Library Repository...

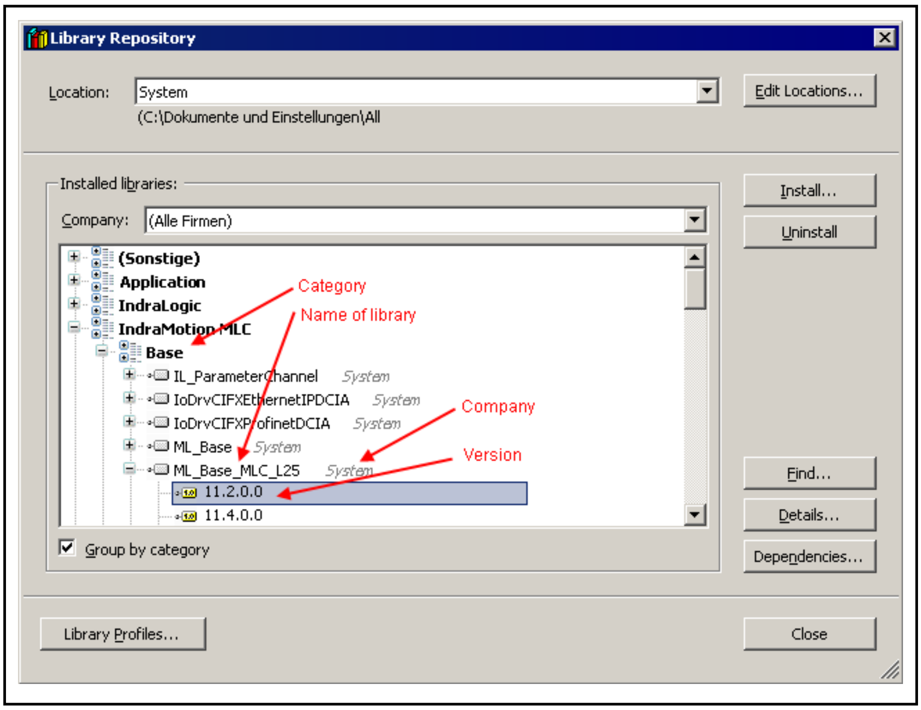
Icon:

This command opens the **Library Repository** dialog, which can also be opened from the Library manager editor window, page 367,.

A library repository is a storage location for libraries that were installed on the local system in order to be included in IndraLogic projects.

☞ A library project "*.library" located in a library repository cannot be opened from the repository for editing or for viewing in the programming system.

To achieve this, the library has to be added to the library manager.


Library Repository

Location: System
(C:\Dokumente und Einstellungen\All

Installed libraries:
Company: (Alle Firmen)

```
(Sonstige)
Application
IndraLogic
IndraMotion MLC
  Base
    IL_ParameterChannel        System
    IoDrvCIFXEthernetIPDCIA    System
    IoDrvCIFXProfinetDCIA      System
    ML_Base                    System
    ML_Base_MLC_L25            System
      11.2.0.0
      11.4.0.0
```

☑ Group by category

Fig.3-136: Main dialog "Library Repository"

The dialog displays the currently defined storage locations (repositories) and the libraries installed there.

This dialog can be used to add new repositories, to modify or remove them and libraries can be installed or uninstalled here.

A list of all of the currently **installed libraries** for the currently selected **storage location**, a directory on the local system in which the library files are located, and for the **company** that is currently set, is displayed. This list displays the respective name (title), version and company for a library as entered in the library information in the library file.

If the **Group according to category** option is selected, the list is grouped according to library categories. The category names appear as nodes that can be expanded or reduced via click. The respective libraries appear below the categories and the respective versions appear below the libraries. If grouping is not performed by category, the libraries are listed alphabetically according to company.

The categories are defined by external description files (*.libcat.xml).

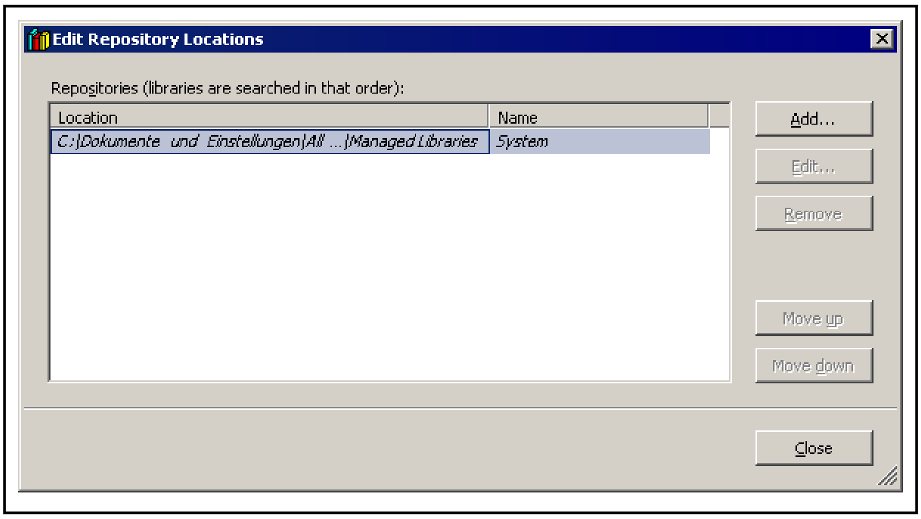
The individual buttons are described in the following:

- **Edit**: Library Repositories, page 187
- **Install**, **Uninstall**: Installing and uninstalling libraries, page 188
- **Details**: Further information on libraries, page 189,
- **Dependencies**: Display of the libraries used, page 189,
- **Library profiles**: Display of the library profiles, page 190, ordered according to placeholder name or compiler version.

Library repositories

One or more repositories can be used for managing libraries. All of the currently defined repositories are in the selection list next to **Storage location**: By default the "System" repository is always created during the installation of IndraLogic.

To change the path or name of a repository, use the **Edit...** buttons to open the **Edit repositories** dialog:



Repositories (libraries are searched in that order):

Location	Name
C:\Dokumente und Einstellungen\All ...\Managed Libraries	System

Fig.3-137: "Edit Repository" dialog

The currently defined storage locations are listed in the repository window. Later, a search for a library is made in the sequence listed here. To change the sequence, use the **Move up** and **Move down** buttons to move the currently selected entry.

Note that the storage location **<All companies>** lists the libraries in all of the currently defined repositories. No installation or uninstallation can be performed in this display.

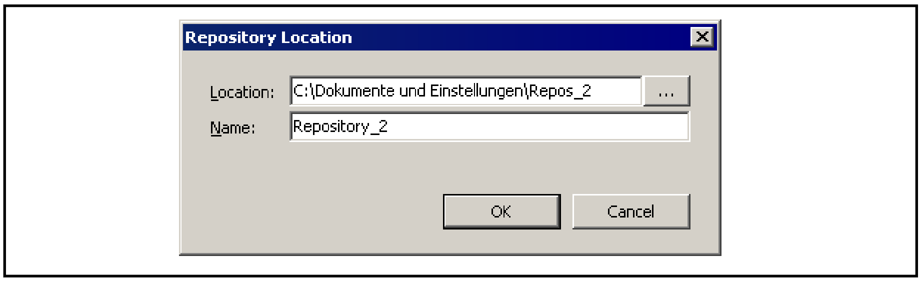
Creating a new repository

To create a new repository, use the **Add** button. The **Repository Location** dialog opens, where the path of the new repository has to be entered in the **Location** field. Use the ... button to search for a suitable directory or create one.

Attention: The selected directory **has to be empty**!

In the **Name** field enter a representative name for the repository, e.g. Libraries for system xy.

To edit an existing repository, select the entry in the Repositories dialog and use the **Edit** button to open the **Repository Locations** dialog, where in this case, the current information for location and name are already entered and can be modified.


Repository Location

Location: C:\Dokumente und Einstellungen\Repos_2

Name: Repository_2

Fig.3-138: Set storage location and name for repository dialog

☞

1. Only empty directories can be used for a new repository.
2. The "System" repository cannot be edited, which is indicated by the italic font used for this entry.

Deleting a repository

If a repository in the list is selected and <Remove> is clicked, a query appears asking if only the entry is to be removed from the list of repositories or if the directory with the library files is also to be deleted from the file system.

Installing and uninstalling libraries

Only one library that is installed on the local system in a library repository can be included in a project using the library manager.

A prerequisite for the installation is that the library file has to contain a title and version information in its library information, page 371,; information on category and company is optional.

To install a library, select the repository in the "Library Repository" dialog to which the library is to be added and click the <Install...> button.

The "Select Library" dialog opens, where it can be searched for the library file.

- By default, the filter is set for "libraries" that contain the appropriate library information and have the default extension *.library.
- The filter can also be set for "compiled libraries" that contain the appropriate library information and have the default extension *.compiled-library.
- The filter can also be set to "IndraLogic libraries (before 2.x)" to find libraries with old formats and with the extension *.lib or to
- "All files".

Select the desired library file and close the dialog. The library is added to the list of installed libraries in the library repository dialog.

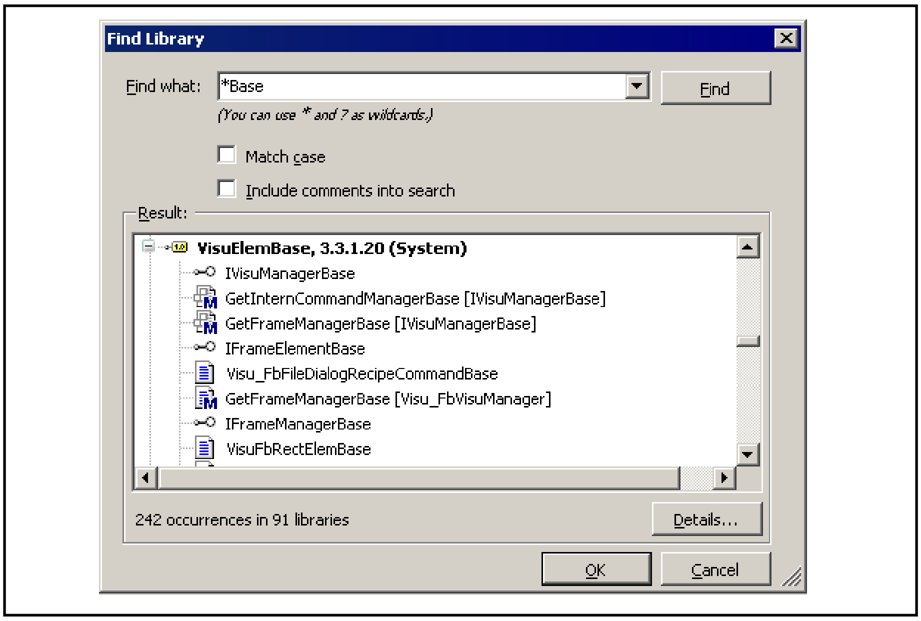
If you try to install a library that cannot be installed, since it does not have the required library information (title, version), the following error message appears:

The library entered is not a managed library. (Reason: No library information could be found.)

To uninstall a library, select it from the list of installed libraries in the library repository dialog and press <Uninstall>.

Search in library

Dialog to search libraries in the specified storage location. This search is used to find function blocks and the related libraries. Apart from the search string for the function block, the placeholders "*" and "?" can be used.



Find Library

Find what: *Base

(You can use * and ? as wildcards.)

□ Match case

□ Include comments into search

Result:

- VisuElemBase, 3.3.1.20 (System)
  - IVisuManagerBase
  - GetInternCommandManagerBase [IVisuManagerBase]
  - GetFrameManagerBase [IVisuManagerBase]
  - IFrameElementBase
  - Visu_FbFileDialogRecipeCommandBase
  - GetFrameManagerBase [Visu_FbVisuManager]
  - IFrameManagerBase
  - VisuFbRectElemBase

242 occurrences in 91 libraries

Fig.3-139: "Find library" dialog

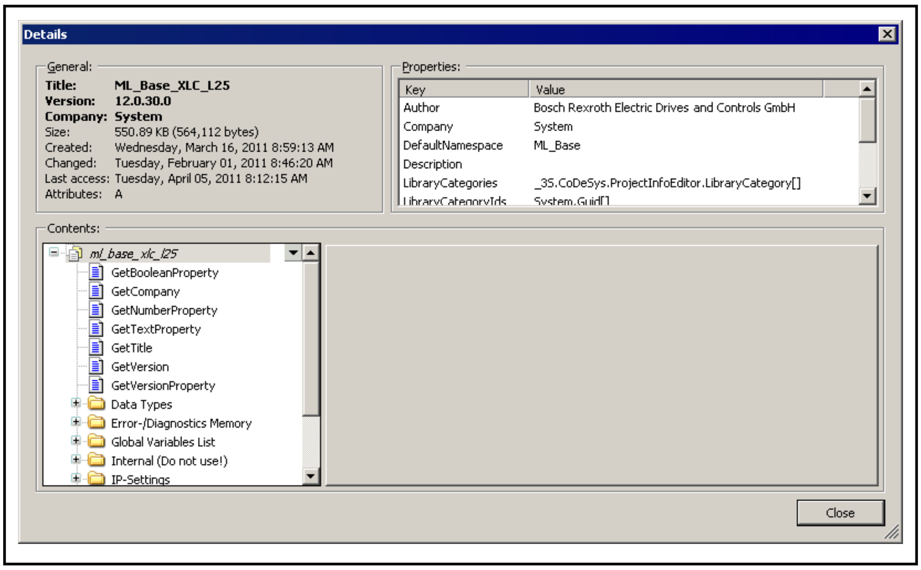
Pay attention to upper and lower cases: This search takes the case of the search string into account

Integrating comments into the search: Not only the name of the function block but also the comment is searched

Details...

When a library is selected from the list of installed libraries - ensure that the line selected contains the version - it button opens a dialog that displays the following details: Title, version, company, size, created (date), changed (date), edited (date of most recent access), properties.

This information origins from the library information, page 371, in the library file.



Details

General:

Title: **ML_Base_XLC_L25**
Version: **12.0.30.0**
Company: **System**
Size: 550.89 KB (564,112 bytes)
Created: Wednesday, March 16, 2011 8:59:13 AM
Changed: Tuesday, February 01, 2011 8:46:20 AM
Last access: Tuesday, April 05, 2011 8:12:15 AM
Attributes: A

Properties:

Key	Value
Author	Bosch Rexroth Electric Drives and Controls GmbH
Company	System
DefaultNamespace	ML_Base
Description	
LibraryCategories	_3S.CoDeSys.ProjectInfoEditor.LibraryCategory[]
LibraryCategoryIds	System.Guid[]

Contents:

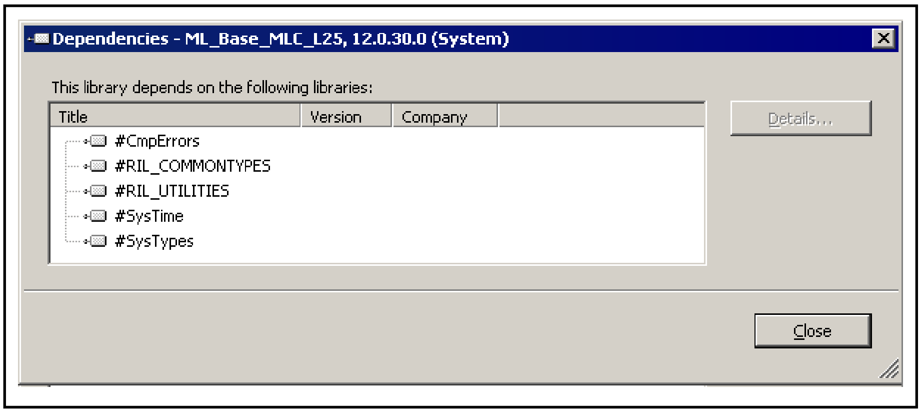
- *ml_base_xlc_l25*
  - GetBooleanProperty
  - GetCompany
  - GetNumberProperty
  - GetTextProperty
  - GetTitle
  - GetVersion
  - GetVersionProperty
  - Data Types
  - Error-/Diagnostics Memory
  - Global Variables List
  - Internal (Do not use!)
  - IP-Settings

Fig.3-140: "Details" dialog

Dependencies...

For the currently selected library, this button opens a dialog that displays the dependencies of other libraries, i.e. the integrated libraries are shown.

The title, version and company are displayed for each library.



Dependencies - ML_Base_MLC_L25, 12.0.30.0 (System)

This library depends on the following libraries:

Title	Version	Company
#CmpErrors		
#RIL_COMMONTYPES		
#RIL_UTILITIES		
#SysTime		
#SysTypes		

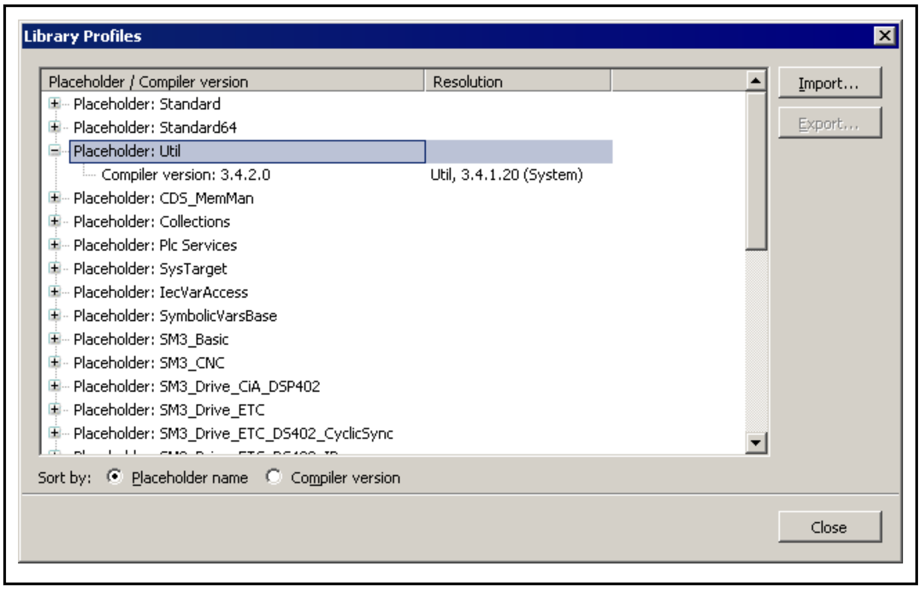
Fig.3-141: "Dependencies" dialog

Library profiles

For the current library repository, the placeholders/compiler versions are displayed.

The list can be ordered according to placeholders or compiler versions. The following two figures show the possibilities using an example.

The dialog simultaneously serves as export/import interface for library profiles.

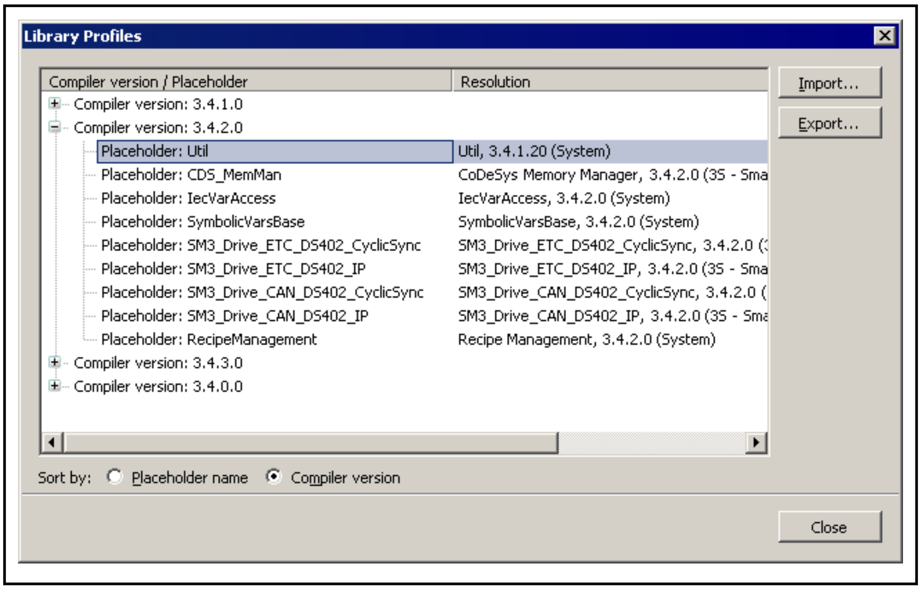


Library Profiles

Placeholder / Compiler version	Resolution
Placeholder: Standard	
Placeholder: Standard64	
Placeholder: Util	
Compiler version: 3.4.2.0	Util, 3.4.1.20 (System)
Placeholder: CDS_MemMan	
Placeholder: Collections	
Placeholder: Plc Services	
Placeholder: SysTarget	
Placeholder: IecVarAccess	
Placeholder: SymbolicVarsBase	
Placeholder: SM3_Basic	
Placeholder: SM3_CNC	
Placeholder: SM3_Drive_CiA_DSP402	
Placeholder: SM3_Drive_ETC	
Placeholder: SM3_Drive_ETC_DS402_CyclicSync	

Sort by: ● Placeholder name ○ Compiler version

Fig.3-142: "Library Profiles"dialog, ordered according to placeholder names



Library Profiles

Compiler version / Placeholder	Resolution
⊞ Compiler version: 3.4.1.0	
⊟ Compiler version: 3.4.2.0	
Placeholder: Util	Util, 3.4.1.20 (System)
Placeholder: CDS_MemMan	CoDeSys Memory Manager, 3.4.2.0 (3S - Sma
Placeholder: IecVarAccess	IecVarAccess, 3.4.2.0 (System)
Placeholder: SymbolicVarsBase	SymbolicVarsBase, 3.4.2.0 (System)
Placeholder: SM3_Drive_ETC_DS402_CyclicSync	SM3_Drive_ETC_DS402_CyclicSync, 3.4.2.0 (:
Placeholder: SM3_Drive_ETC_DS402_IP	SM3_Drive_ETC_DS402_IP, 3.4.2.0 (3S - Sma
Placeholder: SM3_Drive_CAN_DS402_CyclicSync	SM3_Drive_CAN_DS402_CyclicSync, 3.4.2.0 (
Placeholder: SM3_Drive_CAN_DS402_IP	SM3_Drive_CAN_DS402_IP, 3.4.2.0 (3S - Sma
Placeholder: RecipeManagement	Recipe Management, 3.4.2.0 (System)
⊞ Compiler version: 3.4.3.0	
⊞ Compiler version: 3.4.0.0	

Sort by: ○ Placeholder name ● Compiler version

Fig.3-143: "Library Profiles" dialog, ordered according to compiler versions used

3.6.5 Generating a Library in IndraWorks

The libraries of the IndraLogic PLC programming system contain function blocks, functions, data types, variable lists and constants provided by the control manufacturer. The libraries, page 83, are either directly installed during the installation of "IndraWorks Engineering" and are visible in the library manager, page 367, of the relevant application or they can be added from the library repository, page 185,.

This section describes how the user can create a library or compiled library from function blocks, functions, data types, etc. which is equal to the manufacturer's libraries.

For that purpose, the function blocks, functions, data types, etc. are to be arranged in the "General module" folder in a structured form.

In the context menu of the "General module" or in the main menu under "Tools", three menu items are available

- "Generate library…“
- "Generate compiled library…“
- "Generate and install compiled library…“

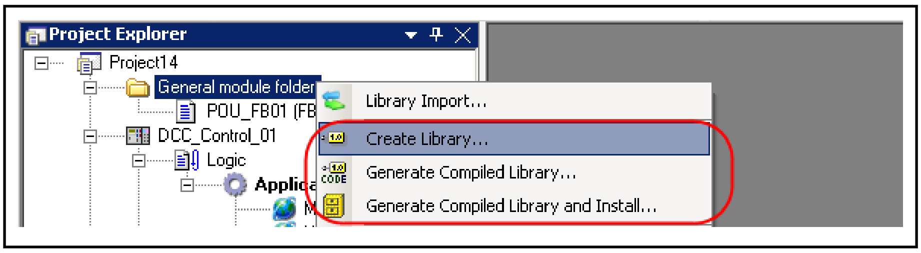


Fig.3-144: Menu items for generating a library or compiled library (context menu)

Menu Items

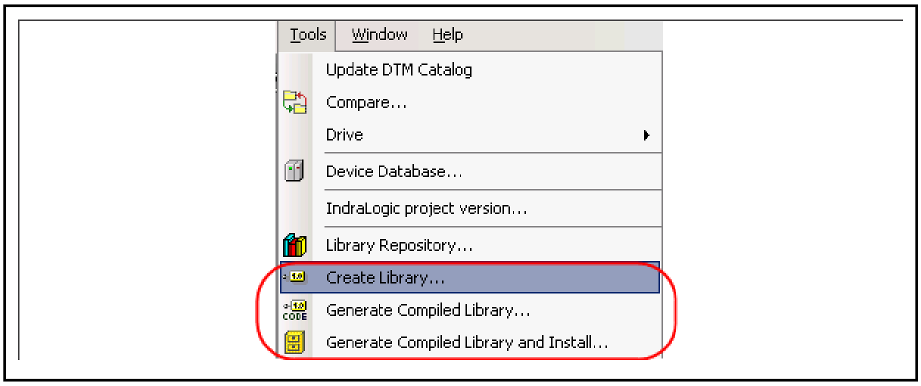
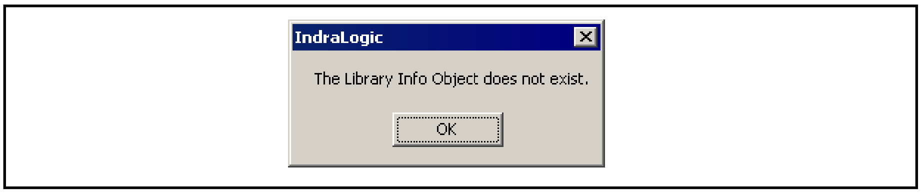


Fig.3-145: Menu items for generating a library or compiled library (main menu)

All menu items can only be used if the opened project contains a "Library info" object, page 371,. If no such object is available and the user selects one of these menu items, the following message is output:



The Library Info Object does not exist.

Fig.3-146: Error message, library info object not available (yet)

In this case, add such object by means of the context menu (or from the library):

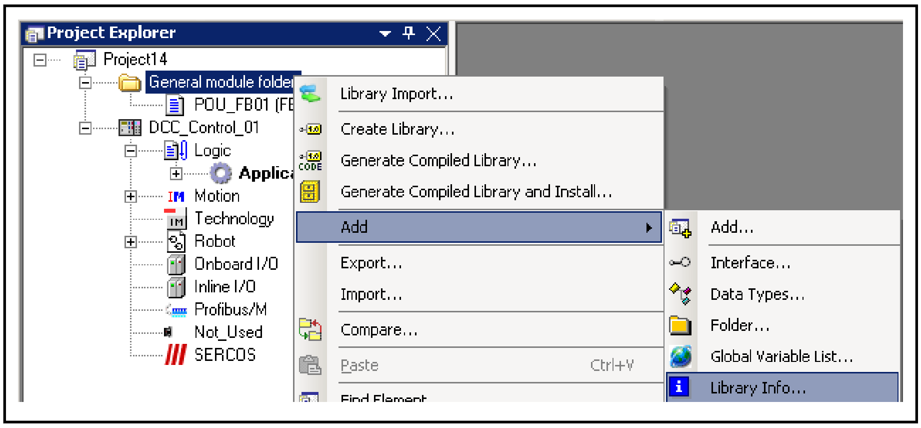
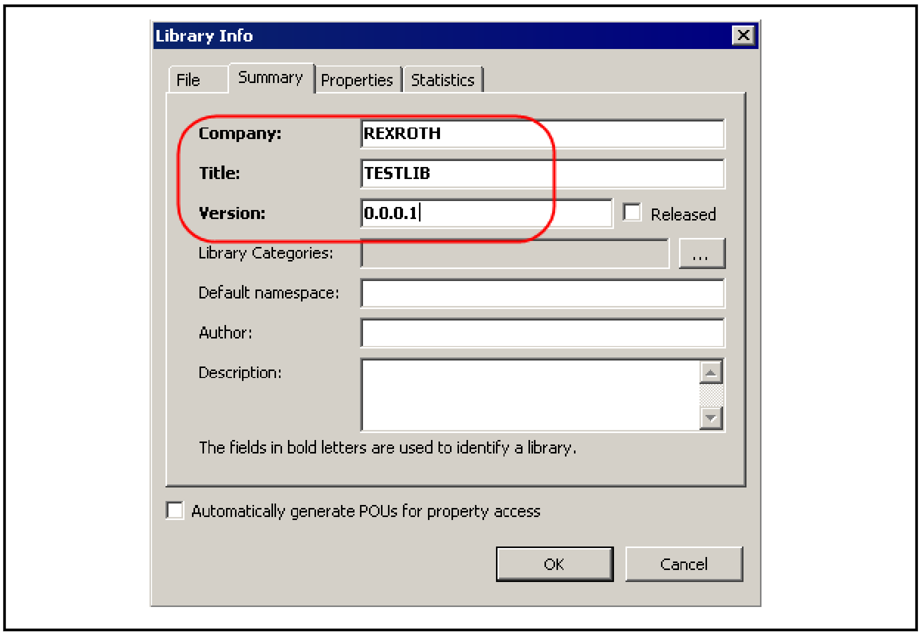


Fig.3-147: Adding a library info object to the planned library

In this object, at least complete the mandatory fields "Company", "Title" and "Version".



Library Info

File | Summary | Properties | Statistics

Company: REXROTH

Title: TESTLIB

Version: 0.0.0.1 □ Released

Library Categories:

Default namespace:

Author:

Description:

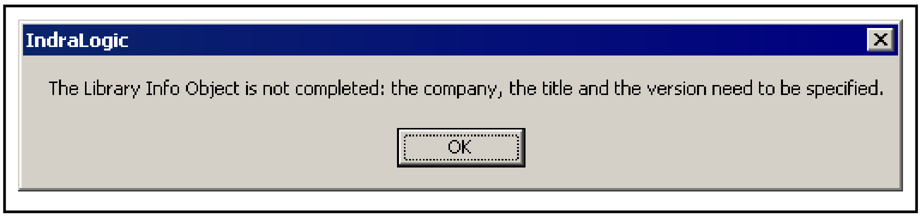
The fields in bold letters are used to identify a library.

□ Automatically generate POUs for property access

OK Cancel

Fig.3-148: Dialog: Library info object

If this was not performed or not performed correctly, the following message is output.



IndraLogic

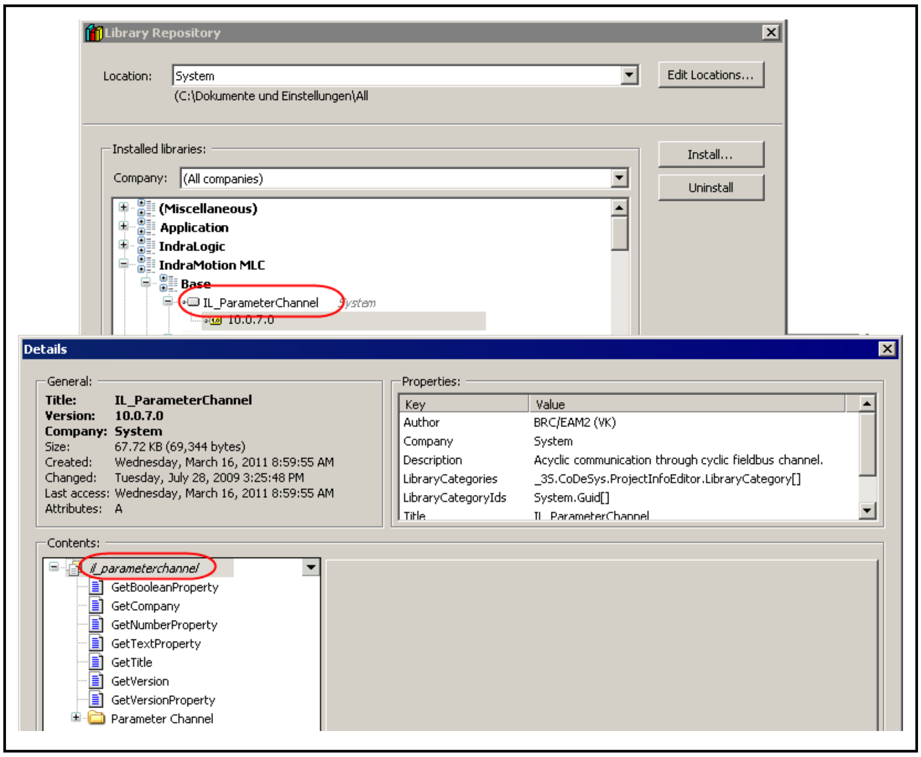
The Library Info Object is not completed: the company, the title and the version need to be specified.

OK

Fig.3-149: Message: Mandatory parts of the library info object are missing

☞ This mandatory data is later used by the library in the library repository.

The name of a library or compiled library is generally displayed in the details of the library repository (normally, title and name should match):



Location: System
(C:\Dokumente und Einstellungen\All

Installed libraries:
Company: (All companies)

General:
Title: IL_ParameterChannel
Version: 10.0.7.0
Company: System
Size: 67.72 KB (69,344 bytes)
Created: Wednesday, March 16, 2011 8:59:55 AM
Changed: Tuesday, July 28, 2009 3:25:48 PM
Last access: Wednesday, March 16, 2011 8:59:55 AM
Attributes: A

Properties:

Key	Value
Author	BRC/EAM2 (VK)
Company	System
Description	Acyclic communication through cyclic fieldbus channel.
LibraryCategories	_3S.CoDeSys.ProjectInfoEditor.LibraryCategory[]
LibraryCategoryIds	System.Guid[]
Title	IL_ParameterChannel

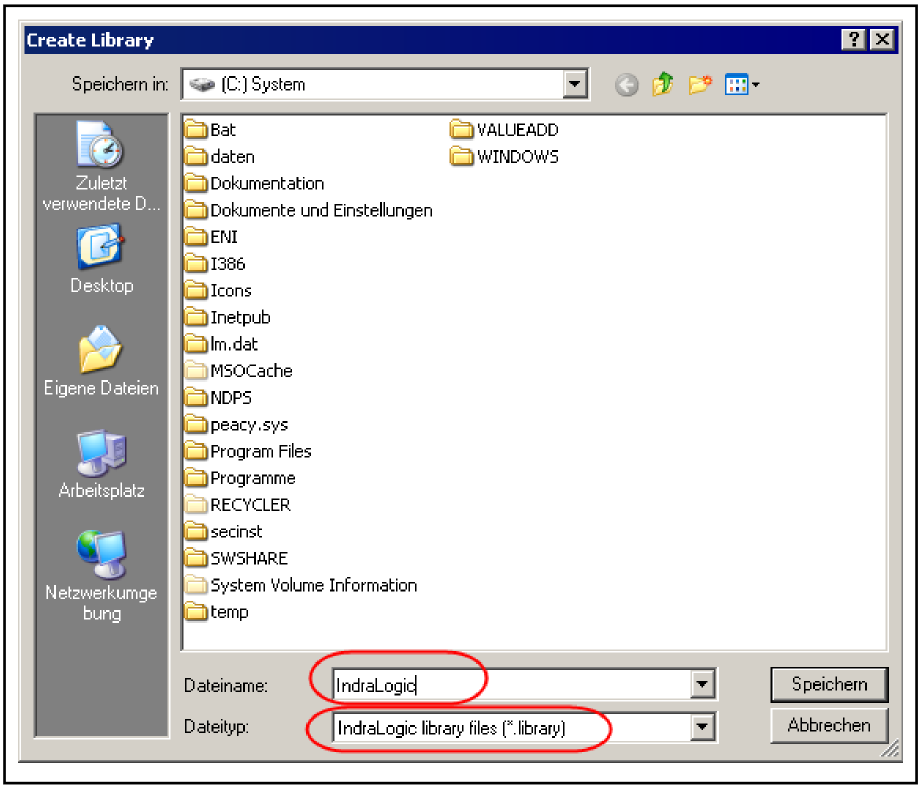
Contents:

Fig.3-150: Library in the library repository

From now, the process and the result of the three menu items differ:

Generate library...

The user first of all selects a directory (and can change the name of the library file) in which the library is to be stored. It has to differ from the IndraLogic directory of the currently opened project. If this is not the case, the user is requested to select a different directory.



Create Library

Speichern in: (C:) System

Dateiname: IndraLogic

Dateityp: IndraLogic library files (*.library)

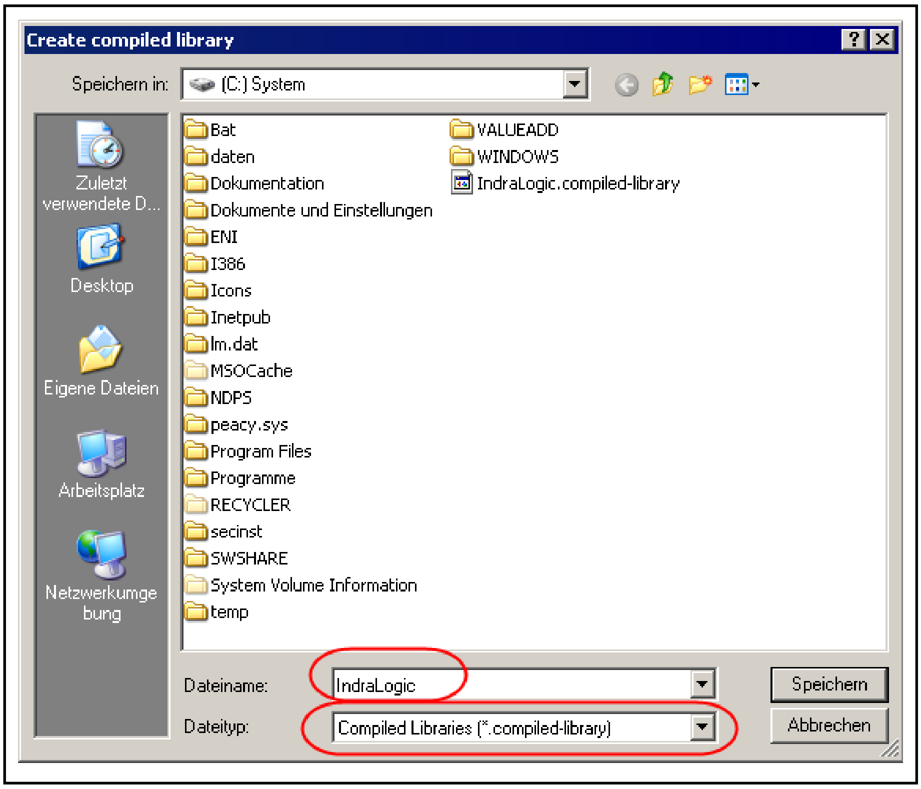
Fig.3-151: Dialog: Selection of the storage location and the name of the library to be generated

After the user has confirmed this dialog, the opened project is first of all saved and then, the library file is generated in the selected directory (in the preceding example in the path "C:\" the file "indralogic.library").

The "storage" name of the library only serves the finding and management outside the library repository. It does not influence the title in the "Library Info".

Generate compiled library...

The user first of all selects a directory (and can change the name of the library file) in which the compiled library is to be stored. It has to differ from the "IndraLogic" directory of the currently opened project. If this is not the case, the user is requested to select a different directory.



Speichern in: (C:) System

Dateiname: IndraLogic

Dateityp: Compiled Libraries (*.compiled-library)

Fig.3-152: Dialog: Selection of the storage location and the name of the "compiled library" to be generated

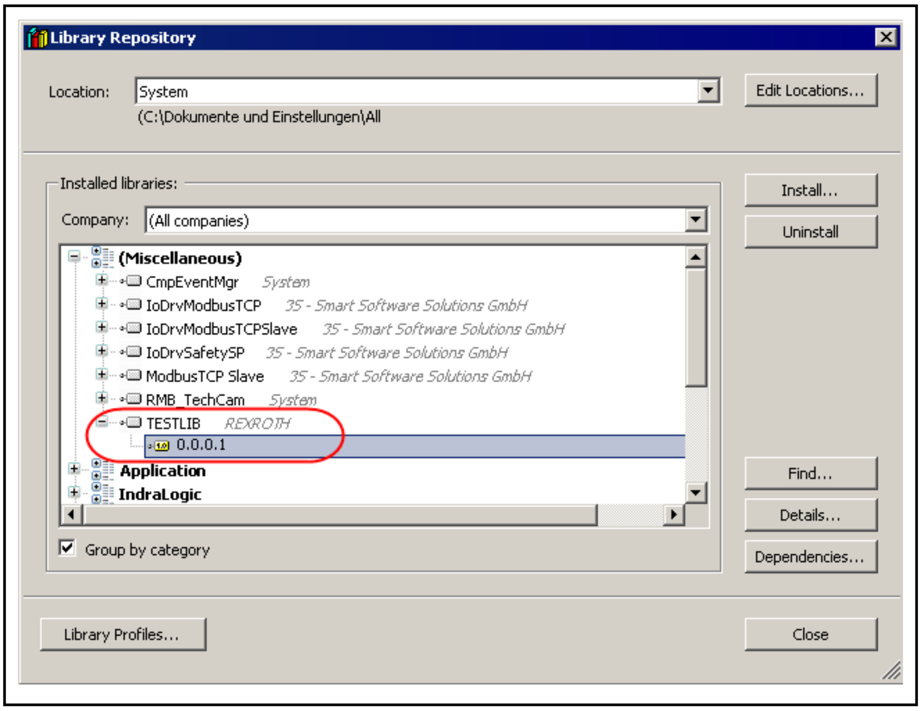
After the user has confirmed this dialog, the opened project is first of all saved and then, the file with the compiled library is generated in the selected directory (in the preceding example in the path "C:\" the file "indralogic.compiled-library").

The "storage" name of the "compiled library" only serves the finding and management outside the library repository. It does not influence the title in the "Library Info".

Generate and install compiled library…

If the user selects this menu item, the "indralogic.compiled-library" file is generated in the "Indralogic" directory of the opened project and immediately installed in the library repository.

The data is taken from the mandatory fields "Company", "Title" and "Version" of the library information.



Location: System
(C:\Dokumente und Einstellungen\All

Installed libraries:

Company: (All companies)

(Miscellaneous)
CmpEventMgr *System*
IoDrvModbusTCP *3S - Smart Software Solutions GmbH*
IoDrvModbusTCPSlave *3S - Smart Software Solutions GmbH*
IoDrvSafetySP *3S - Smart Software Solutions GmbH*
ModbusTCP Slave *3S - Smart Software Solutions GmbH*
RMB_TechCam *System*
TESTLIB *REXROTH*
0.0.0.1
Application
IndraLogic

☑ Group by category

Fig.3-153: Dialog: "Library Repository", new library TESTLIB

3.6.6 Options

Options, General Information

The "Options" dialogs allow to configure the behavior and appearance of the IndraLogic user interface. The default settings are determined by the current profile.

The dialogs can be accessed in **Tools** ▸ **Options**.

The current settings in the options dialog are saved on the local system as default settings and they overwrite the settings made when installing IndraLogic as delivered.

IndraLogic 2 G options

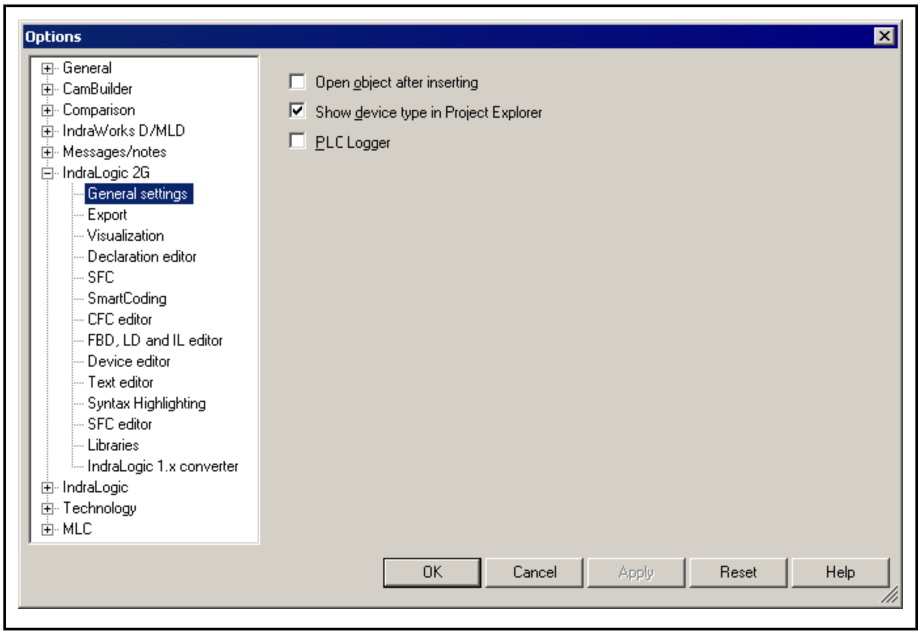
- Options, general settings, page 198
- Export options, page 198
- Visualization options, page 199
- Options, declaration editor, page 201,
- Options, sequential function chart, page 202,
- Options, smart coding, page 205,
- Options, CFC editor, page 207
- Options, FBD, LD and IL editors, page 207
- Options, device editor, page 211
- Options, text editor, page 212
- Options, syntax highlighting, page 219,
- Options, sequential function chart editor, page 219,
- Options, libraries, page 222,
- Options, converter for IndraLogic 1.x projects, page 224

Options, IndraLogic 1.x

- Options, IndraLogic 1.x settings, page 225

Options, General Settings

Menu: Tools ▸ Options ▸ IndraLogic 2G ▸ General settings.



□ Open object after inserting
☑ Show device type in Project Explorer
□ PLC Logger

Fig.3-154: "General settings" option

- Open document when inserting:

  When this option is selected, new objects that are inserted into the Project Explorer are opened immediately.
- Show device type in the Project Explorer

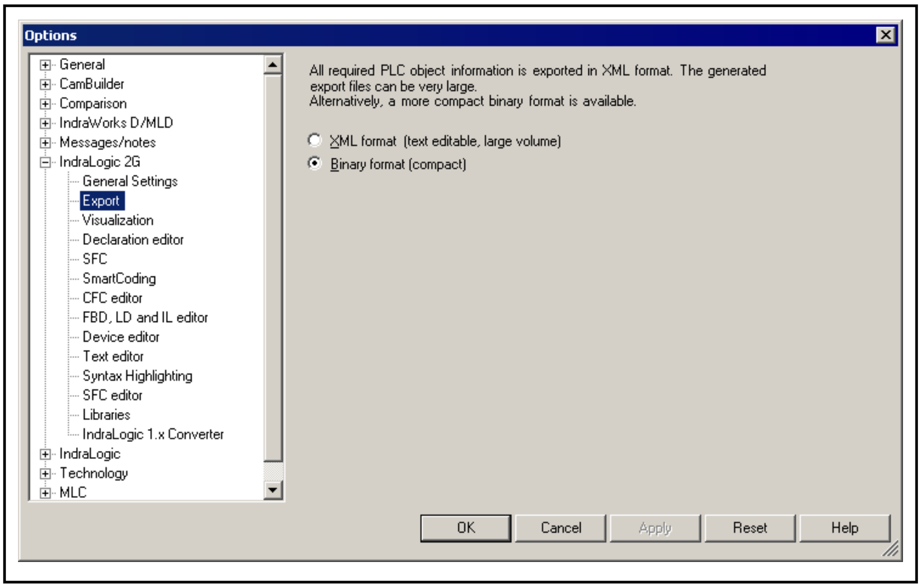
  The name of the object is extended with its device type.
- Activate PLC logger

  Using this button, the "Log" tab of the device dialog of every control can be activated.

Options, Export

Menu: Tools ▸ Options ▸ IndraLogic 2G ▸ Export.

The format of the export files generated by IndraWorks can be set on this option page:



All required PLC object information is exported in XML format. The generated export files can be very large.
Alternatively, a more compact binary format is available.

○ XML format (text editable, large volume)
● Binary format (compact)

Fig.3-155: Options, "Export"

Export format

Format selection

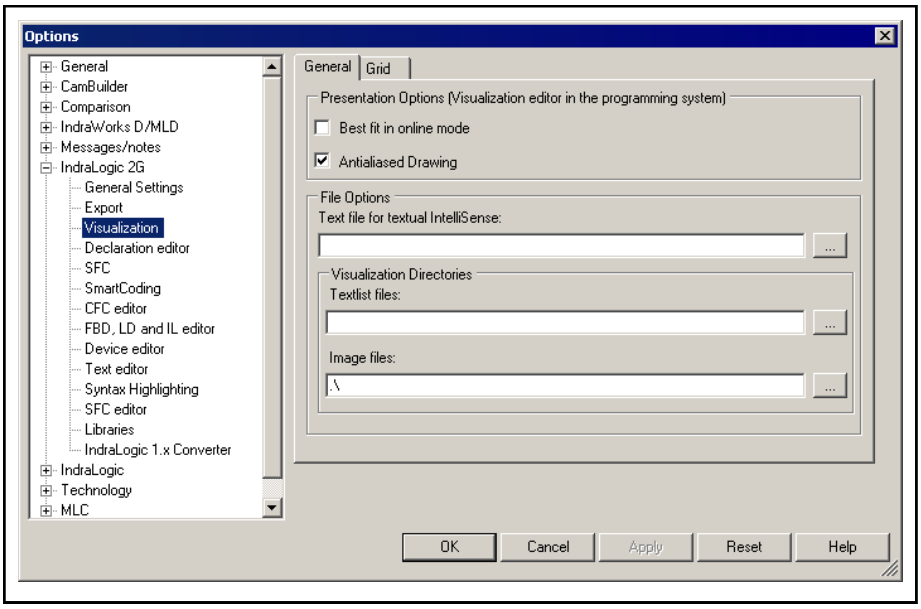
- With the "Binary format (compact)" setting, the objects selected by the user for export are exported in a compact format (as far as possible). The information associated with the PLC objects is integrated into the export file in binary format.
- When the "XML format" setting is applied, all of the objects selected by the user for export are exported in XML format. This could cause very large files.

Options, Visualizations

Menu: **Tools ▸ Options ▸ IndraLogic 2G ▸ Visualizations.**

General settings for working with visualizations in IndraLogic 2G are made in this dialog.

General tab



Presentation Options (Visualization editor in the programming system)

□ Best fit in online mode

☑ Antialiased Drawing

File Options

Text file for textual IntelliSense:

Visualization Directories

Textlist files:

Image files: .\

Fig.3-156: Dialog for visualization options, general information

Display options (visualization editor in the programming system)

- Optimum size in online mode

  This option has to be enabled the visualization is to be adjusted to the current editor window size in online mode so that all elements are always visible.

- Antialiased drawing

  ### in preparation ###

File options

- Text file for textual "List components":

  In the input field located below, the name (including the path) of a file (of .csv format) can be entered. It can then be used as a source for the suggestions contained in the "List components" function during editing of the 'Texts/Text' field within the properties of a visualization element.

  This text file can come from the export of a global text list, page 288,. It has a tabular structure which has to correspond to the structure of text lists which is described on the help page for text lists, page 55,.

- *Visualization directories*

  – Text list files

    Use the ... button to open the dialog for defining an existing or a new directory for the language files that are to be used in the project for configuring text and language in the visualization.

    The files found in this directory are made available when a text list is to be imported, e. g. after it has been processed externally by a translator. Only one directory can be entered here.

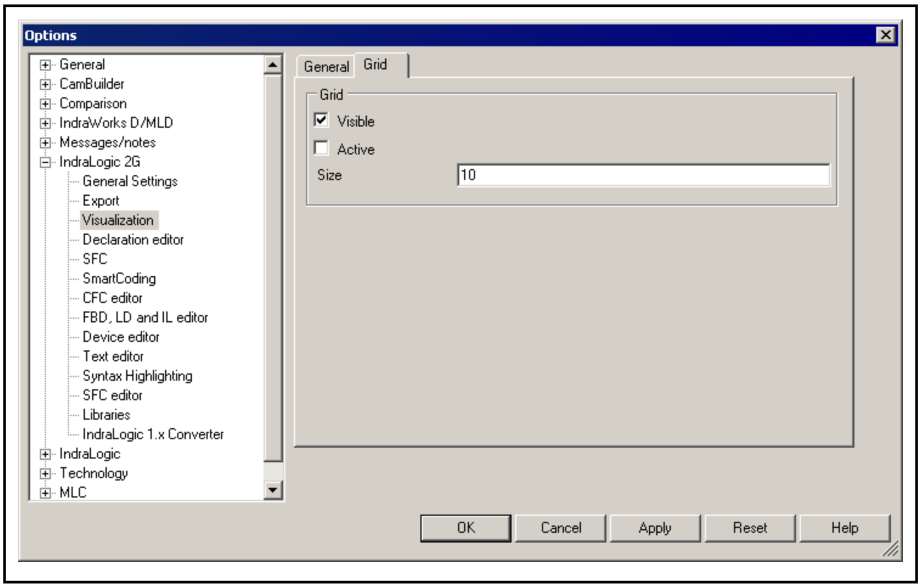
    For text and language configuration, see also the description in Text lists, page 55.

  – Image files

    Use the ... button to open the dialog for defining an existing or a new directory for the image files that are to be accessible in the project for visualizations. Enter several directories here in a list with the items separated by semicolons.

    Regarding image files, see also the description in Image pools, page 61.

Grid tab



Grid

☑ Visible

□ Active

Size 10

Fig.3-157: Dialog for visualization options, Grid

Grid

- **Visible:**

  when this option is enabled, a grid based on the size defined below is shown in the visualization editor for orientation purposes.

- **Active:**

  when this option is enabled, visualization elements can only be positioned on the points in the grid "size" defined below, no matter if the grid is set as "visible" or only "active".

  When inserting or moving an element this means that the center of the element can only be positioned on one grid point.

  When changing an element, this means that the small rectangle on the element edge that is moved in order to change the size or form of the element can only be moved to one grid point.

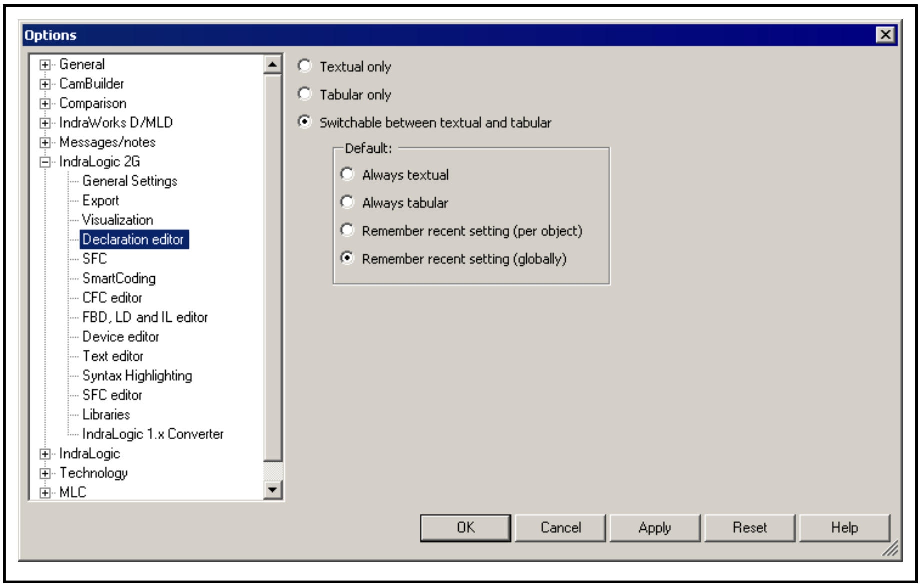
- **Size:**

  vertical and horizontal distance between the grid points in pixels.

Options, Declaration Editor

Menu:**Tools ▸ Options ▸ IndraLogic 2G ▸ Declaration Editor.**

This dialog enables general settings to be made for working in the declaration editor, page 326.



○ Textual only

○ Tabular only

● Switchable between textual and tabular

Default:

○ Always textual

○ Always tabular

○ Remember recent setting (per object)

● Remember recent setting (globally)

Fig.3-158: "Declaration editor options" dialog

The display for the declarations can be specified:

Only textual

Only tabular

Switchable between textual and tabular: If this setting is activated, there are two buttons in the declaration editor window by means of which it can be switched between the textual and the tabular form.

In this case, one of the following settings defines which of the two views is used by default upon opening of an object in the editor:.

- **Always textual**
- **Always tabular**
- **Save latest setting (per object):** If an object is re-opened, the declaration editor appears in the same view as during the last editing process.
- **Save latest setting (global):** If an object is opened, the declaration editor appears in the same view as during its last use, regardless of the object.

Options, SFC

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **SFC.**

The **default settings for SFC objects** can be defined in this dialog.

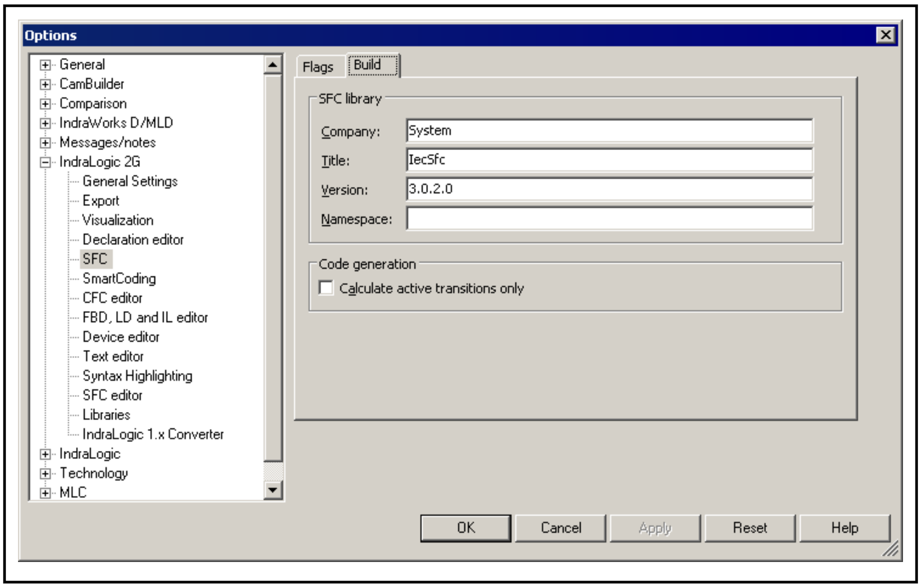
Each **new** SFC object is automatically then provided with these settings in its properties.

☞ The changes to the settings made here have to be transferred explicitly to **existing** SFC objects.

See Modifying the properties of existing POU objects, page 205.

Note that the basic editor options, page 219, i.e. settings for layout and display are managed in a separate options dialog ("SFC editor").

Compilation



SFC library

Company: System

Title: IecSfc

Version: 3.0.2.0

Namespace:

Code generation

☐ Calculate active transitions only

Fig.3-159: SFC options, compilation

SFC library:

The SFC library defined here is used by default for new SFC objects. **Company**, **Title** and **Version** are entered as they are in the project settings for the library as delivered. A **namespace** can be entered as well in order to uniquely address a library. This could be necessary if several versions of the library are available on the system.

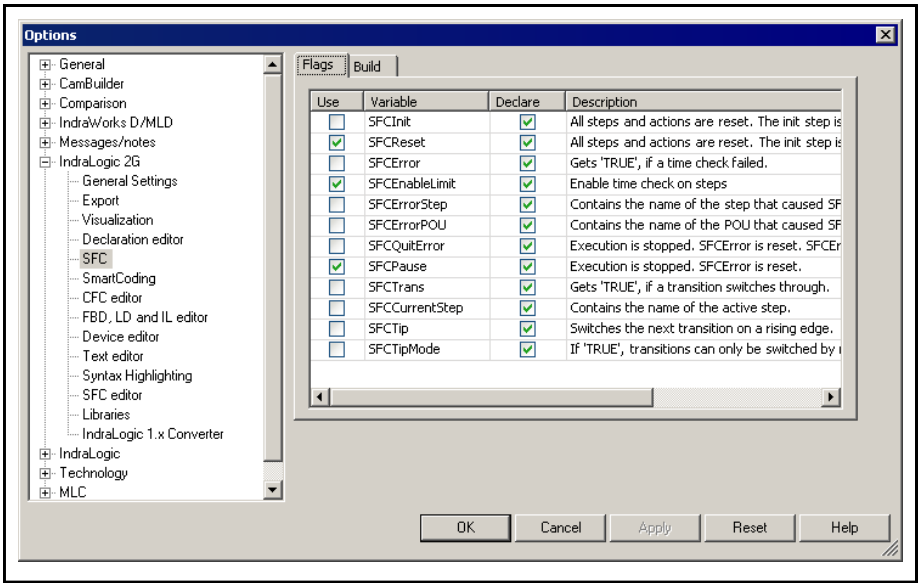
When entering a namespace, however, always ensure that the space entered here is the same as that defined in the Library manager, page 368, for the library!

By default this dialog contains settings for the **Sfclec.library** library, which is delivered along with the standard profile.

Code generation:

Calculate active transitions only: When this option is selected, only the currently active transitions are calculated.

Variables



Use	Variable	Declare	Description
□	SFCInit	☑	All steps and actions are reset. The init step is
☑	SFCReset	☑	All steps and actions are reset. The init step is
□	SFCError	☑	Gets 'TRUE', if a time check failed.
☑	SFCEnableLimit	☑	Enable time check on steps
□	SFCErrorStep	☑	Contains the name of the step that caused SF
□	SFCErrorPOU	☑	Contains the name of the POU that caused SF
□	SFCQuitError	☑	Execution is stopped. SFCError is reset. SFCEr
☑	SFCPause	☑	Execution is stopped. SFCError is reset.
□	SFCTrans	☑	Gets 'TRUE', if a transition switches through.
□	SFCCurrentStep	☑	Contains the name of the active step.
□	SFCTip	☑	Switches the next transition on a rising edge.
□	SFCTipMode	☑	If 'TRUE', transitions can only be switched by

Fig.3-160: SFC options, variables

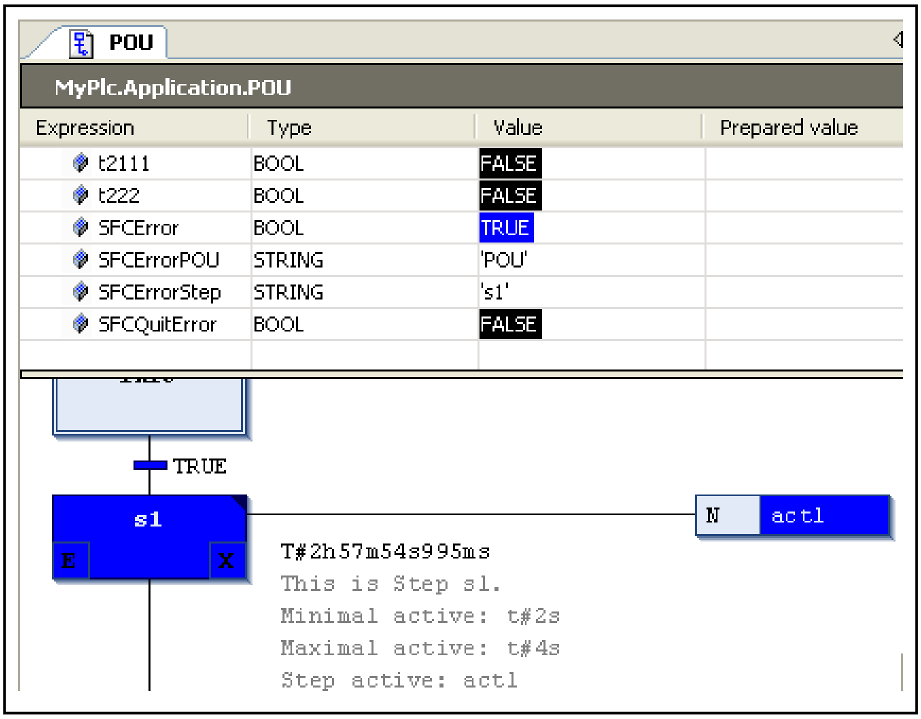
All possible "flags", i.e. implicitly generated variables for monitoring and controlling the processing in an SFC chart are listed here. A short description is included in the dialog.

Detailed information on the flags, see page 414.

Clicking on the checkbox for a flag variable enables automatic declaration (**Declare**) and use (**Use**). These settings are then accepted as default settings for new SFC objects.

If "Declare" is selected but "Use" is not selected, the variable is declared, but the flag has no function during processing.

☞ Note that a flag variable with automatic declaration is visible in the declarations of the SFC editor **only in online mode**!



POU

MyPlc.Application.POU

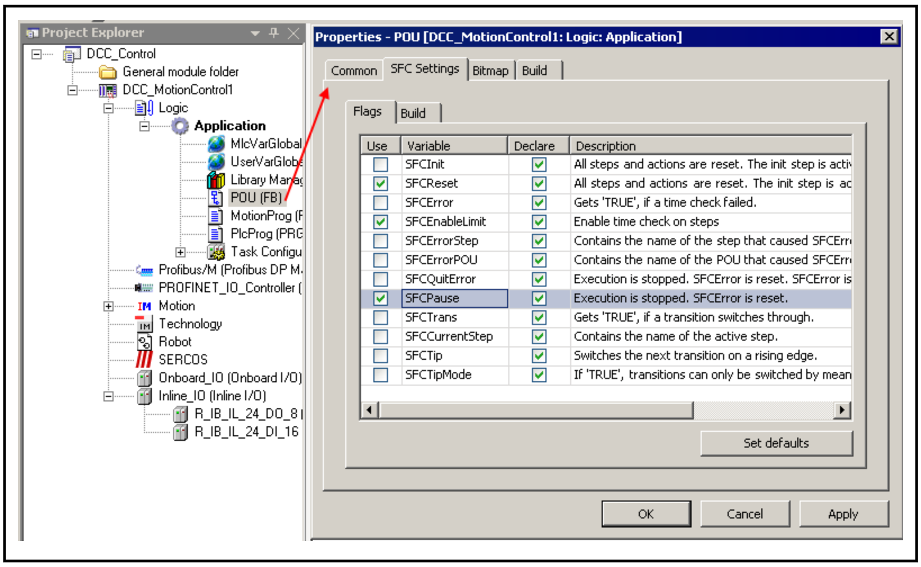
Expression	Type	Value	Prepared value
t2111	BOOL	FALSE	
t222	BOOL	FALSE	
SFCError	BOOL	TRUE	
SFCErrorPOU	STRING	'POU'	
SFCErrorStep	STRING	's1'	
SFCQuitError	BOOL	FALSE	

Fig.3-161: Example of an SFC error flag in the online mode of the editor

A timeout is indicated in step "s1" in the SFC object "POU" by the SFCError flag.

Modifying the properties of existing POU objects

Since modified options are only effective for **newly created** POU objects, **existing** POU objects have to be modified later on (right click, "Properties"):



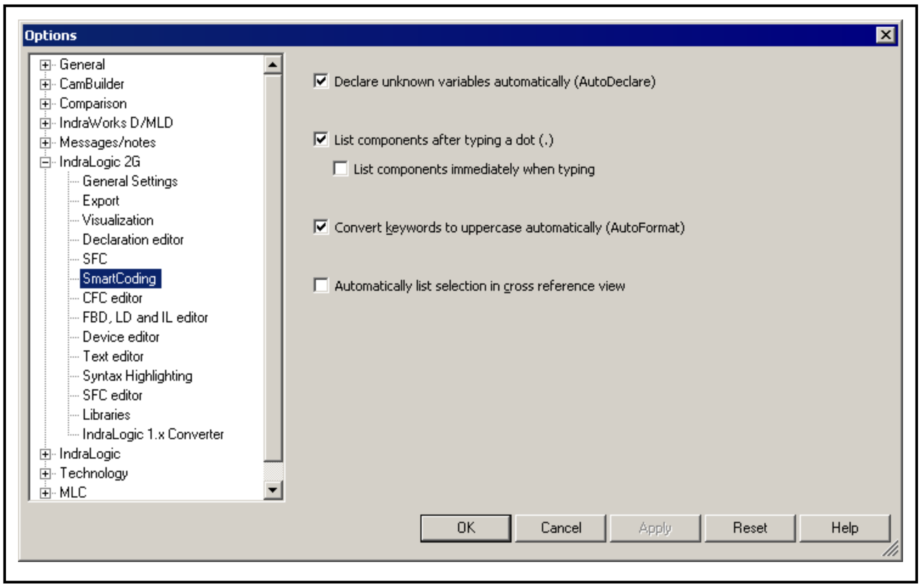
Use	Variable	Declare	Description
□	SFCInit	☑	All steps and actions are reset. The init step is activ
☑	SFCReset	☑	All steps and actions are reset. The init step is ac
□	SFCError	☑	Gets 'TRUE', if a time check failed.
☑	SFCEnableLimit	☑	Enable time check on steps
□	SFCErrorStep	☑	Contains the name of the step that caused SFCErr
□	SFCErrorPOU	☑	Contains the name of the POU that caused SFCErr
□	SFCQuitError	☑	Execution is stopped. SFCError is reset. SFCError is
☑	SFCPause	☑	Execution is stopped. SFCError is reset.
□	SFCTrans	☑	Gets 'TRUE', if a transition switches through.
□	SFCCurrentStep	☑	Contains the name of the active step.
□	SFCTip	☑	Switches the next transition on a rising edge.
□	SFCTipMode	☑	If 'TRUE', transitions can only be switched by mean

Fig.3-162: Modifying the SFC settings for an individual POU object

Options, Smart Coding

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Smart coding.**

This dialog contains settings that makes entering code easier.



☑ Declare unknown variables automatically (AutoDeclare)

☑ List components after typing a dot (.)

□ List components immediately when typing

☑ Convert keywords to uppercase automatically (AutoFormat)

□ Automatically list selection in cross reference view

Fig.3-163: Options, smart coding

Declare unknown variables automatically (AutoDeclare)

When this option is selected, the Declare variable, page100, dialog opens automatically as soon as an identifier that is not yet declared is entered in a language editor and the user moves beyond the input line.

List components after typing a dot (.):

This option enables the List components function, page 103,

That means that if a dot (.) is entered in an editor in a position in which an identifier is expected, a selection list appears with the possible inputs at this position.

List components immediately when typing:

If this option is selected, after any character string is entered in an editor, a selection list appears with the available identifiers and operators. This is also a type of "List components" functionality.

The first entry in the selection list that starts with the same character string as that which was entered is automatically highlighted as the selection and can be accepted at the cursor position by pressing <Enter>.

Otherwise, another entry can be selected from the list.

Convert keywords to upper case automatically (AutoFormat):

If this option is selected, all of the keywords are automatically written in upper case letters.

Example:

When

```
bVar:bool;
```

is input, it is converted to

```
bVar: BOOL;
```

.

Automatically update cross references when changing selection:

If this option is enabled, the cross reference list, page 107, automatically displays a list of the references of the variables that are currently in focus in the editor.

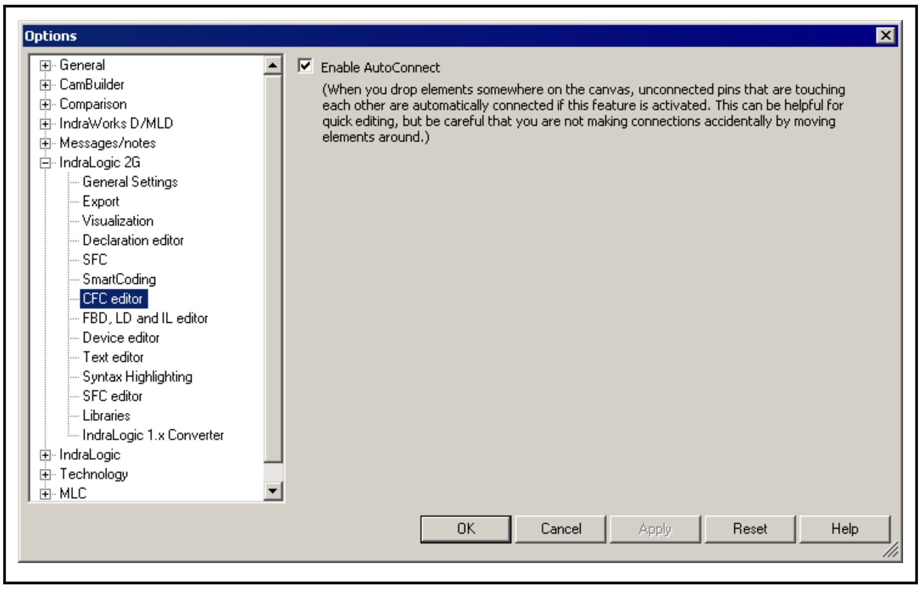
Differences from IndraLogic 1.x with regards to the cross reference list:

- The cross reference list can be displayed in a window (view) next to the editor window.
- The project does not have to be compiled in order to display the cross references.
- Component accesses, enumeration constants and user-defined data types are also displayed.
- A dynamic display of the cross references for the respective variable in focus can be enabled.

Options, CFC Editor

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **CFC editor.**

This dialog enables general settings to be made for working in the CFC editor, page 309.



☑ Enable AutoConnect

(When you drop elements somewhere on the canvas, unconnected pins that are touching each other are automatically connected if this feature is activated. This can be helpful for quick editing, but be careful that you are not making connections accidentally by moving elements around.)

Fig.3-164: "CFC Editor Options" dialog

Enable AutoConnect

If this option is active, the following applies:

If a CFC element is dragged and dropped somewhere on the editor working area, unconnected pins that are "touching" each other are automatically connected. This might support faster editing; however ensure not to generate unwanted links accidentally when shifting elements!

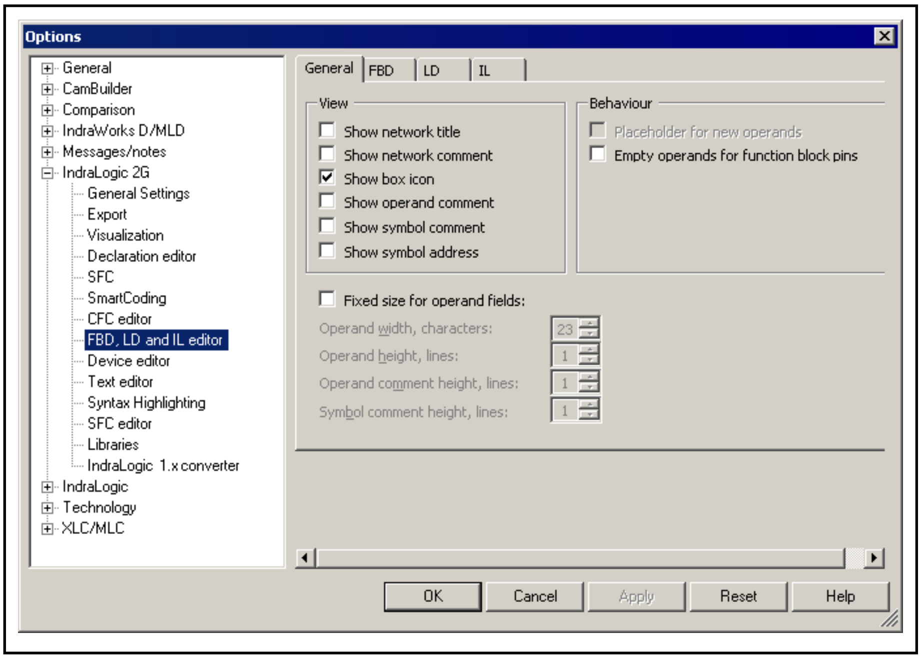
Options, FBD, LD and IL

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **FBD, LD and IL**

The FBD/LD/IL editor is configured in this dialog:

Menu Items

General



View
- □ Show network title
- □ Show network comment
- ☑ Show box icon
- □ Show operand comment
- □ Show symbol comment
- □ Show symbol address

Behaviour
- □ Placeholder for new operands
- □ Empty operands for function block pins

□ Fixed size for operand fields:

Operand width, characters: 23

Operand height, lines: 1

Operand comment height, lines: 1

Symbol comment height, lines: 1

Fig.3-165: Options for displaying the FBD/LD/IL editor

View

Show network title	The network title - if defined - is displayed in the upper left corner of the network.
Show network comment	The network comment - if defined - is displayed in the upper left corner of the network. If the network title is also displayed, the comment appears in the line below.
Display function block symbol	If a function block or a function has been provided with a symbol (bitmap) via a library or via the object properties, it is displayed in the function block element in the FBD and LD editor. The standard operators are provided with symbols, as well

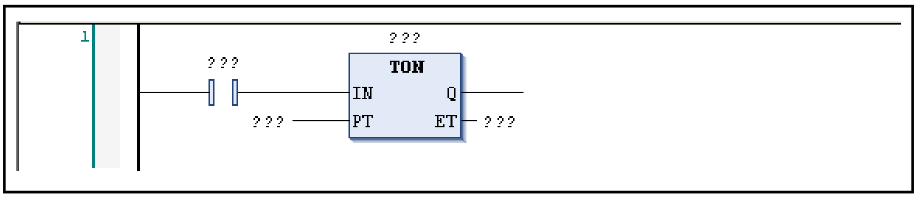


Fig.3-166: "Show function block symbol" option not activated

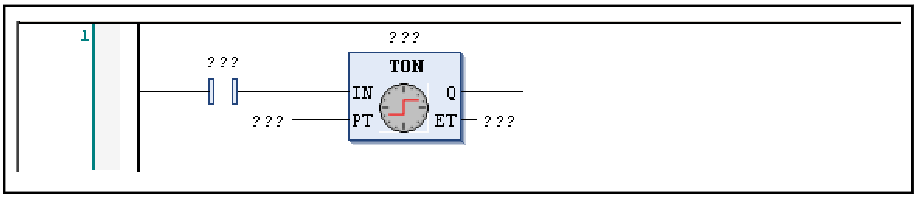


Fig.3-167: "Show function block symbol" option activated

Show operand comment	The comment that might have been added to a variable in the implementation section of the editor is displayed. The operand comment refers only to the local use position of the variable, in contrast to the "symbol comment" which is indicated in the declaration of the variable.

Show symbol comment

For each symbol (variable) which has been provided with a comment in its declaration, this comment is displayed above the variable name. Note that in addition to or instead of this comment, a local "operand comment" can be assigned, as well.

Show symbol address

For each symbol (variable), the address that might have been assigned is displayed above the variable name.

Behavior

Placeholder for new operands: ### in preparation ###

Empty operands for function block pins ### in preparation ###

Fixed size for operand field

The FBD, LD and IL editor automatically adjusts the size of the operand comment fields.

If this option is selected, the following parameters define the display size of the information fields for an operand:

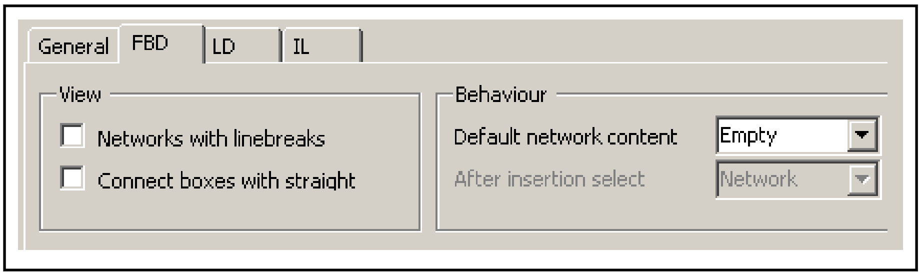
Width of operand field — Maximum number of characters that can be shown for operand name.

Height of operand field — Maximum number of lines that are used for the display of the operand name.

Height of operand comment field — Maximum number of lines that are used for the display of the operand comment.

Height of operand symbol field — Maximum number of lines that are used for the display of the operand symbol.

FBD



□ Networks with linebreaks
□ Connect boxes with straight
Default network content: Empty
After insertion select: Network

Fig.3-168: Options for displaying the FBD editor

View

Network with line breaks

If this option is selected, the networks are displayed with line breaks so that the largest number of function blocks possible can be displayed in the current width of the editor window. Consequently, the networks might require more height.

If the editor window is not wide enough, the networks are not wrapped.

Connect function blocks with straight

lines

If this option is selected, the elements are arranged in a network so that the lines between them receive a fixed, short length and the width required for displaying the networks is reduced to the largest possible extent. Consequently, the function block boxes might be extended vertically in order to create enough space for the input and output elements.

If this option is not selected, the elements maintain their standard size and the connection lines are modified to fit within the space required.

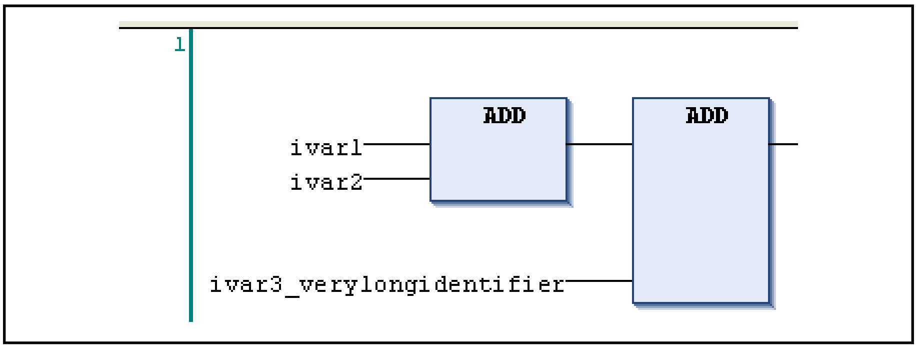


Fig.3-169: Option disabled:

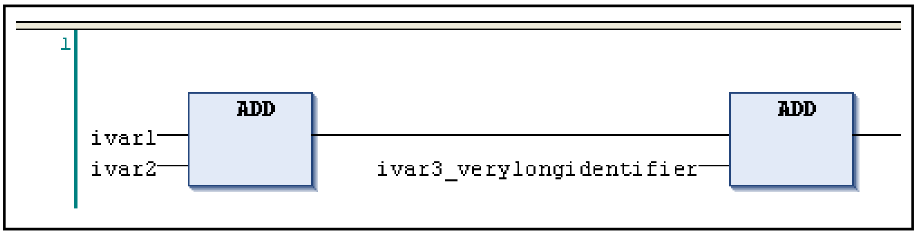


Fig.3-170: Option enabled:

Behavior

Network content

This selection describes which content is displayed after the insertion of a new network in the editor window.

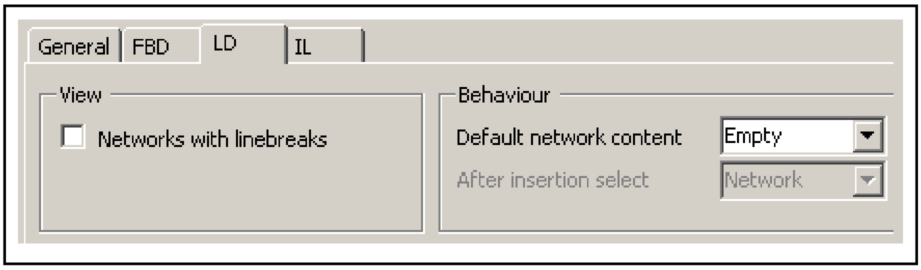
(Empty, assignment, empty block).

Selection after insertion

This selection describes which element is selected after insertion of a new network.

(Element, network).

LD



View

☐ Networks with linebreaks

Behaviour

Default network content: Empty

After insertion select: Network

Fig.3-171: Options for displaying the LD editor

View

Network with line breaks

If this option is selected, the networks are displayed with line breaks so that the largest number of function blocks possible can be displayed in the current width of the editor window. Consequently, the networks might require more height.

If the editor window is not wide enough, the networks are not wrapped.

Behavior

Network content

This selection describes which content is displayed after the insertion of a new network in the editor window.

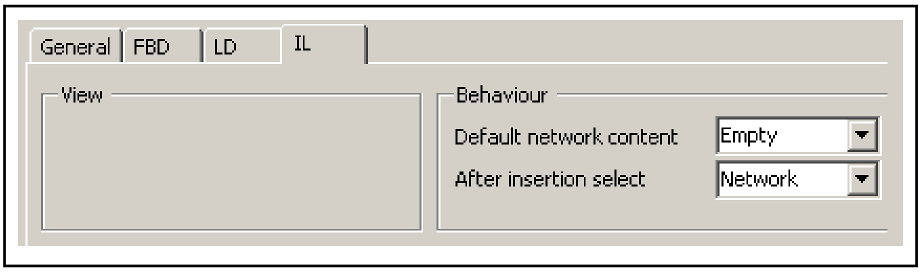
(Empty, contact and coil, empty block).

Selection after insertion

This selection describes which element is selected after insertion of a new network.

(Element, network).

IL



Default network content Empty

After insertion select Network

Fig.3-172: Options for displaying the IL editor

Behavior

Network content

This selection describes which content is displayed after the insertion of a new network in the editor window.

(Empty, LD and ST, CAL).

Selection after insertion

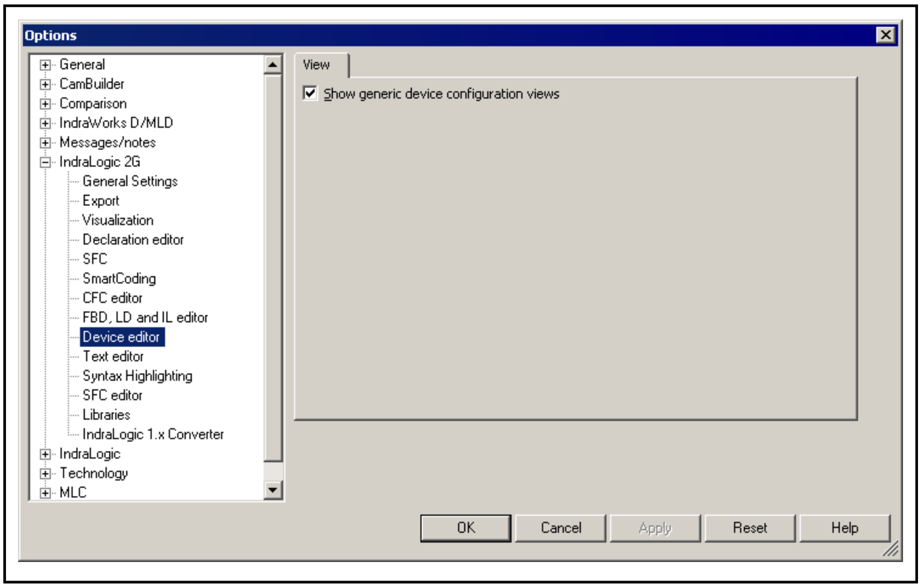
This selection describes which element is selected after insertion of a new network.

(Text, network).

Options, Device Editor

Menu:**Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Device Editor.**

This dialog enables general settings to be made for working in the device editor, page 331.



☑ Show generic device configuration views

Fig.3-173: "Device editor options" dialog

The "View" subdialog contains the **Show generic device configuration editors** setting.

☞ *This option enables*

1. the "Field Bus Diagnostics" tab in the object properties for PROFIBUS DP master and PROFINET IO controllers.
2. For devices that can be parameterized (field bus objects), the configuration dialog is available.

Options, Text Editor

Options, Text Editor, General Information

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Text editor.**

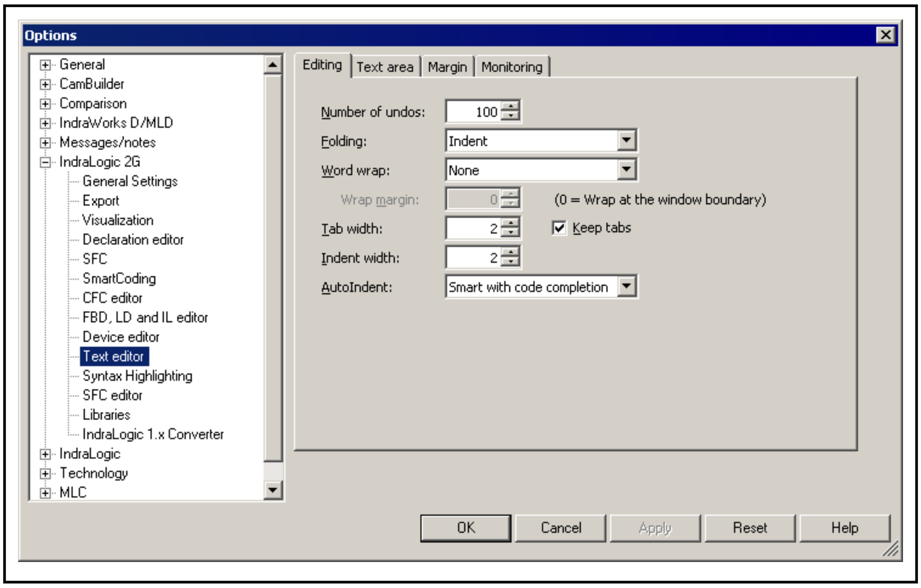
This dialog contains settings for working in one of the text editors:

Tabs

- Editing: Defining Undo steps, tabs, indentation, breaks, etc., page 212
- Text area: Color and font definitions for the text area, page 215
- Margin area: Definitions of colors, font, mouse activity in the margin of the editor, left of the text area, page 217
- Monitoring: Enabling/disabling Inline monitoring, display of monitoring fields, page 218

Options, text editor, editing

Defining Undo steps, tabs, indentation, breaks, etc.



Number of undos: 100

Folding: Indent

Word wrap: None

Wrap margin: 0 (0 = Wrap at the window boundary)

Tab width: 2 ☑ Keep tabs

Indent width: 2

AutoIndent: Smart with code completion

Fig.3-174: Options, text editor, editing

Number of Undo steps

The number of editing steps entered here is saved and can be undone with the "Undo" command.

Fold (outline form):

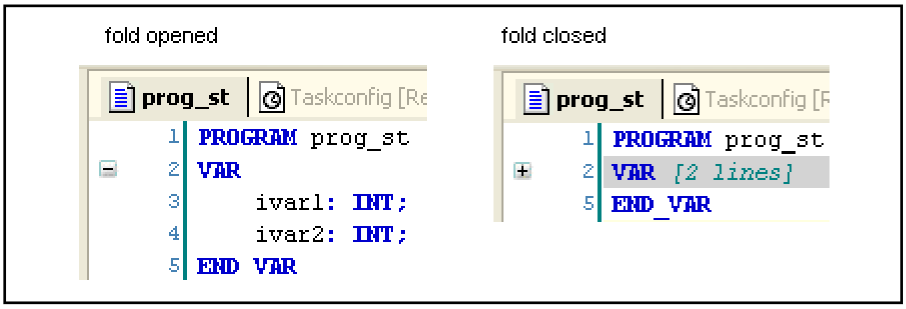
The selected option determines if the code is to be structured using indents. If yes, indented sections or sections that are identified by a special comment can be hidden or shown by using plus and minus signs placed before the first line of the respective section. The following options are possible:

None: No indent.

Indent: All lines that are indented relative to the previous lines are collected into an "indented unit" with a sign placed before the preceding line.

Examples: The lines between VAR and END VAR or between IF and END_IF are indented, Thus, they are collected in an indented unit marked with a minus sign in front of "VAR" when open and a plus sign in front of "VAR" when closed.

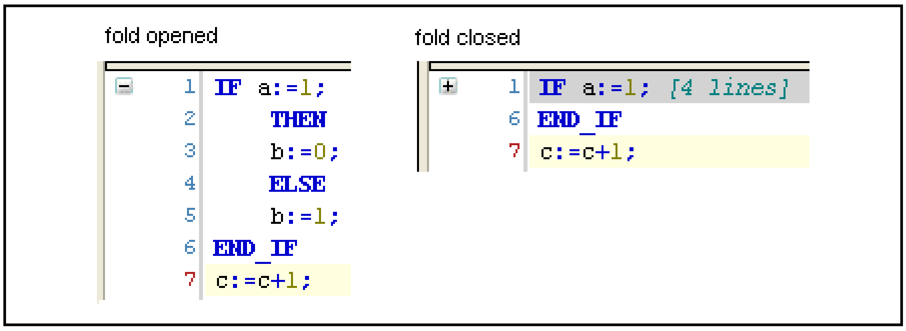
Open and closed indented units:



```
PROGRAM prog_st
VAR
        ivar1: INT;
        ivar2: INT;
END_VAR
```

```
PROGRAM prog_st
VAR [2 lines]
END_VAR
```

Fig.3-175: Opened and closed indented units (declaration)



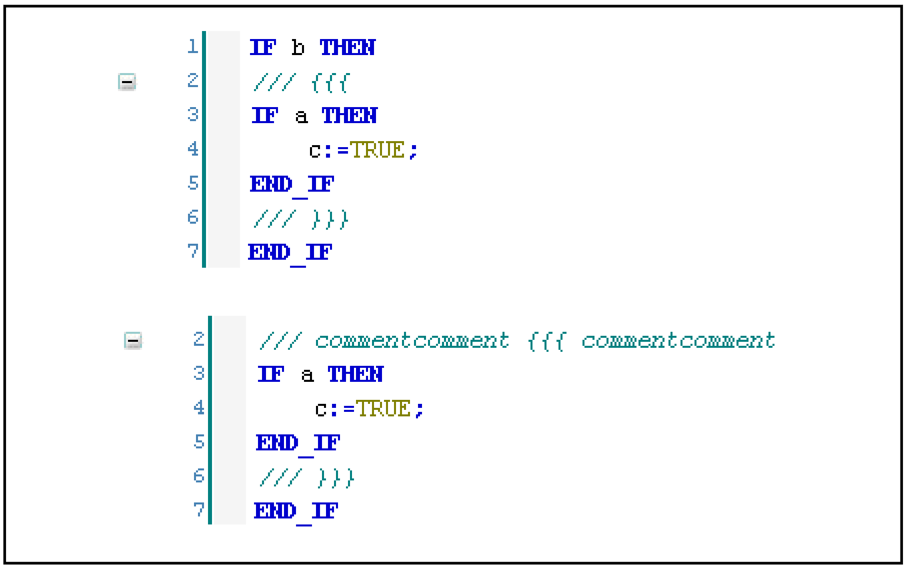
```
IF a:=1;
        THEN
        b:=0;
        ELSE
        b:=1;
END_IF
c:=c+1;
```

```
IF a:=1; [4 lines]
END_IF
c:=c+1;
```

Fig.3-176: Opened and closed indented units (ST)

Explicit: If this option is selected, the code section which includes an indented section is identified explicitly with comments: A comment that includes three opening curly brackets "{{{" has to be located in front the section; another comment that follows the section has to include three closing curly brackets "}}}". The comments can include additional text.

Example of the definition of an indented section by using special comments:



```
IF b THEN
/// {{{
IF a THEN
        c:=TRUE;
END_IF
/// }}}
END_IF
```

```
/// commentcomment {{{ commentcomment
IF a THEN
        c:=TRUE;
END_IF
/// }}}
END_IF
```

Fig.3-177: Indented section explicitly defined by special comments:

Word wrap:

None: The lines are wrapped endlessly.

Soft: The lines are broken at the margin of the editor window if a "0" is entered at the **break position**.

Hard: The lines are broken based on the number of characters entered at the **break position**.

Tab width:

Tab width in number of characters.

Indent width:

If the Auto Indent option (see below) is enabled, an indentation to the right is made here based on the number of characters entered, i.e. a corresponding number of spaces are inserted at the beginning of the line.

Auto indent:

Never: No automatic indentations are made during editing. The character strings entered always start at the left margin of the text area.

Block:

Smart: The lines following a line that contains a keyword (e.g. VAR or IF) are automatically indented based on the Indent width listed above.

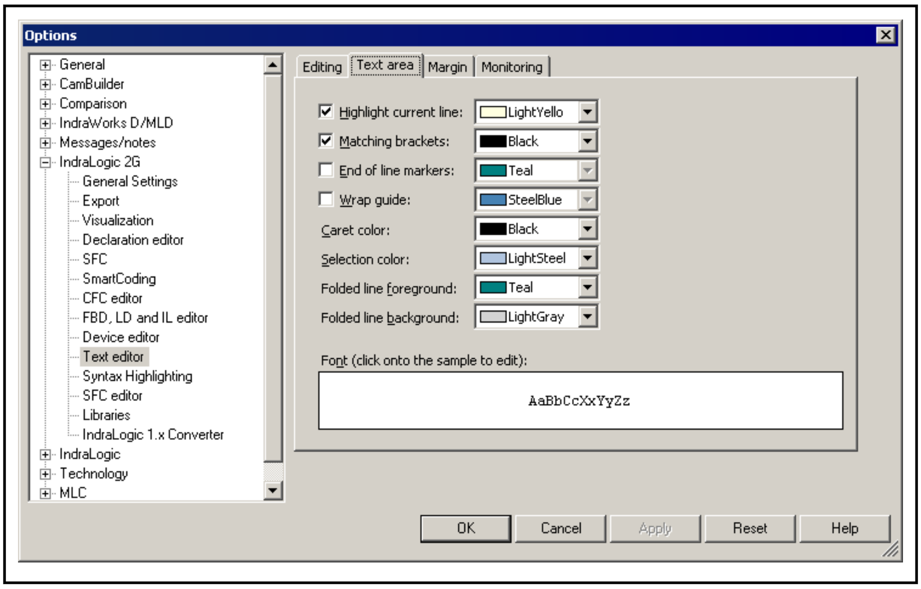
Smart with code completion: The lines following a line that contains a keyword (e.g. VAR or IF) are automatically indented based on the Indent width listed above and in addition, the corresponding closing keyword is inserted (e.g. END_VAR or END_IF).

Keep tabs

If this option is selected, an empty space in a line inserted with the <Tab> key, based on the defined Tab width (see above), is not broken down into individual single spaces. This means that if the tab width is changed, the empty spaces created wit the <tab> key are modified as well.

Options, text editor, text area

Color and font definitions for the text area



☑ Highlight current line: LightYello
☑ Matching brackets: Black
☐ End of line markers: Teal
☐ Wrap guide: SteelBlue
Caret color: Black
Selection color: LightSteel
Folded line foreground: Teal
Folded line background: LightGray
Font (click onto the sample to edit): AaBbCcXxYyZz

Fig.3-178: Options, text editor, text area

The following options can be enabled for better optical support during editing:

Highlight current line: The line that contains the cursor is highlighted with the set color.

Matching brackets: When the cursor is positioned before or after parentheses "(" or ")" in a code line, the matching closing or opening parenthesis is marked with a frame in the set color. Likewise, if the cursor is in the first or last line of a parentheses-enclosed expression[1] the corresponding last or first line of this area is shown with a square frame in this color.

End of line markers: The end of each editing line is marked by a small dash in the set color after the last character in the line (including spaces).

Line break position: If a soft or hard line break is enabled (see above, 'Editing' dialog), the defined line break position is shown with a vertical line in the selected color.

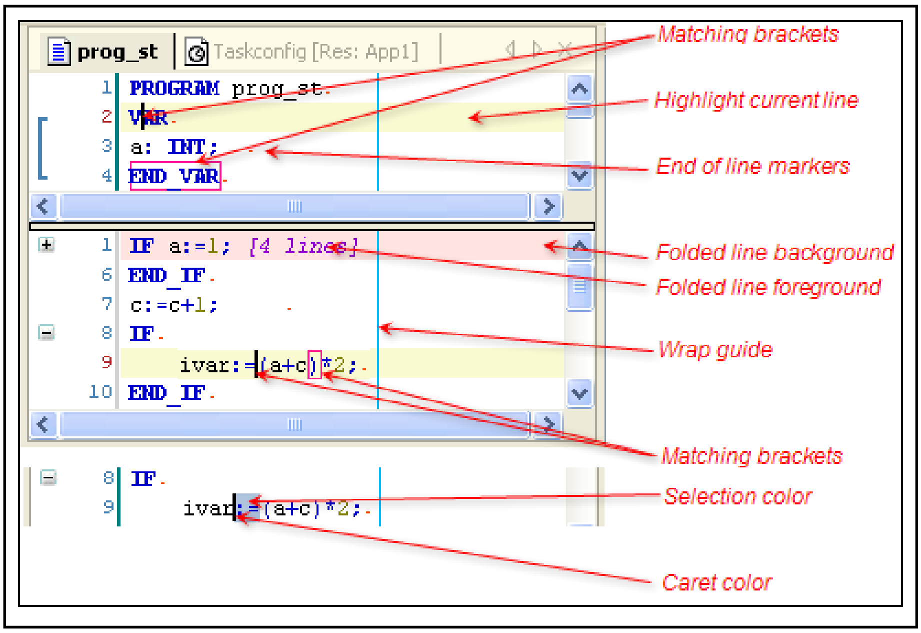
Caret color: Color of the cursor

Selection color: The currently selected text area is highlighted with the selected color.

Folded line foreground: The header of a closed, indented section in the code is displayed in the selected color.

Folded line background: The header of a closed, indented section in the code is highlighted in the selected color.

Fonts: The font set here is used in the text editor. Click on the sample field to open the standard dialog for configuring the font.



```
PROGRAM prog_st
VAR
a: INT;
END_VAR
```

```
IF a:=1; [4 lines]
END_IF
c:=c+1;
IF
    ivar:=(a+c)*2;
END_IF
```

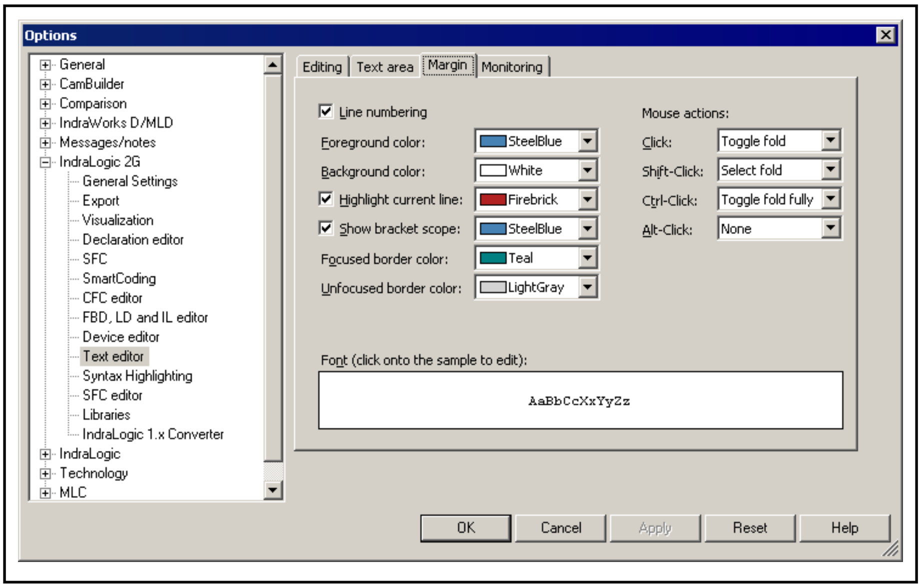
```
IF
    ivar:=(a+c)*2;
```

Fig.3-179: Example settings for the text area

[1] *A parentheses-enclosed expression includes all of the lines between the starting and end keyword of a construct (e.g. all of the lines between IF and END_IF). See also the "Margin area" subdialog.*

Options, text editor, margin area

Definitions of colors, font, mouse activity in the margin of the editor, left of the text area



☑ Line numbering

Foreground color: SteelBlue

Background color: White

☑ Highlight current line: Firebrick

☑ Show bracket scope: SteelBlue

Focused border color: Teal

Unfocused border color: LightGray

Mouse actions:

Click: Toggle fold

Shift-Click: Select fold

Ctrl-Click: Toggle fold fully

Alt-Click: None

Font (click onto the sample to edit):

AaBbCcXxYyZz

Fig.3-180: Options, text editor, margin area

The following settings and options refer to the left margin area in the text editor window separated from the input area by a vertical line.

Line numbering: Line numbers are displayed in the declarations and in the implementation section of the text editors, always starting with 1 in the first line.

Foreground color: Color of the line numbers

Background color: Background color of the margin area

Current line: Color of the line number for the line in which the cursor is currently positioned

Parentheses-enclosed expression: A parentheses-enclosed expression includes the lines between the keywords that open and close a construct, e.g. between IF and END_IF.

The bracket scope function can be enabled using the 'Matching brackets' option in the 'Text area' dialog.

If the cursor is positioned in front of, after or within one of the corresponding keywords, the bracket area is displayed with a bracket in the margin.

Focused border color: Color of the vertical separating line between the margin area and the input area in the currently active section of the text editor window.

Unfocused border color: Color of the vertical separating line between the margin area and the input area in the section of the text editor window that is not currently active.

Mouse actions: One of the following actions can be assigned to each of the specified mouse movements or mouse shorcuts (click, <Shift> + click, <Ctrl> + click, <Alt> + click).

The action is executed if the mouse movement is executed on the plus or minus sign positioned in front of the header of a parentheses-enclosed area:

None: No action.

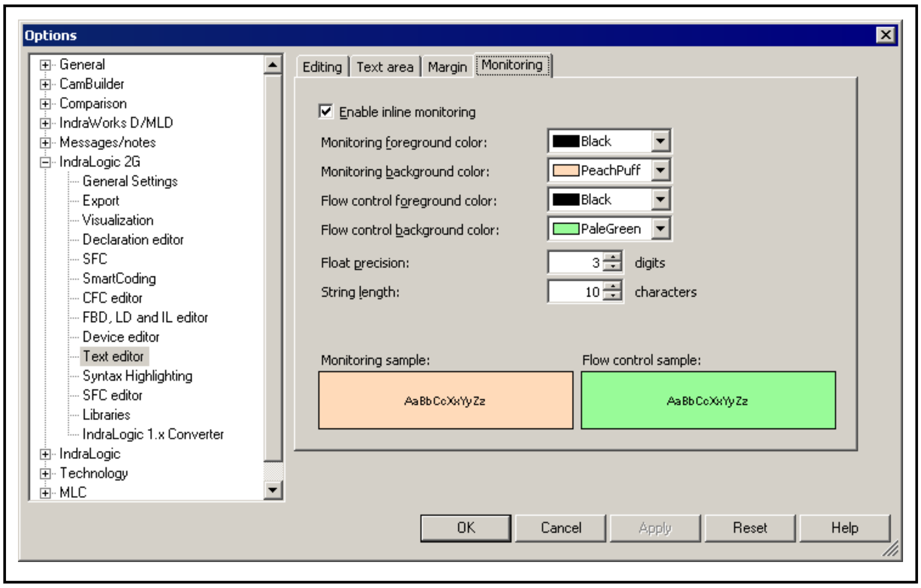
Select fold: All of the lines in the parentheses-enclosed area are selected.

Toggle fold: The parentheses-enclosed area, or in the case of nested enclosed areas, the first level of the parentheses-enclosed area, is opened or closed.

Toggle fold fully: All levels of a nested parentheses-enclosed area are opened or closed.

Options, text editor, monitoring

Enabling/disabling Inline monitoring, display of monitoring fields



☑ Enable inline monitoring

Monitoring foreground color: Black

Monitoring background color: PeachPuff

Flow control foreground color: Black

Flow control background color: PaleGreen

Float precision: 3 digits

String length: 10 characters

Monitoring sample: AaBbCcXxYyZz

Flow control sample: AaBbCcXxYyZz

Fig.3-181: Options, text editor, monitoring

Enable Inline monitoring: The display of monitoring fields after variables in a text editor in online mode can be enabled and disabled.

Foreground color: The value in the monitoring field is displayed in the selected color.

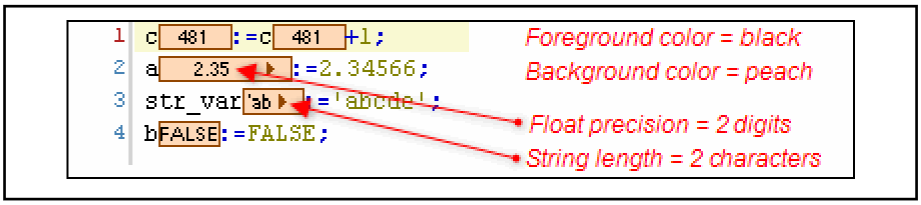
Background color: The background in the monitoring field is displayed in the selected color.

Sequence control foreground color: The value in the monitoring fields at the sequence control positions is displayed in the selected color.

Sequence control background color: The background of the monitoring fields at the sequence control positions is displayed in the selected color.

Floating point precision:	Numbers with decimal points in the monitoring field are displayed to the number of places set here.
String length:	String variable values are displayed in the monitoring field to the maximum length entered here (number of characters)
.. Preview:	Preview of the values currently set for a monitoring field

Preview of the values currently set for a monitoring field



```
1 c 481 :=c 481 +1;
2 a 2.35 :=2.34566;
3 str_var 'ab :='abcde';
4 bFALSE:=FALSE;
```

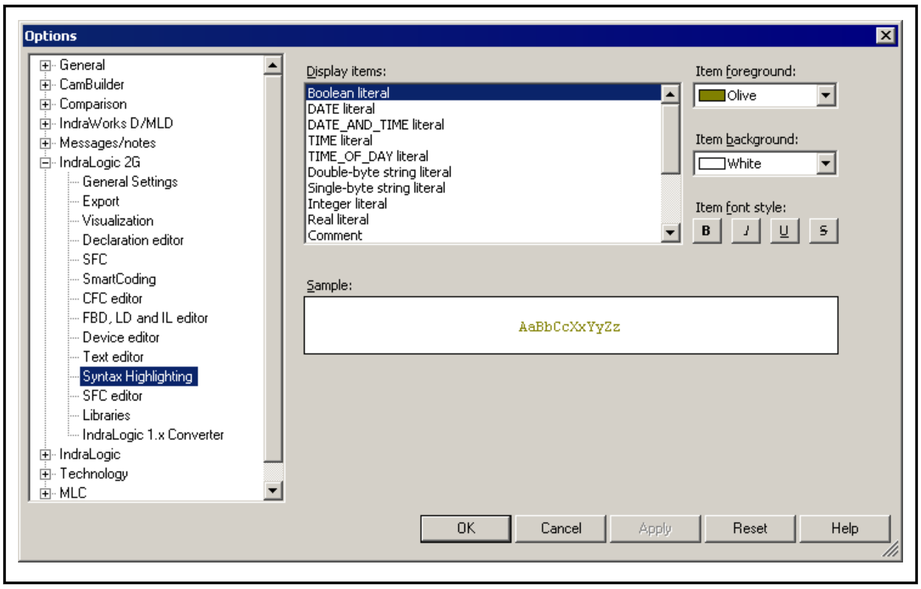
Fig.3-182: Example of Inline monitoring in the ST editor

For a preview with the values just set for the sequence control refer to Sequence control in the different language editors, page 148.

Options, Syntax Highlighting

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Syntax Highlighting.**

This dialog contains color and font settings for the various text elements in an editor (e.g. operands, Pragmas, comments, etc.).



Item foreground: Olive

Item background: White

Sample: AaBbCcXxYyZz

Fig.3-183: "Options" dialog, syntax highlighting category

Both color and font for the text editor display can be set for the text elements listed in **Display items**. A selection list can be used to set the color for the **item foreground** and the **item background**. The **item font style** is defined by clicking the desired symbol for "bold" (**B**), "italic", "underline" and "strikethrough".

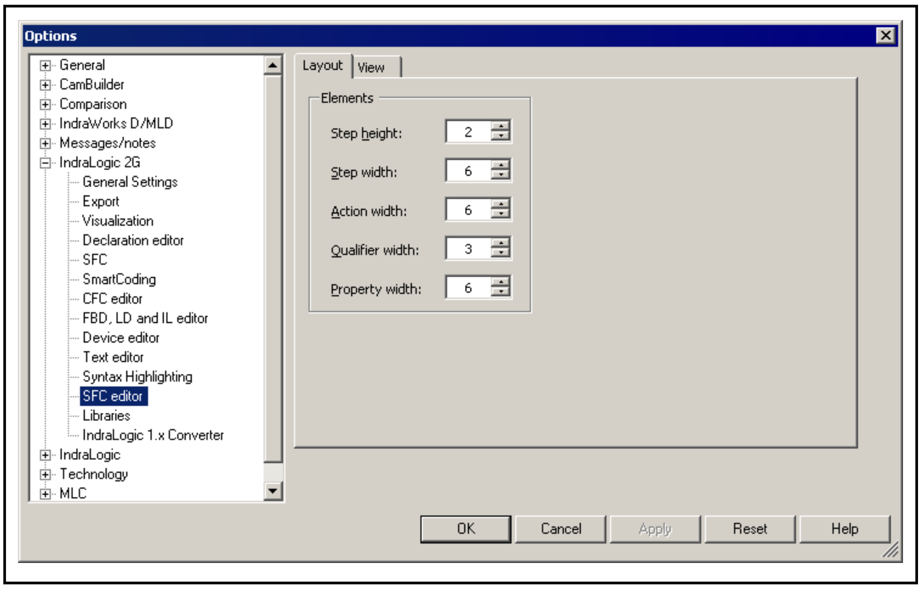
An example of the current settings can be seen in the **Sample (preview)** window.

Options, SFC Editor

Menu: **Tools** ▸ **Options** ▸ **SFC (sequential function chart) editor.**

The dialog contains tabs in which the settings for the SFC editor, page 399, can be made.

Layout



Step height: 2

Step width: 6

Action width: 6

Qualifier width: 3

Property width: 6

Fig.3-184: "SFC editor options" dialog, layout

In the "Layout" tab, the values are entered in "grid units".

1 grid unit = font size currently set in the text editor options (Text area/Font).

Step height: Height of a step element. Possible values: 1-100.

Step width: Width of a step element. Possible values: 2-100.

Action width: Width of an action element. Possible values: 2-100.

Qualifier width: Width of the qualifier. Possible values: 2-100.

Property width: Width of the display area for the step properties. Possible values: 2-100.

The settings are always effective immediately in all currently open SFC editor windows.

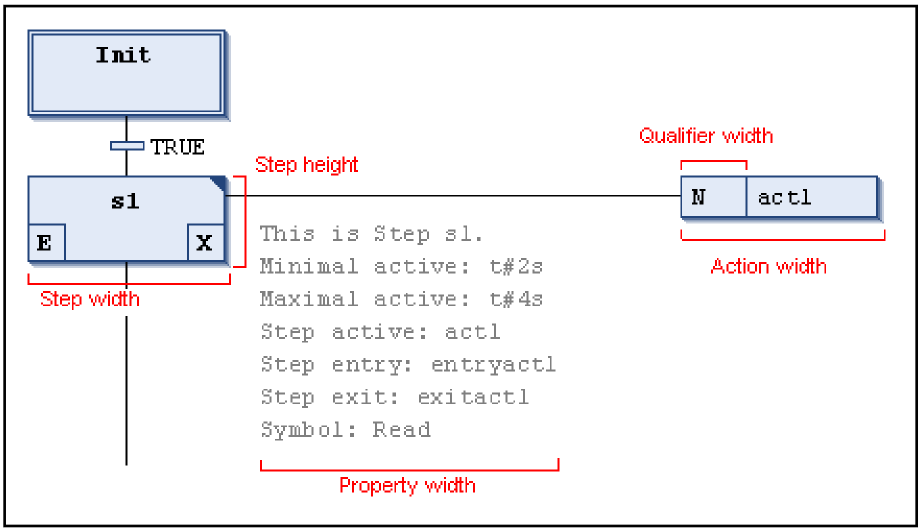
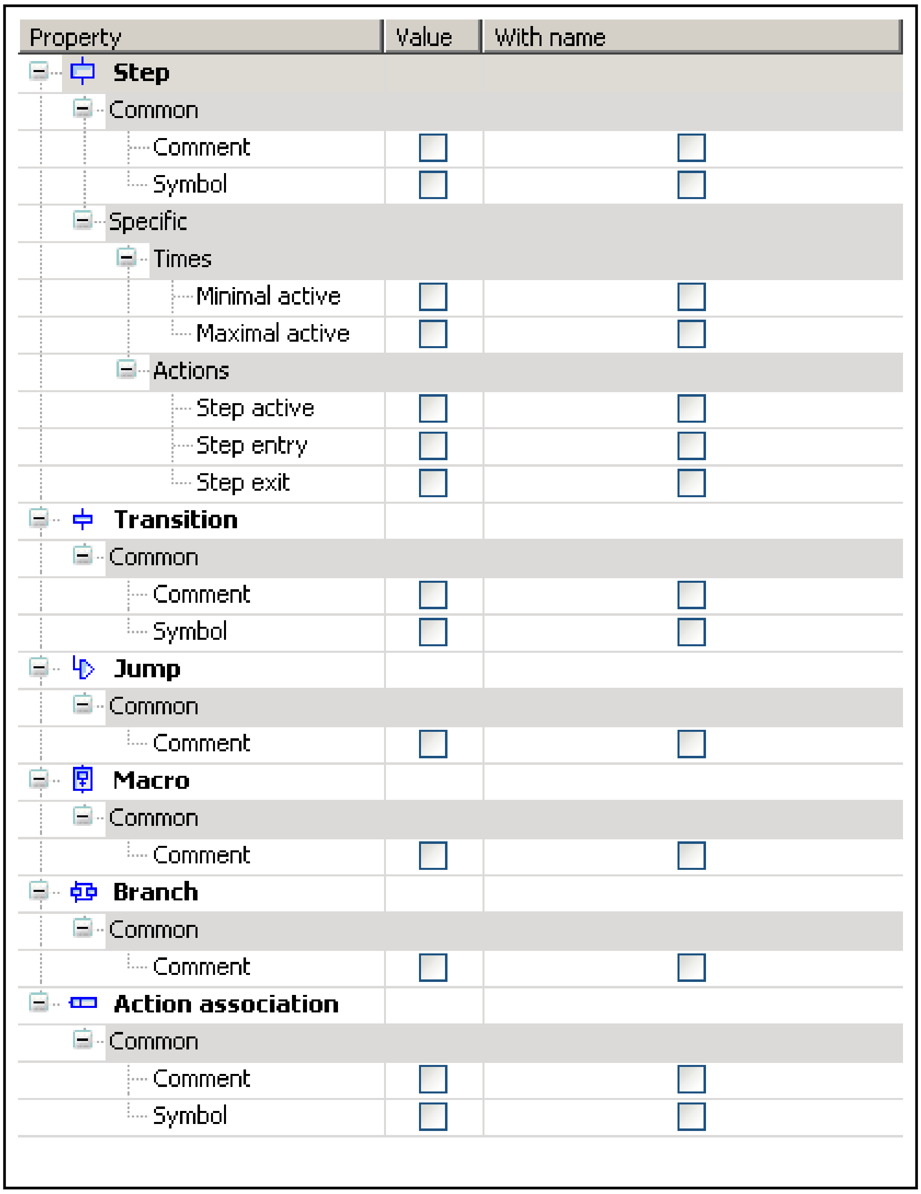


Fig.3-185: Setting parameters for the layout in SFC

View

The "View" tab is described in the following:



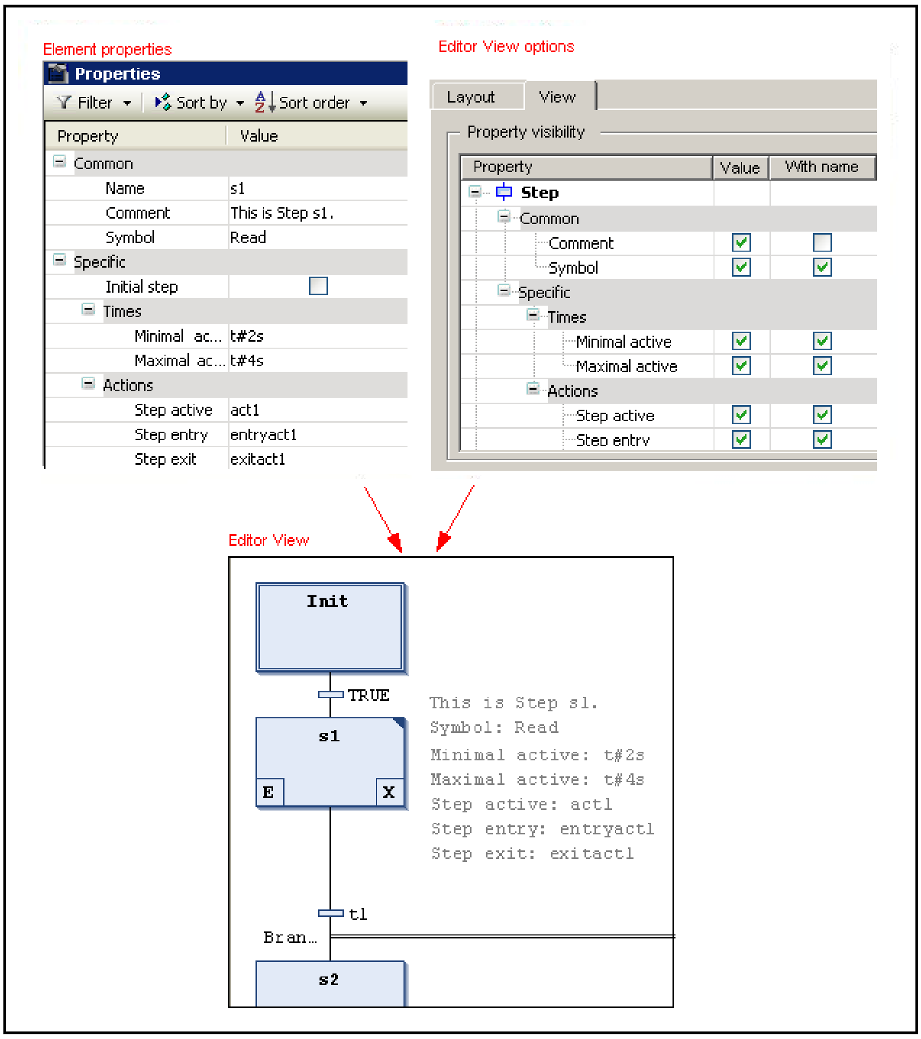
Property	Value	With name
Step		
Common		
Comment	□	□
Symbol	□	□
Specific		
Times		
Minimal active	□	□
Maximal active	□	□
Actions		
Step active	□	□
Step entry	□	□
Step exit	□	□
Transition		
Common		
Comment	□	□
Symbol	□	□
Jump		
Common		
Comment	□	□
Macro		
Common		
Comment	□	□
Branch		
Common		
Comment	□	□
Action association		
Common		
Comment	□	□
Symbol	□	□

Fig.3-186: Visibility of the properties, SFC editor options, view

Visibility of properties:

Define here which element properties are to be displayed next to the element in the SFC chart. For each property, checkmarks can be placed in the "Value" column and in the "With name" column. If the "Value" column is selected, the value is shown in the chart. If the "With name" column is selected, the property name is shown.

Example:



Element properties

Properties

Property	Value
⊟ Common	
Name	s1
Comment	This is Step s1.
Symbol	Read
⊟ Specific	
Initial step	□
⊟ Times	
Minimal ac...	t#2s
Maximal ac...	t#4s
⊟ Actions	
Step active	act1
Step entry	entryact1
Step exit	exitact1

Editor View options

Property visibility

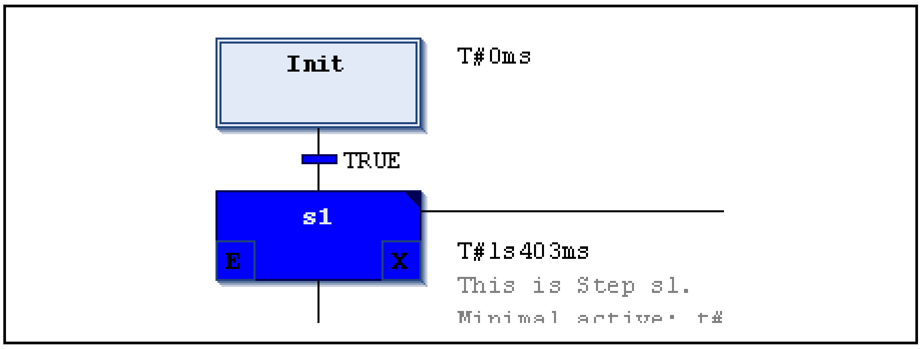
Property	Value	With name
⊟ Step		
⊟ Common		
Comment	☑	□
Symbol	☑	☑
⊟ Specific		
⊟ Times		
Minimal active	☑	☑
Maximal active	☑	☑
⊟ Actions		
Step active	☑	☑
Step entry	☑	☑

Editor View

```
This is Step s1.
Symbol: Read
Minimal active: t#2s
Maximal active: t#4s
Step active: act1
Step entry: entryact1
Step exit: exitact1
```

Fig.3-187: Element properties

Online: If the **Show step time** option is selected, in online mode the current step time is displayed to the right of all of the steps for which time properties have been defined.



```
T#0ms
T#1s403ms
This is Step s1.
Minimal active: t#
```

Fig.3-188: Example display of a step time in online mode

Options, Libraries

Menu: **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Libraries.**

The "**Mappings**" of **library references** that are used when converting an "old" project are managed in this dialog. Until a "mapping" for a specific library is saved here, define it every time an old project is opened that includes this library.

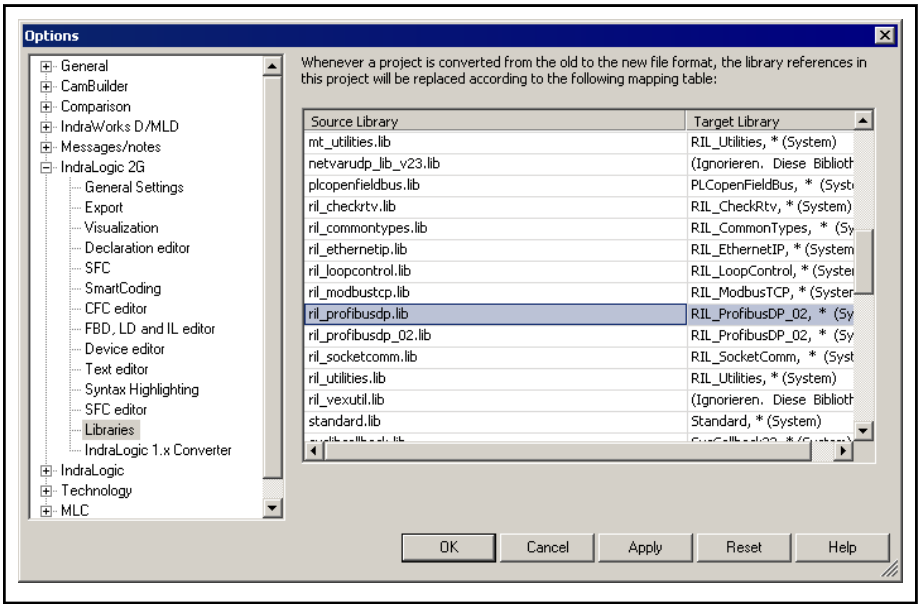
See information on opening and converting projects in the current format, page 116.

See information on library management, page 83.

A "mapping" defines how a library reference should appear after the project is converted into the current format. There are three possibilities:

- The reference is kept, i.e. the library is also converted into the current format (*.library) and installed in the local library repository.
- The reference is replaced by another, i.e. one of the installed libraries replaces the one that was previously included.
- The reference is deleted, i.e. the library is no longer included in the converted project.

If the "mapping" is saved, it is added to the option dialog in the 'Libraries' category:



Whenever a project is converted from the old to the new file format, the library references in this project will be replaced according to the following mapping table:

Source Library	Target Library
mt_utilities.lib	RIL_Utilities, * (System)
netvarudp_lib_v23.lib	(Ignorieren. Diese Biblioth
plcopenfieldbus.lib	PLCopenFieldBus, * (Syst
ril_checkrtv.lib	RIL_CheckRtv, * (System)
ril_commontypes.lib	RIL_CommonTypes, * (Sy
ril_ethernetip.lib	RIL_EthernetIP, * (System
ril_loopcontrol.lib	RIL_LoopControl, * (Syste
ril_modbustcp.lib	RIL_ModbusTCP, * (Syster
ril_profibusdp.lib	RIL_ProfibusDP_02, * (Sy
ril_profibusdp_02.lib	RIL_ProfibusDP_02, * (Sy
ril_socketcomm.lib	RIL_SocketComm, * (Syst
ril_utilities.lib	RIL_Utilities, * (System)
ril_vexutil.lib	(Ignorieren. Diese Biblioth
standard.lib	Standard, * (System)

Fig.3-189: Options for libraries

All mappings listed here are applied to the library references of the next old project that is converted. Thus, the mapping definition does not have to be repeated if the same library is included again in a project to be converted.

Each "mapping" is displayed on one line:

Source library: Library path included in the project before conversion

Target library: Name and storage location of the library that is to be included in the project after conversion. If the reference is to be removed, this is shown in the text "(Ignore. This library reference will not be converted.)".

The mappings list can be edited: To do this, double-click on the respective field or select a field and press <Enter>.

When editing a source library entry, the ... button opens the standard browse dialog in which it can be searched for a library in the "old" format.

When editing a target library entry, the 'Set target library' dialog opens.

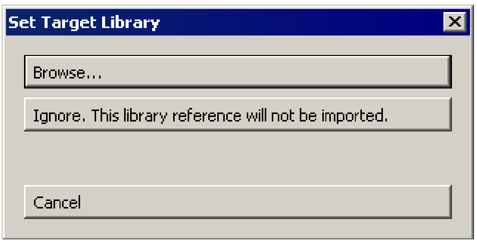


Fig.3-190: 'Set Target Library' dialog

To open a library in the current format, the **Search** button can be used to open the **Select Library** dialog where one of the libraries installed in the local library repository can be selected.

It can be sorted according to company and category, as in the library repository dialog, page 185,.

Use the <Ignore. ...> button to specify that the present source library is to be removed from the project forever when the project is converted.

A **new entry can also be added** to the list directly from this dialog by editing the last line, "**(Insert new mapping here.)**".

Options, Converter for IndraLogic 1.x Projects

This dialog can be accessed using **Tools** ▸ **Options** ▸ **Converter for IndraLogic 1.x projects.**

The "mapping" for converting IndraLogic 1.x projects is defined in this dialog, i.e. it is determined how the components of a project are handled when they are transferred into the current project format.

In the 'Device' tab the **conversion of the device references** (which target system is used) is described:

Until a "mapping" is defined for a device in this dialog, you are prompted to define one if an old project that used this device is opened.

See information on

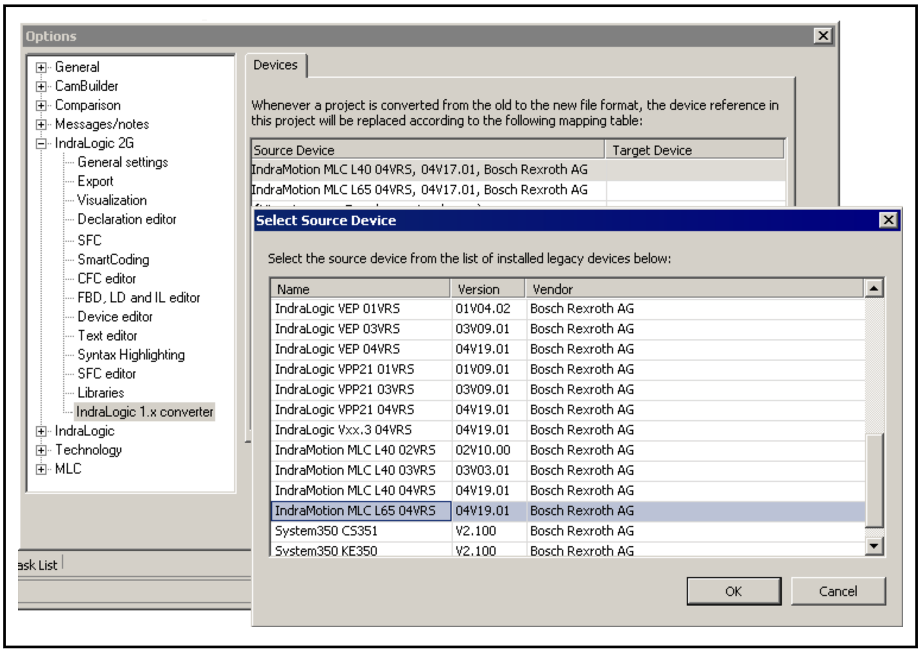
- opening and converting projects in the current format, page 116.
- handling devices in the Project Manager, page 63.

A "device conversion mapping" defines how the reference should appear on a target system after the project is converted, i.e. a "**target device**" (device in the new project) is assigned a "**source device**" (original device in the old project). There are two possibilities:

- In the device tree for the converted project, the source device is replaced by the target device that is currently available on the system, i.e. is installed.
- The source device is not converted (is ignored), i.e. it will not appear in the device tree for the new project. In this case, even application-specific objects such as the task configuration are not transferred into the new project.

A device mapping can be defined and saved ...

- ... while converting an old project that uses the device (see the description of the Data transfer, page116 command)
- ... or by direct entry here in the converter option dialog on the 'Devices' tab:



Whenever a project is converted from the old to the new file format, the device reference in this project will be replaced according to the following mapping table:

Source Device	Target Device
IndraMotion MLC L40 04VRS, 04V17.01, Bosch Rexroth AG	
IndraMotion MLC L65 04VRS, 04V17.01, Bosch Rexroth AG	

Select the source device from the list of installed legacy devices below:

Name	Version	Vendor
IndraLogic VEP 01VRS	01V04.02	Bosch Rexroth AG
IndraLogic VEP 03VRS	03V09.01	Bosch Rexroth AG
IndraLogic VEP 04VRS	04V19.01	Bosch Rexroth AG
IndraLogic VPP21 01VRS	01V09.01	Bosch Rexroth AG
IndraLogic VPP21 03VRS	03V09.01	Bosch Rexroth AG
IndraLogic VPP21 04VRS	04V19.01	Bosch Rexroth AG
IndraLogic Vxx.3 04VRS	04V19.01	Bosch Rexroth AG
IndraMotion MLC L40 02VRS	02V10.00	Bosch Rexroth AG
IndraMotion MLC L40 03VRS	03V03.01	Bosch Rexroth AG
IndraMotion MLC L40 04VRS	04V19.01	Bosch Rexroth AG
IndraMotion MLC L65 04VRS	04V19.01	Bosch Rexroth AG
System350 CS351	V2.100	Bosch Rexroth AG
System350 KE350	V2.100	Bosch Rexroth AG

Fig.3-191: IndraLogic converter options

To edit a field in the mapping table, double-click on the field or select it and press <Enter>. Assign each "source device" a "target device":

For entries in the "source device' column, the '**Select original device**' dialog opens, where all devices that the converter can handle are listed. Select the desired device to insert in into the field.

For entries in the 'target device' column, the '**Select target system**' dialog opens, where one of the currently installed devices can be selected. The dialog is managed like the Device database dialog.

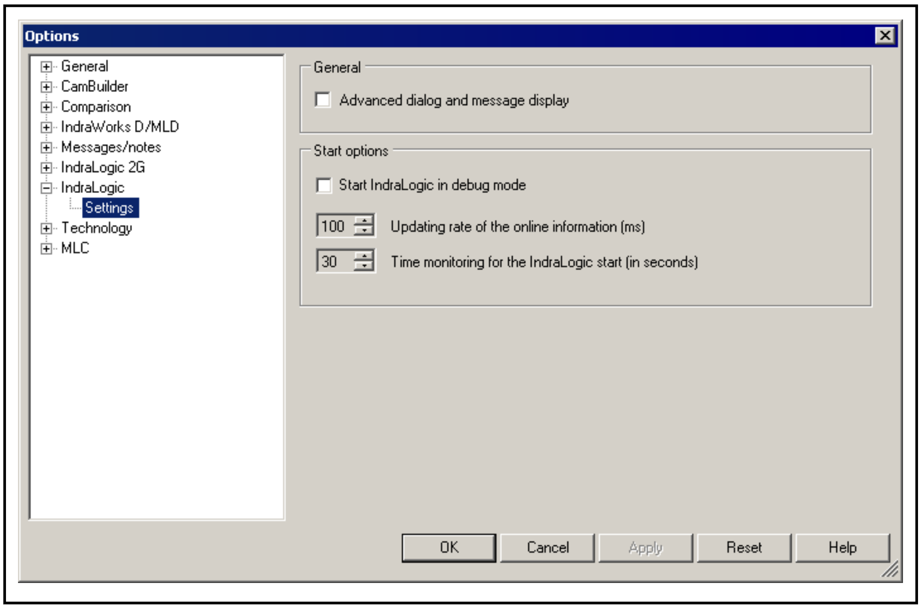
☞ If '**None**' is selected, the source device is **not be available in the new project**, which also means that application-specific objects such as the task configuration are not transferred!

All device mappings listed in the Options dialog are applied the next time an old project is converted. In this way, the mapping definition does not have to be repeated if the same device is used in several projects.

Options, IndraLogic 1.x, Settings

This dialog can be accessed using **Tools** ▸ **Options** ▸ **IndraLogic (1.x), settings.**

☞ These options are only in effect when working with IndraLogic 1.x!



General

☐ Advanced dialog and message display

Start options

☐ Start IndraLogic in debug mode

100 Updating rate of the online information (ms)

30 Time monitoring for the IndraLogic start (in seconds)

Fig.3-192: Options, IndraLogic 1.x settings

General

- Extended dialog and message display

  ### in preparation ###

Start options

- Start IndraLogic in debug mode

  If a project was created in IndraWorks with a 1.x control, double-click the "Logic" node (or its objects), IndraLogic 1.x opens. When it opens, information is displayed in the message window.

  If the checkmark is placed next to "Start IndraLogic in debug mode", these messages are also recorded. The "ILStartingMessages.txt" protocol is stored in the IndraWorks installation directory in the IndraLogic folder.

- Updating rate of the online information

  ### in preparation ###

- Time monitoring for the IndraLogic start

  When IndraLogic 1.x is opened, a monitoring period begins. If IndraLogic does not start within this time period, an error message is output.

3.7 Library Manager - Commands

3.7.1 Library Manager - Commands, General Information

The commands for the Library manager, page 367 can be called from its editor window. Furthermore, the "Add library" command can be accessed in the menu bar as soon as the editor is activated.

If required, the menu configuration can be changed using the dialog in **IndraWorks ▸ Tools ▸ Customize.**

- Add library..., page 227
- Properties..., page 230
- Reload library, page 231.

For general information on the IndraLogic 2G library management, see page 83.

3.7.2 Add Library...

By default, this command is available in the Library menu, page 371, and in the Library Manager window, page 368,.

Use the command to add a library to the project using the currently active library manager.

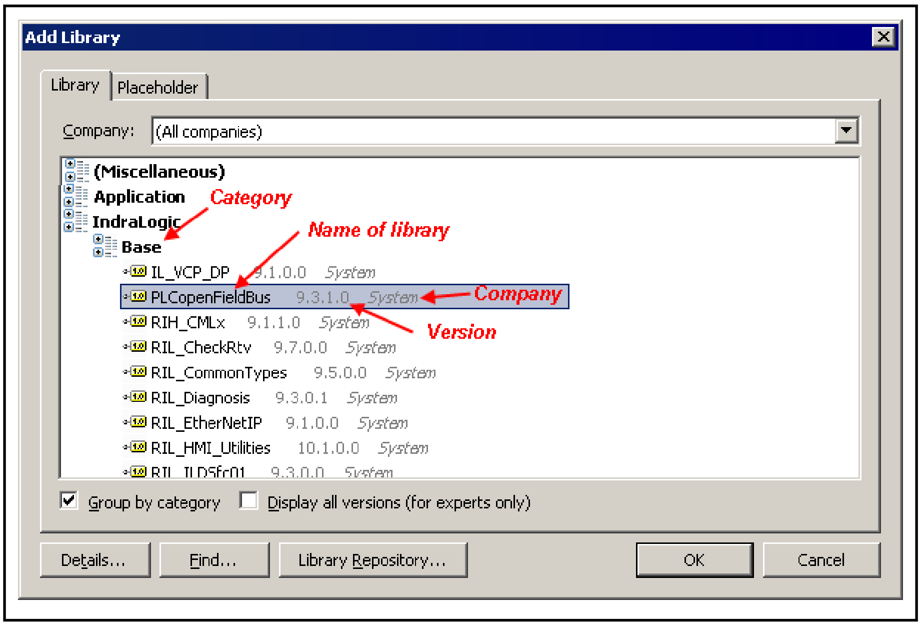
Only libraries installed in a local repository, page 185, can be added. Several versions of a library can be used simultaneously in the project using one or more Library managers.

The command opens the 'Add library' dialog.

To insert a library to your project by default only consider the **Library** tab dialog.

The **Placeholder** tab dialog is only required if a library project is currently created in which a target specific library has to be referenced and this library should not be specified yet.

'Library' tab dialog



Add Library

Company: (All companies)

```
(Miscellaneous)
Application
IndraLogic
  Base
    IL_VCP_DP           9.1.0.0   System
    PLCopenFieldBus     9.3.1.0   System
    RIH_CMLx            9.1.1.0   System
    RIL_CheckRtv        9.7.0.0   System
    RIL_CommonTypes     9.5.0.0   System
    RIL_Diagnosis       9.3.0.1   System
    RIL_EtherNetIP      9.1.0.0   System
    RIL_HMI_Utilities   10.1.0.0  System
    RIL_ILDSfc01        9.3.0.0   System
```

☑ Group by category □ Display all versions (for experts only)

Fig.3-193: 'Add library' dialog, 'Library' tab. Displays only the most recent version.

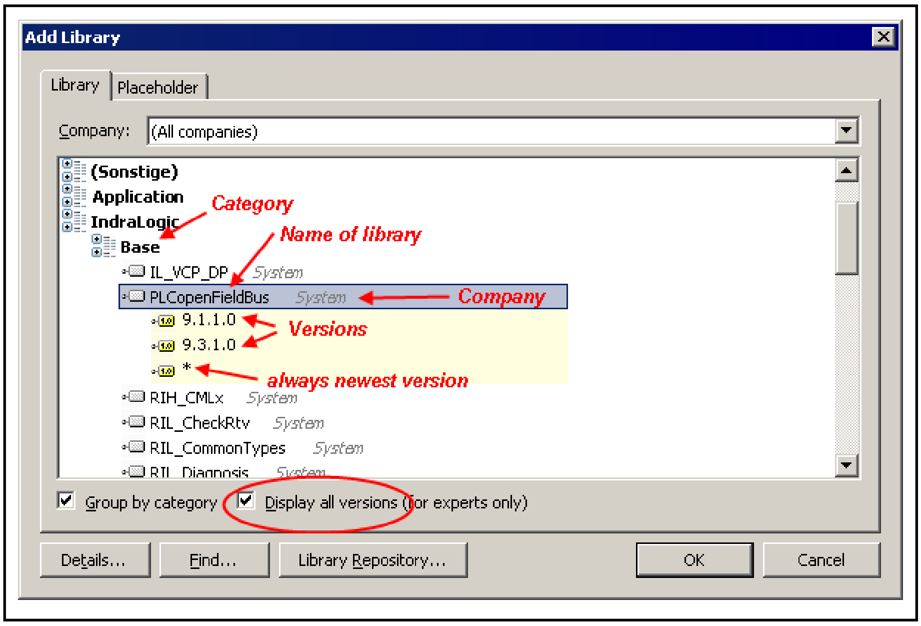


Fig.3-194: 'Add library' dialog, 'Library' tab. Displays all versions.

All installed libraries are shown here. The list can be filtered according to **company**. "(All companies)" displays all available libraries).

If the **Group by category** option is selected, the libraries for the company that is currently set are grouped according to the defined categories. The category designations appear as nodes. A single click on a node opens the list of the related libraries (**Library names**) or subcategories. A single click on a library name opens the list of the available **versions**. If the libraries are not grouped according to categories, the list of all of the libraries is sorted alphabetically.

Select the desired version of the library. A "*" appears in the list in addition to the currently installed versions of a library If the asterisk is selected instead of a concrete version of the library, the **latest version** available in the repository is always used in the project.

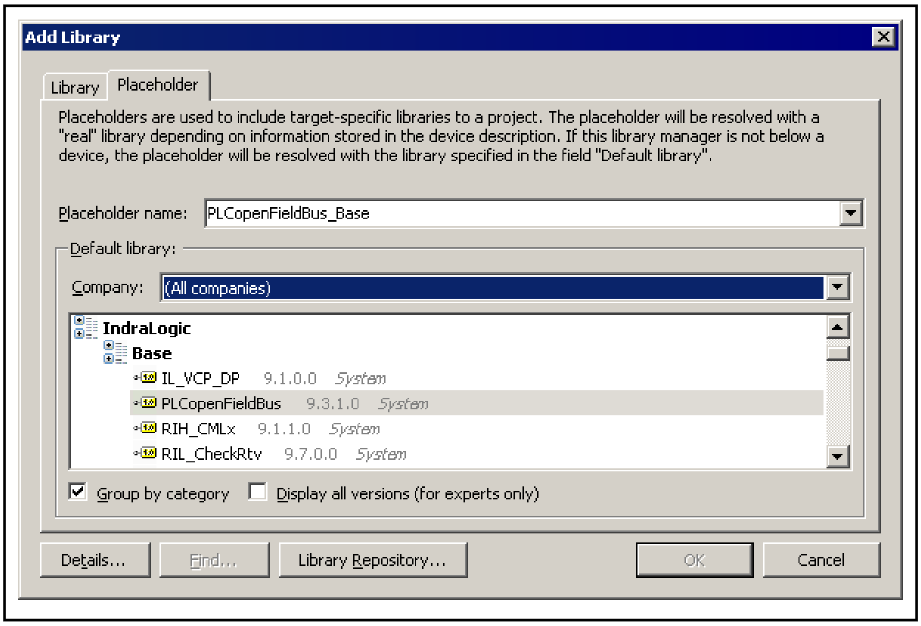
Select the desired library. If Display all versions (for experts only) is selected, all of the currently installed versions of a library are listed. A "*" also appears in the list; it stands for "Newest available version in the repository". In this case, you can also select among the various versions. However, this option is not selected by default. Thus, only the most recent version is displayed. In this case, **several libraries** can be selected: To do this, press and hold the <Shift> key down while selecting the desired libraries.

After confirming the selection with OK, the library version is added to the library manager.

The **Find...** button is used to find libraries containing the requested function blocks. A search dialog opens to enter the name of the function block as well as placeholders like "*" and "?".

To add a library that is not yet installed on the local system, use the **Library Repository** button to open the dialog of the same name in order to carry out the installation.

'Placeholder' tab dialog


Add Library

Library Placeholder

Placeholders are used to include target-specific libraries to a project. The placeholder will be resolved with a "real" library depending on information stored in the device description. If this library manager is not below a device, the placeholder will be resolved with the library specified in the field "Default library".

Placeholder name: PLCopenFieldBus_Base

Default library:

Company: (All companies)

```
IndraLogic
  Base
    IL_VCP_DP        9.1.0.0   System
    PLCopenFieldBus  9.3.1.0   System
    RIH_CMLx         9.1.1.0   System
    RIL_CheckRtv     9.7.0.0   System
```

☑ Group by category ☐ Display all versions (for experts only)

Fig.3-195: 'Add library' dialog, 'Placeholder'

Placeholder with a project

If any project is to be compatible with several exchangeable target devices, the device-specific libraries have to be integrated into the project library manager via placeholders.

As soon as a target device is specified, the placeholders are filled according to the related device description. But even if there is currently no device description available, the placeholders already allow for a syntactic project test.

To add a library to the library manager using placeholders, it has to be selected first from the list provided in the **Default library** section. The selection can be limited by specifying a **company**.

Apart from that, the **placeholder name** has to be entered in the corresponding field. In order to simplify correct entry, all placeholder names currently defined in device descriptions are provided in the selection list.

Placeholder within a library project

If a library project is based on other libraries which depend on the target device, these libraries have to be integrated into the library project by means of placeholders.

That means, a concrete library is not integrated **here**, but instead, just a placeholder. Later, when <library_xy> is used for a specific device in a project, this placeholder is replaced by the name of the device-specific library. However, this name has to be defined in the associated **device description file**. An entry has to be present there that assigns the name of an existing "real" library to the placeholder name.

If, for whatever reason, no device-specific library is available, the placeholder is replaced with the **Default library** entered here in this dialog. (This enables the library project to be compiled without errors, for example, even if there is no existing suitable device description at this point in time.)

In the **Placeholder name** field enter a character string as a name for the placeholder.

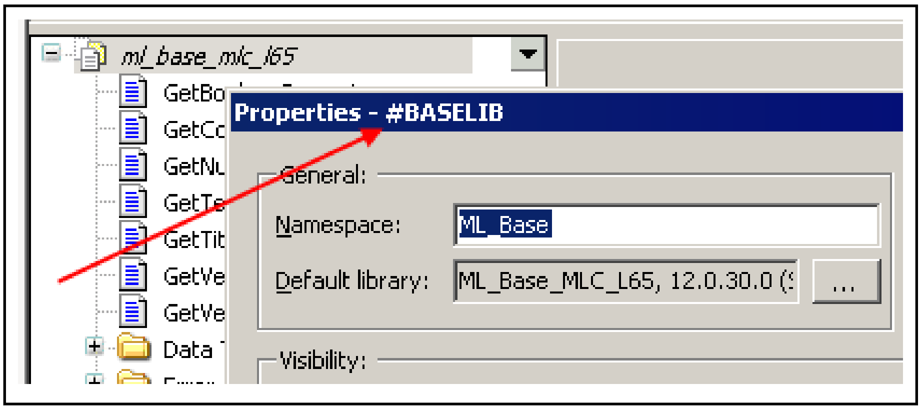
In addition, select one of the currently installed libraries as the **Default library**.

This procedure is like adding a library as in the 'Library' subdialog. As in that dialog, select the "Display all versions" (for experts only) option to display a list of all versions of a library currently installed on the system.

After closing the dialog with **OK**, the "Placeholder library" is added to the "Library Manager" tree.

If the Properties dialog, page 230, is opened for this entry, information on the assigned default library can be accessed via the "Default" field.

The placeholder has not yet been replaced in the following example; later, when the library manager is located below a device with the corresponding device description, the name of the device-specific library appears in the 'Name' field instead of the placeholder.



Properties - #BASELIB

General:

Namespace: ML_Base

Default library: ML_Base_MLC_L65, 12.0.30.0 (

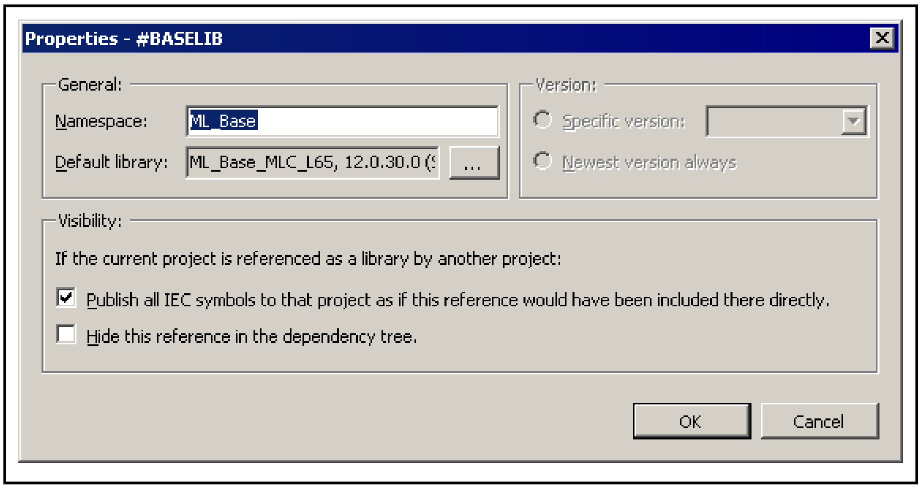
Visibility:

Fig.3-196: Example of a library placeholder in the library manager

3.7.3 Properties...

This command is available in the Library manager, page 368, editor window if a library entry is currently selected.

It opens the **Properties** dialog for the selected library and allows the following settings with regard to namespace, version management and behavior to be made in case the library is referenced in another library and is included in a project via the other library:



Properties - #BASELIB

General:

Namespace: ML_Base

Default library: ML_Base_MLC_L65, 12.0.30.0 (

Version:

○ Specific version:

○ Newest version always

Visibility:

If the current project is referenced as a library by another project:

☑ Publish all IEC symbols to that project as if this reference would have been included there directly.

□ Hide this reference in the dependency tree.

Fig.3-197: "Properties" dialog for a library

General information:

Namespace: The current namespace for the library is displayed. By default, it is identical with the library name, unless a different default namespace was explicitly defined in the project information when the library was created. The namespace can be modified for the local project in this "Properties" dialog at any time.

For further information on the namespace for libraries, see page 368.

Default: If a library placeholder is currently selected in the library manager, this field contains the name of the library that is to replace the placeholder when the device-specific library is available.

For further information on Library placeholders, see page 229,.

Version: Determine here which version of the library is to be used:

Specific version: Exactly this version is used. Select the desired version from the list.

Newest version always: The newest version of the library found in the library repository is always used. This means that the library modules actually used can change, since a newer version of the library is available.

Visibility: These settings are useful when the library is included in another library, i.e. is referenced there. By default, these settings are not selected.

Publish all IEC symbols to that (referencing) project...: As long as this option is not selected if library function blocks are later included in another library, they can be uniquely addressed by using the corresponding namespace path. This is a combination of the namespace of the "father" library and the its own namespace and is appended to the front of the function block name (prefix).

If, for example, the library "RIL_Utilities" references the library "Util" and "Util" contains the function block "BLINK", an instance "bk1" of "BLINK" has to be declared as follows:

```
bk1: RIL_UTILITIES.UTIL.BLINK;
```

For further information about using libraries, see page 83.

The option **should only be selected if** you wish to create a so-called "container library", i.e. a library that does not define its own blocks, but instead, only references other libraries to create a type of library bundle. This can be useful for including several libraries in a project simultaneously, by simply including the container library. In this case, however, it is desirable to position the individual libraries in the package at the top level in the project's library manager in order to shorten the access path for the library blocks. And that is exactly what can be achieved by selecting these options ahead of time.

Hide this reference in the dependency tree: If this option is selected, the library is later included in another library and this library is then added to a project, it is not be displayed in the Library Manager.

This allows **"hidden" libraries** to be added, but it has to be carried out consciously, since errors occur during compilation that are related to errors in the library. There might be problems finding the cause.

3.7.4 Reload Library

This command is available in the Library manager, page 368, editor window if a library is selected that was not loaded correctly when the project was opened.

If, for whatever reason, a library is not available at the defined repository path when a project is opened, a corresponding error message is output.

After having corrected the error and the library is properly available again, select the command 'Reload library' for this library. This way, reload the library without exiting the project.

3.8 Image Pool - Commands

3.8.1 Image Pool - Commands, General Information

Icon:

An image pool, page 61, is an object for managing images.

The "image pool" object is available in the "VI logic objects" folder in the "PLC objects" library. Use drag&drop to insert it below the "Application" node or in the global "General module" folder.

Alternatively, also use **Add ▸ Image pool** in the respective context menu; see the following figure.

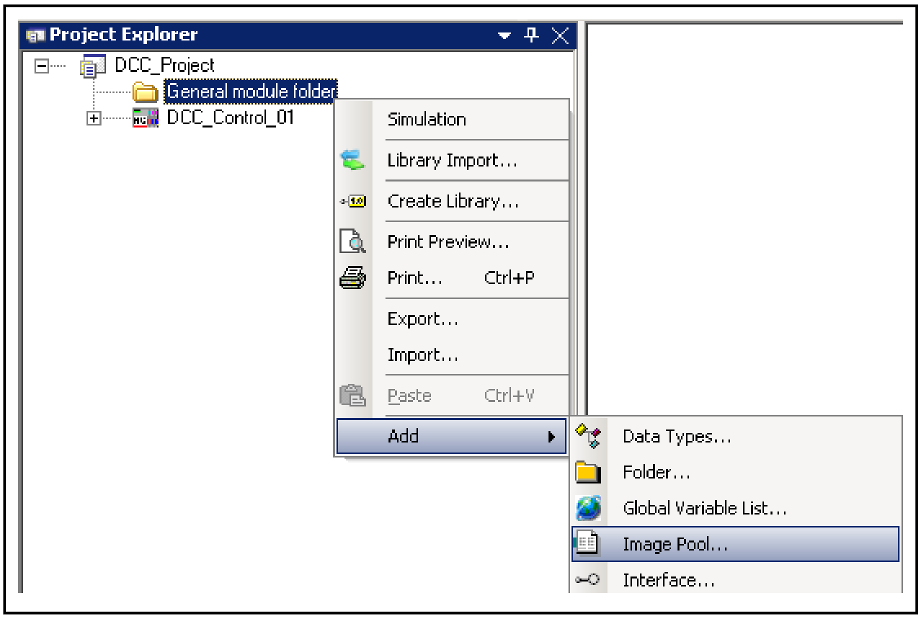


Fig.3-198: Adding an image pool into the project

The menu structure can be reconfigured using the Customize dialog.

Commands:

- Generate global image pool, page 232
- Generate image pool, page 233

3.8.2 Generating a Global Image Pool

☞ Note that "GlobalImagePool" is automatically generated in the "General module" folder when the first use of static image files is defined in a project.

In the main menu click on **VI Logic Visualization ▸ Generate global image pool**to create the global image pool "GlobalImagePool" manually.

Alternatively, **Generate global image pool** is also available in the context menu when the mouse is moved to the working area of a visualization.

image pool, page 61, contains more information on "image pools".

"GlobalImagePool" is a pool of static images that are used in visualizations.

3.8.3 Generating an Image Pool

### in preparation ###

3.9 Recipe Manager - Commands

3.9.1 Recipe Manager - Commands, General Information

A **recipe manager**, an object assigned to an application, can be assigned 1..n recipe definitions (see recipe manager, page 382).

Each of these **recipe definitions** is a table, the columns of which are "**recipes**" and the lines of which are "**variables**".

If a new recipe definition is assigned to the recipe manager, its initial set of columns comes from the recipe manager.

The following recipe manager commands are used to insert additional columns or delete columns or to insert new variables or delete variables.

The commands are available in the context menu when a recipe definition is being processed in the editor. In the basic configuration they are also available in the **VI Logic recipe definition** main menu.

If necessary, the menu structure can be modified in the dialog under **IndraWorks ▸ Tools ▸ Customize ▸ Commands ▸ Recipe Manager.**

Commands:

- Add recipe, page 233
- Add variable, page 233
- Remove variable, page 234
- Remove recipe, page 234

3.9.2 Add Recipe

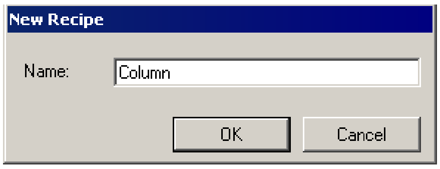
Icon:

This command is used to add a new column for a recipe, page 384 to the recipe definition currently being processed.

Click on **VI Logic recipe definition ▸ Add recipe**to add a new recipe.

Alternatively, the command is also available in the context menu.

The "New Recipe" dialog opens to enter a name for the recipe; see the following figure.



Name: Column

Fig.3-199: New recipe dialog

☞ A new recipe can also be created in online mode using a correspondingly configured visualization element; see "Visualization elements - Properties", page 451.

3.9.3 Add Variable

Icon:

Use "Add variable" to add a new line for a variable entry at the end of the recipe definition that is currently open.

Enter the desired variable in the Variable column; see also recipe definition, page 384.

Click on **VI Logic recipe definition ▸ Add variable**to add a new variable. Alternatively, the command is also available in the context menu.

3.9.4 Remove Recipe

Icon:

"Remove recipe" is used to delete a recipe from the currently open recipe definition, see also "recipe definition", page 384.

Select one of the fields in the recipe column and click on **VI Logic recipe definition ▸ Remove recipe**to remove a recipe. Alternatively, the command is also available in the context menu. The column is removed.

> ☞ Recipes can also be deleted in online mode using a correspondingly configured visualization element; see "Visualization elements - Properties", page 451.

3.9.5 Remove Variables

Icon:

Use "Remove variable" to delete one or more variables from the currently open recipe definition, see also "recipe definition", page 384.

Highlight one of the fields in the line that describes the variable or highlight several lines while holding down the <Ctrl> key and click on **VI Logic recipe definition ▸ Remove variable**to remove variables.

Alternatively, the command is also available in the context menu. All selected lines are removed.

3.10 Object Management - Commands

3.10.1 Object Management - Commands, General Information

The commands for managing objects in a project are described in this section. By default, these commands are included in the context menu of the relevant object, suitable for the object.

The commands:

- Add object, page 234
- Edit object, page 236
- Set active application, page 237.

3.10.2 Adding an Object

This command provides access to the existing object types.

The only object types listed are those that can be inserted at the current position in the Project Explorer.

For example, a data server object can only be inserted if "Application" is selected in the Project Explorer or an action can only be inserted if a function block or program is currently selected.

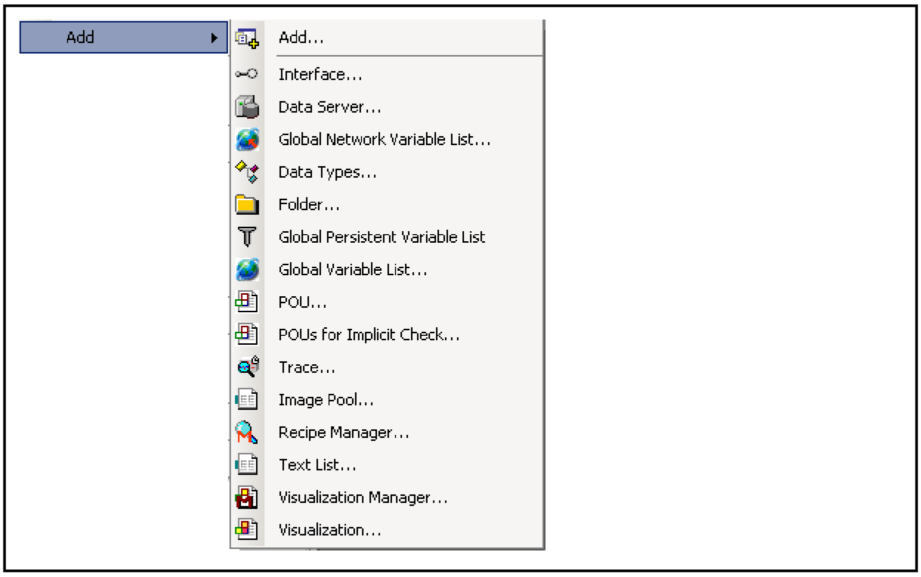


Fig.3-200: Adding an object

Select the desired object type using the respective context menu and in the following "Add object" dialog, define the **Name** and the respective settings for the object.

Please note the Recommendations for names, page 505 to achieve a certain amount of consistency.

The settings that need to be made depend on the object type.

For more information, see the help pages for object type or the corresponding editor:

- Action, page 51
- Application, page 66
- Library manager, page 367
- Image pool, page 61
- Data source, page 318
- Data server, page 71,
- DUT, page 43
- Property, page 46
- Device, page 63
- Global variables list, page 367
- Method, page 45
- Persistent variables, page 54
- POU, page 28
- POUs for implicit checks, page 63,
- Recipe definition, page 384
- Recipe manager, page 382
- Interface, page 49
- Step, page 405,
- Symbol configuration, page 306
- Task configuration, page 423
- Task, page 428

- Text list, page 55
- Transition, page 405,
- Trace, page 431,
- Visualization, page 445
- Visualization manager, page 496.

To **rename** an object in the Project Explorer, click on the object entry to open an editing field or use the <F2> shortcut.

☞ A device object has to be added using the special command: Add device, page 331,.

3.10.3 Opening an Object

Icon:

This command can be used if an object located in the Project Explorer is to be opened for examination or editing.

When the object is selected in the Project Explorer, the "Open" command can be accessed in the context menu.

Alternatively, the object can also be opened by double-clicking it.

The object opens in the corresponding editor.

In online mode, a dialog opens which prompts again about the view to be used to display the object (e.g. implementation/instance 1...).

3.10.4 Implement Interface...

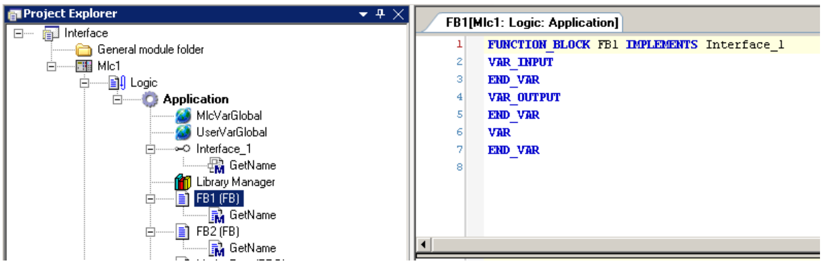
This command is used to update the implemented interfaces, page 38, for a function block.

This means that if the interface definition was modified, e.g. another method was added, this change can be moved into the function block by selecting the function block object in the POU or Project Explorer window and executing the 'Implement interfaces...' command.

Example:

1. Interface_1 contains the GetName interface method of type STRING.

   Upon creation of the FB1 and FB2 function blocks, Interface_1 was in each case implemented and thus, GetName is automatically inserted as POU method also below FB1 and FB2.



```
FUNCTION_BLOCK FB1 IMPLEMENTS Interface_1
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

Fig.3-201: FB1 and FB2 implement ITF1

2. Interface_1 is now extended by method GetID:

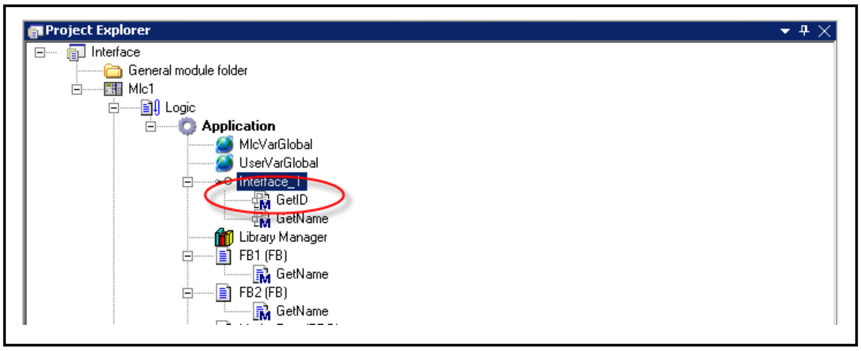


Fig.3-202: Interface modification

3. FB1 and interface ITF1 are updated with the "Implement interface" command (context menu):

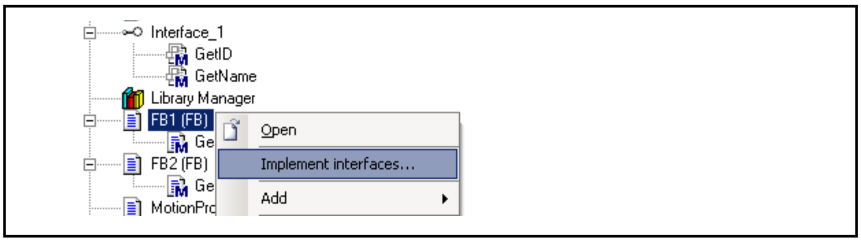
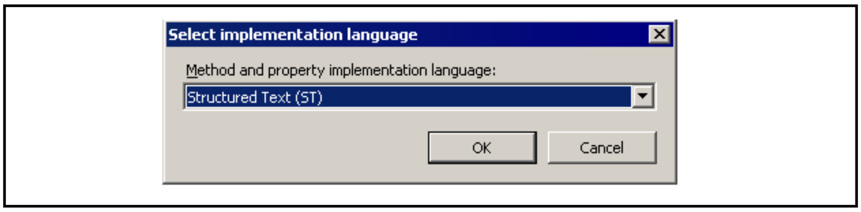


Fig.3-203: Updating the interface in FB1

4. The selection window for the implementation language of the new GetID method appears.


Select implementation language

Method and property implementation language: Structured Text (ST)

Fig.3-204: Selecting the implementation language

The GetID interface method becomes the GetID POU method of the FB1 function block.

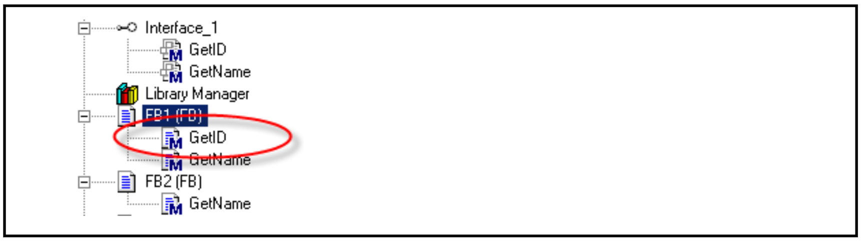


Fig.3-205: Interface in FB1 is updated

3.10.5 Set Active Application

This command is used to define the applications, page 66 located in the **Project Explorer** that are to be active.

If an application is currently selected in the Project Explorer the command can be selected in the context menu and by default in the project menu. After it is executed, the object entry of the application is displayed in bold in the tree.

☞ Note that in all cases online actions apply only to the "active application"!

3.10.6 Object Properties

Object Properties, General Information

If objects (e.g. devices, applications or visualization objects) are selected in the Project Explorer, information on these objects can be found in the "Properties" window.

Examples:

- Device properties, page 238
- Application properties, page 240
- Object properties, general information, page 242
- File properties, page 245
- Visualization object properties, general information, page 245
- GVL properties, network variables, page 246, sender
- GVL properties, network variables, page 248, receiver

Device Properties

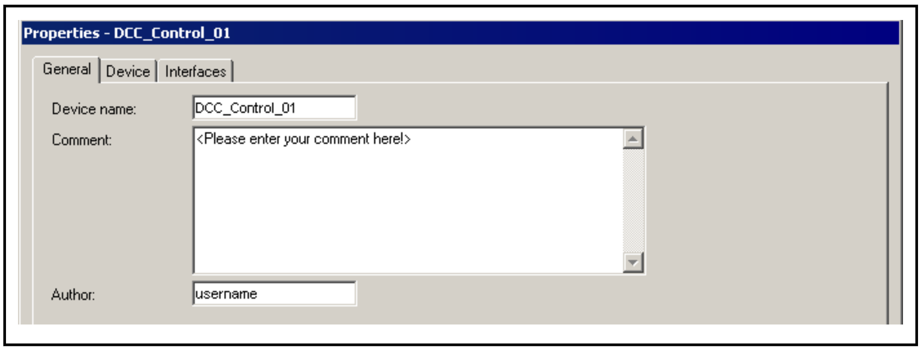
Icon:

Context menu: **Properties**.

The properties of the device currently selected in the Project Explorer are displayed on tabs. A few settings can also be changed in these dialogs:

Highlight the device node and activate the command to open the Properties dialog.

"General" tab



Device name: DCC_Control_01

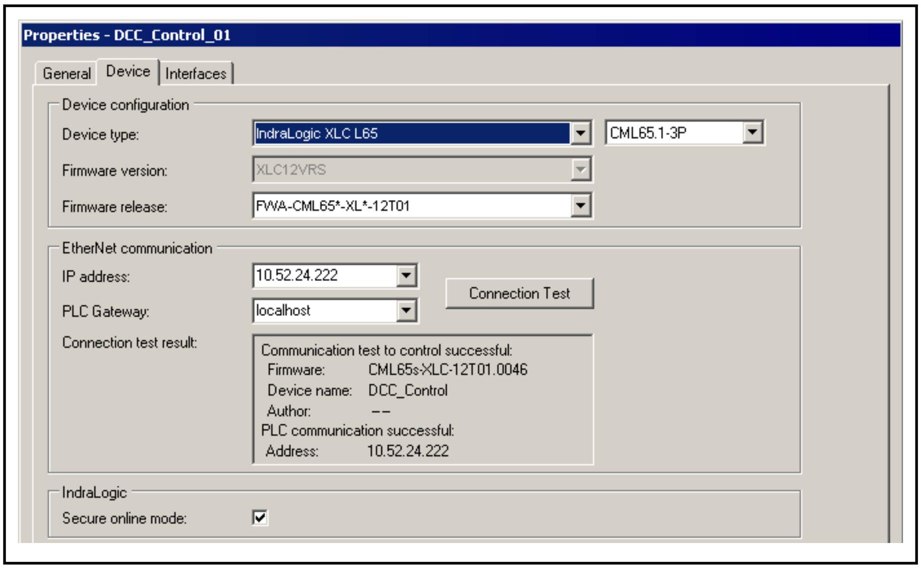
Comment: <Please enter your comment here!>

Author: username

Fig.3-206: General device properties

- **Device name:** Name of the device in the project tree, cannot be edited. The device name can only be changed in the project tree.
- **Comment:** Insert a comment here that describes the device in detail. This entry is optional.
- **Creator:** Enter the name of the project creator here. This entry is optional.

"Device" tab



Properties - DCC_Control_01

Device configuration

Device type: IndraLogic XLC L65 | CML65.1-3P

Firmware version: XLC12VRS

Firmware release: FWA-CML65*-XL*-12T01

EtherNet communication

IP address: 10.52.24.222

PLC Gateway: localhost

Connection test result:

Communication test to control successful:
Firmware: CML65s-XLC-12T01.0046
Device name: DCC_Control
Author: --
PLC communication successful:
Address: 10.52.24.222

IndraLogic

Secure online mode: ☑

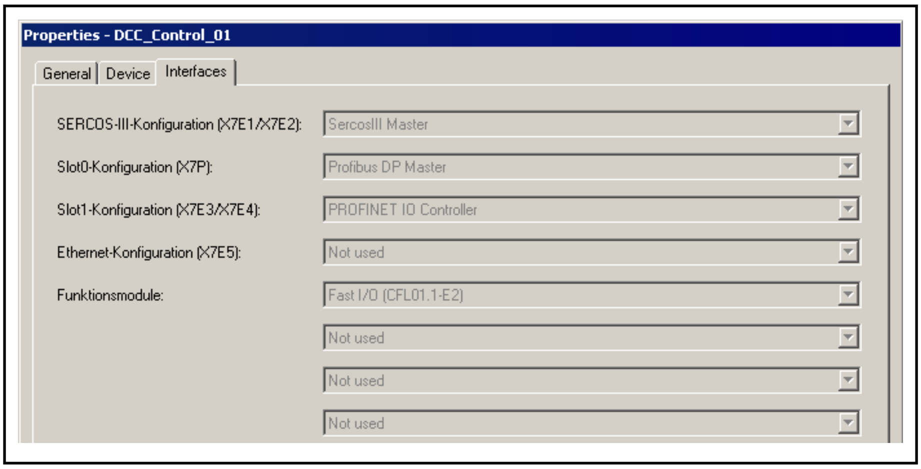
Fig.3-207: Device properties Device

With this dialog, the device IP address can be modified, the PLC Gateway can be set and a connection test (ping) to the device addressed can be carried out.

The **Secure online mode** option adds an additional prompt for activities that are relevant to security. See also Debugging, Overview, page 126.

"Interfaces" tab

The entries in this dialog are for information purposes only and cannot be edited.



Properties - DCC_Control_01

SERCOS-III-Konfiguration (X7E1/X7E2): SercosIII Master

Slot0-Konfiguration (X7P): Profibus DP Master

Slot1-Konfiguration (X7E3/X7E4): PROFINET IO Controller

Ethernet-Konfiguration (X7E5): Not used

Funktionsmodule: Fast I/O (CFL01.1-E2)
Not used
Not used
Not used

Fig.3-208: Interfaces tab

- **SERCOS III configuration (X7E1/X7E2):** Current SERCOS III configuration that can be selected when the device is added. The selection includes "Sercos III Master" or "Not used".
- **Slot0 configuration (X7P):** Current PROFIBUS configuration that can be selected when adding a device. "PROFIBUS DP Master", "PROFIBUS DP Slave" or "Not Used" can be selected.
- **Slot1 configuration (X7E3/X7E4):** The following communications protocols can be configured using this interface:

  PROFINET IO controller, PROFINET IO device, EtherNet/IP scanner, EtherNet/IP adapter, Ethernet interface or "Not used".

- **Ethernet configuration(X7E5):** Engineering interface that can be configured as EtherNet/IP adapter or "Not used" for exclusive engineering interface.
- **Function modules:** Depending on the control type, different function modules are available for extension.

  Here note the numbering, "function module 1" to "function module 4" which have to correspond to the switch positions for the slot numbers and the position of the function modules.

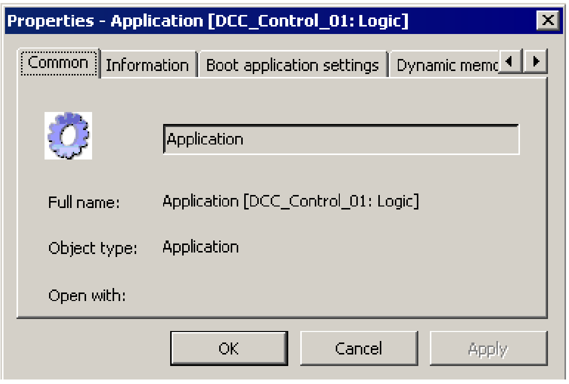
Application Properties

Icon:

Context menu: **Properties**.

Highlight the "Application" node and activate the command to open the "Properties" dialog.

"Common (general)" tab



Properties - Application [DCC_Control_01: Logic]

Application

Full name: Application [DCC_Control_01: Logic]

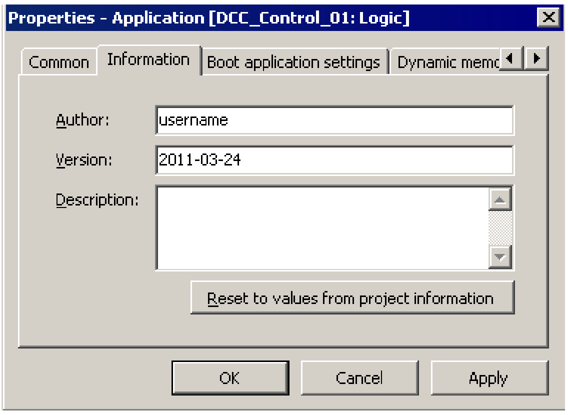
Object type: Application

Open with:

Fig.3-209: General object properties

- **Full name:** Name of the object as it appears in the Project Explorer.
- **Object type:** Type of object (e.g. "POU", "Application", "Interface" etc.),
- **Open with:** Type of editor in which the object is to be displayed or edited.

"Information" tab


Common
Information
Boot application settings
Dynamic memc
Reset to values from project information
OK
Cancel
Apply

Properties - Application [DCC_Control_01: Logic]

Author: username

Version: 2011-03-24

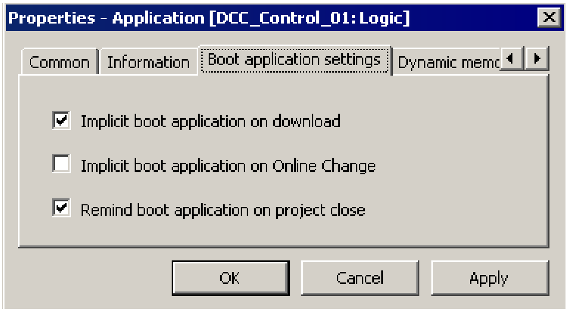
Description:

Fig.3-210: Information tab

The user can use this tab to store information about the application: author, version and description of the application.

"Reset to values from project information" deletes this information.

"Boot application settings" tab



Properties - Application [DCC_Control_01: Logic]

Common | Information | Boot application settings | Dynamic memo

☑ Implicit boot application on download

☐ Implicit boot application on Online Change

☑ Remind boot application on project close

Fig.3-211: Boot application settings tab

During every download the active application is automatically saved as a file called <application>.app in the target system directory.

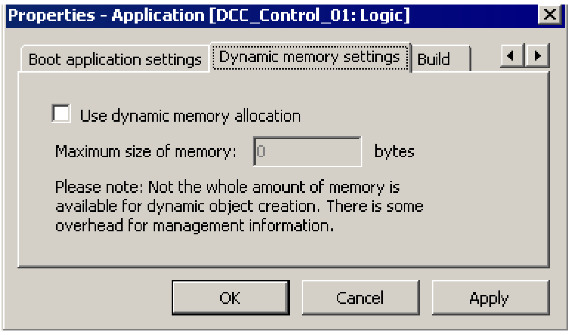
A boot application is started automatically when the control is started.

To do this, the application project on the control has to be available in a file <ProjectName>.app. To create the file, use **Debug ▸ Generating a boot application.**

In this tab, the user can specify how to handle the boot application file:

- Transfer the boot application implicitly into the control with each download.
- Transfer the boot application implicitly into the control with each online change.
- When the project is closed, ask the user if the file is to be transferred into the control.

"Dynamic memory setting" tab



Properties - Application [DCC_Control_01: Logic]

Boot application settings | Dynamic memory settings | Build

☐ Use dynamic memory allocation

Maximum size of memory: 0 bytes

Please note: Not the whole amount of memory is available for dynamic object creation. There is some overhead for management information.

Fig.3-212: Dynamic memory setting tab

"Use dynamic memory allocation":

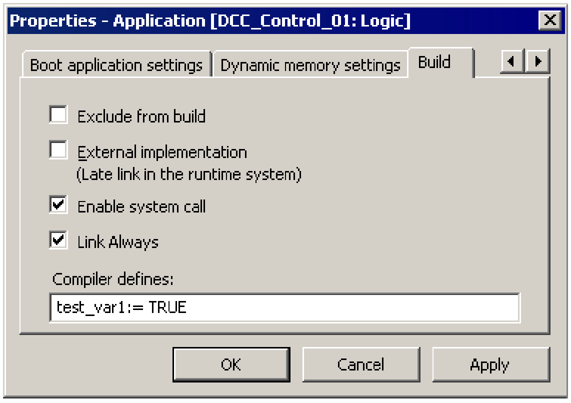
Select this option if memory is to be reserved for the application automatically. In this case, specify the desired "Maximum memory size" (bytes).

☞ Note that not the entire memory is available for the dynamic object generation but one part is used for management information.

The current settings are transferred to the control in every download or online change.

"Compile" tab

In the "Compile" tab, options regarding object compilation can be selected.



Properties - Application [DCC_Control_01: Logic]

Boot application settings | Dynamic memory settings | Build

□ Exclude from build

□ External implementation
(Late link in the runtime system)

☑ Enable system call

☑ Link Always

Compiler defines:

test_var1:= TRUE

Fig.3-213: Compile tab

- **Exclude from compilation**: The object is not considered in the next compilation run.
- **External implementation (late link in the runtime system)**: When the project is compiled, no code is generated for this object. The object is only considered if the project is running on the target system, providing the object is present there (e. g. in a library).
- **Enable system calls**: The reason is that in contrast to IndraLogic 1. x, the ADR operator can now be used with function, program, function block and method names and it replaces the INSTANCE_OF operator.

  For more information here, see Function pointers, page 557 in Data types.

  However, it is not possible to call function pointers within "IndraLogic 2G". To enable a system call (runtime system), the "Enable system calls" option has to be selected for the function object.
- **Always link**: The object is highlighted in the compiler. Thus, it is always included in the compile information. This means that it is always compiled and loaded to the PLC. This option is relevant if the object is located in an application or is referenced by libraries that are also located in an application. The compile information is also used as the basis for the selectable variables of the symbol configuration.
- **Compiler definitions**: "Defines" (see {define} instructions) and conditions for object compilation can be entered here (conditional compilation). In the "Declaration" section is a description of the available "conditional Pragmas", page 546.

  The "expr" expression, which is used in such Pragmas, can also be entered here and several entries are possible in a list of items separated by commas.

  For example, it might be desired to compile an application depending on a value of a certain variable. (In the example above: test_var1.)

Object Properties

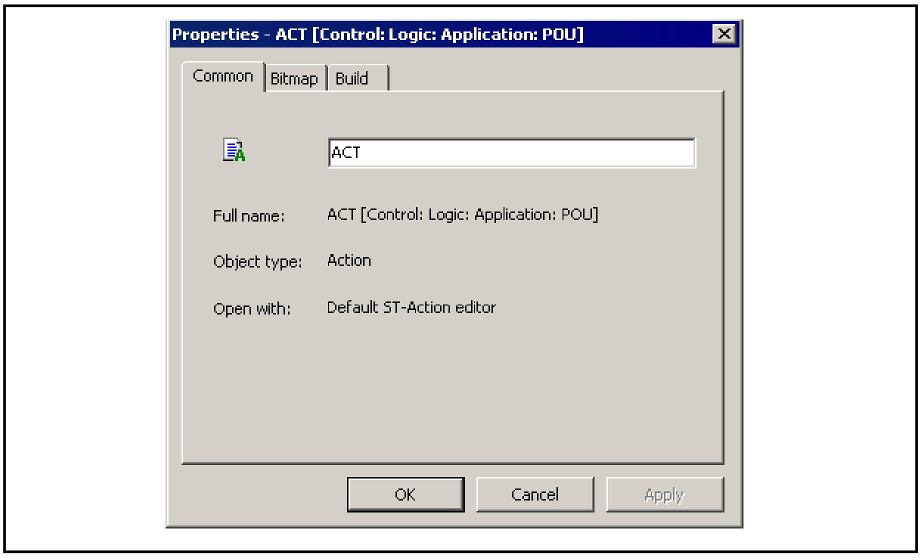
Icon:

Context menu: **Properties**.

Various objects in the Project Explorer have a "Properties" dialog in the context menu.

Highlight the desired object in the Project Explorer and activate the command to open the Properties dialog.

"Common (general)" tab


Properties - ACT [Control: Logic: Application: POU]

ACT

Full name: ACT [Control: Logic: Application: POU]

Object type: Action

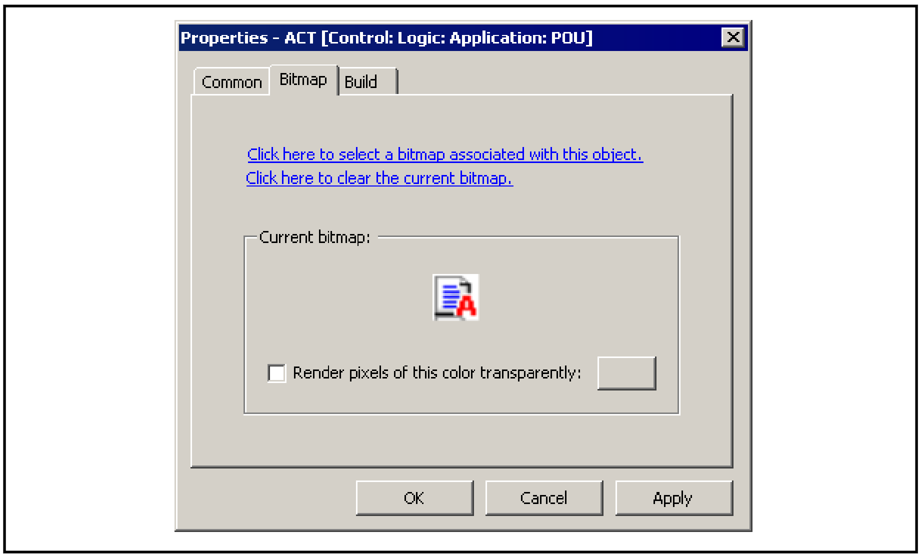
Open with: Default ST-Action editor

Fig.3-214: General object properties

- **Full name:** Name of the object as it appears in the Project Explorer. The object can be renamed here.
- **Object type:** Type of object (e.g. "Logic", "Program", "Task configuration" etc.),
- **Open with:** Type of editor in which the object is to be displayed or edited.

"Bitmap" tab

The "Bitmap" tab allows a bitmap to be assigned to the associated object, e.g. a symbol assigned to the graphical symbol in the function block diagram for a function block, an action or a program.


Properties - ACT [Control: Logic: Application: POU]

Click here to select a bitmap associated with this object.
Click here to clear the current bitmap.

Current bitmap:

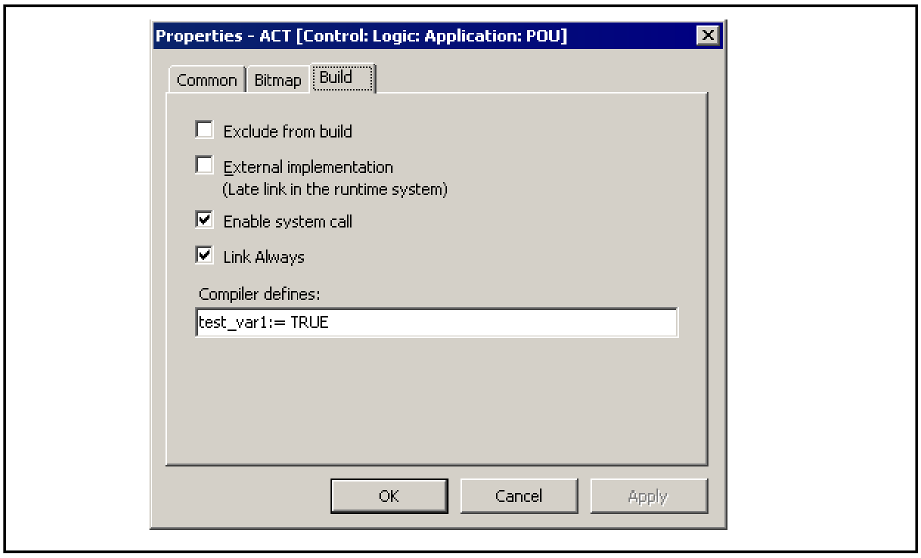
□ Render pixels of this color transparently:

Fig.3-215: Object properties, bitmap

Here, assign to or remove an image file from by means of which it is displayed in the graphical view of the library manager, page 367, and in the toolbox of the FBD/LD/IL editor, page 355,. Transparency of the figure can be obtained by selecting a color that is then displayed transparently. For that purpose, the "Transparency color" option has to be selected. Then, the default dialog for selecting a color can be opened using the rectangular button on the right side.

"Compile" tab

In the "Compile" tab, options on the object compilation can be selected.



Properties - ACT [Control: Logic: Application: POU]

□ Exclude from build

□ External implementation
(Late link in the runtime system)

☑ Enable system call

☑ Link Always

Compiler defines:

test_var1:= TRUE

Fig.3-216: Compile tab

- **Exclude from compilation**: The object is not considered in the next compilation run.
- **External implementation (late link in the runtime system)**: When the project is compiled, no code is generated for this object. The object is only considered if the project is running on the target system, providing the object is present there (e. g. in a library).
- **Enable system calls**: The reason is that in contrast to IndraLogic 1. x, the ADR operator can now be used with function, program, function block and method names and it replaces the INSTANCE_OF operator.

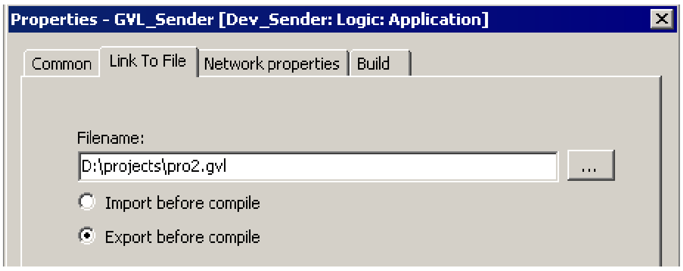
  For more information here, see Function pointers, page 557 in Data types.

  However, it is not possible to call function pointers within "IndraLogic 2G". To enable a system call (runtime system), the "Enable system calls" option has to be selected for the function object.
- **Always link**: The object is highlighted in the compiler. Thus, it is always included in the compile information. This means that it is always compiled and loaded to the PLC. This option is relevant if the object is located in an application or is referenced by libraries that are also located in an application. The compile information is also used as the basis for the selectable variables of the symbol configuration.
- **Compiler definitions**: "Defines" (see {define} instructions) and conditions for object compilation can be entered here (conditional compilation). In the "Declaration" section is a description of the available "conditional Pragmas", page 546.

  The "expr" expression, which is used in such Pragmas, can also be entered here and several entries are possible in a list of items separated by commas.

  For example, it might be desired to compile a function block depending on a value of a certain variable. (In the example above: test_var1.)

File Properties



Properties - GVL_Sender [Dev_Sender: Logic: Application]

Filename: D:\projects\pro2.gvl

○ Import before compile

● Export before compile

Fig.3-217: Properties dialog, linking to file

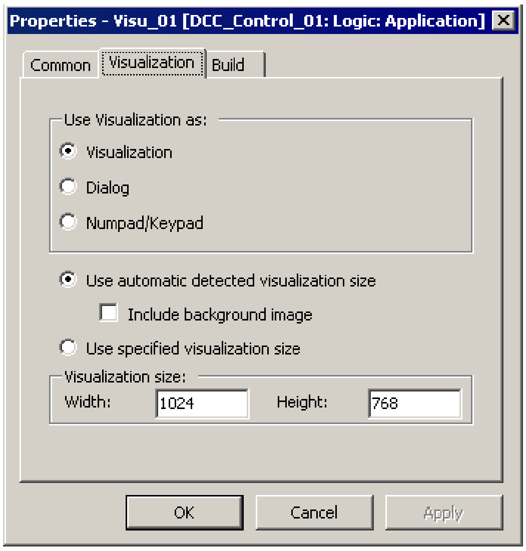
Global variable lists, page 52, can be defined by means of an external file in text format. Such file can be generated using the export functionality that is available in the Properties dialog of the relevant variable list.

- If the **Export before compilation** option is selected, a file with the extension "gvl" is stored automatically in every project compilation (e.g. via <F11>) in the path contained in the File name field.
- If the **Import before compilation** option is selected, an existing export file can be imported in every project compilation. This allows for the import of a GVL that has been defined in another projection in order to set-up communication via network variables, page 248,.

Properties of Visualization Objects

Icon:

Context menu: **Properties**.



Properties - Visu_01 [DCC_Control_01: Logic: Application]

Use Visualization as:

● Visualization

○ Dialog

○ Numpad/Keypad

● Use automatic detected visualization size

□ Include background image

○ Use specified visualization size

Visualization size:

Width: 1024 Height: 768

Fig.3-218: Visualization tab

For the currently selected visualization, the following settings are displayed, which can also be changed:

- **Use the automatically determined visualization size**: The size of the visualization which shows all currently contained visualization elements and the background image is determined. If the "Include background image" option is, however, deactivated, the background image is cut off if it exceeds the area including all visualization elements.

- **Use specified visualization size**: The visualization size is defined by the values displayed above (width, height) whereas it is not considered whether all visualization elements and the background image fit into this area, i.e. are completely visible.
- **Visualization size:** current "width" and "height" of the visualization (number of pixel)

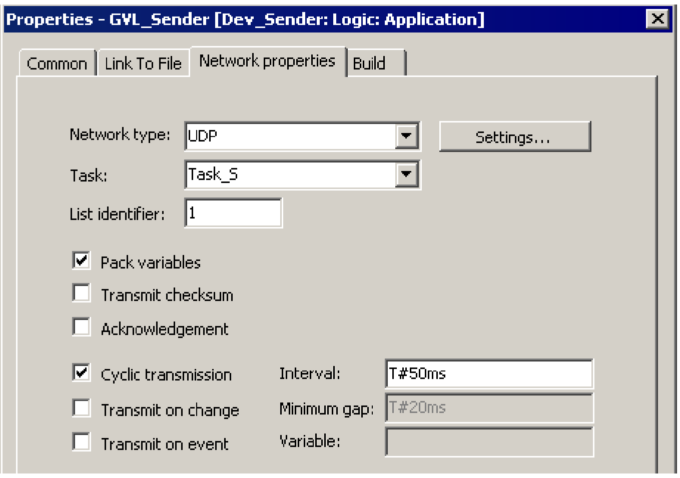
Please note that you can define in the visualization manager, page 496, whether the visualization size that is defined here is to be adjusted to the image size of the visualization client automatically.

Network Variable Properties

Network Variable Properties, Sender End

If the network functionality is supported by the device, the network properties of a global variable list (GVL) can be displayed and edited in the Properties dialog.

Defining network properties for a GVL means making the variables declared in the GVL available as network variables. For that purpose, a GVL has to be provided by the "sender" of the network variables. The receiver as counterpart has to define a corresponding global network variable list containing the same variable declarations. See information on network variables, page 71.



Properties - GVL_Sender [Dev_Sender: Logic: Application]

Common | Link To File | Network properties | Build

Network type: UDP Settings...

Task: Task_S

List identifier: 1

☑ Pack variables

☐ Transmit checksum

☐ Acknowledgement

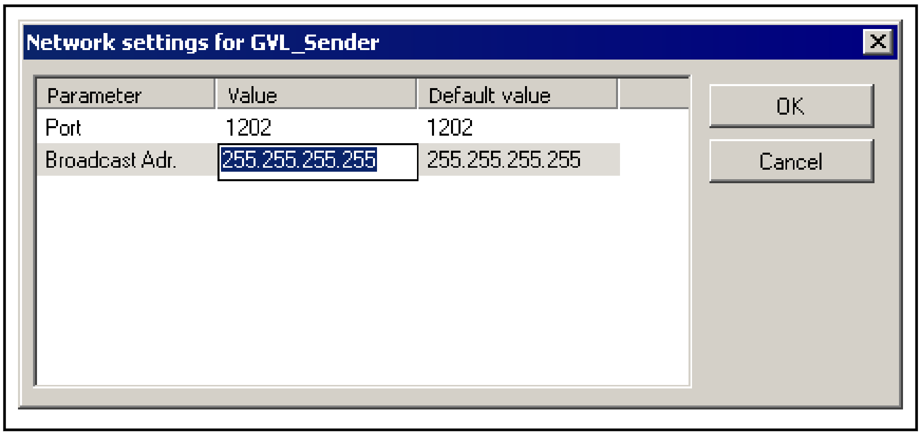
☑ Cyclic transmission Interval: T#50ms

☐ Transmit on change Minimum gap: T#20ms

☐ Transmit on event Variable:

Fig.3-219: Setting GVL network properties, sender

- **Network type:** Select the desire type from the target system-dependent list. For example "UDP" for a UDP system.
- **Task:** From the selection list, choose the task of the current application that controls the variables to be sent. The variables are always sent at the end of one task cycle.
- **Variable list code:** Number of the list that is to be sent first (default=1). More lists are numbered consecutively.
- **Settings:** Protocol-specific settings; the possible entries depend on the relevant network library.



Parameter	Value	Default value
Port	1202	1202
Broadcast Adr.	255.255.255.255	255.255.255.255

Fig.3-220: Setting GVL network properties, settings

For "UDP" networks, the following parameters have to be set:

- **Port:** Number of the port that is used for the data exchange with other network participants; the **preset value** is "1202"; the current value can be changed in the Value field at any time (select the field and press <space> in order to access the entry mode); ensure that the other nodes in the network define the same port! If more than one UDP connection is defined in the project, the port number is automatically adjusted in all configurations to the value set here.
- **Broadcast addr.:** The **preset value** is "255 . 255 . 255 . 255" means that data is exchanged with all network participants. The current value can be changed in the "Value" field (select the field and press <space> in order to access the entry mode) where to enter the address or the address range of a subnetwork (e.g. "197 . 200 . 100 . 255" in case of communication with all nodes having IP addresses in the range from "197 . 200 . 100 . x").

☞ Note for Win32 systems that the broadcast addresses have to comply with the subnet mask of the TCP/IP configuration of the computer!

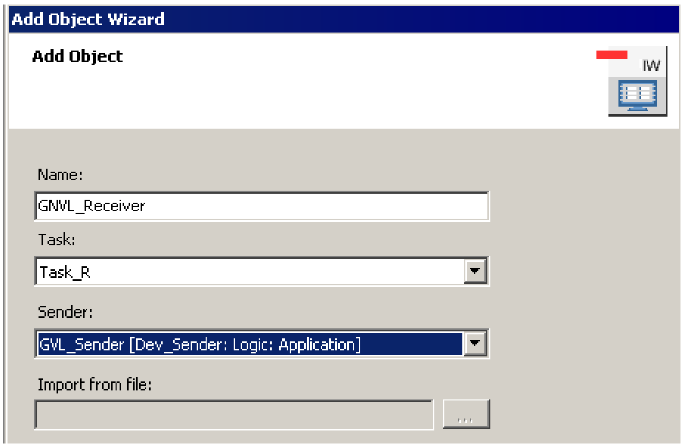
The following options can be enabled or disabled for configuring the transfer behavior of the variables:

- **Zip variables:** For the transfer, the variables are bundled in packages (telegrams) the size of which depends on the network type. If the option is disabled, one package per variable is generated.
- **Transfer checksum:** A checksum is attached to every variable package. The checksum is verified by the receiver in order to ensure that the variable definitions of sender and receiver comply with each other. A package with incorrect checksum is not accepted.
- **Cyclic transfer:** The variables are sent within the specified interval. (time specification e.g. "T#70ms").
- **Confirmation:** A confirmation message is sent for every data package received. If the sender has not received a confirmation before resending, an error is written into the diagnostic structure.
- **Transfer in case of change:** The variables are only sent if their values have changed. The **minimum distance** can be used to specify how much time has to pass at least between two transfers.
- **Event-controlled transfer:** The variables are sent as soon as the specified variable becomes TRUE.

☞ Note that the network variables are automatically sent upon each system start. That means that the current variable values are also transferred if this were not the case at that time due to the configured transfer triggers (change, result).

Network Variable Properties, Receiver End

If the network functionality is supported by the device, the current network settings for a GNVL (global network variable list, page 53) can be displayed and changed in the Properties dialog. These are the settings that have been made upon insertion of the GNVL in the 'Add object' dialog. (See more information on network variables, page 71).



Name: GNVL_Receiver

Task: Task_R

Sender: GVL_Sender [Dev_Sender: Logic: Application]

Import from file:

Fig.3-221: Properties dialog, network settings, receiver

- **Sender:** Name of the global variable list of the sender device that is referenced in the present GNVL. The square brackets contain the name of the device and the application or "import from file".
- **Task:** Name of the task of the current device controlling the network variables.
- **Import from file:** Enables the transfer of data as XML list e.g. from an IndraLogic-1.x system.

  See also file properties, page 245.

3.11 FBD/LD/IL - Commands

3.11.1 FBD/LD/IL - Commands, Overview

The commands described below are used to work in the FBD/LD/IL editor, page 339.

This editor is the common editor for the three programming languages function diagram (FBD), page 339, ladder diagram (LD), page 340, and instruction list (IL), page 340.

The menu item appears in the main menu when the FBD/LD/IL editor is active.

The menu item commands are also available in the context menu, depending on position.

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize ▸ Commands** dialog.

☞ The display of the elements in FBD/LD/IL networks is configured in the "Options, FBD, LD and IL", page 207,. To do this, in the main menu click on **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **FBD, LD and IL.**

☞ Note that in contrast to IndraLogic 1.x, networks are also used in the IL editor as structural units.

The commands:

- Add network, page 250
- Add network below, page 250
- Commenting on/off, page 250,
- Add assignment, page 250
- Add FB call, page 251
- Add empty block, page 254
- Add jump, page 255
- Add jump label, page 256
- Add return, page 256
- Add function block input, page 257
- Only LD: Add coil, page 257
- Only LD: Add set coil, page 257
- Only LD: Add set coil, page 257
- Only LD: Add contact, page 257
- Only LD: Add negated contact, page 259
- Only LD: Add contact (right), page 259
- Only LD: Add parallel contact (below), page 259
- Only LD: Add parallel negated contact (below), page 259
- Only LD: Add parallel contact above, page 259
- Only LD: Add contacts: Add below, page 260
- Only LD: Add contacts: Add right (after), page 260
- Only LD: Add contacts: Add above page 261
- Only IL: Add IL line after, page 261
- Only IL: Delete IL line, page 261
- Negation, page 261,
- Edge detection, page 262
- Set/Reset, page 263
- Determine further connection, page 264
- Insert line branching
- Insert line branching above
- Insert line branching below
- Update parameters, page 265
- Remove unused FB call parameters, page 266
- View: Show as function block diagram, page 266
- View: Show as ladder diagram, page 267

- View: Show as instruction list, page 267

3.11.2 Add Network

Icon:

Default shortcut: <Ctrl>+<Shift>+<B>

Menu: **FBD/LD/IL ▸ Adding a network**

Use this command to add a network, page 356, **above** the highlighted network in the editor.

If the cursor is currently positioned in the editor window but not in a network, the new network is attached at the end of the existing network list. The network numbering is automatically updated.

Note that in contrast to IndraLogic 1.x, networks are also used in the IL editor networks, page 346, as structural units.

3.11.3 Add Network Below

Icon:

Default shortcut: <Ctrl>+<Shift>+<A>

Menu: **FBD/LD/IL ▸ Insert Network below**

Use this command to add a network, page 356, in the FBD/LD/IL editor. If the cursor is positioned in an existing network, the new network is added **below** the existing network.

If the cursor is currently positioned in the editor window but not in a network, the new network is attached at the end of the existing network list. The network numbering is automatically updated.

Note that in contrast to IndraLogic 1.x, networks are also used in the IL editor networks, page 346, as structural units.

3.11.4 Commenting On/Off

Icon:

Default shortcut: <Ctrl>+<Shift>+<C>

Menu: **FBD/LD/IL ▸ Commenting on/off**

Use this command in the FBD/LD/IL editor to "comment out" a network or to return it to normal status again. The "Commenting on/off" command affects the network in which the cursor is located.

A network that is "commented out" is displayed and treated as a comment, i.e. the network is not processed.

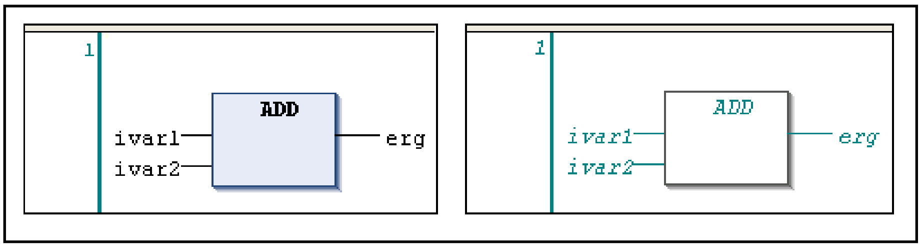


Fig.3-222: Network in normal status and in comment status

3.11.5 Add Assignment

Icon:

Default shortcut: <Ctrl>+<W>

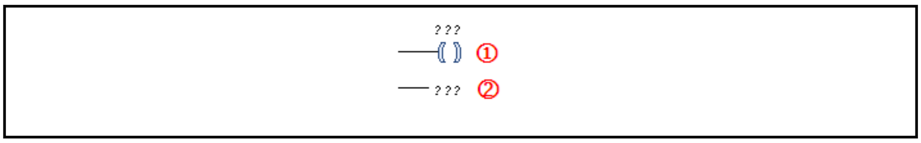
Menu: **FBD/LD/IL ▸ Insert Assignment**

Use this command to insert an assignment in the FBD or LD editor. It is not available in the IL view.

☞ In the LD view, an assignment is inserted as a coil using the command "Add coil", page 257.

Depending on the current cursor position, page 351, the assignment is inserted directly before a selected input (cursor position 2), directly after a selected output (cursor position 4) or, if the entire network or subnetwork is highlighted, at the end of the respective network (cursor position 6 or 11).

In the FBD editor an assignment is inserted as a line followed by three question marks; in the LD editor it is inserted as a coil, page 361 and three question marks.

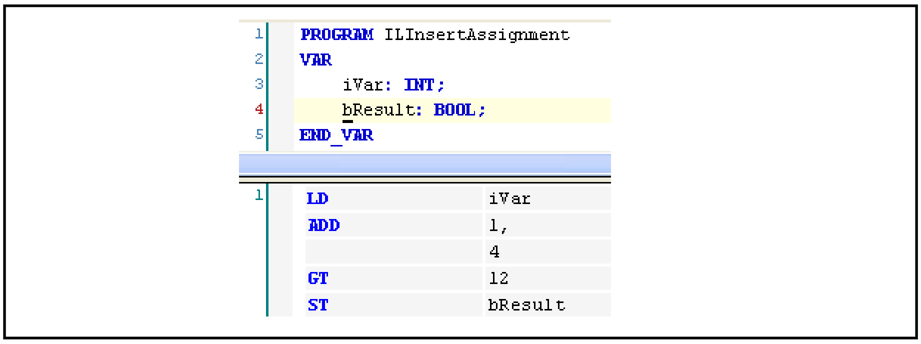


① Assignment in LD
② Assignment in FBD

Fig.3-223: Assignments in the LD and FBD editors

To define an assignment, select the placeholder text "???" and replace it with the name of the variable that is to be assigned the signal (value) coming from the left. The input assistance can be accessed with the ... button.

In the IL editor, an assignment is inserted as an ST operator, page 341; see the following example.



```
PROGRAM ILInsertAssignment
VAR
    iVar: INT;
    bResult: BOOL;
END_VAR
```

```
LD      iVar
ADD     1,
        4
GT      12
ST      bResult
```

Fig.3-224: Assignment as ST operator in the IL editor

3.11.6 Add FB Call

Icon:

Default shortcut: <Ctrl>+<B>

Menu: **FBD/LD/IL ▸ Call FB call**

Use this command to add a function block element into an FBD, LD or IL network. This applies to calling an operator function block, a program, a function block, a function or an interface.

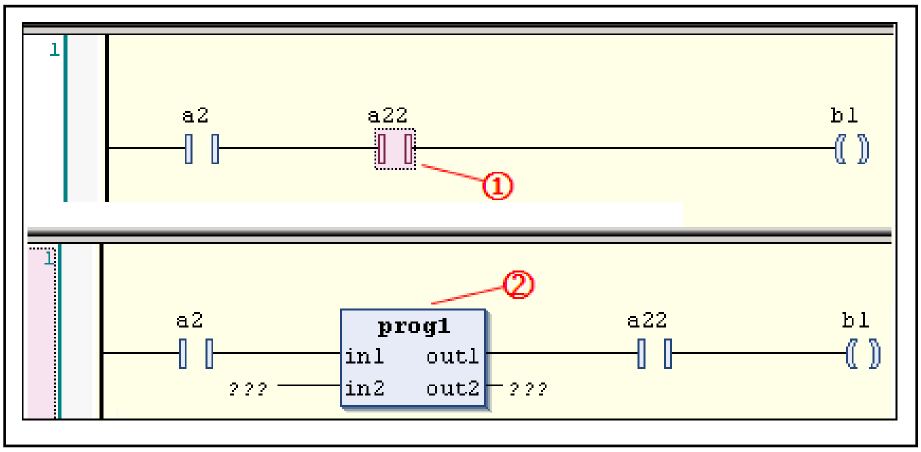
The corresponding instructions are added in the IL editor.

After selecting the "Add FB call" command, the Input assistance, page 98 dialog opens, which lists the available function block categories. When selecting a function block and confirming it with "OK", a corresponding box with inputs and outputs or the corresponding IL instructions is inserted at the current cursor position in the network.

Alternatively, standard blocks can be dragged out of the Tools, page 355, tab directly into the editor window. See the following information on editor-specific details:

FBD/LD

- "Program" or "function block" type function blocks are always added in a series; that means that the processing line is connected with the topmost input and the topmost output for the function block inserted.



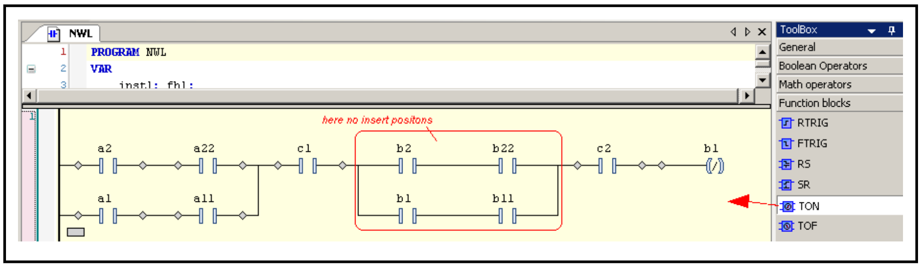
① Select contact
② prog1 program function block added
Fig.3-225: Adding a function block in LD

- The text in the inserted function block element indicates the **function block type** (e.g. "F_TRIG") and can be edited, page 351.

  By replacing this text with the type name of another valid function block, **replace** this function block call with a different one. A replacement can also be made by highlighting the existing function block element and inserted a different one in this position. Please note that the inputs and outputs **already assigned** for a function block remain in their previous order from top to bottom, unless the greatest number of inputs for the new function block is lower. In this case, the inputs and outputs that can no longer be linked are deleted, starting from the bottom.

- If it is available for the relevant function block and if the Show function block symbol, page 207, option is selected, a symbol is displayed within the function block element.

- Within the parallel connections in an LD network, there is no insertion position for a function block from the "ToolBox" tab, since a function block call requires a direct connection to the electrical power supply (vertical lines at left and right in the LD network list).



```
PROGRAM NWL
VAR
    instl: fbl:
```

① Insertion not possible
Fig.3-226: Insertion positions for FB call in an LD network

- **Function block calls with EN/ENO:** There is a special command for the insertion of a FB call with EN input and ENO output: Add FB call with EN/ENO, page 254.

☞ Consider the following when structuring the program: At the inputs of an EN/ENO FB, no FB call can be inserted! If the output of a function block is to be used as input of an EN/ENO function block, it has to be written to a variable in advance and the latter has then to be used as input of the EN/ENO function block.

- **VAR_IN_OUT parameters** of an inserted POU FB call are marked with an arrow pointing in both directions.
- **For function blocks**, the text field above the box in which the local instance variable has to be entered can also be edited. If a function block instance element of this type is replaced by inserting a different function block type, you have to enter the instance definition again.
- The **formal names of the inputs and outputs** are displayed in the boxes of function calls and function block calls. The output of a function (return value) is, however, shown without name.
- If the **interface of a function block** has changed (e.g. modified number of outputs), the function block parameters can be updated with the "Update parameters" command; see "Update parameters, page 265.

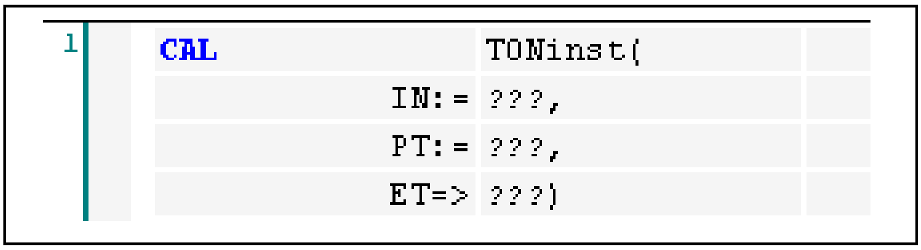
☞ The function block parameters are not updated automatically as with IndraLogic 1.x!

- Insertion positions: A new FB call is inserted at the current cursor position as follows:
  - If an input is selected (cursor position 2, page 351), the block box is inserted before it and connected into the existing line branch with its first input and output.
  - If an output is selected (cursor position 4, page 351), the block box is inserted after it and connected into the existing line branch with its first input and output.
  - If a block box is selected (cursor position 3, page 351), this function block is replaced by the new one. As far as possible, the assignments of the inputs and outputs remain in their order from top to bottom. If the old function block had more inputs or outputs than the new one, those that are not linked are deleted (starting at the bottom).
  - If a jump or return element is selected (cursor position 3, page 351), the block box is inserted before it and connected into the existing line branch with its first input and output.
  - If an entire network or subnetwork is selected, (cursor positions 6 and 11, page 351), the function block box is inserted after the last element and connected with its first input. The output has no connection.
- All function block inputs that do not automatically receive assignments during insertion are given "???" as text. Select this text and replace it with the name of variable or with a constant.
- If you insert a function block at the end of a network, its output is not linked.

IL editor

In the IL editor, "Add FB call" can be used to insert a function block. After selecting the "Add FB call" command, the Input assistance, page 98 dialog opens, which lists the available function block categories. If the "Insert with arguments" option is selected in the "Input Assistance" dialog, a CAL instruction, page 341, is inserted for example along with the corresponding input and output parameters; see the following example.

Example:



```
CAL          TONinst(
        IN:= ???,
        PT:= ???,
        ET=> ???)
```

Fig.3-227: FB call in IL

The "TON" function block was selected with input assistance. The input parameters are automatically specified in the following lines and can be defined. In this example, the user replaced the "???" in the CAL line with "TONinst" (local instance of TON).

FB calls with EN/ENO cannot be inserted in the IL editor.

3.11.7 Add FB Call with EN/ENO

Symbol:

Shortcut: <Ctrl>+<Shift>+<E>

Menu: **FBD/LD/IL ▸ Insert FB call with EN/ENO**

This command is only available in the FBD editor. It can be used in order to add, page 251 a FB call with an EN input and an ENO output to a network.

Processing EN/ENO function blocks: If at the time of the FB call EN has the value FALSE (0), the operations defined in the function block are not executed. Otherwise, i.e. if EN has the value TRUE (1), these operations are carried out. The ENO output acts as repeater of the EN input.

3.11.8 Add Empty Block

Symbol:

Menu: **FBD/LD/IL ▸ Add empty block**

This command is used to add an empty block element into an FBD, LD or IL network.

See Add FB call, page 251, for detailed information on adding FBs in the FBD/LD/IL editor. In contrast to the "Add FB call" command, the input assistance is not opened automatically, but the instance filed above the FB element.

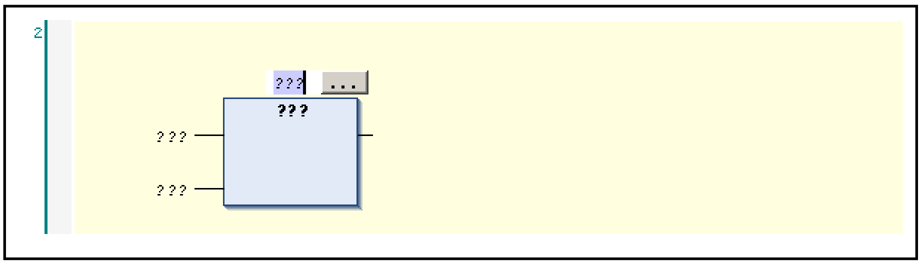


Fig.3-228: Empty function block

Decide then which function type is necessary:

- If there is to be a **function block**, enter the desired instance variable name and close the field by means of "Enter". The input assistance can be used to select the name of an already existing instance variable. After entry of an already declared instance variable, the block is shown accordingly. If an instance name has been entered that is still unknown, you have to specify the name of the desired function block as well. After closing the input field, the focus is for that purpose automatically set into the input field within the box. In this case, the input assistance only provides function blocks.

- If the block is to represent an **operator**, a **program**, a **function** or an **interface**, press the <down arrow key> when the mouse cursor is located in the instance input field. The input focus changes to the function block type field (within the function block element) where the desired operator, program, function or interface name can be entered directly or via the input assistance. After completing this entry, the block is displayed in the network accordingly.

The corresponding instructions are added in the IL editor.

3.11.9 Add Jump

Icon: →

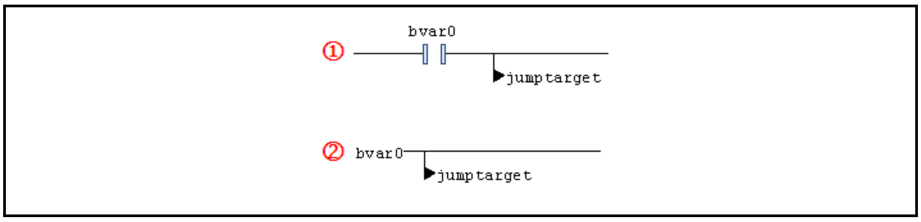
Default shortcut: <Ctrl>+<J>

Menu: **FBD/LD/IL ▸ Add jump**

Use this command to add a jump. The jump target is another network that is indicated here by its label, page 357, (jump label).

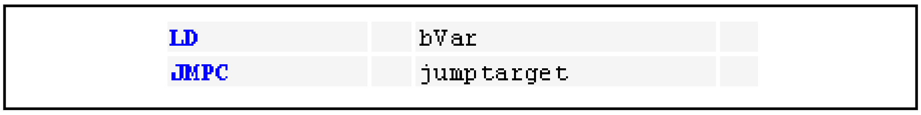
In the FBD and LD editors the jump element is inserted based on the current cursor position, either directly before a selected input (cursor position 2, page 351), directly after a selected output (cursor position 4, page 351,) or if the entire network or subnetwork is highlighted, at the end (cursor position 6 or 11, page 351).

A new jump element contains the text "???" that can be selected via click and replaced with the name of label for the network that is the jump target; see the following figure.



① Jump label in LD
② Jump label in FBD
Fig.3-229: Jump with jump label

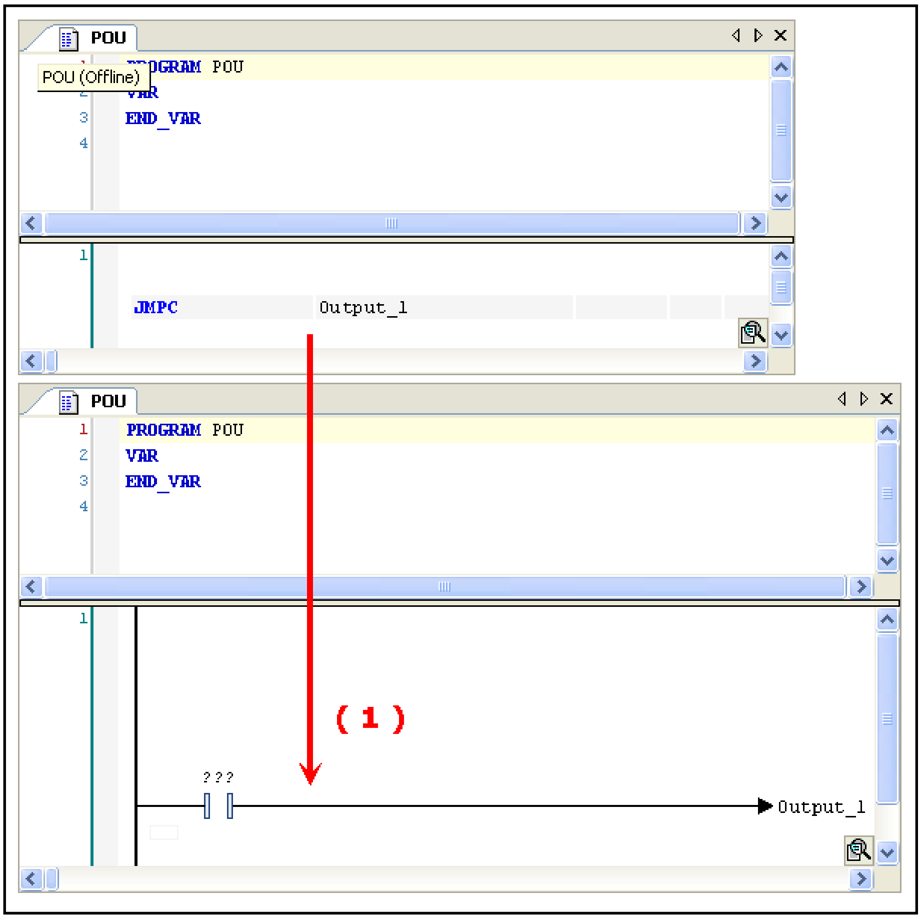
In the IL editor, a jump is programmed with the JMP operator, page 341,.



```
LD      bVar
JMPC    jumptarget
```

Fig.3-230: Jump with JMP operator

If a JMP operator that was previously inserted within the IL editor without a preceding LD is later converted in the LD editor, a dummy operator, "???", is inserted.



```
PROGRAM POU
VAR
END_VAR
```

```
JMPC        Output_1
```

```
PROGRAM POU
VAR
END_VAR
```

(1) Conversion from IL to LD

Fig.3-231: Example: Conversion of a JMP operator

3.11.10 Add Jump Label

Icon:

Menu: **FBD/LD/IL ▸ Add jump label**

Use this command to add a jump label, page 357 to the currently selected network. The jump label can be used as the target for a jump, page 357.

When calling the "Add jump label" command, by default the text "Label:" is entered in the associated text field which can be changed as desired.

3.11.11 Add Return

Icon: RET

Menu: **FBD/LD/IL ▸ Add return**

Use this command to insert a RETURN instruction at the current cursor position.

FBD/LD Based on the current cursor position the return element is inserted directly before a selected input (cursor- position 2, page351), directly after a selected output (cursor position 4, page 351), directly before a branch (cursor position 5, page 351), or at the end of a network or subnetwork (cursor position 6 or 11, page 351).

IL In the IL editor an instruction is added with the "RETURN", page 341, operator.

3.11.12 Add Block Input

Icon:

Default shortcut: <Ctrl>+<Q>

Menu: **FBD/LD/IL ▸ Call FB call**

This command adds another input to an already added extendable block (AND, OR, ADD, MUL, SEL) in the FBD or LD editor. The command is not available in the IL editor.

The maximum number of inputs depends on the block type (ADD e.g. may have 2 or more inputs).

To insert the input at a certain position relative to the existing input, highlight the input above which the new one is to be inserted (cursor position 1, page 351).

To insert another input for a function block that is way at the bottom, highlight the function block body (cursor position 3, page 351).

The new input is initially assigned the text "???". Replace this text with the name of a variable or with a constant. The input assistance can be accessed with the ... button.

3.11.13 Add Coil

Icon:

Default shortcut: <Ctrl>+<W>

Menu: **FBD/LD/IL ▸ Add coil**

Use this command to insert a coil, page 361 parallel to the existing coils.

If the highlighted position, page 351 is a connection between contacts, page 360 and coils, the new coil is inserted at the end. If the highlighted position is a coil, the new coil is inserted parallely and directly above it.

By default, the coil is assigned the text "???". To change this text, click on it to edit it or use the Input assistant, page 98,. The input assistance can be accessed with the ... button.

3.11.14 Add Set Coil

Icon:

Menu: **FBD/LD/IL ▸ Add set coil**

Use this command to insert a set coil, page 362,.

The "Set/Reset", page 263, command can convert a coil into a "set" coil.

3.11.15 Add Reset Coil

Icon:

Menu: **FBD/LD/IL ▸ Add reset coil**

Use this command to insert a reset coil, page 362,.

The "Set/Reset", page 263, command can convert a coil into a "reset" coil.

3.11.16 Add Contact

Icon:

Default shortcut: <Ctrl>+<K>

Menu: **FBD/LD/IL ▸ Add contact**

Use this command to add a contact element to a ladder diagram network.

The "Add contact" command is only available in the LD editor. When switching to the FBD or IL view, a contact is converted accordingly.

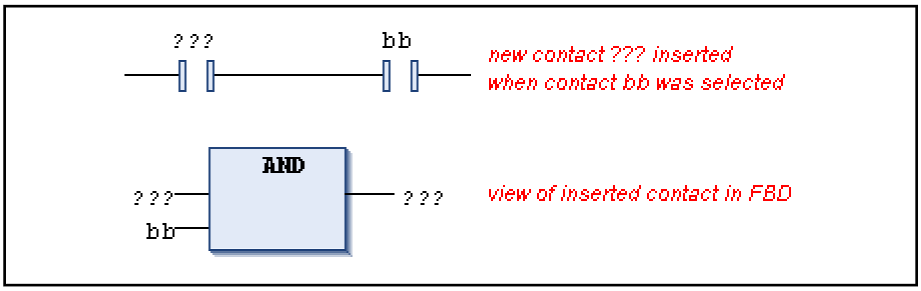


Fig.3-232: Switching from LD to FBD

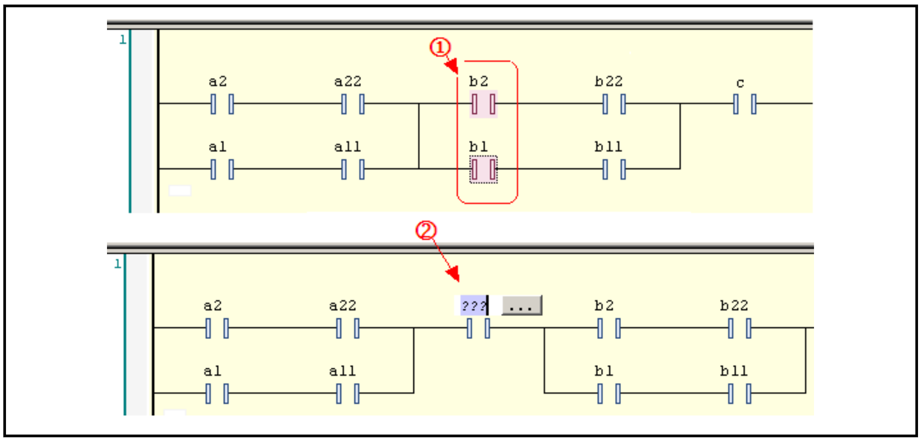
The new contact is inserted at the current cursor position in the sequence **before** the existing contact or a function block box. If the cursor position is in the branch of an existing parallel connection, the new contact is inserted into it.

Alternatively, insert a contact element from the Tools, page 355, tab. However, if it is to be inserted not within, but before, after or between existing parallel connections, only the "Add contact" command can be used.

To do this, one of the existing contact elements in each branch of the parallel connection has to be highlighted (multiple selection with <Ctrl> key) and the "Add contact" command used; see the following example "Add contact with parallel connection".

Example:

Adding a contact with parallel connection



① Highlight b1 and b2 while pressing and holding the <Ctrl> key.
② Use the "Add contact" command.
Fig.3-233: Adding a contact with parallel connection

Note also the following commands:

- Add contact (right), page 259
- Add parallel contact (above), page 259
- Add parallel contact (below), page 259.

By default, a new contact element contains the text "???". To modify this text, highlight the three question marks and enter the desired variable name or an address (depending on the current settings in the "FBD, LD and IL options", page 207) or a constant. In addition, the input assistance can be accessed with the ... button.

3.11.17 Add Contact (Right)

Icon:

Default shortcut: <Ctrl>+<R>

Menu: **FBD/LD/IL ▸ Add contact (right)**

Use this command to add a contact element into a ladder diagram network. The same features apply as for the "Add contact" command, except in this case the new element is not inserted to the left, but instead, to the right of the current cursor position; see Add contact, page 257.

3.11.18 Add Parallel Contact (Below)

Icon:

Default shortcut: <Ctrl>+<D>

Menu: **FBD/LD/IL ▸ Add parallel contact (below)**

Use this command to insert a parallel contact at the current cursor position in an LD network. The same features apply as for the "Add parallel contact (above)", except in this case the new contact appears below the selected position; see Add parallel contact (above), page 259.

3.11.19 Add Negated Contact

Icon:

Menu: **FBD/LD/IL ▸ Add negated contact**

Use this command to insert a negated contact, page 360,.

☞ An existing contact can also be converted into a negated contact with the Negation, page 261, command.

3.11.20 Add Parallel Negated Contact (Below)

Icon:

Menu: **FBD/LD/IL ▸ Add parallel negated contact (below)**

Use this command to add a parallel negated contact, page 360, below the current cursor position.

☞ An existing contact can also be converted into a negated contact with the Negation, page 261, command.

3.11.21 Add Parallel Contact (Above)

Icon:

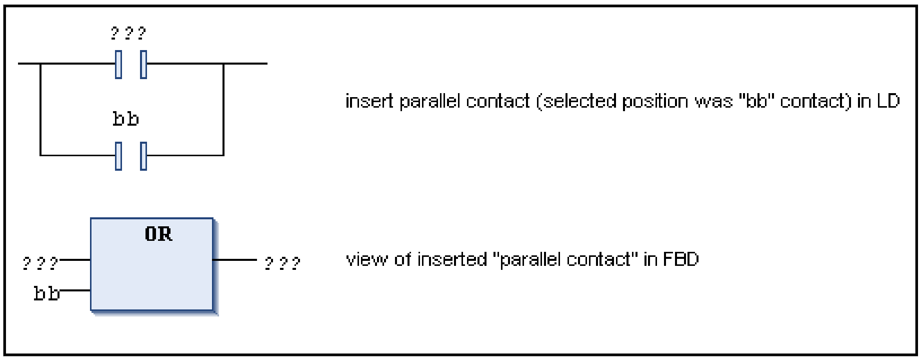
Default shortcut: <Ctrl>+<E>

Menu: **FBD/LD/IL ▸ Add parallel contact (above)**

Use this command to insert a parallel contact (parallel connection) at the current cursor position in an LD network. Several existing elements can also be highlighted in order to insert a new contact parallel to them and above the current cursor position.

☞ The "Add parallel contact (above)" command is only available in the LD editor. However, when switching to the IL or FBD editor, the contact is converted accordingly.

By default, a new contact element contains the text "???". To modify this text, highlight the three question marks and enter the desired variable name or an address (depending on the current settings in the "FBD, LD and IL options") or a constant. In addition, the input assistance can be accessed with the ... button.


???
bb
insert parallel contact (selected position was "bb" contact) in LD
OR
???
bb
???
view of inserted "parallel contact" in FBD

① Inserting a parallel contact in the LD view (the highlighted position was contact "bb")

② Display of the inserted parallel contact in the FBD view

Fig.3-234: Inserting a parallel contact

3.11.22 Add Contacts: Add Below

Default shortcut: <Ctrl>+<Shift>+<D>

Menu: FBD/LD/IL ▸ Add contacts ▸ Add below

Use this command to add contacts or a section of the network, which was previously placed on the clipboard using the "Copy" or "Cut" command.

The contacts are inserted below the area currently highlighted. This corresponds with the general "Insert" command.

☞ The "Insert below" command is only available in the LD editor. However, when switching to the IL or FBD editor, the contact is converted accordingly.

To insert contacts below the selected area, highlight one or more contacts and execute the "Copy" or "Cut" command. Then activate the command.

Information about highlighting in networks can be found in the editors under Working in the FBD and LD editors, page 339,.

3.11.23 Add Contacts: Add Right (After)

Default shortcut: <Ctrl>+<Shift>+<R>

Menu: FBD/LD/IL ▸ Add contacts ▸ Add right (after)

Use this command to add contacts or a section of the network, which was previously placed on the clipboard using the "Copy" or "Cut" command.

The contacts are inserted to the right of the area currently highlighted.

This corresponds with the "Insert" command.

☞ The "Add right (after)" command is only available in the LD editor. However, when switching to the IL or FBD editor, the contact is converted accordingly.

To insert contacts below the selected area, highlight one or more contacts and execute the "Copy" or "Cut" command. Then activate the command.

Information about highlighting in networks can be found in the editors under Working in the FBD and LD editors, page 339,.

3.11.24 Add Contacts: Add Above

Default shortcut: <Ctrl>+<Shift>+<E>

Menu: **FBD/LD/IL** ▸ **Add contacts** ▸ **Insert above**

Use this command to add contacts or a section of the network, which was previously placed on the clipboard using the "Copy" or "Cut" command.

The contacts are inserted above the area currently highlighted.

This corresponds with the "Insert" command.

☞ The "Insert above" command is only available in the LD editor. However, when switching to the IL or FBD editor, the contact is converted accordingly.

To insert contacts above the selected area, highlight one or more contacts and execute the "Copy" or "Cut" command. Then activate the command.

Information about highlighting in networks can be found in the editors under Working in the FBD and LD editors, page 339,.

3.11.25 Add IL Line After

Icon:

Menu: **FBD/LD/IL** ▸ **Add IL line after**

Use this command to insert another instruction line below the line in which the cursor is currently located.

☞ The "Add IL line after" command is only available in the IL editor. When switching to the LD or FBD editor, the function of the line is converted accordingly.

3.11.26 Delete IL Line

Icon:

Default shortcut: <Ctrl>+<Del>

Menu: **FBD/LD/IL** ▸ **Delete IL line**

Use this command to delete the instruction line in which the cursor is located.

☞ The "Delete IL line" command is only available in the IL editor. When switching to the LD or FBD editor, the function of the line is converted accordingly, and in this case, deleted.

3.11.27 Negation

Icon: (FBD), (LD)

Default shortcut: <Ctrl>+<T>

Menu: **FBD/LD/IL ▸ Negation**

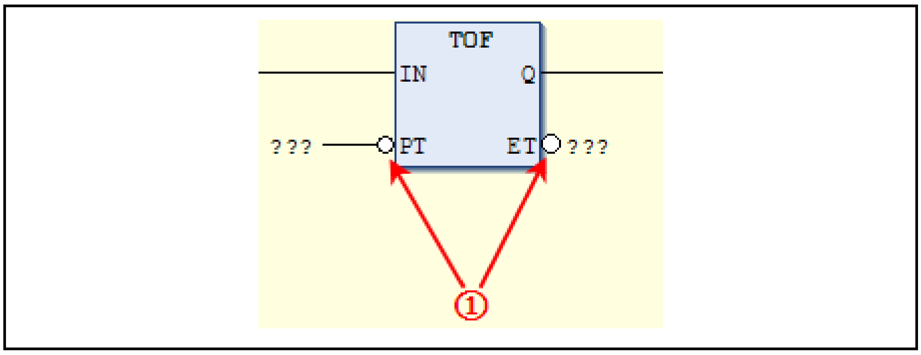
Use this command to toggle the currently selected input, output, jump or RETURN instruction between "negated" and "not negated".

☞ The "Negation" command is not available in the IL editor. In the IL editor, the corresponding modifiers have to be set. For information about the modifiers, see Modifiers and operators in IL, page 339.

☞ When switching back and forth among the FBD, LD and IL views, the negations of some constructs might be reset, since a unique conversion is not possible.

Types of display for negation

- For the function block, jump and RETURN instruction elements, a negation is shown with a small circle icon at the respective input or output connection; see the following figure of a "negated block".



① Negation

Fig.3-235: Negated block

To negate a jump or a RETURN instruction, highlight the most recent, previous output (cursor position 4, page 351).

- In the LD editor, a negated contact is indicated by a slash in the contact icon; see the following figure of a "negated contact".

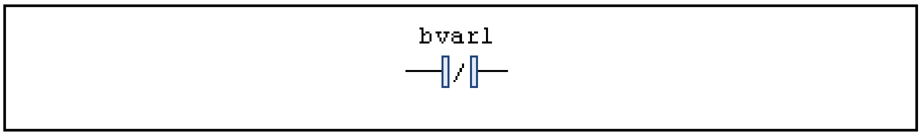


Fig.3-236: Negated contact

The "Tools", page 355, tab also provides negated contact elements in the "ladder diagram elements" category. Add elements from the "ToolBox" tab by using the mouse to drag them to the corresponding position.

- In LD, a negated coil is indicated with a slash in the coil icon; see the following figure of a "negated coil".

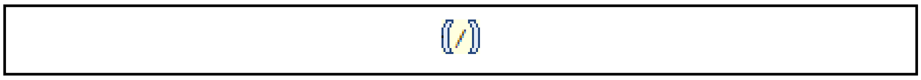


Fig.3-237: Negated coil

To negate an input or output, highlight it at cursor position 2 or 4, page 351.

3.11.28 Edge Detection

Icon: (FBD), (LD)

Default shortcut: <Ctrl>+<Shift>+<G>

Menu: **FBD/LD/IL ▸ Edge Detection**

Use this command to insert a detector for rising or falling edge at a Boolean input in the FBD or LD editor. This is comparable to using the R_TRIG function block (rising edge: FALSE → TRUE) and the F_TRIG function block (falling edge TRUE → FALSE).

☞ The "Edge detection" command is not available in the IL editor. FBD or LD networks that contain edge detection are accepted without modification in the IL view.

Execute the "Edge detection" command several times until the desired icon appears. The following are in the sequence:

- ⊣▷| for rising edges
- ⊣◁ for falling edges
- no icon

Example:

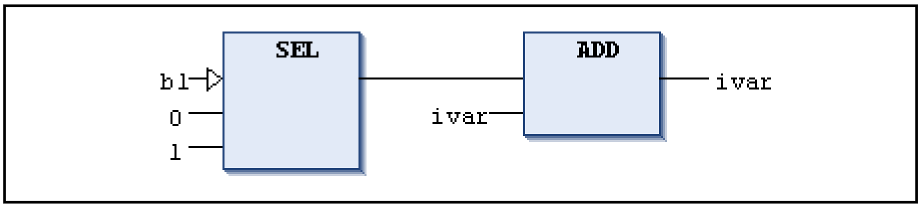


Fig.3-238: Detector for rising edges at the SEL operator

In this example, the input at "b1" was highlighted and the "Edge detection" command executed. The detector for rising edges was inserted, i.e. the output of "SEL" delivers "1" if "b1" changes from FALSE to TRUE.

The command is not available in the IL editor. FBD or LD networks that contain edge detection are accepted without modification in the IL view.

3.11.29 Set/Reset

Icon: S

Default shortcut: <Ctrl>+<Shift>+<S>

Menu: **FBD/LD/IL ▸ Set/Reset**

Use this command to define Boolean outputs as "set" or "reset" outputs and coils to "set" or "reset" coils.

☞ The "Set/Reset" command is not available in the IL editor. In the IL editor, the corresponding modifiers have to be set. For information about the modifiers, see "Modifiers and operators in IL," page 341.

If the "Set/Reset" command should be be executed at the same output several times in a row, "Set", "Reset" and normal state is set in sequence.

☞ Further information about this command is located in Set/Reset, page 355.

3.11.30 Determine Further Connection

Icon:

Default shortcut: <Ctrl>+<Shift>+<W>

Menu: **FBD/LD/IL ▸ Determine further connection**

Use this command for function block elements with several outputs to determine which of these outputs are to be connected with the continuing processing line in the network.

☞ The "Determine further connection" command is not available in the IL editor.

Highlight the output that is to be connected further and activate the command in order to specify the further connection.

Note that the output assignments are displaced when the further connection is redefined; see the following figure.

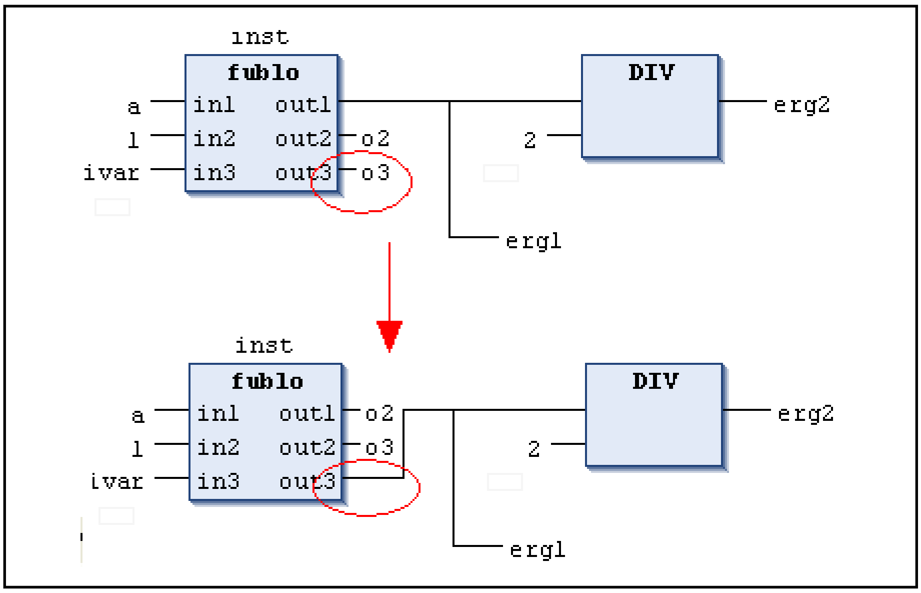


Fig.3-239: Further connection is positioned at out3.

3.11.31 Insert Line Branching

Icon:

Menu: **FBD/LD/IL ▸ Insert line branching**

This command branches the current processing line in a network, page 343, in FBD.

The processing line is divided into two subnetworks, page 264,: Another processing line is inserted below the existing one. The two subnetworks are processed in online mode according to their position from top to bottom.

If a "line branching" is dragged from the toolbox or from any other position in the editor, the possible insertion positions, page 343, are displayed by gray position labels.

A branch can be inserted at the input connectors of function blocks that are not located within a subnetwork, at output connectors (cursor position 4) of a block if it is not (not indirectly either) connected to the input of another function block in the subnetwork, at the connection between contacts and coils (cursor position 10) or at a contact (cursor position 8). A branch cannot be inserted within an "ORed" contact group and within multiple assignment groups cursor positions, page 351).

Every subnetwork gets an own "label", an upright rectangular symbol serving as possible cursor position (11) for selection of the subnetwork.

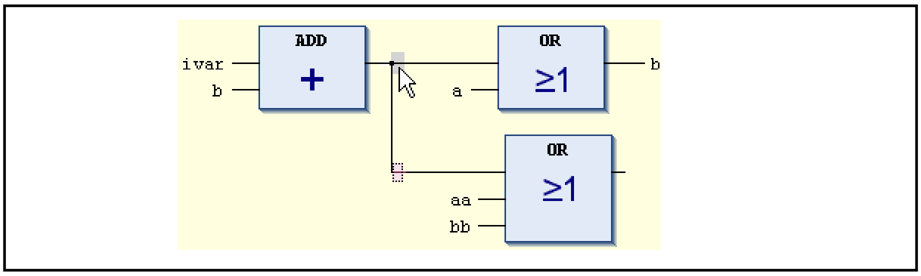


Fig.3-240: Subnetwork label in an FBD network

Every subnetwork can again be branched which allows for the generation of a widely ramified network and "subnetworks" within the main network.

For adding subnetworks to subnetworks see the commands:

Insert line branching above, page 265, and

Insert line branching below, page 265.

See branching in FBD/LD/IL, page 358, for more information on branching and subnetworks.

Connections between different branching arms or subnetworks can be removed or established by means of the Separate connections and Connect connections commands.

The command is not available in the IL editor. Networks with branching cannot be converted into IL.

3.11.32 Insert Line Branching Above

Icon:

Menu: **FBD/LD/IL ▸ Insert line branching above**

This command is used to insert a subnetwork, page 264, within the existing one, namely above the current cursor position.

3.11.33 Insert Line Branching Below

Icon:

Menu: **FBD/LD/IL ▸ Insert line branching below**

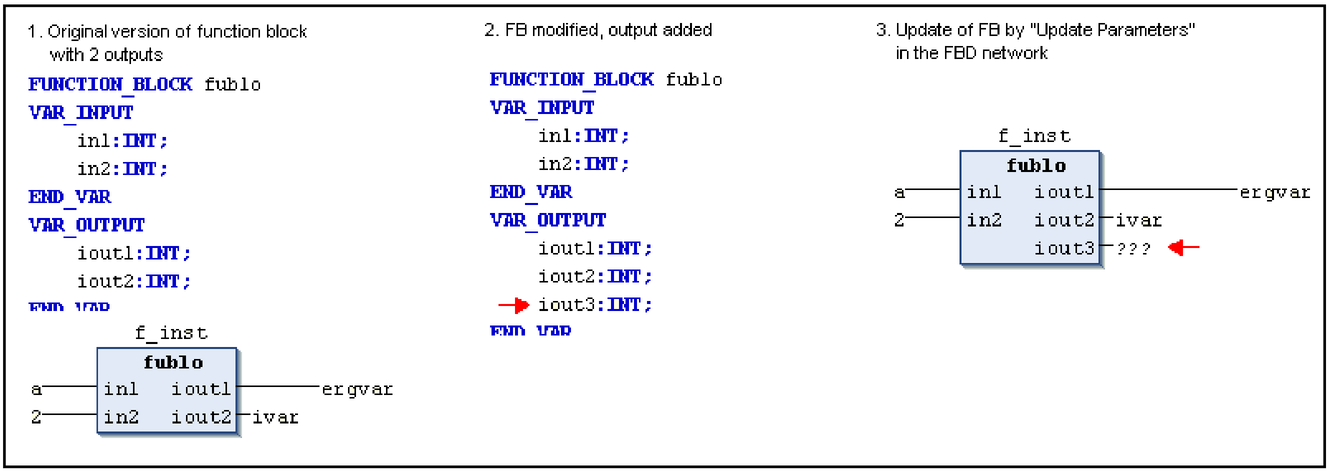
This command is used to insert a subnetwork, page 264, within the existing one, namely below the current cursor position.

3.11.34 Update Parameters

Default shortcut: <Ctrl>+<Shift>+<P>

Menu: **FBD/LD/IL ▸ Updating parameters**

Use this command to update the input and output parameters for previously inserted blocks if their interface has changed (if, for example, an output was inserted).



1. Original version of function block with 2 outputs

```
FUNCTION_BLOCK fublo
VAR_INPUT
    in1:INT;
    in2:INT;
END_VAR
VAR_OUTPUT
    iout1:INT;
    iout2:INT;
END_VAR
```

2. FB modified, output added

```
FUNCTION_BLOCK fublo
VAR_INPUT
    in1:INT;
    in2:INT;
END_VAR
VAR_OUTPUT
    iout1:INT;
    iout2:INT;
    iout3:INT;
END_VAR
```

3. Update of FB by "Update Parameters" in the FBD network

Fig.3-241: Updating parameters

The existing connections to the remaining inputs or outputs remain intact. If an input or output is added, it appears with the text "???" and can then be assigned.

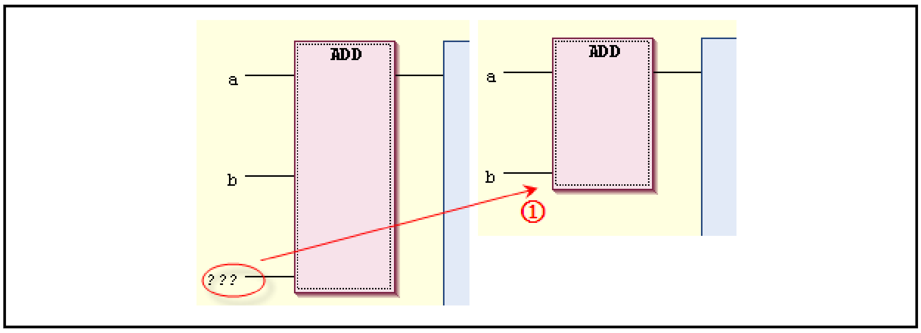
3.11.35 Remove Unused FB Call Parameters

Icon:

Menu: **FBD/LD/IL ▸ Remove unused FB call parameters**

Use this command to remove all unused inputs or outputs for a highlighted function block (all inputs and outputs that contain the text "???"). The lowest amount of necessary inputs or outputs for a function block are retained; see the figure below.

The "Remove unused FB call parameters" command is not available in IL.



① Unused input removed

Fig.3-242: Command - Remove unused FB call parameters

3.11.36 View: Show as Function Block Diagram

Default shortcut: <Ctrl>+<Shift>+<F>

Menu: **FBD/LD/IL ▸ View ▸ Show as function block diagram**

Use this command to switch among the FBD/LD/IL editors, page 339, in offline and online mode in the FBD view.

The "Show as function block diagram" command allows an LD or IL network to be converted into an FBD network.

However, note that some special elements cannot be converted and remain provided only in the original editor view. In addition, not all constructs can be converted uniquely; see also "Show as instruction list", page 267.

To switch back to the LD view, use the command "Show as ladder diagram", page 267.

To switch back to the IL view, use the command "Show as instruction list", page 267.

⚠ CAUTION	**Error-free conversion requires code with correct syntax. Otherwise, parts of the implementation might be lost.**

3.11.37 View: Show as Ladder Diagram

Default shortcut: <Ctrl>+<Shift>+<L>

Menu: **FBD/LD/IL ▸ View ▸ Show as ladder diagram**

Use this command to switch among the FBD/LD/IL editors, page 339 in offline and online mode in the LD view.

The "Show as ladder diagram" command allows an IL or FBD network to be converted into an LD network. However, note that some special elements cannot be converted and remain provided only in the original editor view. In addition, not all constructs can be converted uniquely; see also "Show as instruction list", page 267.

FBD elements that cannot be displayed as LD element (e.g. XOR) are transferred into the LD network as FBD blocks.

To switch back to the FBD view, use the command "Show as function block diagram", page 266.

To switch back to the IL view, use the command "Show as instruction list", page 267.

⚠ CAUTION	**Error-free conversion requires code with correct syntax. Otherwise, parts of the implementation might be lost.**

3.11.38 View: Show as Instruction List

Default shortcut: <Ctrl>+<Shift>+<I>

Menu: **FBD/LD/IL ▸ View ▸ Show as instruction list**

Use this command to switch among the FBD/LD/IL editors, page 339 in the IL view.

The "Show as instruction list" command allows an LD or FBD network to be converted into an IL network.

☞ Since there are elements that cannot be converted, the network affected continues to be displayed in the original editor view.

Example: Operators CALC, CALCN

If there are syntax errors, conversion is not possible either, and a corresponding error message is output. A few constructs cannot be converted uniquely. Thus, they are "normalized" when switching back and forth among IL and FBD/LD. This applies to negation and explicit/implicit assignments for function block inputs and outputs.

To switch back to the FBD view, use the command "Show as function block diagram", page 266.

To switch back to the LD view, use the command "Show as ladder diagram", page 267.

⚠ CAUTION

Error-free conversion requires code with correct syntax. Otherwise, parts of the implementation might be lost.

3.12 CFC Commands

3.12.1 CFC Commands, Overview

The commands of the CFC Editor, page 309, (Continuous Function Chart) are described in this section.

By default, they are included in the **CFC menu** which is available when the CFC editor is active. Alternatively, they are provided in the context menu for the CFC elements in the working sheet or toolbox.

If required, the menu structure can be reconfigured via the **IndraWorks** ▸ **Tools** ▸ **Customize** ▸ **Commands** dialog.

The commands:

- Edit Working Sheet..., page 269
- Negation, page 270
- EN/ENO, page 270
- Set/Reset:
  - None, page 271
  - R - Reset, page 272
  - S - Set, page 272
- Execution order:
  - At the Beginning, page 272
  - At the End, page 273
  - One Ahead, page 273
  - One Back, page 273
- Setting Execution Order, page 276
- Arranging by Dataflow, page 273
- Arranging Topologically, page 275
- Connecting Selected Connections, page 277

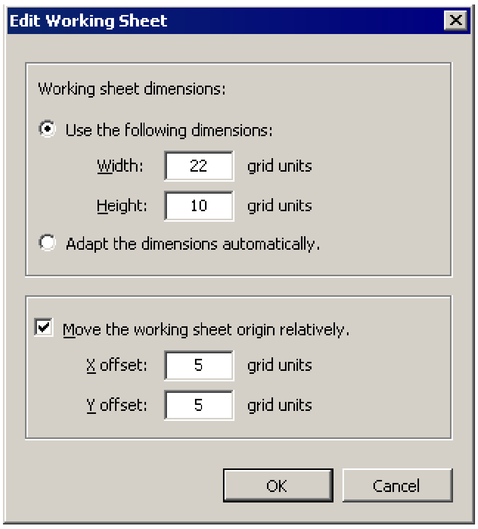
- Resetting Connections, page 277
- Removing Unconnected Connections, page 278
- Editing Parameters..., page 276

3.12.2 Edit Working Sheet...

This command opens the **Edit Working Sheet** dialog, in which the size of the current CFC working sheet can be modified.

The size of the rectangular working area that includes all existing CFC elements is defined by entering height and width, where the origin (X: 0, Y: 0) is located at the upper left corner of the editor window. Height and width are given in integer **grid units**. The size of a grid unit cannot be modified by the user. The height value (Y) increases from top to bottom and the width value (X) from left to right.

The maximum size is 2048 grid units for both the height and the width.



Edit Working Sheet

Working sheet dimensions:

● Use the following dimensions:

Width: 22 grid units

Height: 10 grid units

○ Adapt the dimensions automatically.

☑ Move the working sheet origin relatively.

X offset: 5 grid units

Y offset: 5 grid units

Fig.3-243: Dialog - Edit Working Sheet...

Working sheet dimensions:

Use the following dimensions:

If this option is selected, the working sheet is sized according to the dimensions given for height and width.

- **Width:** Contains the current width in grid units. The width can be changed; however, it is not possible to enter a value that is less than required for the currently existing elements. The width value (X) increases from left to right.
- **Height:** Contains the current height in grid units. The height can be changed; however, it is not possible to enter a value that is less than required for the currently existing elements. The height value (Y) increases from top to bottom.

Adapt the dimensions automatically.:

This option is selected by default. The height of the working sheet is defined by the element at the bottom; the width is defined by the element positioned furthest to the right. The origin (X=0, Y=0) is identical to the upper left corner.

Move the working sheet origin relatively:

After this option is selected, the working area can be moved vertically or horizontally relative to the origin based on the offset values entered (in grid units).

Restrictions:

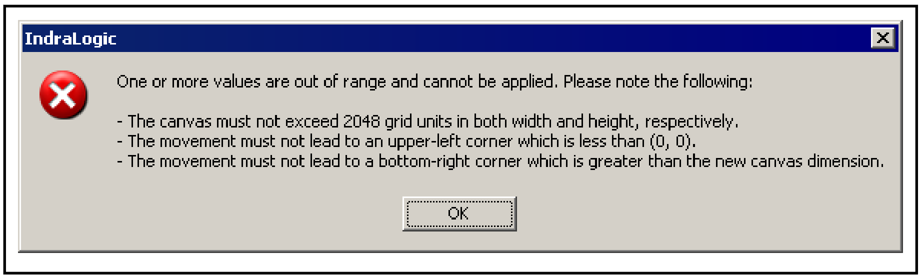
The offsets have to be integers and may not move the upper left corner of the working area out of the window.

When the 'Use the following dimensions' option in the upper dialog section is used, the movement must not exceed the values defined there for width and height.

If the 'Adapt dimensions automatically' option is selected, the movement may exceed the current values for height and width; these are automatically adjusted afterward.

- **X-offset:** By default, 0. Entering a positive number moves the working area to the right and might thus become wider. Entering a negative number moves the canvas to the left. Thus, it is only possible if there is enough space between the element positioned furthest to the left and the left window margin.
- **Y-offset:** By default, 0. Entering a positive number moves the working area down and and might it thus enlarge. Entering a negative number moves the working area up. Thus, it is only possible if there is enough space between the element positioned furthest towards the top and the upper window margin.

If entering invalid dimension values, an error message is output listing the restriction explained above.



IndraLogic

One or more values are out of range and cannot be applied. Please note the following:

- The canvas must not exceed 2048 grid units in both width and height, respectively.
- The movement must not lead to an upper-left corner which is less than (0, 0).
- The movement must not lead to a bottom-right corner which is greater than the new canvas dimension.

Fig.3-244: Error message in case of dimension values that are not permissible

3.12.3 Negation

Icon: -○|

Menu: **CFC ▸ Negation**

This command is used to negate inputs, outputs, jumps or RETURN instructions.

The icon for negation is a small circle on the connection line.

To insert the negation, select the corresponding input or output pin for the respective element and execute the command.

A detailed description of the possible cursor positions is located on the help page cursor positions in CFC. page 310.

A negation can be canceled by executing the command again.

3.12.4 EN/ENO

Icon: EN

Menu: **CFC ▸ EN/ENO**

This command is used to give the selected function block (cursor position 3, page 310) an additional Boolean "Enable" input EN and an additional Boolean output ENO (enable out).

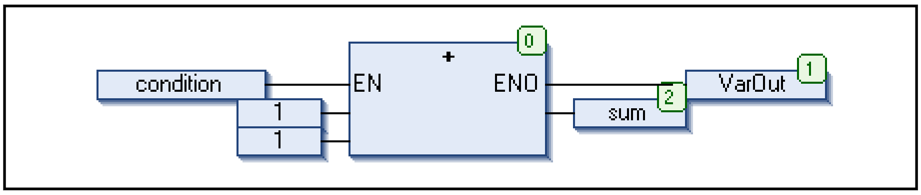


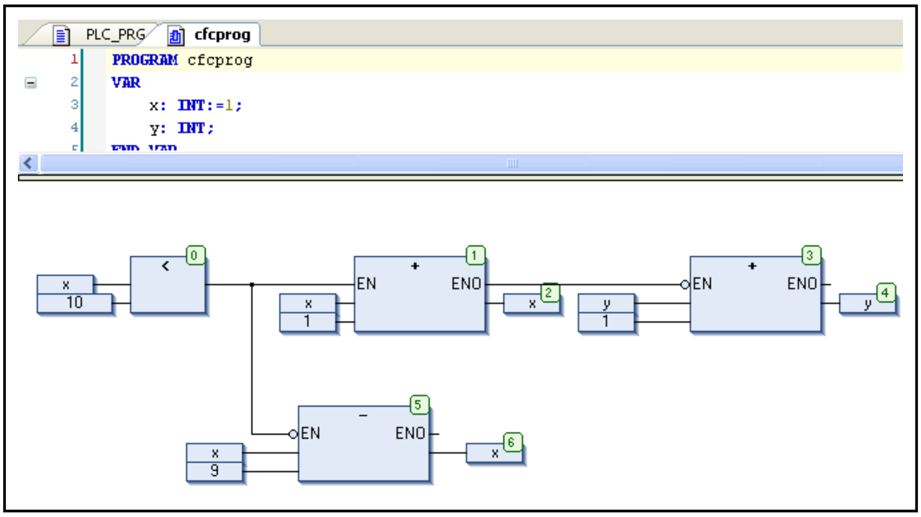
Fig.3-245: ADD function block with EN/ENO

In this example, ADD is only executed if the Boolean variable **condition** is TRUE.

VarOut is set to TRUE after ADD is executed.

If **condition** is set to FALSE again later, ADD is not executed again and **VarOut** becomes FALSE as well!

The following example shows how the ENO value can be used for other function blocks:



```
PROGRAM cfcprog
VAR
    x: INT:=1;
    y: INT;
END_VAR
```

Fig.3-246: Example use of EN/ENO

For this example, initialized **x** with "1".

The numbers in the upper right corner of every function block indicate the execution order in the network.

As long as **x** is less than 10 (0), it increases by 1 (1).

As soon as **x = 10**, the output from LT(0) delivers the value FALSE and SUB (6) and ADD (4) are executed.

x is reset to "1" and **y** increases to 1.

LT (0) continues to be executed as long as **x** is less than 10.

In this way **y** counts the number of times **x** passes through the value range 1 through 10.

3.12.5 Set/Reset

None (Set/Reset)

Icon:

Menu: **CFC** ▸ **Set/Reset** ▸ **None**

This command is used to remove a Set or Reset from an output element. To do this, select the input pin for the output element and execute the command. The "Set" or "Reset" icon at the input will then disappear.

A detailed description of the possible cursor positions is located on the help page cursor positions in CFC. page 310.

R - Reset

Icon:

Menu: **CFC ▸ Set/Reset ▸ R (Reset)**

This command is used to add a "Reset" to a Boolean output element. That means that when the input provides the value TRUE, the output is set to FALSE and retains this value.

To insert a reset, select the input pin for the output and execute the command.

A detailed description of the possible cursor positions for a selection is located in cursor positions in CFC. page 310.

The reset output is designated by a small "R" icon on its input.

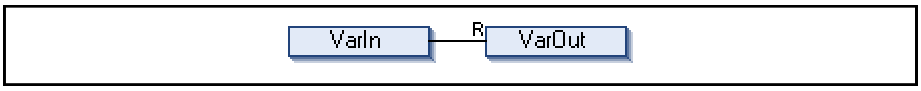


Fig.3-247: Example

In this example, **VarOut** is set to FALSE as soon as **VarIn** delivers the value TRUE. **VarOut** retains this value, even if **VarIn** becomes FALSE again.

Alternatively, either a S (Set) or a None command can be used to define the Set/Reset properties of an output.

S - Set

Icon:

Menu: **CFC ▸ Set/Reset ▸ S (Set)**

This command is used to add a "Set" to a Boolean output element. That means that when the input becomes TRUE, the output is also set to TRUE and retains this value.

To insert a reset, select the input pin for the output and execute the command.

A detailed description of the possible cursor positions for a selection is located in cursor positions in CFC. page 310.

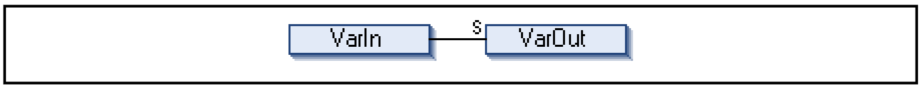


Fig.3-248: Example

In this example, **VarOut** is set to TRUE as soon as **VarIn** delivers the value TRUE. **VarOut** retains this value, even if **VarIn** becomes FALSE again.

Alternatively, either a R - Reset or aNone command can be used to define the Set/Reset properties of an output.

3.12.6 Execution Order

At the Beginning

Icon:

Menu: **CFC ▸ Execution Order ▸ At the beginning**

This command is use to move all of the selected elements in the CFC editor to the beginning of the execution order, page 315,; the sequence among the selected elements is retained. Likewise, the order within the group of elements that were not selected is retained.

At the End

Icon:

Menu: **CFC ▸ Execution Order ▸ At the end**

This command is use to move all of the selected elements in the CFC editor to the end of the execution order, page 315,; the sequence among the selected elements is retained. Likewise, the order within the group of elements that were not selected is retained.

One Ahead

Icon:

Menu: **CFC ▸ Execution Order ▸ One ahead**

This command is used to move all of the selected elements in the CFC editor one place ahead in the execution order, except for the elements that are at the beginning of the execution order, page 315,.

One Back

Icon:

Menu: **CFC ▸ Execution Order ▸ One back**

This command is used to move all of the selected elements in the CFC editor one place back in the execution order, except for the elements that are at the beginning of the execution order, page 315,.

Arrange by Dataflow

Menu: **CFC ▸ Arrange by dataflow**

This command is used to arrange all of the currently selected elements in the CFC editor in the execution order, page 315, (indicated by the element numbers in the upper right corner of each element).

This command is applied to all elements in the CFC editor.

The execution order is determined by the dataflow of the elements and not by their positions.

The following figure shows elements arranged topologically:

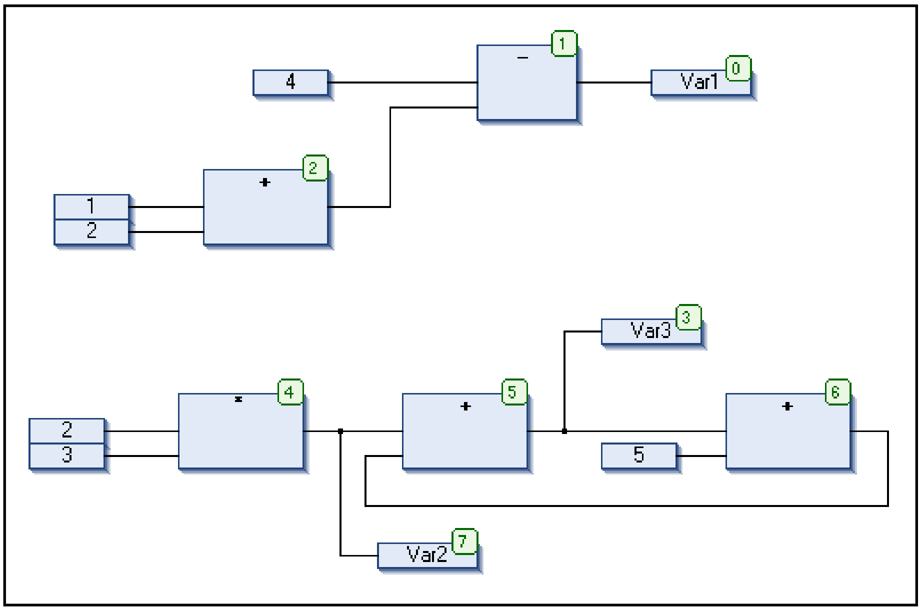


Fig.3-249: Example of topological execution order

After the command is executed, it determines the following order based on dataflow:

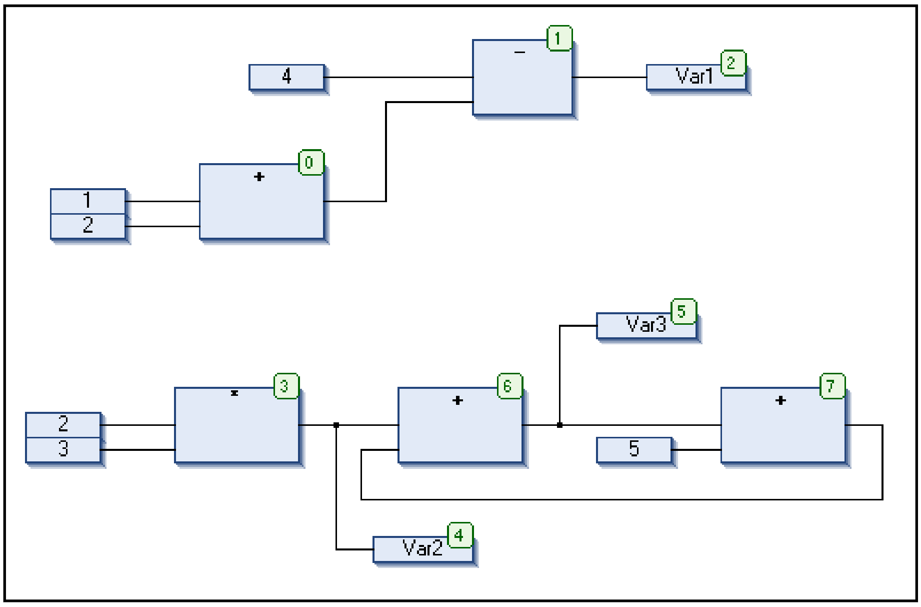


Fig.3-250: Example of execution order by dataflow

When the command is selected, the following happens internally first:

All elements are sorted topologically. Then a new processing list is put together:

based on the known values of the inputs it is determined which of the elements that are not yet numbered can be processed next.

In the upper "network", for example, the function block ADD (2) can be processed immediately, since the values at its inputs ("1" and "2") are known. Only then can the function block SUB (1) be processed, since the result of ADD has to be known, etc.

Feedback, however, is added last. In this way, the new execution order is formed according to dataflow.

The advantage of a sequence based on dataflow is that the output box, which is connected with the output of a function block, follows that output immediately during execution, which is not always the case with topological arrangement.

☞ The topological sequence might also provide a different result than the sequence according to dataflow, as shown from the examples above.

Arrange Topologically

Menu: **CFC ▸ Arrange topologically**

This command is used to arrange all of the currently selected elements in the CFC editor in the topologic execution order, page 315, (indicated by the element numbers in the upper right corner of each element).

Topological arrangement means that processing occurs from left to right and from top to bottom, i.e. with topologically ordered elements, the element numbers that indicate the position of an element in the processing list increase from left to right and from top to bottom. The position of the connections are not significant. Only the position of the elements is important.

When executing the order, first all of the selected elements are removed from the processing list in an internal procedure; then they are added back into the remaining list one by one from bottom right to top left. Each highlighted element is inserted in the processing list before the topological successor. This is shown in an example.

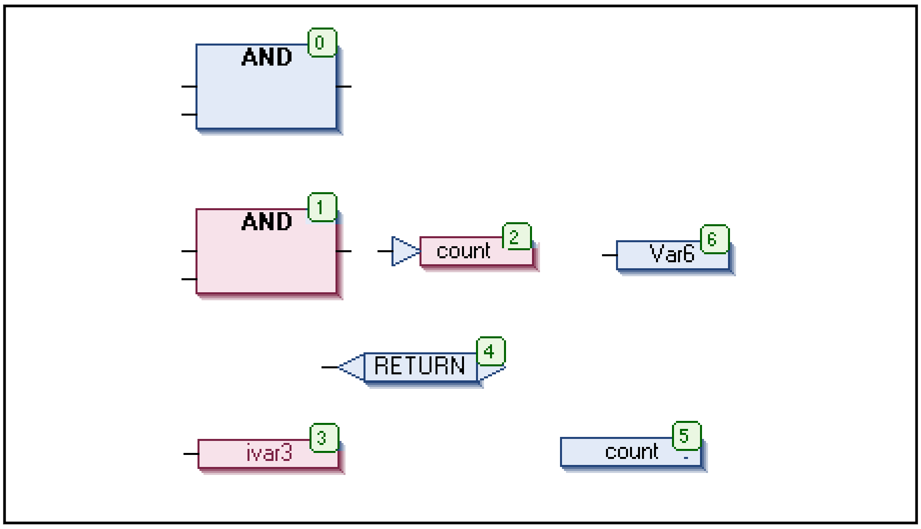


Fig.3-251: Example of topological arrangement, before

The elements with numbers 1, 2 and 3 are selected.

If the 'Arrange topologically' command is selected now, the three selected elements are first removed from the processing list. The following re-insertion is completed in reverse order as compared to the removal:

First of all, ivar3 is arranged in front of the count label, i.e. it becomes 4 and consequently RETURN falls back to 3.

Then, the count jump is arranged in front of Var6 and therefore receives 5.

Consequently, the count label (which just was on 5), ivar3 output and RETURN are in each case moved one downward.

Finally, the AND block is re-inserted in front of the count jump and therefore receives 4.

Therefore, the count label (which just was on 4), ivar3 output and RETURN fall by 1 again. This results in the following new processing order: sequence after

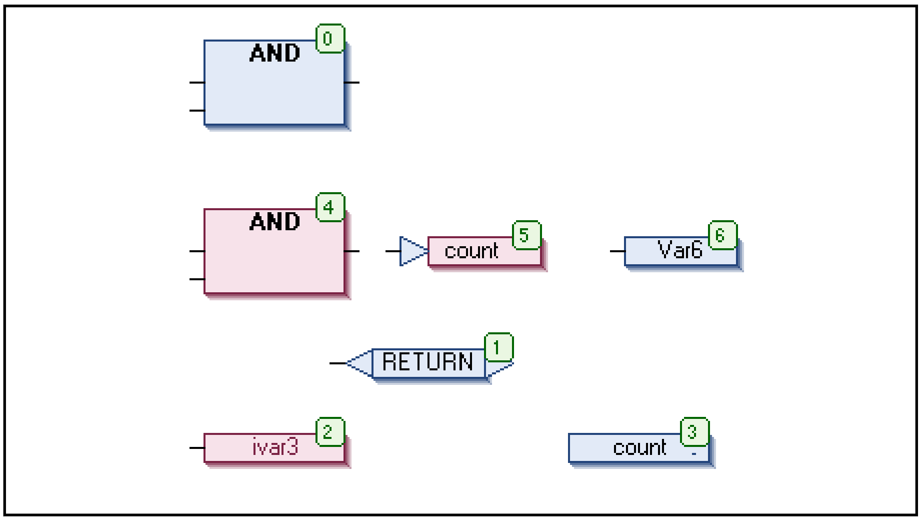


Fig.3-252: Example of topological arrangement, after

When adding a new element, by default it is inserted before its topological successor in the processing list.

☞ The topological sequence might also provide a different result than the sequence according to dataflow, as shown from the examples above.

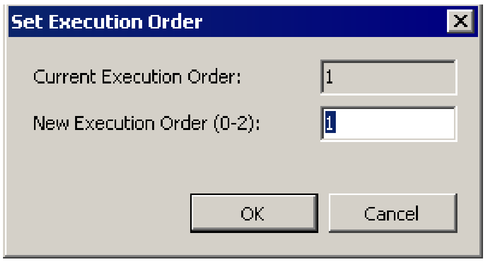
Set Execution Order

Menu: **CFC ▸ Set execution order...**

With this command, the element number of the currently selected element can be redefined in order to change the position of the element within the execution order, page 315,.

The command opens the 'Set execution order" dialog.

The current element number is displayed in the **Current Execution Order** field and the new number can be entered in **New Execution Order**. The possible values are given in parentheses.


Set Execution Order

Current Execution Order: 1

New Execution Order (0-2): 1

Fig.3-253: 'Execution order' dialog

3.12.7 Edit Parameters

Menu: **CFC ▸ Edit Parameters...**

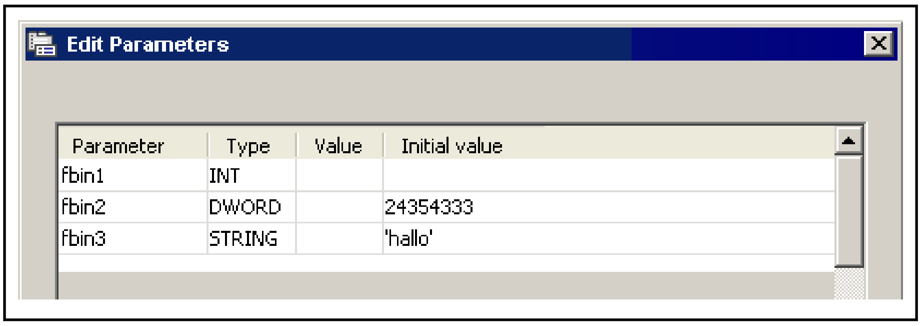
Constant input parameters (VAR_INPUT CONSTANT) of functions and function blocks are not directly displayed in the CFC editor (see the following example for a declaration).

Example for declaration

Declaration of VAR_INPUT CONSTANT

```
FUNCTION_BLOCK fublo2
VAR_INPUT CONSTANT
 fbin1:INT;
 fbin2:DWORD:=24354333;
 fbin3:STRING:='hallo';
END_VAR
...
```

In order to access these parameters, use the 'Edit parameters' command which opens the dialog with the same name. Here the name (parameter), data type (type) and initialization value are displayed for each parameter.



Edit Parameters

Parameter	Type	Value	Initial value
fbin1	INT		
fbin2	DWORD		24354333
fbin3	STRING		'hallo'

Fig.3-254: 'Edit parameters' dialog

The values of the constant input parameters (VAR_INPUT CONSTANT) can be changed here.

To do this, click on the respective field in the value column to select the value and click again (or press <Enter>) to open the input field. Enter the desired value and confirm it with the <Enter key>. <Cancel> discards the entry. <OK> saves all of the changes made.

☞ This functionality and the declaration of variables with the keyword "VAR_INPUT CONSTANT" applies only for the CFC editor.

In the FBD editor all of the input parameters are always displayed at the function block, no matter if they are declared as VAR_INPUT or VAR_INPUT CONSTANT. Even for text editors this does not make any difference.

3.12.8 Connect Selected Connections

Icon:

Menu: **CFC ▸ Connect Selected Connections**

This command can only be executed if exactly one output and several inputs are selected.

Press and hold the <Ctrl> key and click on the desired inputs and outputs to select them.

When the command is called, a connection is established between the connection of the output and the connections of the inputs.

3.12.9 Reset Connections

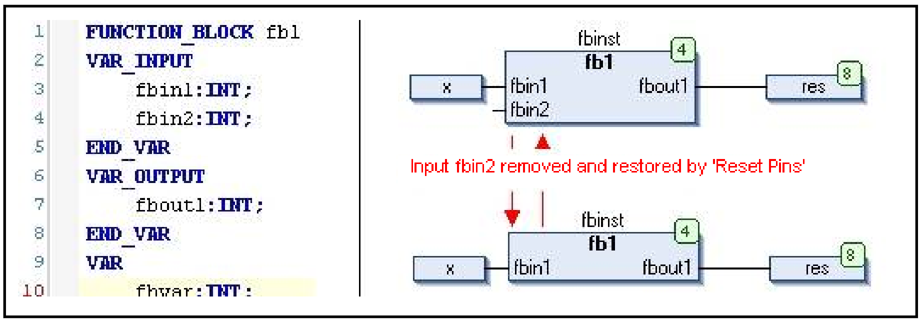
Icon:

Menu: **CFC ▸ Reset connections**

If an input pin or an output pin was removed from a function block, e.g. since it was not used, or if the interface of a function block has changed, this command can be used to display all of the inputs and outputs of the function block defined in its implementation again.

The command can also be used to display the type VAR_IN_OUT parameters for a function block, which are hidden by default.

In the following example, the input pin fbin2 of the function block instance was deleted in the CFC editor, since it was not used. When the instance element is selected and the "Reset connections" command is executed, fbin2 is displayed again.



```
FUNCTION_BLOCK fb1
VAR_INPUT
    fbin1:INT;
    fbin2:INT;
END_VAR
VAR_OUTPUT
    fbout1:INT;
END_VAR
VAR
    fbvar:INT;
```

Fig.3-255: Example of resetting connections

3.12.10 Remove Unconnected Connections

Icon:

Menu: **CFC ▸ Remove Unconnected Connections**

In the currently selected editor, this command removes the unconnected connections from the program call, function block or for actions that are not local.

However, the disconnected connections are not removed from calls of functions, methods or operators, since it would result in invalid syntax.

3.13 Sequenctial Function Chart - Commands

3.13.1 Sequential Function Chart - Commands, Overview

The commands for Sequential Function Chart (SFC) are available for programming in the SFC editor, page 399,.

By default, they are included in the SFC menu which appears in the menu bar if the SFC editor is active.

Alternatively, they are provided in the context menu for the SFC elements in the working sheet.

An SFC elements Toolbox, page 405 is ### in preparation ###.

If necessary, use the dialog under **IndraWorks ▸ Tools ▸ Customize ▸ Commands ▸ SFC**to change the menu configuration.

The commands:

- Initial step, page 279
- Add input action, page 279
- Add output action, page 280
- Add step transition, page 280
- Add step transition after, page 281
- Parallel, page 281,
- Alternate, page 282
- Add branching, page 282
- Add branch right, page 282,
- Add action association, page 283,

- Add action association after, page 284
- Add jump, page 284
- Add jump after, page 284
- Add macro, page 285,
- Add macro after, page 285
- Display macro, page 285,
- Exit macro, page 286,

3.13.2 Initial Step

Icon:

Menu: **SFC ▸ Initial step**

This command is used to perform the currently selected step into the initial step, page 405,.

After the command is executed, the frame of the step element becomes a double line.

The step that was previously the initial step automatically becomes a normal step and is displayed with a simple frame.

The command can be useful if converting the chart.

When a new SFC object is created, it automatically contains an initial step, followed by a transition (TRUE) and a jump back to the initial step.

The "Initial step" property404, can also be enabled or disabled in the element properties of a step element; however, there is no automatic adjustment of this setting for other steps.

Note that the diagram to the initial step can be reset with the following variables, page 414, "SFCInit" or "SFCReset".

3.13.3 Add Input Action

Icon:

Menu: **SFC ▸ Add input action**

This command is used in the SFC editor to add an input action to the currently selected position.

"<stepname>_entry" is the suggested name. It can be changed. The language for implementing the action can also be selected.

An input action is only executed once, right after the step has become active. A step with an input action is designated by an "E" in the upper left corner.

In the Project Explorer, the input action appears under the POU that contains the SFC.

To edit an input action in the sequence, double-click the corner where the "E" is located in the step.

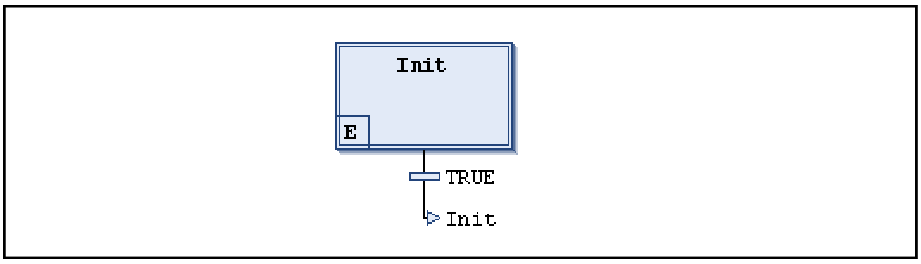


Fig.3-256: Input action added

3.13.4 Add Output Action

Icon:

Menu: **SFC ▸ Add output action**

This command is used in the SFC editor to add an output action to the currently selected position.

"<stepname>_exit" is the suggested name. It can be changed. The language for implementing the action can also be selected.

An output action is only executed once, before the step is disabled. A step with an output action is designated by an "X" in the lower right corner.

In the Project Explorer, the output action appears under the POU that contains the SFC.

To edit an output action in the sequence, double-click the corner where the "X" is located in the step.

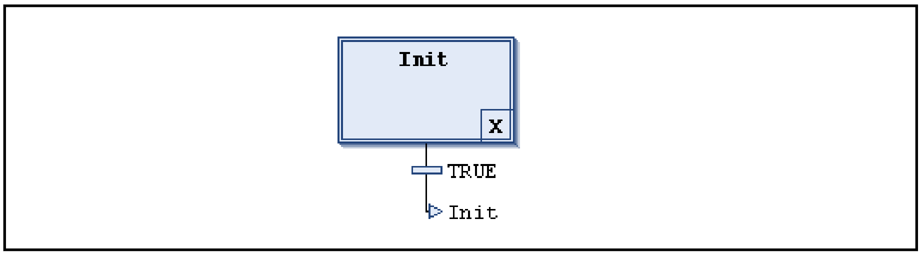


Fig.3-257: Output action added

3.13.5 Add Step Transition

Icon:

Menu: **SFC ▸ Add step transition**

This command is used in the SFC editor to add a step and a transition **before** the currently selected position.

The arrangement (order) of the new step-transition combination depends on which step or which transition is currently selected when adding them. The basic order step-transition-step-transition-... is automatically used.

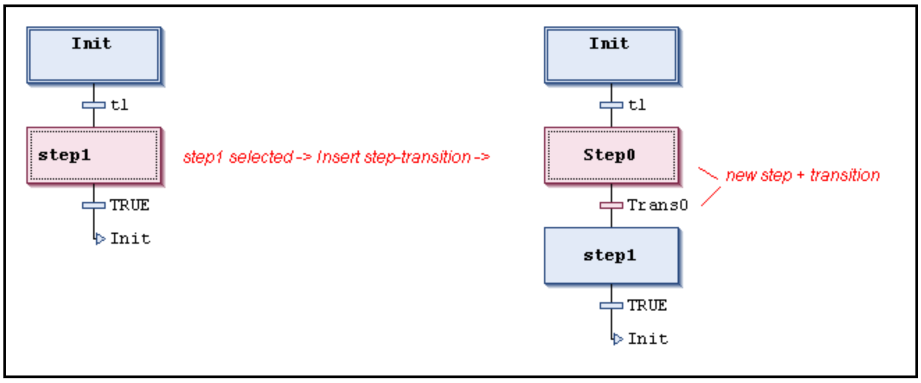


Fig.3-258: Step-transition added after step

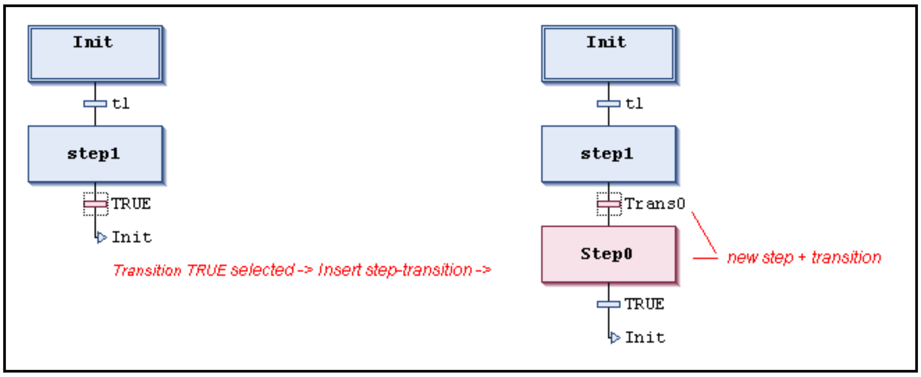


Fig.3-259: Step-transition added after transition

By default, the new step is named "Step<n>".

n is a consecutive number, starting with "0" for the first step, that is added to the initial step.

By default, the new transition is accordingly named "Trans<n>".

The default names can be edited directly (click on the text to open an input field).

3.13.6 Add Step Transition After

Icon: ⇣

Menu: **SFC ▸ Insert Step Transition Later**

This command is used in the SFC editor to add a step and a transition **after** the currently selected position.

The arrangement (order) of the new step-transition combination depends on which step or which transition is currently selected when adding them. The basic order step-transition-step-transition-... is automatically used.

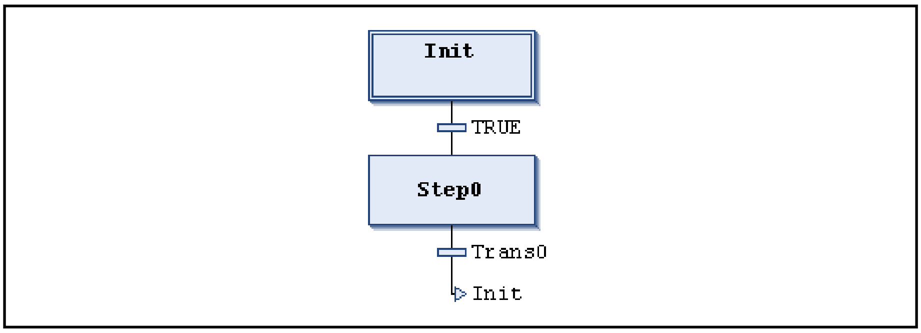


Fig.3-260: Example of a step-transition added "after"

By default, the new step is named "Step<n>". n. "n" is a consecutive number, starting with "0" for the first step, that is added to the initial step.

By default, the new transition is accordingly named "Trans<n>".

The default names can be edited directly (click on the text to open an input field).

3.13.7 Parallel

Icon: ⊟

Menu: **SFC ▸ Parallel**

This command is used in the SFC editor to convert the currently selected "alternate branch" into a "parallel branch".

☞ After a branch is converted, the arrangement of the steps and transitions before and after the branch has to be checked and adapted.

3.13.8 Alternate

Icon:

Menu: **SFC ▸ Alternative**

This command is used in the SFC editor to convert the currently selected "parallel branch" into an "alternate branch".

☞ After a branch is converted, the arrangement of the steps and transitions before and after the branch has to be checked and adapted.

3.13.9 Add Branch

Icon:

Menu: **SFC ▸ Add branch**

This command is used in the SFC editor to add a "branch" **to the left** of the current position.

See also the description for Add branch right.

3.13.10 Add Branch Right

Icon:

Menu: **SFC ▸ Add branch right**

This command is used in the SFC editor to add a "branch" **to the right** of the current position. (To add the branch to the left, use the Add branch command).

- If the topmost element among the currently selected elements is a transition or an alternate branch, an "alternate branch" is added.
- If the topmost element among the currently selected elements is a step, a macro, a jump or a parallel branch, a "parallel branch" with the **jump label** "Branch<x>" is added, where x is a consecutive number, starting with "0" for the first label in the diagram. This default name for the jump label can be edited. The jump label can be indicated as the target for a jump.
- If a common element is currently selected in an existing branch (horizontal line), the new branch is added as new arm at the complete right. If an entire arm of an existing branch is currently selected, the new branch is added directly to the right as new arm.

☞ Note that a branch added with the command "Alternate" or "Parallel" can be transferred into the other type.

Example:

Parallel branch:

In the following figure is a parallel branch that was just added, created with the "Add branch right" command while step11 was selected. A step ("Step2" in the example) is automatically added.

Processing in online mode: If t2 delivers the value TRUE, Step2 is immediately executed after step11 and before t3 is evaluated.

In this case, in contrast to alternate branches, both branches are executed.

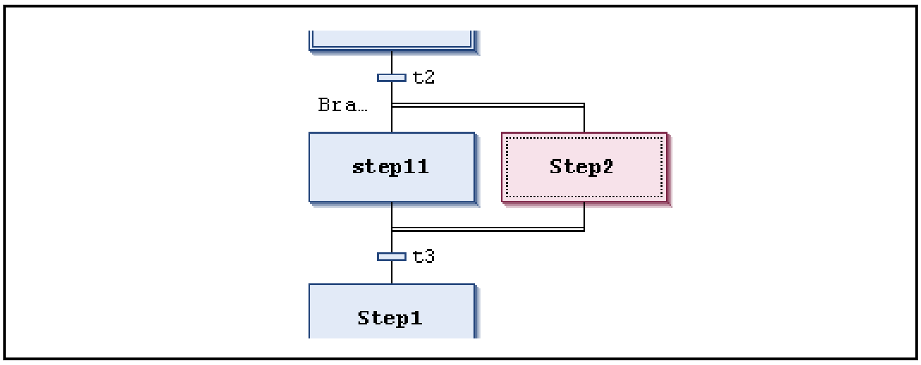


Fig.3-261: Parallel branch

Example:

Alternate branch:

In the follow figure is an alternate branch that was just added, created with the "Add branch right" command while Transition t4 was selected. A step ("Step32" in the example), and two transitions (t41, t42) are added, one before and one after.

Processing in online mode: When Step3 is active, the subsequent transitions (t4, t41) are evaluated from left to right. The first arm of the branch in which the first transition delivers the value TRUE is executed. In this way, in contrast to a parallel branch, only one arm is executed.

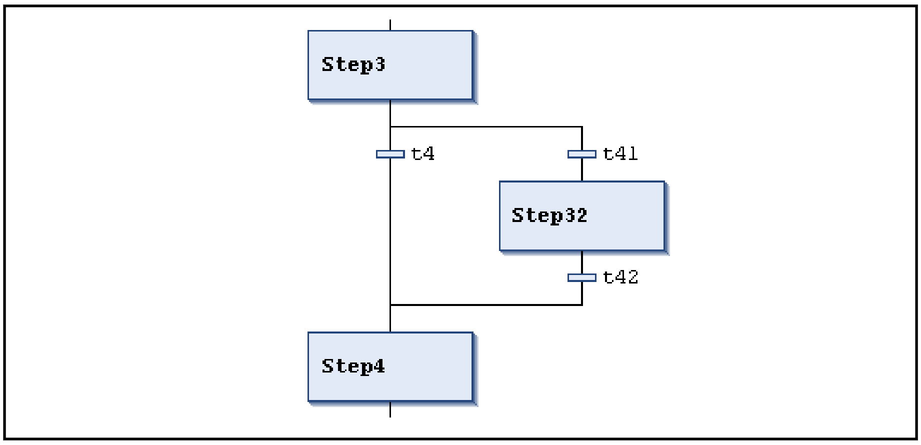


Fig.3-262: Alternative branch

3.13.11 Add Action Association

Icon:

Menu: **SFC ▸ Add action association**

This command is used in the SFC editor to assign an "IEC action" to a step.

The action element is added to the right, next to the currently selected step element.

If the step has been assigned one or more actions, they are displayed in an "actions list". The new action is positioned as follows:

- If the step element is selected, it is positioned as the first action of the step, i.e. in the top position in the action list.
- If one of the existing actions in the action list for the step is selected, it is positioned directly before this action, i.e. directly above.

See also Add action association after.

The left section of the action element contains the **qualifier**, "N" by default. Enter the **action name** in the right section. To do this, click in the box to obtain an editing frame.

An action with the name entered has to be created in the project as a POU. See the Add object234 command.

The qualifier can also be edited.

See the List of valid qualifiers, page 414.

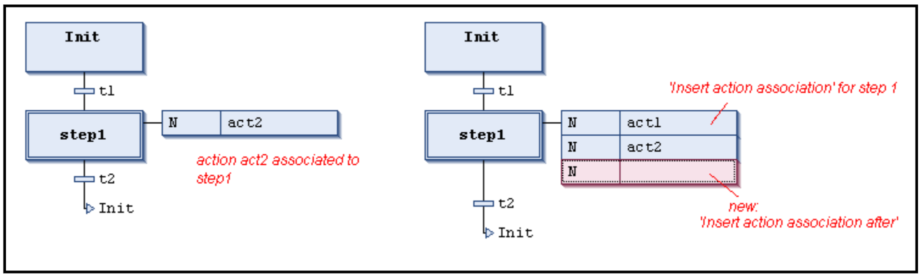

Init
t1
step1
N
act2
action act2 associated to step1
t2
Init
Init
t1
'Insert action association' for step 1
step1
N
act1
N
act2
N
t2
Init
new: 'Insert action association after'

Fig.3-263: Example of actions assigned to a step

3.13.12 Add Action Association After

Icon:

Menu: **SFC ▸ Add action association after**

This command is used in the SFC editor to assign another "IEC action" to a step.

See also in this regard the description for the Add action association command. The difference is that with "add after", the new action is not placed in the first position, but instead in the last position in the actions list, i.e. not above, but instead, below the action currently selected in this list.

SFC elements, ToolBox, page 405

3.13.13 Add Jump

Icon:

Menu: **SFC ▸ Add jump**

This command is used in the SFC editor to add a "jump element" **before** the currently selected element.

The jump is automatically added with the jump target "Step". Replace "Step" with the name of a step or the jump label of a "parallel branch" to which the jump is to be made.

SFC elements, ToolBox, page 405

3.13.14 Add Jump After

Icon:

Menu: **SFC ▸ Add jump after**

This command is used in the SFC editor to add a "jump element" **after** the currently selected element.

The jump is automatically added with the jump target "Step". Replace "Step" with the name of a step or the jump label of a "parallel branch" to which the jump is to be made.

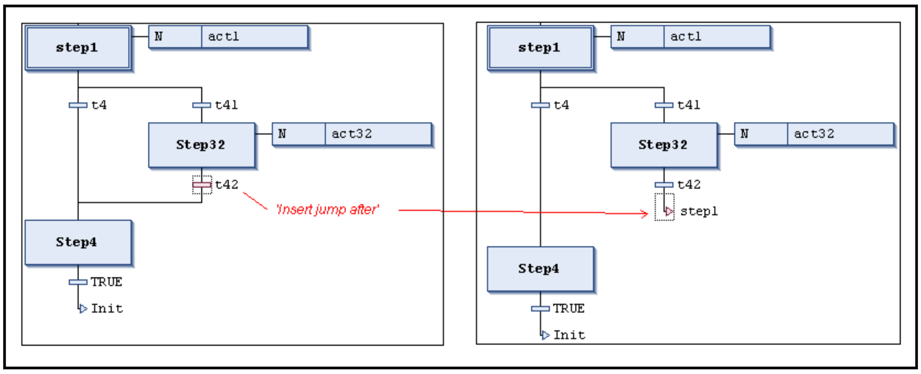


Fig.3-264: Example of a jump added after

SFC elements, ToolBox, page 405

3.13.15 Insert Macro

Icon: ⊞↑

Menu: **SFC ▸ Insert macro**

This command is used in the SFC editor to add a macro element **before** the currently selected element.

By default, the macro is named "Macro<x>", where x is a consecutive number, beginning with "0" for the first macro inserted.

This name can be edited.

To edit or display a macro, the Display macro command can be used to open the macro editor.

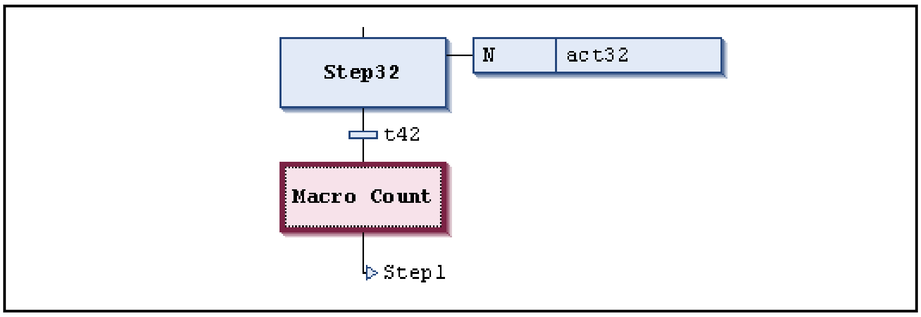


Fig.3-265: Example of macro in an SFC chart (currently selected)

3.13.16 Add Macro After

Icon: ⊞↓

Menu: **SFC ▸ Add macro after**

This command is used in the SFC editor to add a macro after the currently selected element. See the related information at Add macro.

3.13.17 Display Macro

Icon: ▷

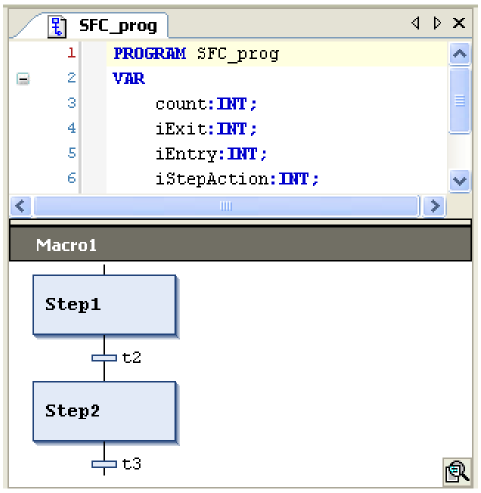
Menu: **SFC ▸ Display macro**

This command is available in the SFC editor if a macro element is selected in the SFC chart.

It is used to open the macro in the macro editor to view or edit it.

The main view of the SFC editor is closed and the macro editor is opened. It is also an SFC editor in that you can now view and edit a section of the SFC diagram that is only displayed in the main view as a macro box. A zoom menu is also available here in the lower right corner.

To return to the main view of the SFC editor, use the Exit macro command.



```
PROGRAM SFC_prog
VAR
    count:INT;
    iExit:INT;
    iEntry:INT;
    iStepAction:INT;
```

Fig.3-266: Example of the macro editor

The command can be used in offline and online mode.

3.13.18 Exit Macro

Icon: ◁

Menu: **SFC ▸ Exit macro**

This command is available in the SFC editor when the macro editor is currently open (using the Display macro command).

The macro editor is closed again and you return to the main view of the SFC editor.

The command can be used in offline and online mode.

3.14 Text Lists - Commands

3.14.1 Text Lists - Overview of Commands

The commands for working with text lists are described in this section.

By default, these commands are available in a 'Text list' menu as soon as a text list is being edited or partially in the 'VI-Logic visualization' menu.

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize.**

Commands:

- Create global text list, page 287
- Add language, page 287
- Remove language, page 288
- Import/Export text lists, page 288
- Update visualization text IDs, page 293
- Check visualization text IDs, page 293

- Remove unused text list entries, page 294

3.14.2 Create Global Text List

Icon:

Use "Create global text list" to create a global text list explicitly.

The object is named "GlobalTextList" and is added to the "General module" folder.

To create a global text list, in the main menu click on **VI Logic Visualization ▸ Create global text list**. Alternatively, "Create global text list" is available in the context menu if the cursor is in the working area of the visualization editor.

The "GlobalTextList" object is **automatically** created as soon as the first text is defined when configuring a visualization.

Refer to text list, page 55 for detailed information.

3.14.3 Add Language

Icon:

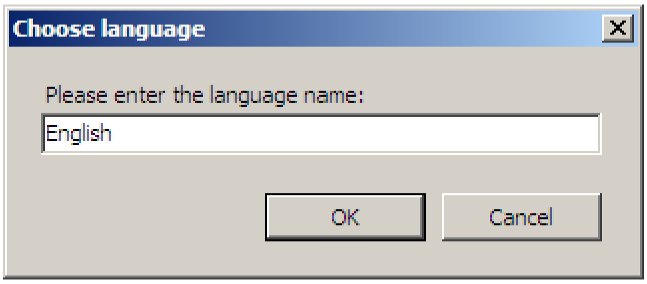
Use "Add language" to add a column in a text list for an additional local country language.

To add a new language in a text list, open the editor for the "text list" or "GlobalTextList" object in the working area first.

Then, in the main menu, click on **VI Logic text list ▸ Add language**.

This opens the "Choose language" dialog.

In the "Choose language" dialog, enter a name (column heading in the text list) for the new language and confirm your entry with the "OK" button.

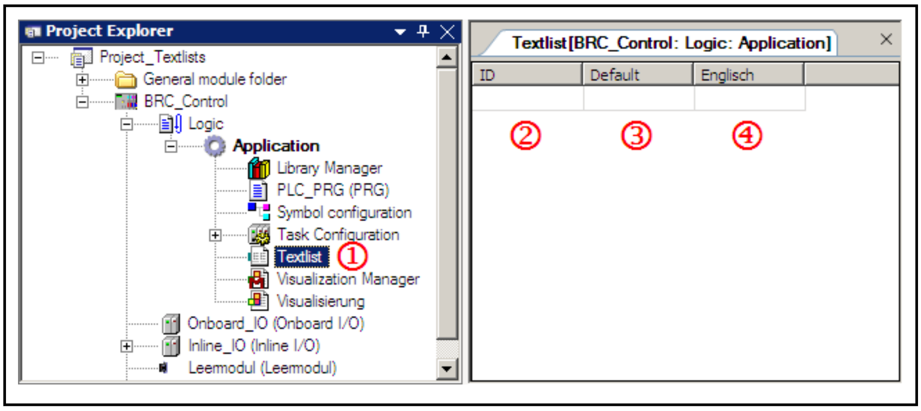


Choose language

Please enter the language name: English

Fig.3-267: 'Choose language' dialog

Then the corresponding column is created in the text list.



① Object text list

② Unique ID (qualifier, index) specified in the configuration of a visualization element.

③ Default language, always available. If there is no entry in the text list that matches the language currently set in IndraLogic, the entry defined as default is used.

④ Newly defined language, English in this case

Fig.3-268: Object text list

Refer to text list, page 55 for detailed information.

3.14.4 Remove Language

Icon:

Use "Remove language" to remove the column for a certain language from the text list.

To remove a language from a text list, place the cursor in one of the column cells and click in the main menu on **VI Logic text list ▸ Remove language**.

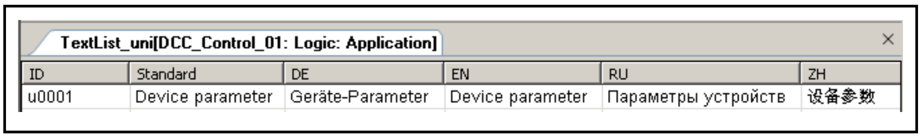
Alternatively, "Remove language" is also available in the context menu.

3.14.5 Import/Export Text Lists

Icon:

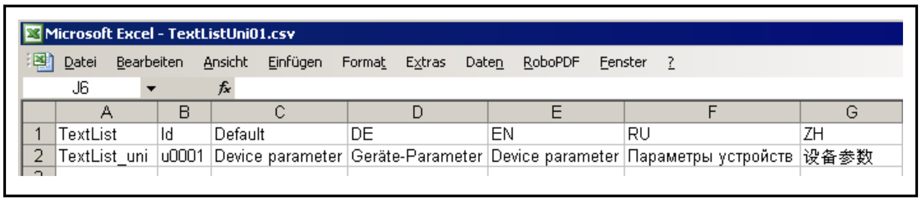
Use "Import/Export text lists" to exchange data with other programs, e.g. Excel.

The data format used is .csv ("Comma Separated Values").


TextList_uni[DCC_Control_01: Logic: Application]

ID	Standard	DE	EN	RU	ZH
u0001	Device parameter	Geräte-Parameter	Device parameter	Параметры устройств	设备参数

Fig.3-269: Unicode text list object in IndraWorks

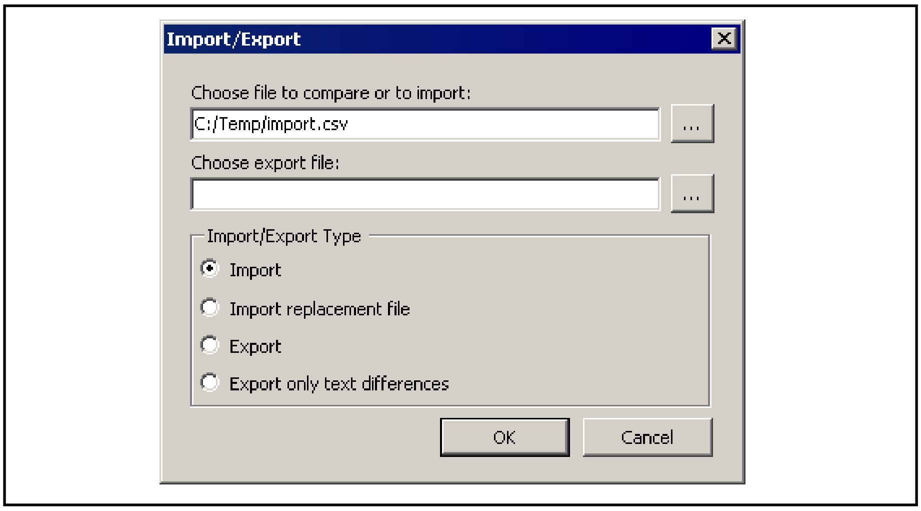

Microsoft Excel - TextListUni01.csv
Datei Bearbeiten Ansicht Einfügen Format Extras Daten RoboPDF Fenster ?
J6
fx

	A	B	C	D	E	F	G
1	TextList	Id	Default	DE	EN	RU	ZH
2	TextList_uni	u0001	Device parameter	Geräte-Parameter	Device parameter	Параметры устройств	设备参数

Fig.3-270: Unicode text list object opened in MS Excel

☞ Both the text list object and the csv import/export format can be used for the characters.

The following dialog appears when calling the command



Import/Export

Choose file to compare or to import: C:/Temp/import.csv

Choose export file:

Import/Export Type

● Import

○ Import replacement file

○ Export

○ Export only text differences

Fig.3-271: Import/Export dialog

The files for an import, export or for comparison can be determined by specifying the corresponding path or by using the input assistance (...). Specify the actions is to be carried out by a corresponding selection in the lower section of the dialog:

Import:

When an external file is imported, the data sets of the external file are compared with the data sets of the project. The data sets in the project are adapted according to the following rules:

- If the text content of the files is the same, the data set is not changed.
- If a translation has been added to the external file, it is accepted into the data set in the project.
- If the text has been modified within a translation, the change is accepted into the data set in the project.
- If translated texts are missing in the external file, the data set in the project is not changed.
- If a new line has been added to the external file, the new data set is accepted into the project.
- If the project contains an additional data set, it is retained.
- A change within the Default column can be considered the same as adding a new text. If text passages have, however, several spaces instead of one, this is not regarded as change!

See example - importing a .cvs file, page 289.

Example - Importing a .cvs file

Example:

Data content in the external file:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel

Text list	ID	Standard	German	English
TextList1	1	Door		
TextList2	2	Light		

Data content of the text lists for the current project before import:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door	Tür	Door
TextList2	2	Seat	Sitz	Seat

During import, all differences are accepted into the project. At this time, both lists are compared and adapted so that the content of the following text lists in the project is created:

Result:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door	Tür	Door
TextList1	2	Light		
TextList2	3	Seat	Sitz	Seat

Import list of changes (replacement file)

When importing text lists, a change within the Default column is considered the same as adding a new line. The reason for this is that the Default column serves as the key for comparing lines during export/import.

Note: If text passages in the "Default" column have, however, several spaces instead of one, this is not regarded as change!

If the text within the Default column is to be changed (correcting a typing error or supplementing existing text), a list of changes (replacement file) is required.

Example - Importing a list of changes

Default, old	Default, new	Command
Cancel ?	Cancel	REPLACE
Do you want to save ?	Do you really want to save ?	REPLACE_AND_REMOVE
Do you really want to save ?!	Do you really want to save ?	REPLACE_AND_REMOVE

The list of changes is processed from top to bottom. This way, the change history can also be considered.

The command describes what is to be done with the text line.

Currently, only the **REPLACE** command is available. It has the following effect:

Normally, the text in the "Default" column is replaced by the new text. In the example, "Cancel ?" is replaced by "Cancel" and "Do you want to save ?" by "Do you really want to save ?".

At the same time, the texts for all of the visualization elements are adapted, i.e. the old entries are replaced by the new texts here as well.

If the new default text is already contained in a field of the "Default" column in another line in the text list, the line that contains the entry to be replaced is deleted entirely. The respective visualization elements receive entries from the remaining column with the same default entry.

In the example, this occurs for the entry "Do you really want to save ?!", which is to be replaced by "Do you really want to save ?". Since the REPLACE command is called for the entry "Do you really want to save ?!", based on the history of the changes, at this point in time there is already a default entry "Do you really want to save ?". To prevent this key from **appearing multiple times**, the line with the old default entry "Do you really want to save ?!" is **deleted** entirely from the text list.

Export

When exporting text lists, all of the changes within the project are compared with an external comparison file. A new export file is created, which follows these rules:

1. If the text contents are identical, the data set is exported unchanged.
2. If a translation has been added to the project, it is accepted as new.
3. If the text has been modified within a translation, the changed text from the project is used for the new data set.
4. If translated texts are missing in the project, the translations of the template are used for the new data set.
5. If a line has been added to the project, it is accepted as a new data set.
6. If an additional line is contained in the external file, it is exported again.
7. A change within the "Default" column can be considered the same as adding a new text.

See example - exporting a .cvs file, page 291.

Example - Exporting a .cvs file

Example:

Data content in the external file:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel		
TextList1	0	Cancel	Abbrechen	Abort
TextList1	1	Door	Tür	Door
TextList2	2	Seat	Sitz	Seat

Data content of the text lists for the current project before export:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel

Text list	ID	Standard	German	English
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door		
TextList1	3	Light		
TextList2				

When exporting, all differences are accepted into the external file. At this time, both lists are compared and adapted such that the following external file is created.

Data content of the external file after the export:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Abort
TextList1	1	Door	Tür	Door
TextList1	3	Light		
TextList2	2	Seat	Sitz	Seat

Export only text differences:

If this option is selected, only the lines that differ from each other in different versions are accepted into the export file. Such difference files are e.g. suitable as template for new compilations to be prepared. As it is to be kept as small as possible, missing entries in the current lists are not considered as deviation.

Also see example - exporting the text differences, page 292.

☞ To find the corresponding data set, the "Default" column is used for the **GlobalTextList** and the ID column is used for all other text lists. For this reason, the ID column is empty for all data sets of the GlobalTextList.

Example - Exporting the text differences

Example:

Data content in the external file:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel		
TextList1	0	Cancel	Abbrechen	Abort
TextList1	1	Door	Tür	Door
TextList2	2	Seat	Sitz	Seat

Data content of the text lists for the current project before export:

Text list	ID	Standard	German	English
GlobalTextList		Automobile	Automobil	Automobile
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	1	Door		
TextList1	3	Light		
TextList2				

During export, all of the lines that differ from each other in different versions (lines 2, 3 and 5 of the current list) are accepted into the export file.

Data content of the external file after the export:

Text list	ID	Standard	German	English
GlobalTextList		Steering wheel	Lenkrad	Steering wheel
TextList1	0	Cancel	Abbrechen	Cancel
TextList1	3	Light		

3.14.6 Update Visualization Text IDs

Icon:

If a static text is modified within a visualization element, the visualization and perhaps the GlobalTextList have to have write access (see TextList/GlobalTextList, page 55).

☞ If they are modified even though no changes were made to the write permission, it can happen that the text IDs no longer match the texts within a visualization element.

The Check visualization text IDs, page 293, command can determine these kinds of errors within all visualizations.

The "Update visualization text IDs" command can automatically correct these cases of error. To do this, all visualizations with error cases and the GlobalTextList have to have write access.

3.14.7 Check Visualization Text IDs

Icon:

If a static text is modified within a visualization element, the visualization and perhaps the GlobalTextList have to have write access (see TextList/GlobalTextList, page 55).

If they are modified even though no changes were made to the write permission, it can happen that the text IDs no longer match the texts within a visualization element.

The "Check visualization text IDs" command can determine these kinds of errors within all visualizations.

The Update visualization text IDs, page 293, command can automatically remedy these kinds of errors within all visualizations.

3.14.8 Remove Unused Text List Entries

Icon:

This command is used to remove texts within the GlobalTextList that are not used in any visualization element.

3.15 Trace - Commands

3.15.1 Trace - Commands, Overview

The commands in the menu item "Trace" are used to work with the trace editor, page 431.

The menu item appears in the main menu or as a context menu (right click in the left section of the window) when the trace editor is active.

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize ▸ Commands** dialog.

Commands:

- Login trace (download), page 294,
- Start/stop trace, page 294,
- Reset trigger, page 295
- Cursor, page 295
- Scrolling with mouse, page 296,
- Zooming with mouse, page 297,
- Reset display, page 297
- Compress 297,
- Expanding, page 298,
- Multi-channel, page 298,
- Upload trace..., page 299
- Load trace..., page 299
- Save trace..., page 299
- Shortcuts in the trace diagram, page 444.

3.15.2 Login Trace (Download)

This command is used to load the code generated for the trace explicitly in the control in order to enable tracing there. This is required for the first use of the trace functionality on an application and later as well if the trace configuration or the application program have been modified.

See also trace editor in online mode, page 442.

3.15.3 Start/Stop Trace

Icons: ▶, ❚❚

This command toggles between the start and stop tracing commands.

If tracing is stopped, the ▶ symbol is displayed. Usage of the command then starts tracing. This restarts the data acquisition on the runtime system and the current values are continuously displayed in the trace editor.

If tracing is just running, the ❚❚ symbol is displayed. Use of the command stops tracing on the runtime system and the latest data acquired is then displayed in the trace editor.

3.15.4 Reset Trigger

Icon:

This command resets the trigger so that the trace is restarted after the trigger has been enabled.

3.15.5 Cursor

Icon:

If the "Cursor" command is activated, the cursors can be managed in the trace diagram of the trace editor, page 431,.

The command is disabled when one of the commands Scrolling with mouse, page 296, or Zooming with mouse, page 297, is enabled.

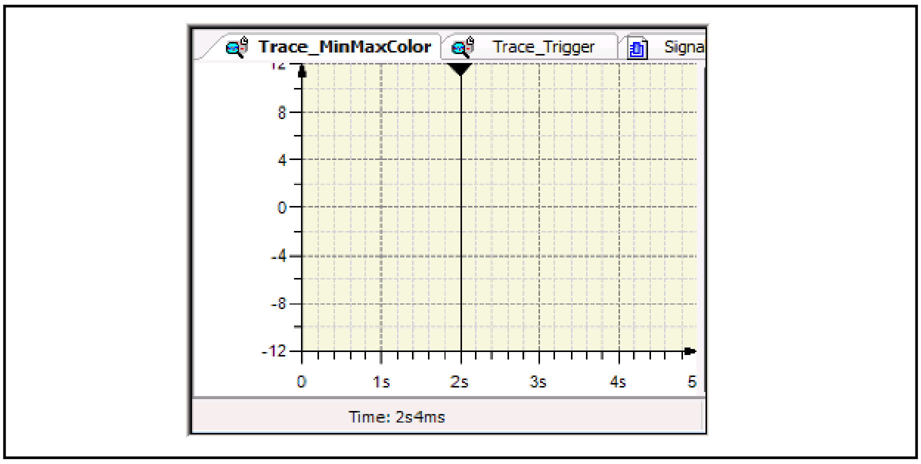
All trace shortcuts are listed in the Shortcuts in the trace diagram, page 444.

One cursor

If there is no cursor, one is crated by clicking into the trace diagram.

One cursor is displayed as black triangle located at the upper edge of the diagram and connected with a fine black line ending at the lower edge of the diagram. The related time value is displayed in the status bar. To move the cursor, click and drag it to the desired position while keeping the left mouse button pressed.

It can also be approached using the <Left> or <Right> arrow keys. The selected cursor is moved step by step as long as the key is held down.



Time: 2s4ms

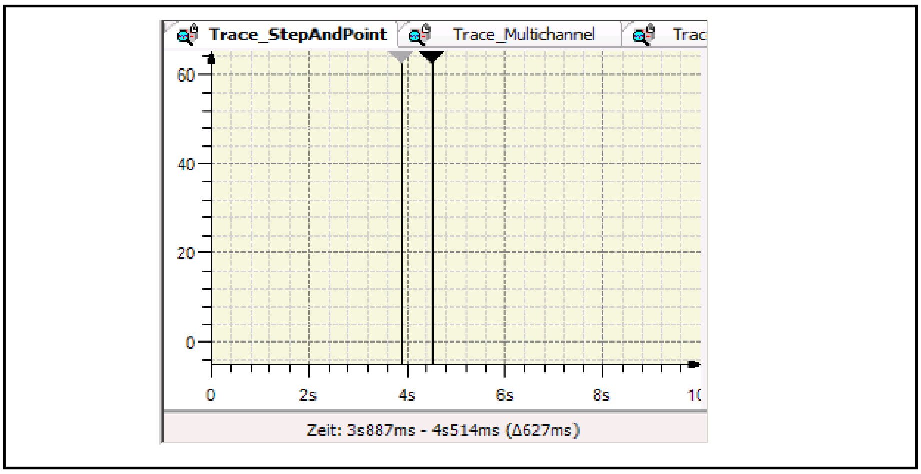
Fig.3-272: Trace diagram with one cursor

The commands are also available if the "Zooming with mouse" or "Scrolling with mouse" mode is enabled.

If you are in the cursor mode, you can also click at the desired position and the cursor exactly positions at this place.

Two cursors

If there is one cursor, can ca generate a second one by clicking on the command. The related time of both cursors as well as their time difference in brackets is now displayed in the status bar as long as the cursor mode is enabled.



Zeit: 3s887ms - 4s514ms (Δ627ms)

Fig.3-273: Trace diagram with ***two*** *cursors*

Move one cursor by clicking and moving it along the time axis while keeping the left mouse button pressed.

This is also possible using the <Left> or <Right> arrow keys to move the **selected** cursor.

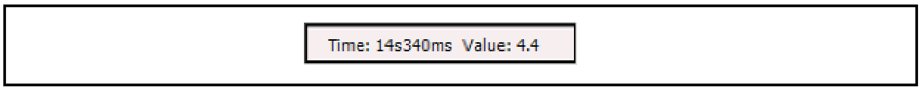
In order to move the **unselected** cursor, use the <Left>/<Right> arrow keys while keeping the <Shift> key pressed.

Deleting the cursor

If there are two cursors, they can both by deleted by clicking on ↖.

If <Del> or ✕ is used, only the **selected** cursor is removed. These commands can also be used if you are in the ✛ "Zooming with mouse" or ✋ "Scrolling with mouse" mode.

If there is no cursor in the trace diagram, the time value and the y-value e of the mouse pointer position are displayed in the status bar.



Time: 14s340ms Value: 4.4

Fig.3-274: Status bar: Time and y-value of the mouse pointer position

3.15.6 Scrolling with Mouse

Icon: ✋

This command activates the "Scrolling with mouse" mode

If it is activated and you click into the trace diagram while keeping the left mouse button pressed simultaneously, the mouse cursor is displayed as double arrow and you can scroll the time axis of the currently displayed trace by means of mouse movements. Using the <Left>/<Right> arrow keys while keeping <Alt> pressed simultaneously has the same effect.

If you click into the diagram and hold the left mouse button and the <Ctrl> key pressed at the same time, the mouse cursor is displayed as double up/down arrow and you can scroll the y-axis of the currently displayed trace by means of mouse movements. The same is achieved by using the <Up> or <Down> arrow keys while keeping <Ctrl> key pressed.

If the "Multi-channel" option is enabled, you can scroll the time and y-axis of all diagrams in the same way with mouse or keyboard. Horizontal scrolling acts on the time axis, vertical scrolling acts on the y-axis of the selected diagram.

To disable the "Scrolling with mouse" mode, activate the "Cursor" mode, page 295 or the "Zooming with mouse" mode, page 297,.

To reset the display, use the Reset display, page 297 command.

All keyboard commands are listed in Shortcuts in the trace diagram, page 444.

3.15.7 Zooming with Mouse

Icon:

This command activates the "Zooming with mouse" mode.

If it is enabled, the mouse cursor is displayed as and by keeping the left mouse button pressed, draw a rectangle in the trace window to reset the area of the displayed trace graph. If the mouse button is released, the diagram is zoomed in time and y-direction so that the content of the rectangular now fills the entire diagram.

To disable the zoom mode, enable the cursor mode, page 295 or the "Scrolling with mouse" mode, page 296,.

To reset the display, use the Reset display, page 297 command.

All keyboard commands are listed in Shortcuts in the trace diagram, page 444.

Note that the **scroll mouse** or the keyboard can also be used to zoom in the trace window:

- Spinning the mouse wheel enlarges/reduces the size of the coordinate system along the x- **and** y-axis. The same action can be performed in the number pad using the <+> and <-> keys.
- It is only zoomed along the x-axis by spinning the mouse wheel while pressing and holding the <Shift> key. The same action can be performed in the number pad by using the <+> and <-> keys while holding down the <Shift> key.
- It is only zoomed along the y-axis by spinning the mouse wheel while pressing and holding the <Ctrl> key. The same action can be performed in the number pad by using the <+> and <-> keys while holding down the <Ctrl> key.

3.15.8 Reset Display

Icon:

This command can be used to reset the display of the recording(s) to the default values after they have been changed by zooming, for example.

The default settings are defined in the configuration dialog, page 434,.

3.15.9 Compress

Icon:

Using this command, the time area that is displayed in the trace editor can be enlarged by a fixed percentage. The graph is compressed. The command can be executed several times.

Use this command to display the values in the sampled graphs in an compressed fashion. With further compressions performed one after another the size of the section of the trace displayed in the window can be enlarged even more.

This command is the counterpart to expand.

3.15.10 Expand

Icon:

Using this command, the size of the time area that is displayed in the trace editor can be reduced by a fixed percentage. The graph is expanded.

This command is the counterpart to Compress.

3.15.11 Multi-Channel

This command enables the view of the Trace graphs to be changed. By default, the graphs of all variables are shown in the same diagram. In Multichannel view, the graph of each variable is visualized in its own diagram with with an identical X-axis. Zooming and scrolling simultaneously affect the X-axis of all diagrams.

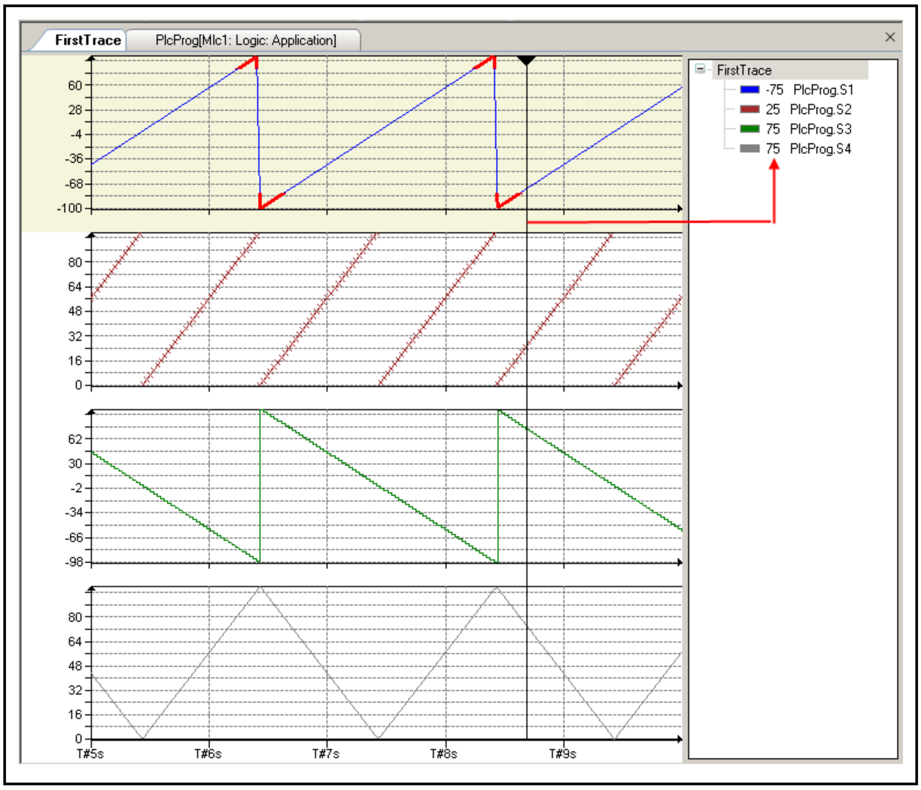


Fig.3-275: Trace in the multi-channel mode

All keyboard commands are listed in Shortcuts in the trace diagram, page 444.

3.15.12 Online List...

This command opens a dialog listing al currently existing control traces. If the trace object is located under an application, all traces from there are listed. If the trace object is placed below a device, the traces of all applications are displayed and moreover the device-specific traces that are implemented in the device.

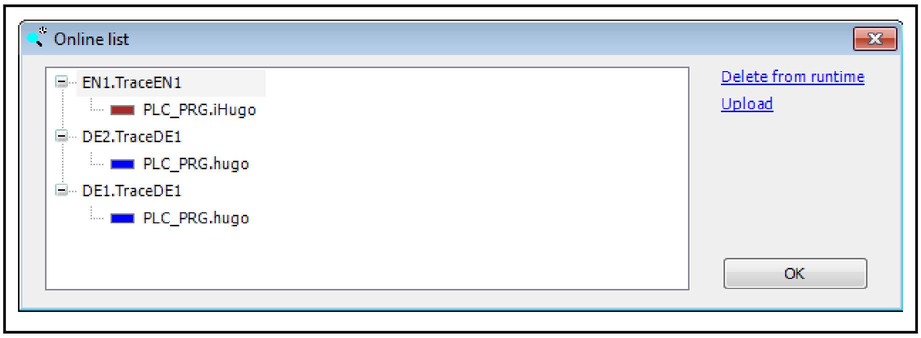


Fig.3-276: "Online list' dialog

Remove from runtime system: By means of this command, the selected trace on the runtime system is removed.

Upload: By means of this command, the selected trace is uploaded from the runtime system into the trace editor.

This command is only available if tracing in the runtime system is carried out by the 'CmpTraceMgr' component.

3.15.13 Upload Trace

This command loads all existing traces. If the trace object is positioned below an application, a trace of the runtime system with the same name as the trace object is uploaded into the trace editor and the previous trace configuration and record is overwritten. If the trace object is positioned below a device, every trace that ran on the runtime system can be loaded into the trace editor.

The extended name with instance path provides for unique names (e.g. "Application.Trace.MyRecord").

This command is only available if tracing is carried out in the 'CmpTraceMgr' runtime system component.

3.15.14 Load Trace...

Use this command to load the trace previously saved in a file into the editor again, completely with configuration, data and time stamp.

The file can be generated using the Save trace..., page command.

Calling the command opens the "Load Trace" dialog (default dialog for browsing the file system) and a trace file can be searched.

Choose between the trace dump (*trace) and trace file (*.trace) file formats from a dropdown list. Then, a suitable file can be selected from the file list and the dialog can be finished by means of <Open>.

The graph is then displayed according to the trace configuration in the trace editor as it is saved in the file.

3.15.15 Save Trace...

This command is used to save the current traces in a file.

Clicking it opens the Save trace dialog (default dialog for saving files). There, determine a file name and a storage location and choose between the trace file (*.trace) and text file (*.txt) file format in the dropdown list. Then, the dialog is to be finished by means of Save.

Traces that are saved as **trace file** (TRACE format: *.trace) contain the configuration settings with the recorded values which can then be loaded back into the trace editor. There, carry out an offline analysis of the trace values.

The configuration can be loaded back to the editor from a trace file with the command Load trace...., page .

Traces that are saved as **text file** (TXT format: *.txt) do not contain any configuration data; all recorded values are, however, listed. This file can be loaded and processed by tools supporting the CSV format. The trace editor itself cannot read this format. Loading back into the editor is not possible.

3.16 Visualization - Commands

3.16.1 Visualization - Commands, General Information

The commands for working in the visualization editor, page 445, are described here.

They can be found in the main menu **VI Logic Visualization** and in the context menu.

If required, the menu structure can be reconfigured via the **IndraWorks ▸ Tools ▸ Customize ▸ Commands ▸ Visualization commands.**

Commands:

- Add visualization element, page 301
- Frame selection, page 302
- Interface editor, page 303
- Keyboard operation, page 303,
- Element list, page 304,
- Background, page 304.

Selection:

- Select All, page 301
- Deselect All, page 301
- Group, page 301
- Ungroup, page 301

Alignment:

- Align left, page 305
- Align at top, page 305
- Align right, page 305
- Align at bottom, page 306
- Center horizontally, page 306
- Center vertically, page 306

Order:

- One level up, page 305
- Place on top, page 304
- Send to back, page 305
- One level down, page 305
- Enable keyboard operation, page 306-

3.16.2 Add Visualization Element

This command is used to add a visualization element in the visualization editor.

This corresponds to add items from the ToolBox window, page 446 via drag&drop.

The command opens a submenu in which the desired element can be selected from the currently available elements. The element is then inserted into the upper left corner of the editor window.

3.16.3 Select All

This command can be used to select all of the visualization elements in the visualization that is currently being edited in the editor.

Position the mouse in the visualization editor and in the context menu select **Select All**.

Also refer to

- Deselect All, page 301.

3.16.4 Deselect All

Use this command to suspend the current selection of visualization elements.

Also refer to

- "Select All, page 301".

3.16.5 Group

Icon:

This command is used to group the currently selected visualization elements and to display the group as a single selected object.

To select multiple items, press and hold the <Shift> key while clicking on the desired elements. Alternatively, click in the editor window outside of an element and hold down the mouse key while drawing a rectangle around the desired elements.

To suspend the group, use the command Ungroup, page 301.

The following figure shows grouping (from left to right) and ungrouping (from right to left) for two rectangle elements:

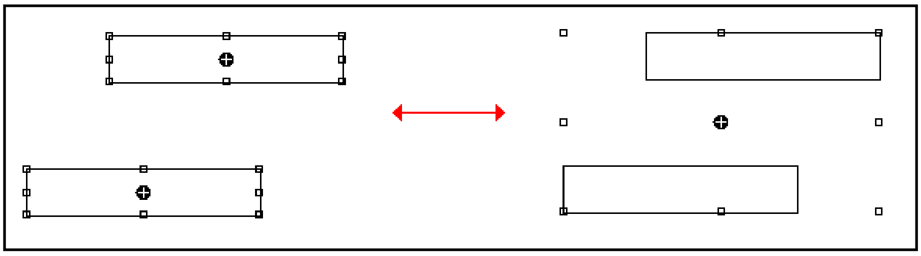


Fig.3-277: Group and Ungroup

3.16.6 Ungroup

Icon:

This command is used to suspend a group of visualization elements.

The individual elements are displayed as individually selected.

A group can be formed with the Group, page 301,command.

3.16.7 Frame Selection

This command can be used to configure "frame" elements.

A "frame" element is used to defined a sub-unit of a visualization. This sub-unit contains one or more visualizations, of which one is always displayed in online mode. Switch back and forth among the available visualizations.

The first visualization in the list of those assigned to the frame is displayed. One of the other visualizations in the frame can be displayed if the user makes an entry on another visualization element that is appropriately configured. This allows to switch back and forth among a variety of displays within one visualization, which contrasts with IndraLogic 1.x, where users had to make a complete jump from one visualization to another.

☞ Further information about "Toggle frame visualizations" and "Input properties" can be found in the visualization editor; see Visualization elements - Properties, page 451.

The visualizations in a frame are references, i. e. instances of the original visualizations, and placeholders defined in the original visualization can be replaced in the frame by local, suitable values.

See also Frames, References, Interfaces, Placeholders, page 628.

The selection of the visualizations in a frame is made in the "Frame configuration" dialog.

To do this, highlight the frame element and select "Frame selection" in the context menu.

The "Configuring the frame visualization" opens; see Dialog - Configuring the frame visualization, page 302.

Alternatively, open the dialog "Configuring the frame visualization" in the main menu using**VI Logic Visualization ▸ Frame Selection** .

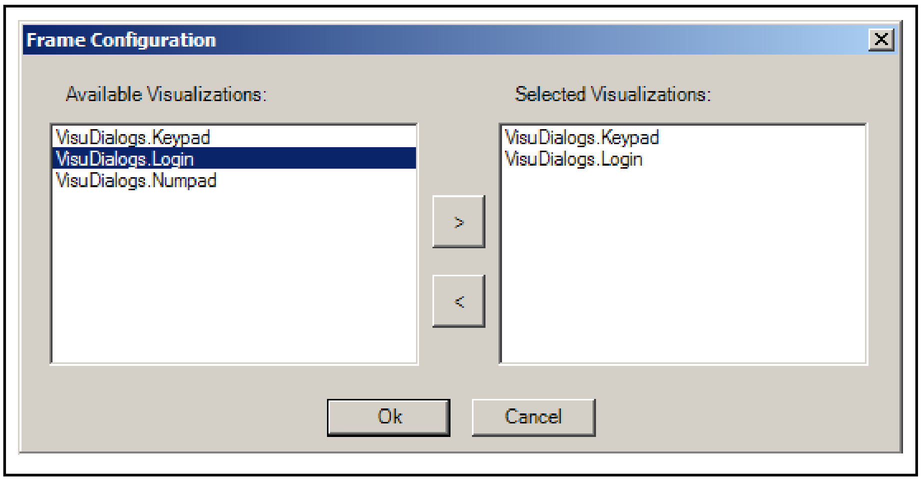


Fig.3-278: Dialog - Configuring the frame visualization

On the left side are the "Available Visualizations" in the project. Select those that are to be referenced in the frame. To do this, click on the arrow buttons to insert or remove visualizations into or from the "Selected visualizations" list.

It is recommended to assign only visualizations from the **global** pool to a frame.

Otherwise problems could occur if a device or application object is renamed later and the path of the assigned visualization is no longer valid.

The order of the selected visualizations from top to bottom determines the automatically generated index numbers for the visualizations. The topmost is assigned "0" and the following "1", "2", etc.

The index numbers are required for configuring the toggle function (Toggle frame visualization, page 451) for another element.

The visualization is initially displayed with index "0".

Example:

Visualization with button bar and display field:

Configure a visualization that contains a button bar and a display field, which displays another visualization in online mode, depending on which button is clicked. To do this, carry out the following steps:

1. Create a visualization "visu_xy" and three other visualizations "visu1", "visu2", "visu3", which are to be displayed later in "visu_xy".
2. Insert three rectangles (buttons) and a frame into "visu_xy". Configure the frame element with the "Frame selection" function and assign it visualizations "visu1", "visu2" and "visu3".
3. Configure the buttons such that when they are clicked, "visu1", "visu2" or "visu3" is called in the frame (input configuration, page 451, OnMouseClick property, toggle frame visualization).

☞ Detailed information about configuring buttons can be found in the visualization editor, see Visualization elements - Properties, page 451.

3.16.8 Interface Editor

Icon:

Default shortcut: <Alt>+<F6>

The interface editor is used to define "placeholder variables" in a visualization, which are then inserted into another visualization in the frame element, i. e. they are referenced.

In the main menu click on **VI Logic Visualization ▸ Interface editor**to open the interface editor.

A detailed description of the interface editor, page 491 can be found in the editors.

3.16.9 Keyboard Operation

Icon:

This command opens the configuration editor for the keyboard operation, page 494, for the current visualization. It is displayed in a tab view in the upper part of the visualization editor.

3.16.10 Element List

Icon:

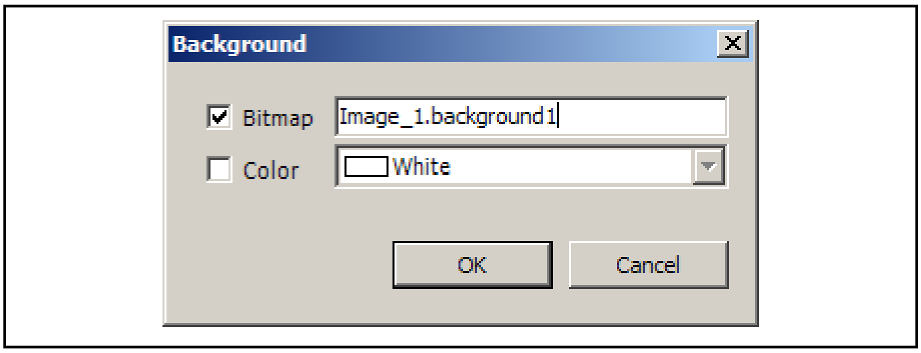
This command opens the element list editor, page 496, for the current visualization. It is displayed in a tab view in the upper part of the visualization editor.

3.16.11 Background

Icon:

A color and an image for the visualization background can be selected in the "Background Information" dialog.

In the main menu click on **VI Logic Visualization ▸ Background information**to open the "Background Information" dialog. Alternatively, open the "Background" dialog by positioning the mouse in the working area of the visualization editor and selecting **Background information** in the context menu.



Background

☑ Bitmap Image_1.background1

☐ Color White

Fig.3-279: Background dialog

Select the desired options:

- **Bitmap:**

  To define a background image, enter the path of an image file available in an image pool in the project here.

  In addition, enter the name of the image pool and the ID of the image file - separated by a dot ".": - On:

  `<ImagePool>.<ID>`

  (e.g.. `Image_1.background1`, `Images_1.43`).

- **Graph color:**

  To define the background color of the visualization, select the desired color from the color selection list.

3.16.12 Order

Place on Top

Icon:

Use this command to position the selected element, page 449, in the foreground of the visualization, i.e. on the "top" level. Elements on lower levels are covered by those on higher levels.

Highlight the visualization elements to be placed in the foreground and in the context menu click on **Order ▸ Place on top**. Alternatively, click in the main menu on **Order ▸ Place on top** .

One Level up

Icon:

Use this command to position the selected element, page 449, one level higher, i.e. closer to the foreground in the visualization. Elements on lower levels are covered by those on higher levels.

Highlight the visualization elements to be placed one level further in the foreground and in the context menu click on **Order** ▸ **One level up**. Alternatively, click in the main menu on **Order** ▸ **One level up** .

Send to Back

Icon:

Use this command to position the selected element, page 449, in the background of the visualization, i.e. on the "deepest" level. Elements on lower levels are covered by those on higher levels.

Highlight the visualization elements to be placed in the background and in the context menu click on **Order** ▸ **Send to Back**. Alternatively, click in the main menu on **Order** ▸ **Send to Back** .

One Level down

Icon:

Use this command to position the selected element, page 449, one level lower in the visualization, i.e. closer to the background. Elements on lower levels are covered by those on higher levels.

Highlight the visualization elements to be placed one level further in the background and in the context menu click on **Order** ▸ **One level down**. Alternatively, click in the main menu on **Order** ▸ **One level down** .

3.16.13 Alignment

Align Left

Icon:

This command is used to align all of the selected visualization elements, page 449, at the left edge of the element positioned farthest to the left.

Highlight the visualization elements to be aligned at the left and in the context menu click on **Direction** ▸ **Align left**. Alternatively, click in the main menu on **Direction** ▸ **Align left** .

Align at Top

Icon:

This command is used to align all of the selected visualization elements, page 449, at the upper edge of the element positioned farthest toward the top.

Highlight the visualization elements to be aligned at the top and in the context menu click on **Direction** ▸ **Align top**. Alternatively, click in the main menu on **Direction** ▸ **Align top** .

Align Right

Icon:

This command is used to align all of the selected visualization elements, page 449, at the right edge of the element positioned farthest to the right.

Highlight the visualization elements to be aligned at the right and in the context menu click on **Direction** ▸ **Align right**. Alternatively, click in the main menu on **Direction** ▸ **Align right** .

Align at Bottom

Icon:

This command is used to align all of the selected visualization elements, page 449, at the lower edge of the element positioned farthest toward the bottom.

Highlight the visualization elements to be aligned at the bottom and in the context menu click on **Direction** ▸ **Align bottom**. Alternatively, click in the main menu on **Direction** ▸ **Align bottom** .

Center Horizontally

Icon:

This command is used to align all of the selected visualization elements, page 449, based on their common middle point along the horizontal axis.

Highlight the visualization elements to be aligned at the horizontal center and in the context menu click on **Direction** ▸ **Center horizontally**. Alternatively, click in the main menu on **Direction** ▸ **Center horizontally** .

Center Vertically

Icon:

This command is used to align all of the selected visualization elements, page 449, based on their common middle point along the vertical axis.

Highlight the visualization elements to be aligned at the vertical center and in the context menu click on **Direction** ▸ **Center vertically**. Alternatively, click in the main menu on **Direction** ▸ **Center vertically** .

3.16.14 Enable Keyboard Operation

Icon: ⌨

This command is in the provided in menu bar for an integrated visualization. It activates or deactivates keyboard operation in online mode, page 657, of a visualization.

When keyboard operation is activated, input to elements and element selection can be performed using specific shortcuts. In this case, other commands given using the keyboard are not executed as long as the visualization editor is active and in online mode.

3.17 Symbol Configuration

The symbol configuration is used to create **symbols** with certain access rights that can be used from external locations to access project (application) variables, e. g. from an OPC server.

The description of the symbols (symbol information) is made available in an xml file (symbol file) of the project directory and simultaneously loaded to the control with the application.

To do this, a **symbol list** is generated that is exported into an XML file in the project directory and, during the application download, loaded into a file on the target system that is not visible to users.

The XML file is named according to the following syntax:

`<ProjectName>.<DeviceName>.<ApplicationName>.xml` (in the following example: IndraLogic.IndraMotionMlp1.Application.xml).

Creating a symbol configuration

A "**Symbol configuration**" object is included below the respective **application** in the device window in the project.

The variables that are to be exported as symbols, can be defined either in the symbol configuration editor, page 398, or by using Pragmas {attribute 'linkalways'}, page 536, which are added to the variable declarations.

The SFC editor provides another possibility: Here, in the element properties, page 414, specify implicitly generated element flags that can be exported to the symbol configuration.

The **symbol name** is generated in the symbol configuration according to the following syntax:

`<ApplicationName>.<POUName>.<VariableName>`

in the following example: "Application.MOTIONPROG.en_PowerRA1" .

☞ When accessing the variable, the complete symbol name always has to be entered using this spelling.

Symbol information

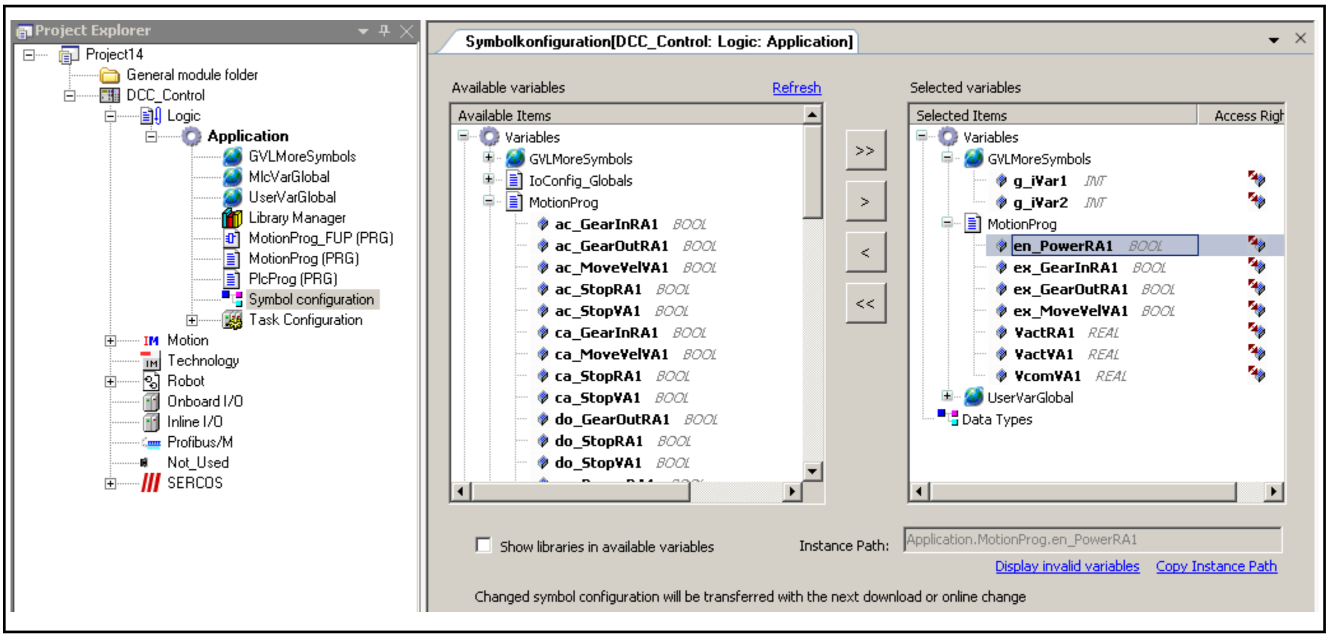
The symbols defined for an application are exported into an xml file in the project directory (**symbol file**).

This file is name according to the following syntax:

`IndraLogic.<DeviceName>.<ApplicationName>.xml`,

(in the following example: IndraLogic.DCC_Control.Application.xml).

The information is **loaded** together with the application onto the control.



□ Show libraries in available variables

Instance Path: Application.MotionProg.en_PowerRA1

Changed symbol configuration will be transferred with the next download or online change

Fig.3-280: Example symbol configuration

If the symbol configuration is modified in online status, it can be explicitly reloaded; see the Download, page 398, button in the editor window.

4 Editors

4.1 CFC Editor

4.1.1 CFC Editor, General Information

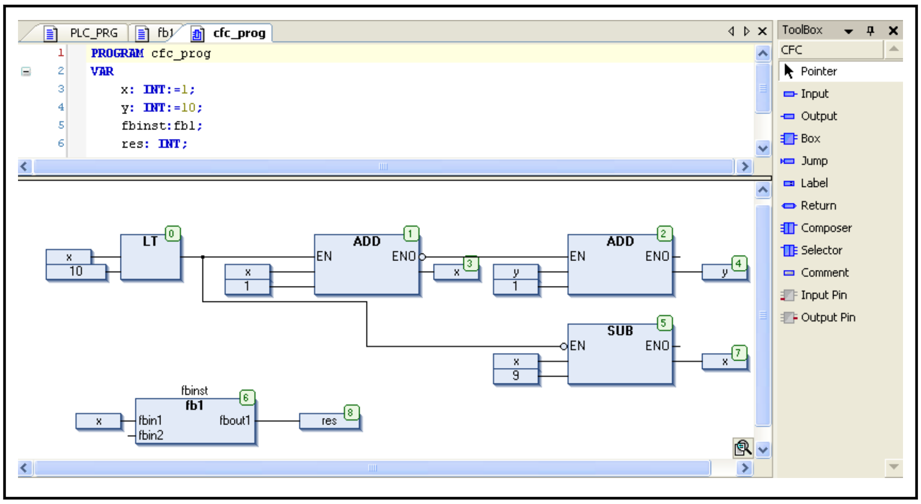
The CFC editor is used to program objects in the **Continuous Function Chart** (CFC) language available in the standard languages of IEC 61131-3.

The **programming language** for a new object is specified when the object is created using the Add object dialog, see page 234,.

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 declaration editor, see page 510.



```
PROGRAM cfc_prog
VAR
    x: INT:=1;
    y: INT:=10;
    fbinst:fb1;
    res: INT;
```

Fig.4-1: CFC editor

In contrast to the network-based editors, the CFC editor allows free positioning, page 315, 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.

The following elements are located in a Ttoolbox, page 311, 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: Inserting and arranging elements, page 315.

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.

☞ The Option "Enable AutoConnect", page 207, 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 function block diagram language, page 339, (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.

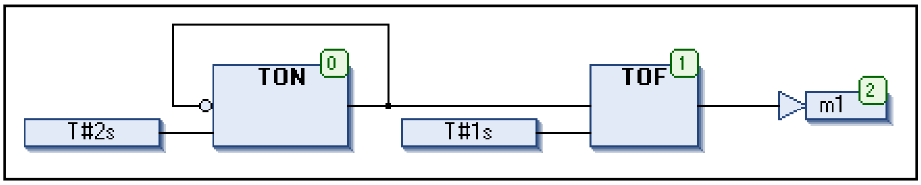


Fig.4-2: Example of a CFC network

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.

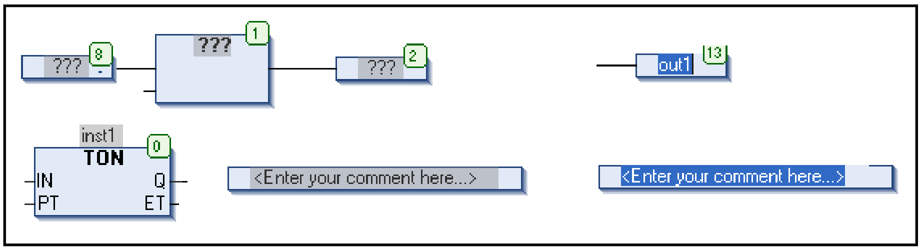


Fig.4-3: Possible cursor positions and two examples of selected text in the CFC

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.

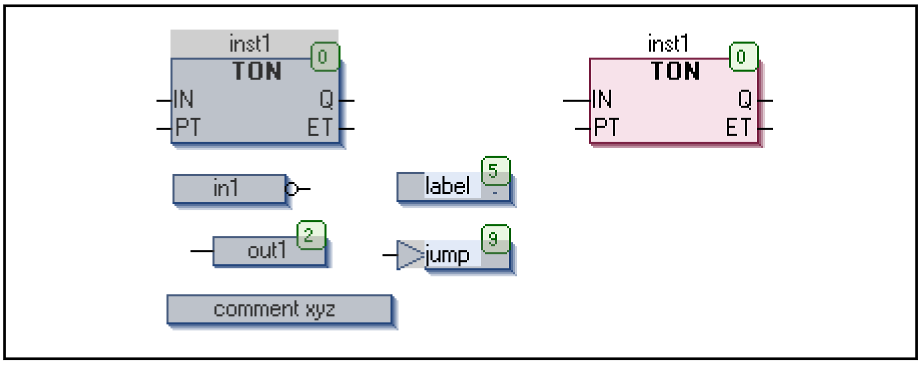


Fig.4-4: Possible cursor positions and an example of a selected function block (box)

3. 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.

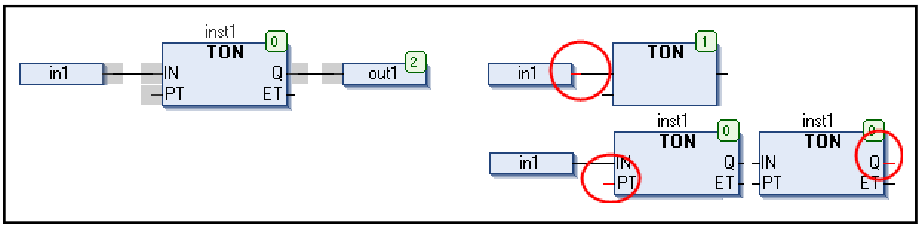


Fig.4-5: Possible cursor positions and examples of selected input and output pins

4.1.4 CFC Elements/Tools

The graphical elements for programming in the CFC editor 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**

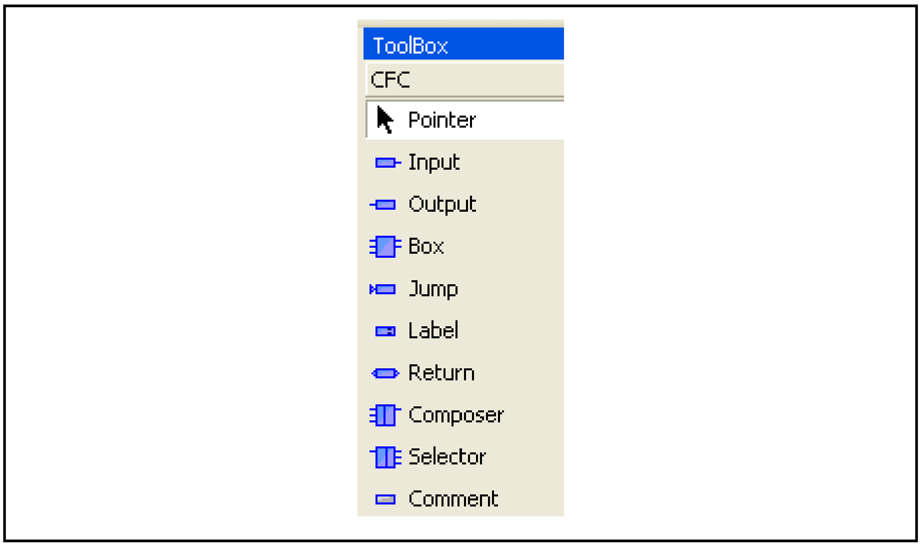
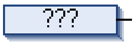
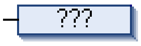
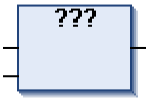
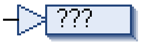
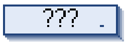
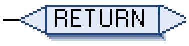
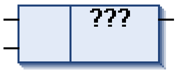


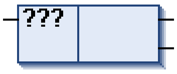
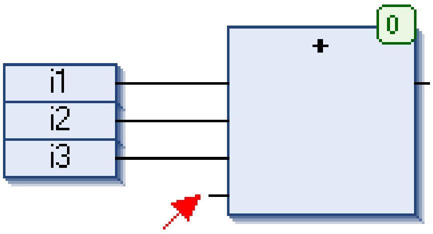
Fig.4-6: CFC tools, standard configuration

The desired element is selected in the toolbox and inserted 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 position and edit them.

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)		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. For function blocks, "???" are also displayed above the function block symbol. These have to be replaced with the name of the function block instance. 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.
	Jump		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). The "???" text has to be replaced as well with the name of the label.
	Label		A label indicates a position to which the program execution can jump using a jump element (see above). In online mode, a RETURN label is automatically placed at the end of a CFC function block.
	Return		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.
	Composer		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. Example: Composer, page 313.

	Selector		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. Example: Selector, page 314.
	Comment		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>.
	Function block input (Input Pin)		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.
	Function block output (Output Pin)		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.

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);
```

CFC program "cfc_prog", declaration and implementation:

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

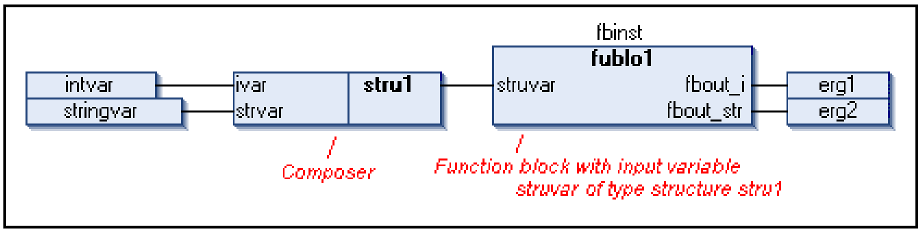


Fig.4-7: Example: Composer

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
```

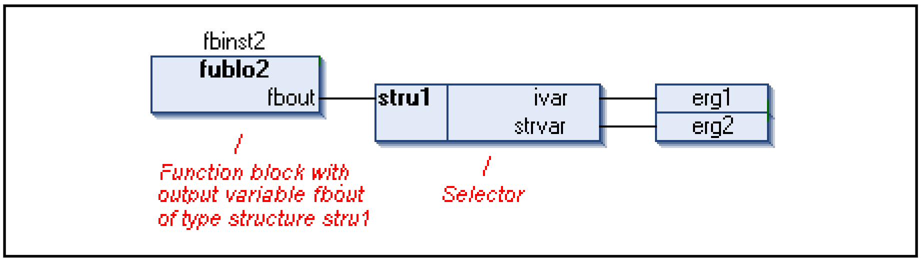

fbinst2
fublo2
fbout
stru1
ivar
strvar
erg1
erg2
Function block with output variable fbout of type structure stru1
Selector

Fig.4-8: Example: Selector

4.1.5 Inserting and Arranging Elements

The elements, page 311, 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 cursor positions, page 310, 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 POUs. This causes that the existing assignments can be deleted.

Moving

To move an element, select it by clicking on the body (see possible cursor positions, page 310) 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 input assistance, page 98,.

Deleting

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

Execution sequence, element numbers

The sequence in which the elements of a CFC network are executed in online 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 commands, page 268, 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.

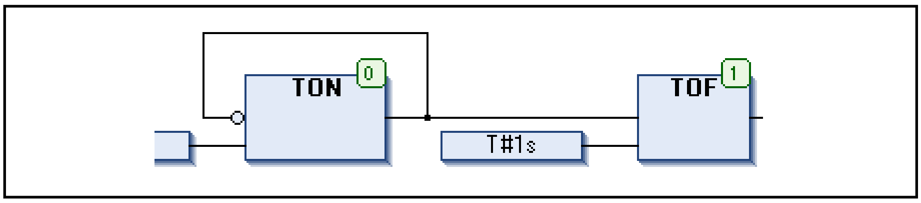


Fig.4-9: Example of element numbering

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 either by selecting and Moving 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.

('Edit Working Sheet' dialog, page 269).

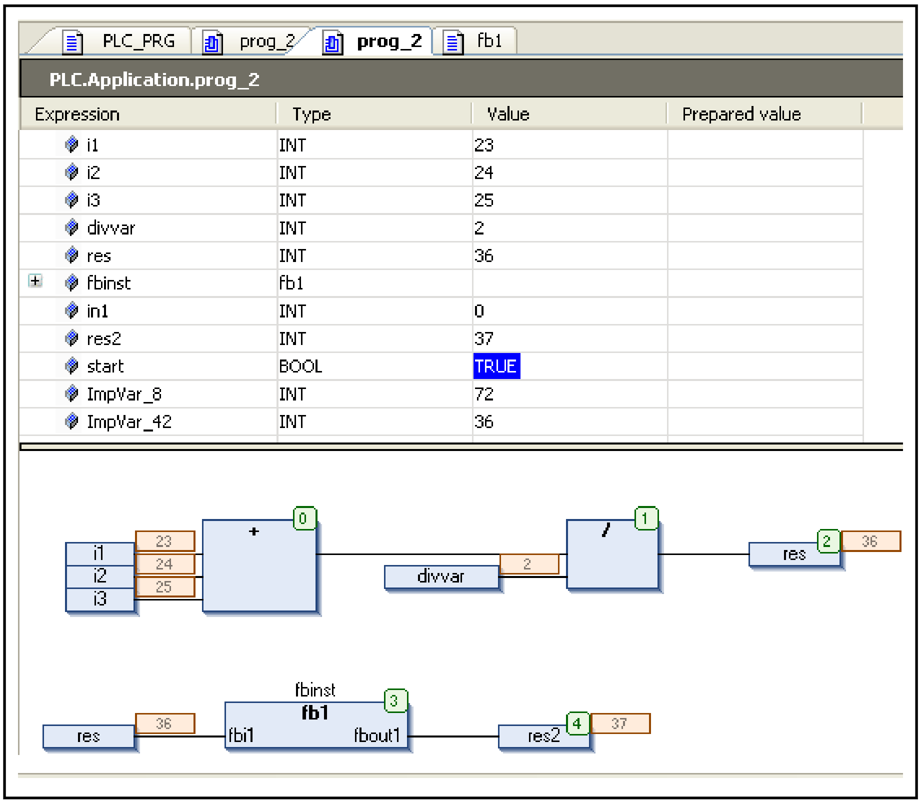
4.1.6 CFC Editor in Online Mode

In online mode, the CFC editor provides views for monitoring and for writing, page144, and forcing, page 145, 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).



PLC.Application.prog_2

Expression	Type	Value	Prepared value
i1	INT	23	
i2	INT	24	
i3	INT	25	
divvar	INT	2	
res	INT	36	
fbinst	fb1		
in1	INT	0	
res2	INT	37	
start	BOOL	TRUE	
ImpVar_8	INT	72	
ImpVar_42	INT	36	

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:

Users can set a breakpoint, page82, 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:

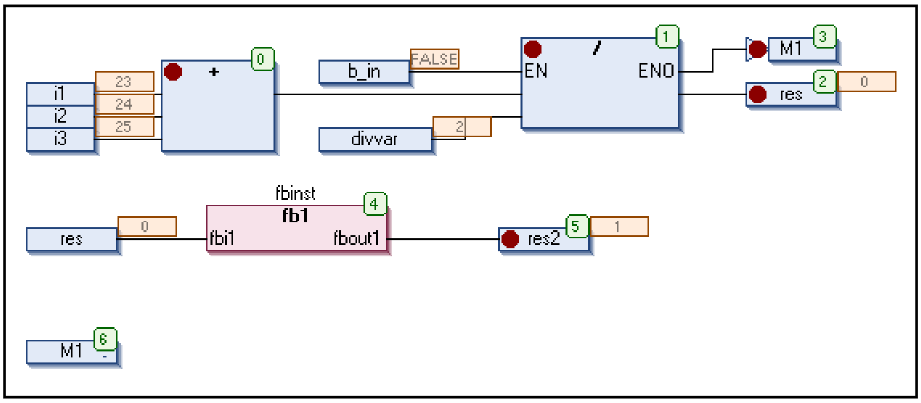


Fig.4-11: Breakpoint positions in the CFC editor

☞ A breakpoint is automatically set in all methods, page 45, 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

A "data server" can be added to an application, page66, 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 visualized 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** has to be inserted 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 inserted as objects below the data server.

The data source editor provides dialogs for configuring, page 318, 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

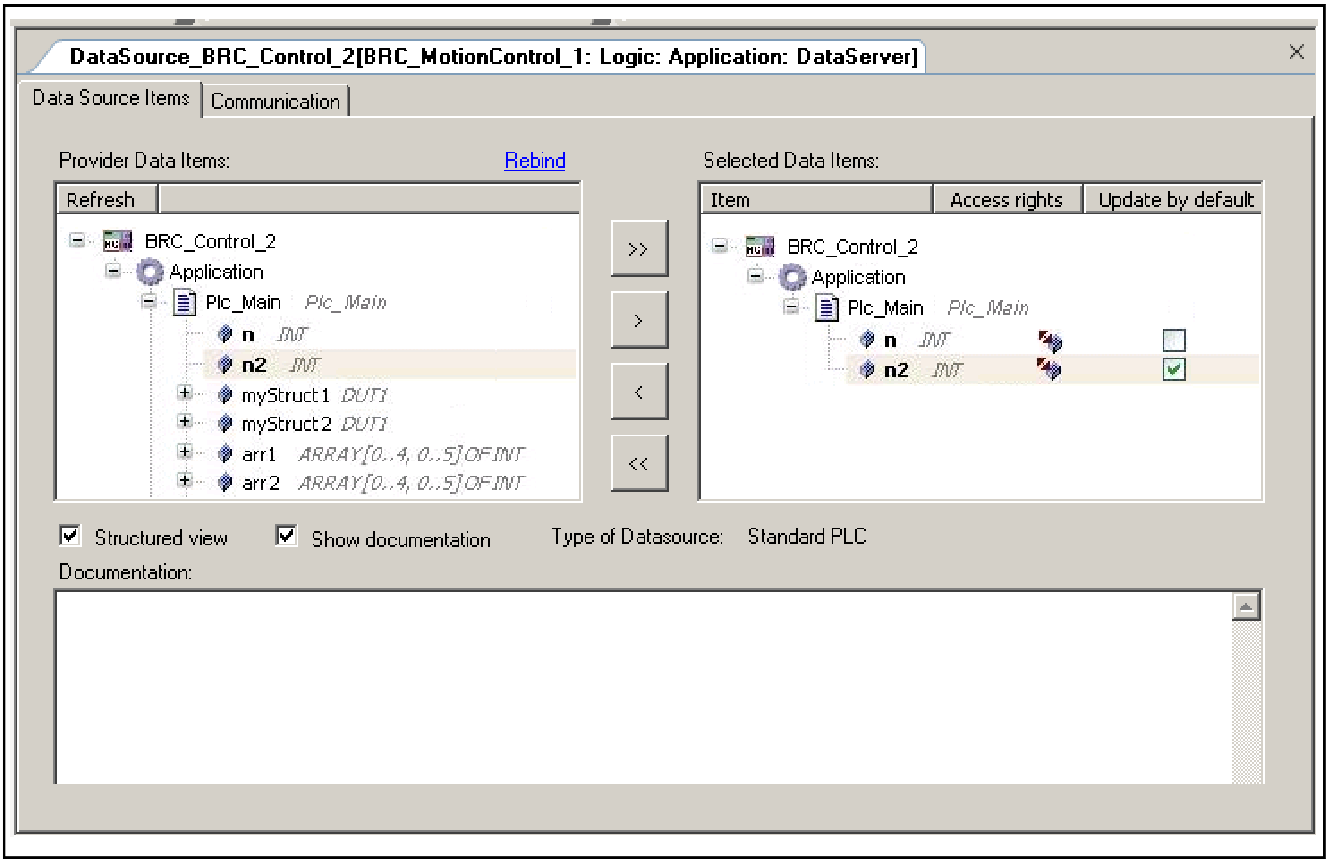
4.2.2 Configuring a Data Source

The data source objects assigned to a data server 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



DataSource_BRC_Control_2[BRC_MotionControl_1: Logic: Application: DataServer]

Data Source Items | Communication

Provider Data Items:

- BRC_Control_2
  - Application
    - Plc_Main *Plc_Main*
      - n *INT*
      - n2 *INT*
      - myStruct1 *DUT1*
      - myStruct2 *DUT1*
      - arr1 *ARRAY[0..4, 0..5] OF INT*
      - arr2 *ARRAY[0..4, 0..5] OF INT*

Selected Data Items:

Item	Access rights	Update by default
BRC_Control_2		
Application		
Plc_Main *Plc_Main*		
n *INT*		□
n2 *INT*		☑

☑ Structured view ☑ Show documentation Type of Datasource: Standard PLC

Documentation:

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.

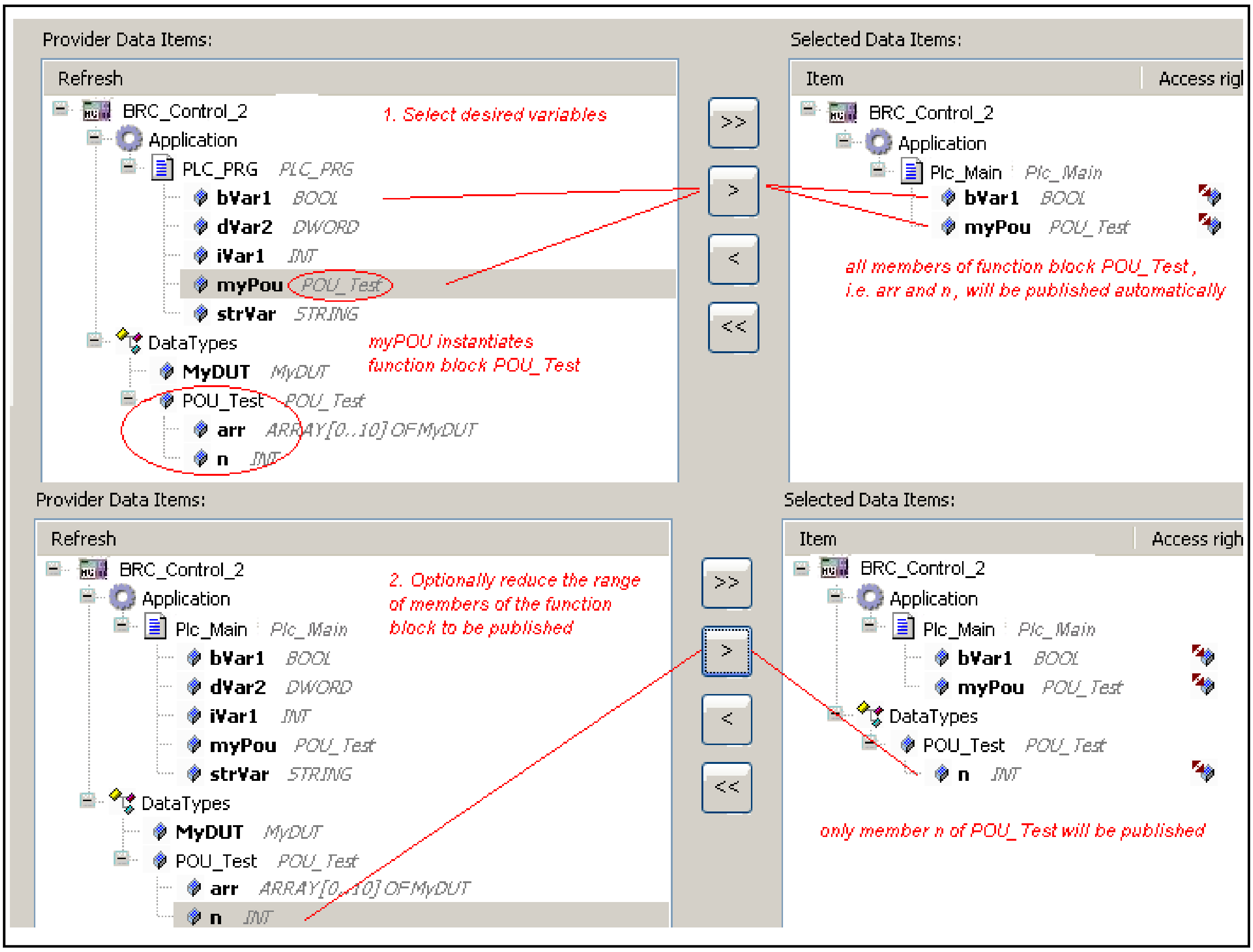
To select the variables that the local application is supposed to access:

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 move them into the "Select 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



1. Select desired variables

myPOU instantiates function block POU_Test

all members of function block POU_Test , i.e. arr and n , will be published automatically

2. Optionally reduce the range of members of the function block to be published

only member n of POU_Test will be published

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.

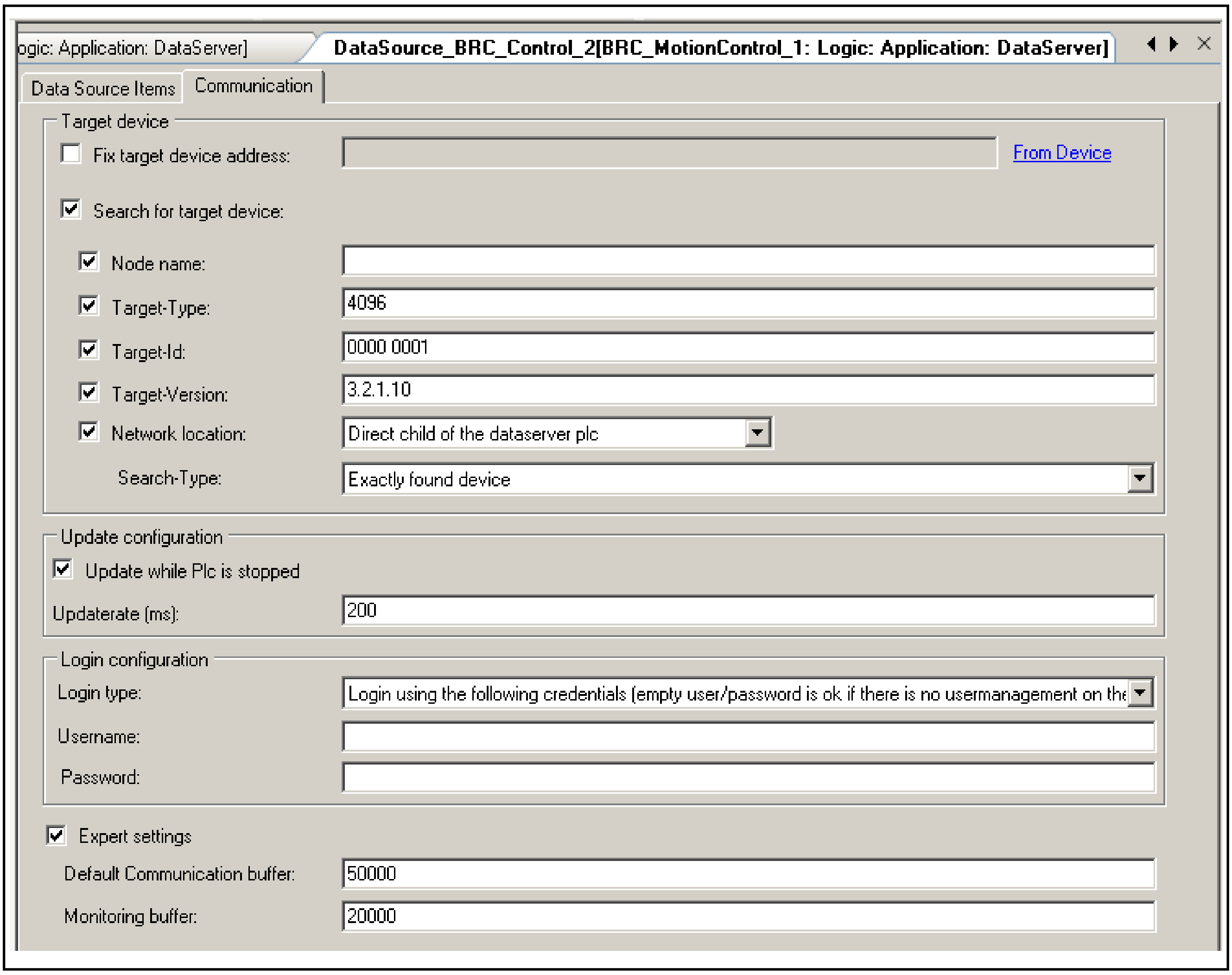
Updating by default

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, is 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



ogic: Application: DataServer] | DataSource_BRC_Control_2[BRC_MotionControl_1: Logic: Application: DataServer]

Data Source Items | Communication

Target device

☐ Fix target device address: From Device

☑ Search for target device:

☑ Node name:

☑ Target-Type: 4096

☑ Target-Id: 0000 0001

☑ Target-Version: 3.2.1.10

☑ Network location: Direct child of the dataserver plc

Search-Type: Exactly found device

Update configuration

☑ Update while Plc is stopped

Updaterate (ms): 200

Login configuration

Login type: Login using the following credentials (empty user/password is ok if there is no usermanagement on the

Username:

Password:

☑ Expert settings

Default Communication buffer: 50000

Monitoring buffer: 20000

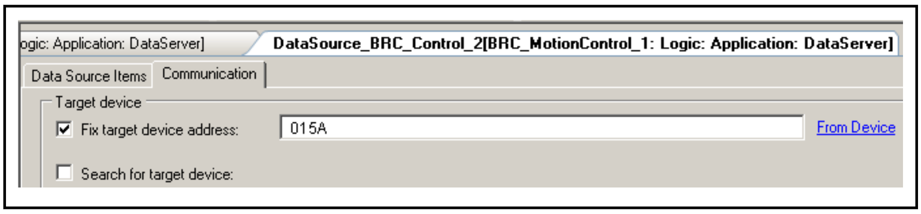
Fig.4-14: "DataSource" dialog, communication parameters

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.

- Click on "**From Device**" to automatically enter the data from the respective, currently connected data source device as configured in the communication settings, page 67.

- **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".


ogic: Application: DataServer]
DataSource_BRC_Control_2[BRC_MotionControl_1: Logic: Application: DataServer]
Data Source Items
Communication
Target device
From Device

☑ Fix target device address: 015A

□ Search for target 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.

- 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.

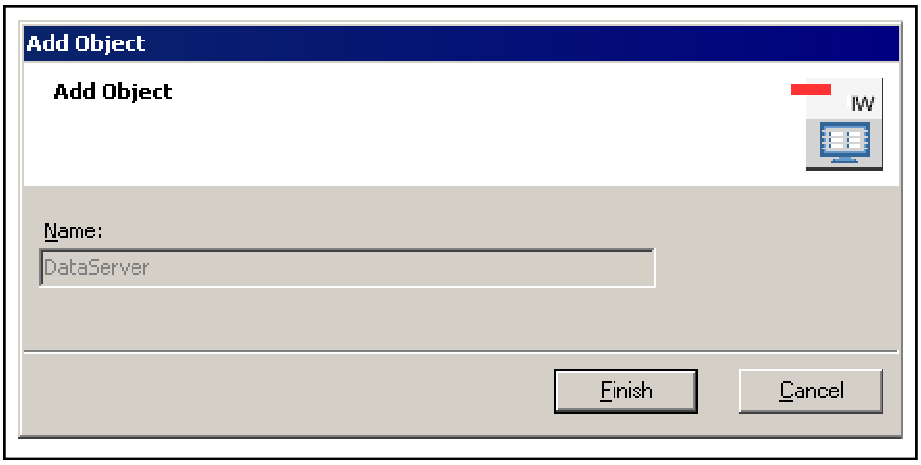
  Default: 50000.
- **Size of the monitoring buffer:** Buffer size for the monitoring data in bytes.

  Default: 20000.

4.2.3 Adding a Data Server

A Data Server object can be added to an application with the Add, page 234, 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.


Add Object

Add Object

Name:

DataServer

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

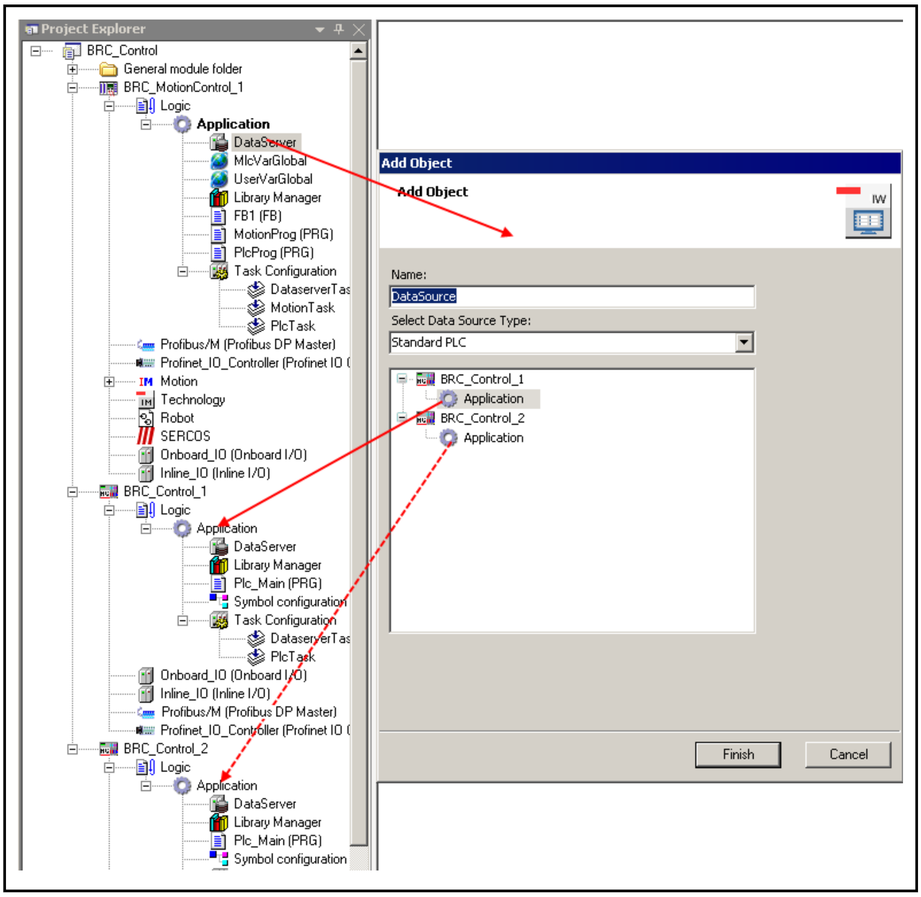
The necessary data server libraries in the global project library manager are automatically added with the data server object and a **data server task** is created in the task configuration of the application.

☞ The task priority has to be specified.

The command Add (Data Source) can be used to add one or multiple **data source** objects to the application.

4.2.4 Adding a Data Source

A data source, page 318, can be assigned to the data server, page 318, of an application using the Add, page 234, command.



Name: DataSource

Select Data Source Type: Standard PLC

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

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 (Fig. "Add Object" dialog, adding a data server, page 323).

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

Select the data sources that correspond to this data server (Fig. "Add Object" dialog, adding a data source, page 324).

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

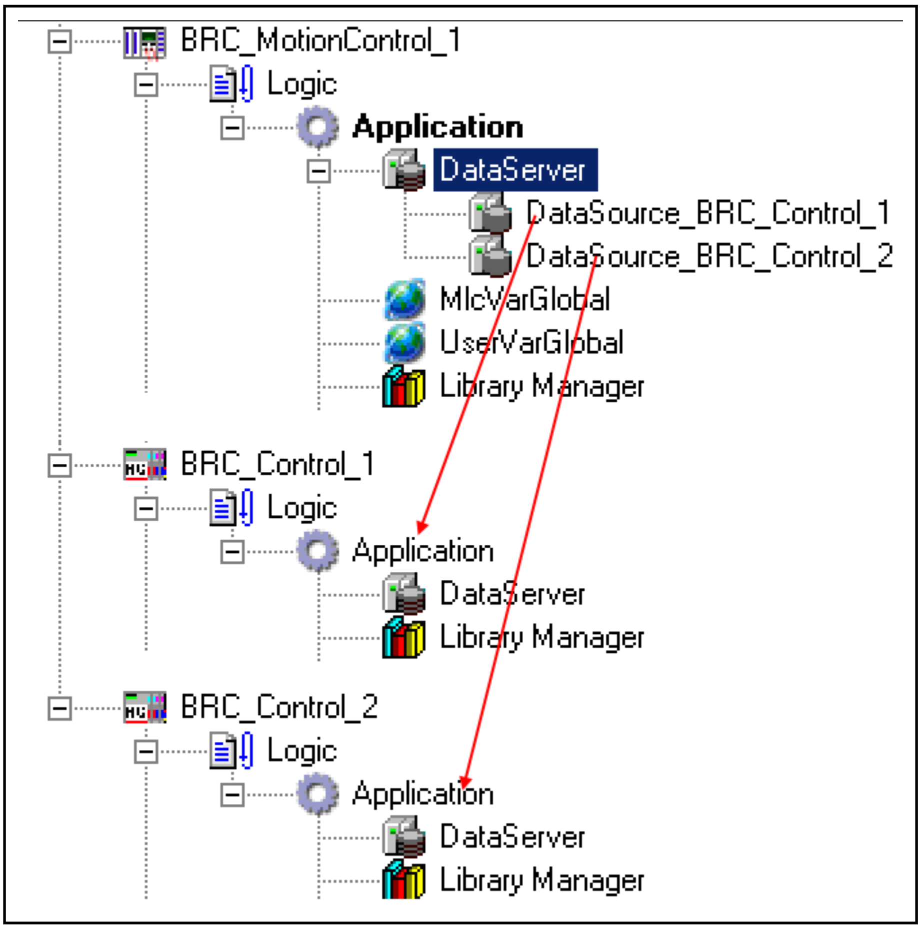


Fig.4-18: Example of data server and data sources

4.2.5 Using Data Sources in Visualizations

All visualizations assigned to an application can use all data sources, page 318, that are also assigned to the application.

The variables of the data source configured 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:

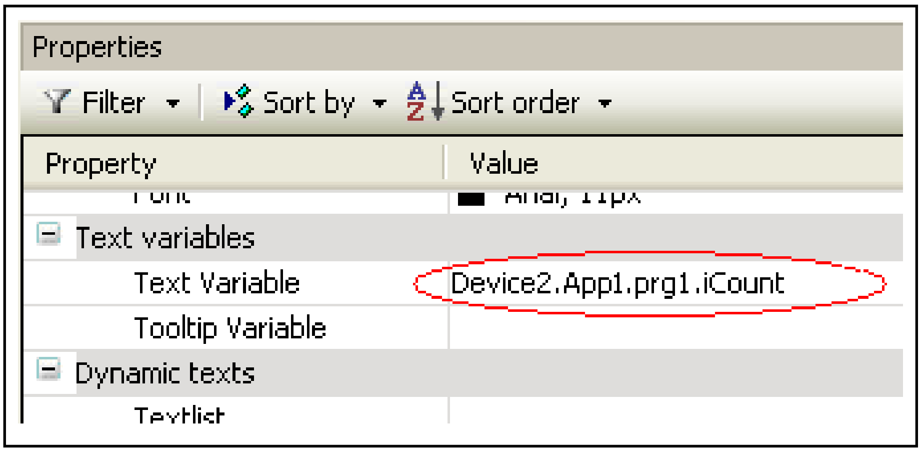
`<Device name for the data source>.<Function block name>.<Variable name>`.

☞ 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".



Property	Value
Text variables	
Text Variable	Device2.App1.prg1.iCount
Tooltip Variable	
Dynamic texts	
Textlist	

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, is 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.

Depending on the settings in the declaration editor options, page 201, 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.

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 online mode, page 330,.

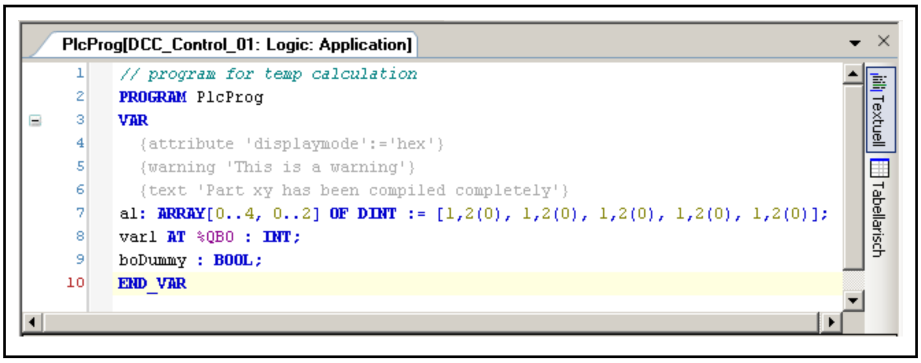
The declaration header describes the POU type (e.g. PROGRAM, FUNCTION_ BLOCK, FUNCTION ...) and can be extended by POU global pragma attributes.

In online mode, the declaration editor is structured as a monitoring view, page 499

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 variable declaration, page 503.

Text declaration editor



PlcProg[DCC_Control_01: Logic: Application]

```
// program for temp calculation
PROGRAM PlcProg
VAR
  {attribute 'displaymode':='hex'}
  {warning 'This is a warning'}
  {text 'Part xy has been compiled completely'}
a1: ARRAY[0..4, 0..2] OF DINT := [1,2(0), 1,2(0), 1,2(0), 1,2(0), 1,2(0)];
var1 AT %QB0 : INT;
boDummy : BOOL;
END_VAR
```

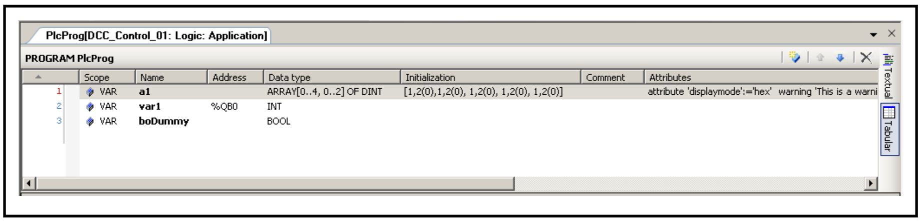
Fig.4-20: Text editor view

Behavior and appearance of the textual editor are determined by the current text editor settings in the declaration editor option, page 201 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 IntelliMouse, page 510 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



PlcProg[DCC_Control_01: Logic: Application]

PROGRAM PlcProg

	Scope	Name	Address	Data type	Initialization	Comment	Attributes
1	VAR	**a1**		ARRAY[0..4, 0..2] OF DINT	[1,2(0),1,2(0), 1,2(0), 1,2(0), 1,2(0)]		attribute 'displaymode':='hex' warning 'This is a warni
2	VAR	**var1**	%QB0	INT			
3	VAR	**boDummy**		BOOL			

Fig.4-21: Tabular editor view

The tabular version of the editor displays columns for common definitions required for a variable declaration, page 503,: "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.

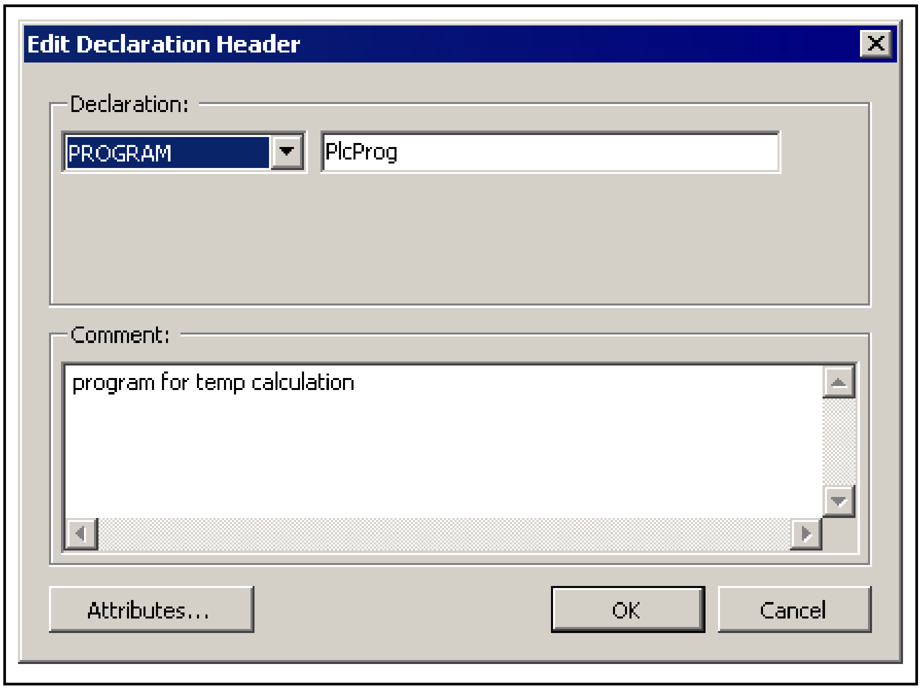


Fig.4-22: "Edit Declaration Header" dialog

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.

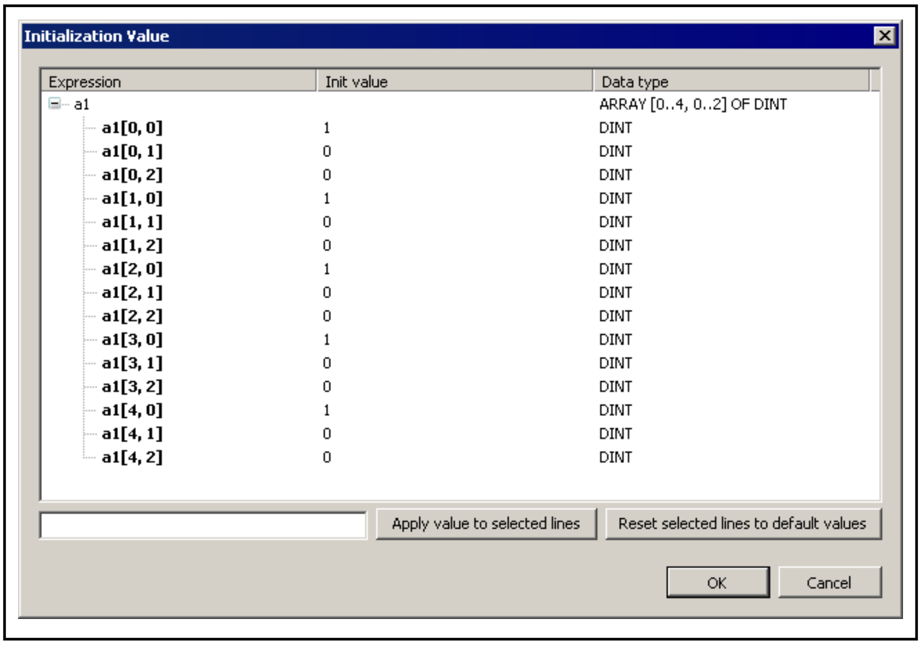
To add a new declaration at the very bottom of the table, click on the last existing line and use the "Insert" command as well.

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 input assistance, page 98, or the array assistant, page 100.

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


Initialization Value
Apply value to selected lines
Reset selected lines to default values
OK
Cancel

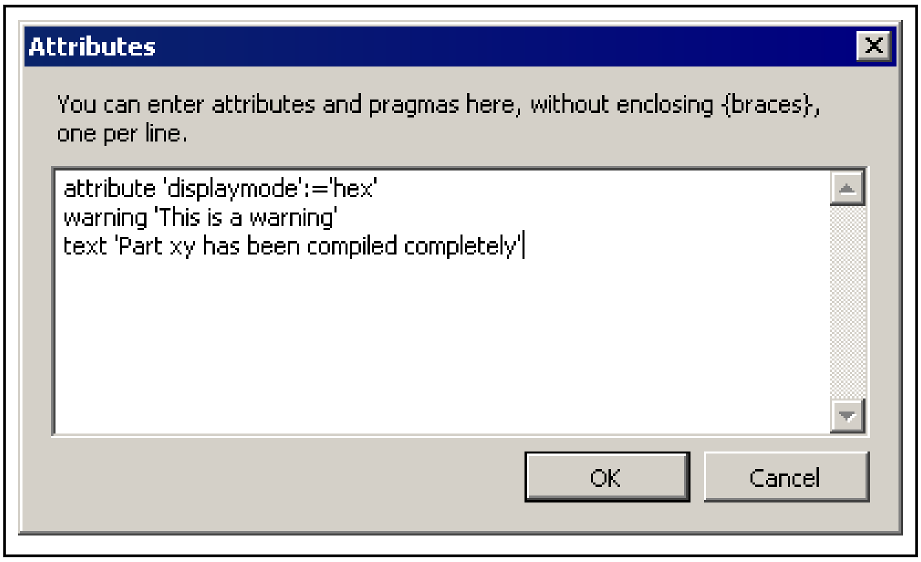
Expression	Init value	Data type
a1		ARRAY [0..4, 0..2] OF DINT
a1[0, 0]	1	DINT
a1[0, 1]	0	DINT
a1[0, 2]	0	DINT
a1[1, 0]	1	DINT
a1[1, 1]	0	DINT
a1[1, 2]	0	DINT
a1[2, 0]	1	DINT
a1[2, 1]	0	DINT
a1[2, 2]	0	DINT
a1[3, 0]	1	DINT
a1[3, 1]	0	DINT
a1[3, 2]	0	DINT
a1[4, 0]	1	DINT
a1[4, 1]	0	DINT
a1[4, 2]	0	DINT

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".



You can enter attributes and pragmas here, without enclosing {braces}, one per line.

```
attribute 'displaymode':='hex'
warning 'This is a warning'
text 'Part xy has been compiled completely'
```

Fig.4-24: "Attributes" dialog

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

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 <Del> 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 monitoring windows, page 500.

The header always contains the current object path:

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

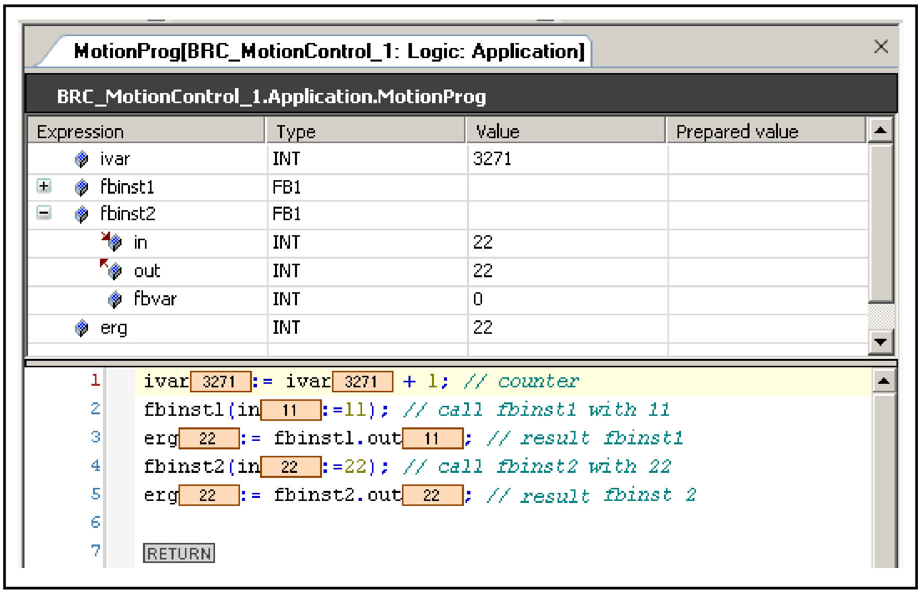
The table displays the data type and the current value for each expression (watch expression), and - if it is currently set - the prepared value for forcing, page 145, or writing, page 144,.

To prepare a value for a variable, use the Prepare value, page 146, 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.



MotionProg[BRC_MotionControl_1: Logic: Application]

BRC_MotionControl_1.Application.MotionProg

Expression	Type	Value	Prepared value
ivar	INT	3271	
⊞ fbinst1	FB1		
⊟ fbinst2	FB1		
in	INT	22	
out	INT	22	
fbvar	INT	0	
erg	INT	22	

```
ivar 3271 := ivar 3271 + 1; // counter
fbinst1(in 11 :=11); // call fbinst1 with 11
erg 22 := fbinst1.out 11 ; // result fbinst1
fbinst2(in 22 :=22); // call fbinst2 with 22
erg 22 := fbinst2.out 22 ; // result fbinst 2

RETURN
```

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 device, page 63, 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:

- Applications, page 331: List of the applications on the control.
- Log, page 332: Displays PLC log file.

  The tab can be switched off and on via **Options** ▸ **IndraLogic 2G** ▸ **General Settings** ▸ **Enable PLC Logger**.
- PLC settings, page 334
- Configuration, floating point number exceptions in the PLC program, page 335,
- Information, page 337: 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.

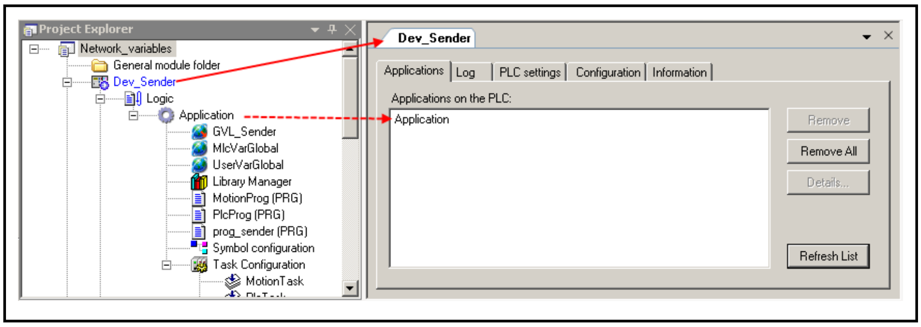


Fig.4-26: Device dialog, Applications

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.

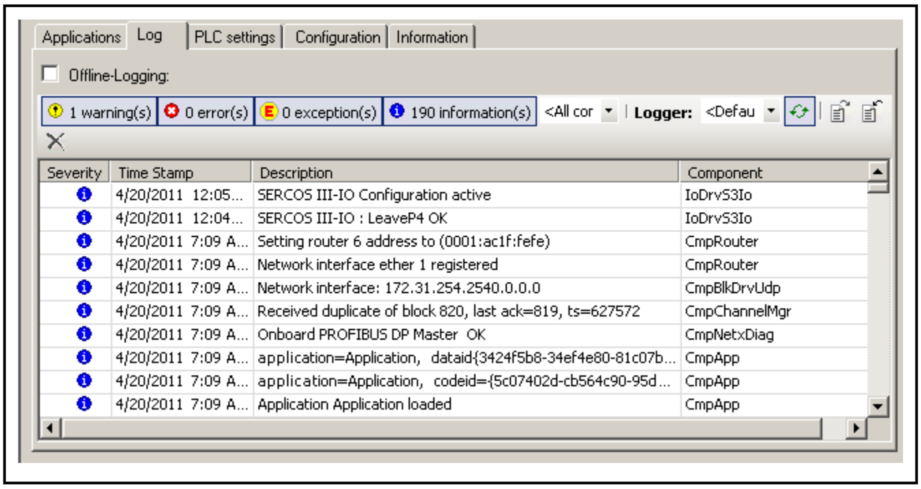
4.4.3 Log

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



Severity	Time Stamp	Description	Component
ℹ	4/20/2011 12:05...	SERCOS III-IO Configuration active	IoDrvS3Io
ℹ	4/20/2011 12:04...	SERCOS III-IO : LeaveP4 OK	IoDrvS3Io
ℹ	4/20/2011 7:09 A...	Setting router 6 address to (0001:ac1f:fefe)	CmpRouter
ℹ	4/20/2011 7:09 A...	Network interface ether 1 registered	CmpRouter
ℹ	4/20/2011 7:09 A...	Network interface: 172.31.254.2540.0.0.0	CmpBlkDrvUdp
ℹ	4/20/2011 7:09 A...	Received duplicate of block 820, last ack=819, ts=627572	CmpChannelMgr
ℹ	4/20/2011 7:09 A...	Onboard PROFIBUS DP Master OK	CmpNetxDiag
ℹ	4/20/2011 7:09 A...	application=Application, dataid{3424f5b8-34ef4e80-81c07b...	CmpApp
ℹ	4/20/2011 7:09 A...	application=Application, codeid={5c07402d-cb564c90-95d...	CmpApp
ℹ	4/20/2011 7:09 A...	Application Application loaded	CmpApp

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.

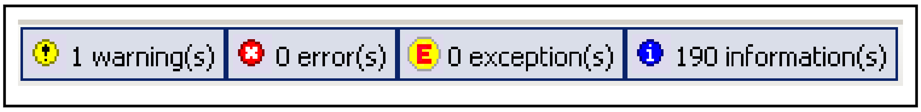


Fig.4-28: Severity of event

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

Description: Description of the event, e.g. Import function failed of <CmpFileTransfer>

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.

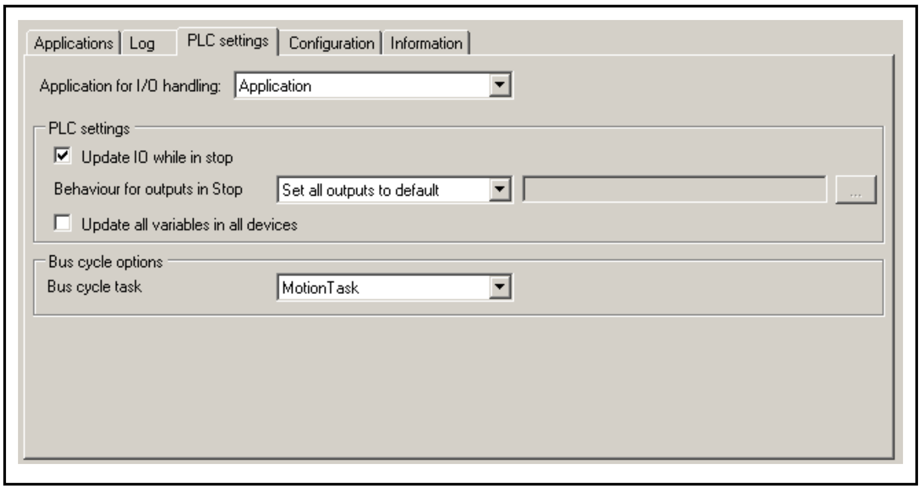
Currently, the list has to be refreshed using the button.

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**!

4.4.4 PLC Settings

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



Application for I/O handling: Application

PLC settings

☑ Update IO while in stop

Behaviour for outputs in Stop: Set all outputs to default

□ Update all variables in all devices

Bus cycle options

Bus cycle task: MotionTask

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

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

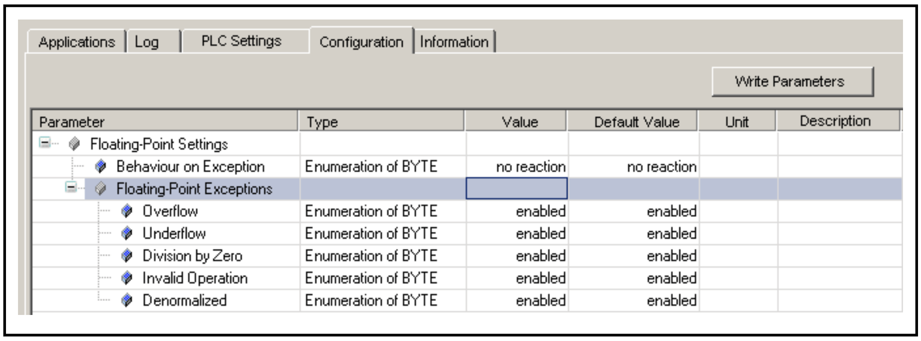
In the IEEE standard for floating point calculations, different types of floating point exceptions are defined. These are supported by floating point processors according to IEEE standard.

- 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.



Parameter	Type	Value	Default Value	Unit	Description
Floating-Point Settings					
Behaviour on Exception	Enumeration of BYTE	no reaction	no reaction		
Floating-Point Exceptions					
Overflow	Enumeration of BYTE	enabled	enabled		
Underflow	Enumeration of BYTE	enabled	enabled		
Division by Zero	Enumeration of BYTE	enabled	enabled		
Invalid Operation	Enumeration of BYTE	enabled	enabled		
Denormalized	Enumeration of BYTE	enabled	enabled		

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

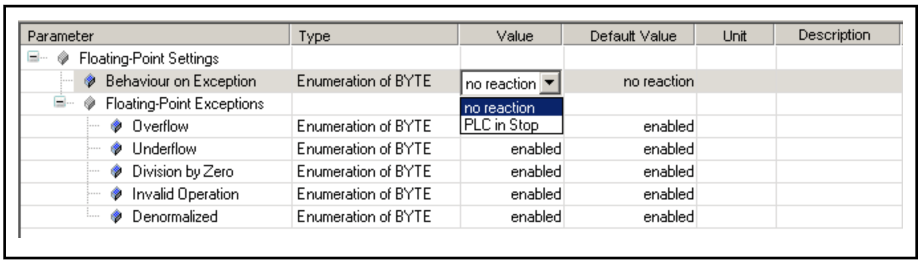
No reaction: All exception conditions occurring at floating point calculations are ignored.

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.

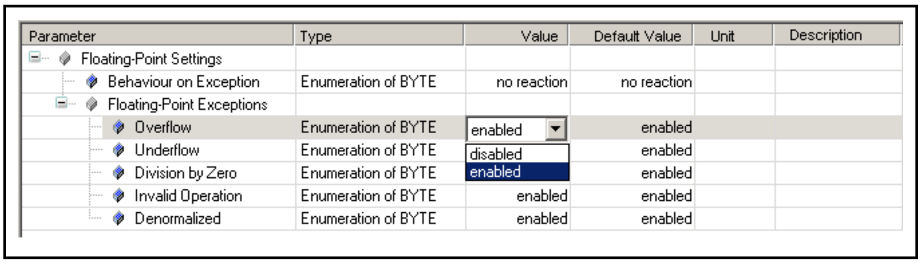


Parameter	Type	Value	Default Value	Unit	Description
Floating-Point Settings					
Behaviour on Exception	Enumeration of BYTE	no reaction	no reaction		
Floating-Point Exceptions					
Overflow	Enumeration of BYTE		enabled		
Underflow	Enumeration of BYTE	enabled	enabled		
Division by Zero	Enumeration of BYTE	enabled	enabled		
Invalid Operation	Enumeration of BYTE	enabled	enabled		
Denormalized	Enumeration of BYTE	enabled	enabled		

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:



Parameter	Type	Value	Default Value	Unit	Description
Floating-Point Settings					
Behaviour on Exception	Enumeration of BYTE	no reaction	no reaction		
Floating-Point Exceptions					
Overflow	Enumeration of BYTE	enabled	enabled		
Underflow	Enumeration of BYTE		enabled		
Division by Zero	Enumeration of BYTE		enabled		
Invalid Operation	Enumeration of BYTE	enabled	enabled		
Denormalized	Enumeration of BYTE	enabled	enabled		

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

> The floating point exception "Inexact“ is not displayed. It is always disabled.

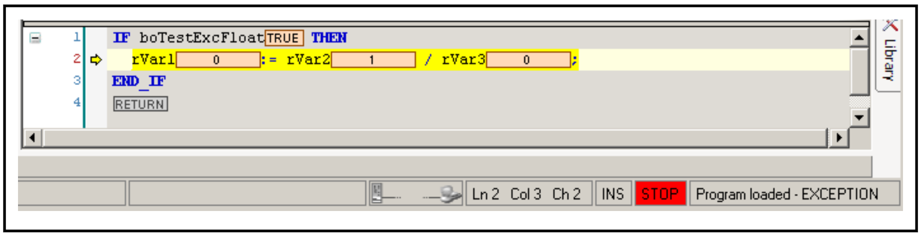
Default behavior

The reaction of the PLC when a floating point exception occurs is the same as for the previous exceptions.

- The PLC goes into the "STOP" state and EXCEPTION.
- The cause of the exception is entered in the logger.
- The program location causing the exception is displayed in the monitor.
- "STOP" and "EXCEPTION ERRORS" are displayed in the status of the programming system.

Example:

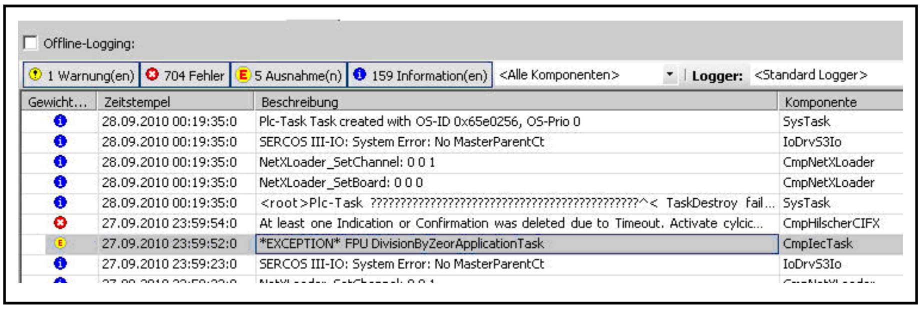
Floating point exception "Division by 0“, exception **enabled**



```
IF boTestExcFloat[TRUE] THEN
   rVar1[   0   ]:= rVar2[   1   ] / rVar3[   0   ];
END_IF
RETURN
```

Ln 2 Col 3 Ch 2 | INS | STOP | Program loaded - EXCEPTION

Fig.4-33: PLC in STOP, EXCEPTION ERROR, Display of the cause in the program code



☐ Offline-Logging:

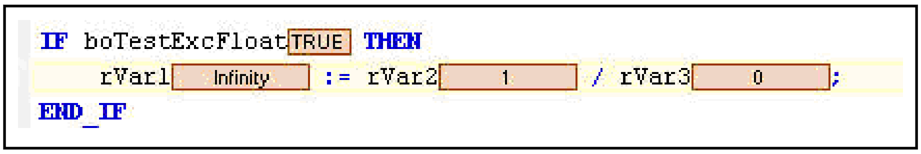
1 Warnung(en) | 704 Fehler | 5 Ausnahme(n) | 159 Information(en) | <Alle Komponenten> | Logger: <Standard Logger>

Gewicht...	Zeitstempel	Beschreibung	Komponente
ⓘ	28.09.2010 00:19:35:0	Plc-Task Task created with OS-ID 0x65e0256, OS-Prio 0	SysTask
ⓘ	28.09.2010 00:19:35:0	SERCOS III-IO: System Error: No MasterParentCt	IoDrvS3Io
ⓘ	28.09.2010 00:19:35:0	NetXLoader_SetChannel: 0 0 1	CmpNetXLoader
ⓘ	28.09.2010 00:19:35:0	NetXLoader_SetBoard: 0 0 0	CmpNetXLoader
ⓘ	28.09.2010 00:19:35:0	<root>Plc-Task ??^< TaskDestroy fail...	SysTask
⊗	27.09.2010 23:59:54:0	At least one Indication or Confirmation was deleted due to Timeout. Activate cylcic...	CmpHilscherCIFX
E	27.09.2010 23:59:52:0	*EXCEPTION* FPU DivisionByZeorApplicationTask	CmpIecTask
ⓘ	27.09.2010 23:59:23:0	SERCOS III-IO: System Error: No MasterParentCt	IoDrvS3Io

Fig.4-34: Logger message

Example:

Floating point exception "Division by 0“, exception **disabled**



```
IF boTestExcFloat[TRUE] THEN
   rVar1[ Infinity ] := rVar2[   1   ] / rVar3[   0   ];
END_IF
```

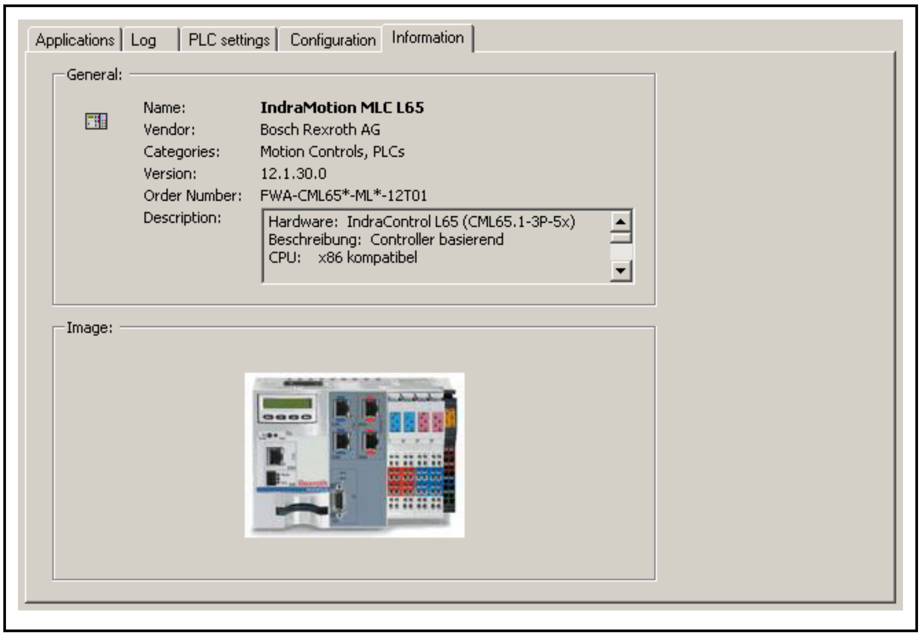
Fig.4-35: Program code, status display at running program

If no exceptions are enabled (as now), the PLC program continues running.

The result of the calculation is this case is the value "Infinity“.

4.4.6 Information

This tab in the **Device dialog** displays some general information taken from the device description file on the device currently selected in the Project Explorer: Name, Vendor, Categories, Version, Ordering number, Description, figure if available.



Name: **IndraMotion MLC L65**
Vendor: Bosch Rexroth AG
Categories: Motion Controls, PLCs
Version: 12.1.30.0
Order Number: FWA-CML65*-ML*-12T01
Description: Hardware: IndraControl L65 (CML65.1-3P-5x)
Beschreibung: Controller basierend
CPU: x86 kompatibel

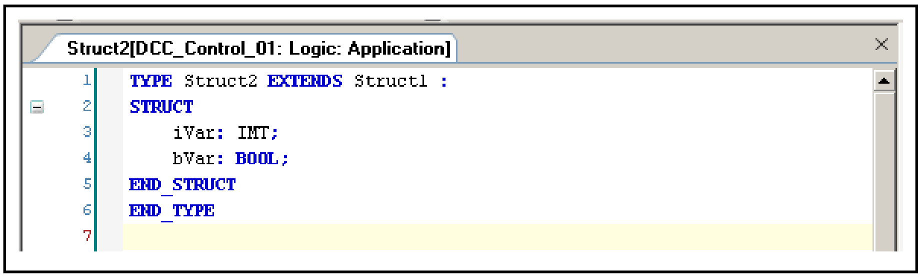
Fig.4-36: Device editor, Information

4.5 Data Type Editor

User-defined data types, page 552, (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 text editor options, page 212,.

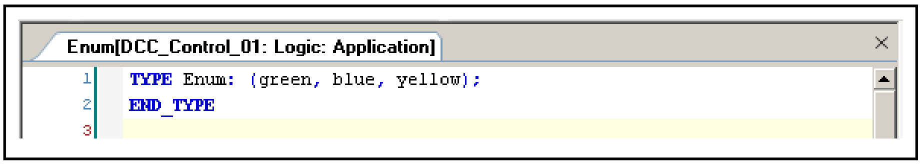
The data type editor window opens if an existing DUT object is opened for editing.

The editor already contains the predefinition of an extended structure definition, page 43 which can be modified into a simple structure definition or into a definition of another data type, e.g. an enumeration.



```
TYPE Struct2 EXTENDS Struct1 :
STRUCT
    iVar: INT;
    bVar: BOOL;
END_STRUCT
END_TYPE
```

Fig.4-37: DUT editor window with structure



```
TYPE Enum: (green, blue, yellow);
END_TYPE
```

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**.

A description of the initialization of arrays is located in arrays, page 560 or in the "Initialization value" dialog of the declaration editor, page 329.

A description of the initialization of structures is located under Structures, page 563.

A description of the initialization of enumerations is located under Enumerations, page 564 subrange types.

A description of variable initialization using the desired expressions is located under Expressions for variable initialization, page 510.

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 (Function block diagram, page 339), LD (Ladder diagram, page 340) and IL (Instruction list, page 340).

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 (View, page 266) 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 Options dialog, page 207,. 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 declaration editor, see page 510.

The **programming language** for a new object is specified when the object is created using the Add object dialog, see page 234,.

- Working in the FBD and LD editor, page 343.
- Working in the IL editor, page 343.

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.

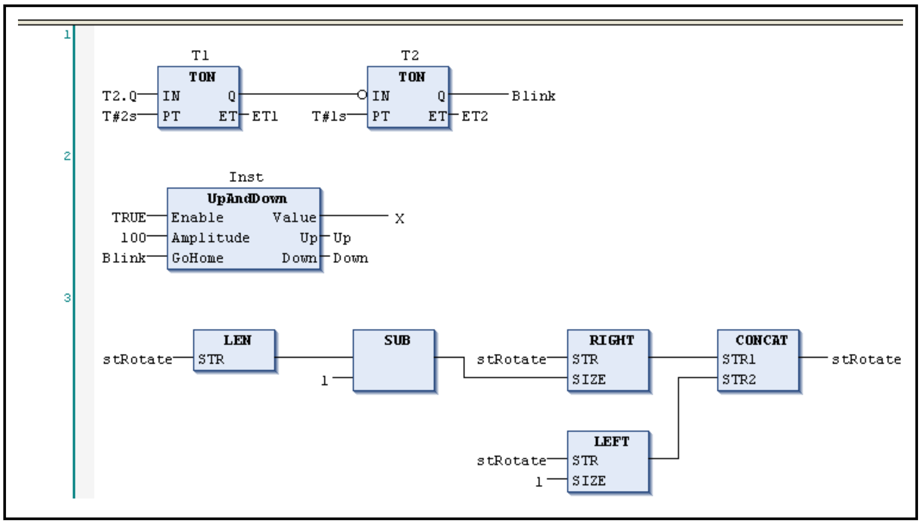


Fig.4-39: Example of a few FBD networks

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.

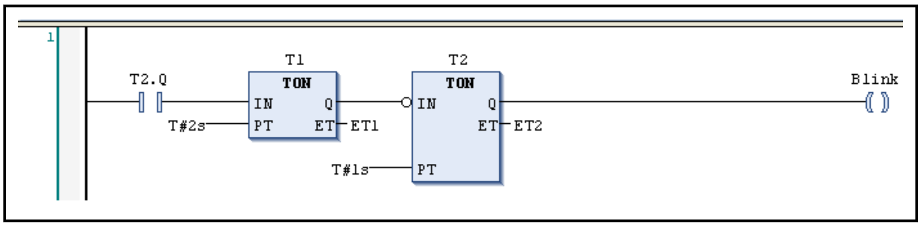


Fig.4-40: Example of an LD network

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.

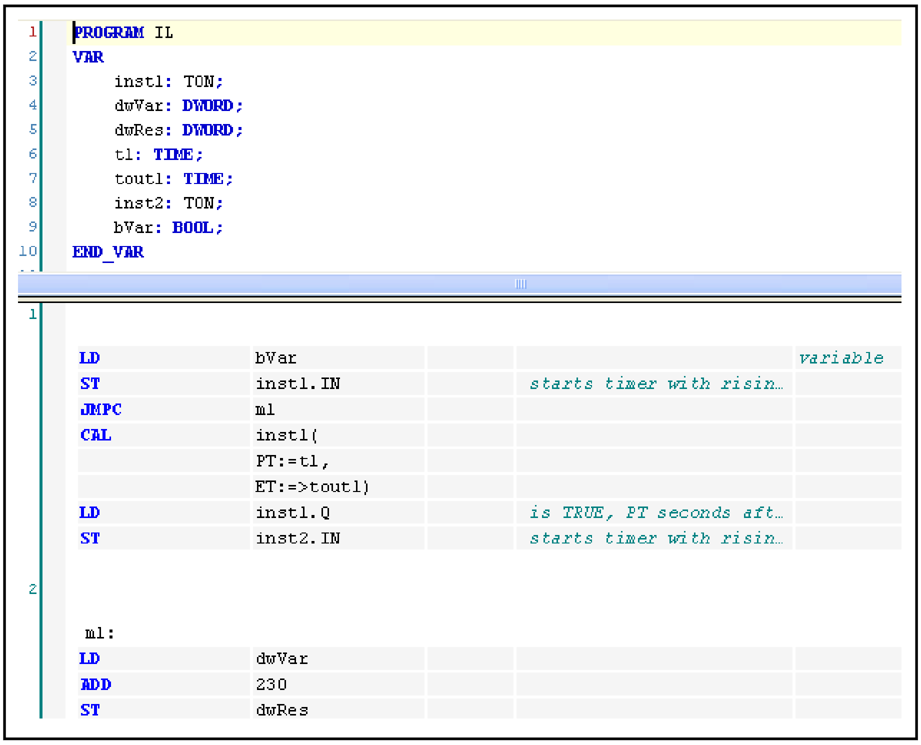
For programming conditional executions or loops, the instruction list supports relational operators (EQ, GT, LT, GE, LE, NE) and jumps. Jumps can be unconditional (JMP) or conditional (JMPC / JMPCN). For conditional jumps, a check is made to see if the value in the accumulator is TRUE or FALSE.

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.



```
PROGRAM IL
VAR
    inst1: TON;
    dwVar: DWORD;
    dwRes: DWORD;
    t1: TIME;
    tout1: TIME;
    inst2: TON;
    bVar: BOOL;
END_VAR
```

```
1
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…

2
 m1:
LD      dwVar
ADD     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 FBD/LD/IL editor.

- Working in the IL editor, page 346
- Modifiers and operators in IL, page 341

4.6.3 Modifiers and Operators in IL

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

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.

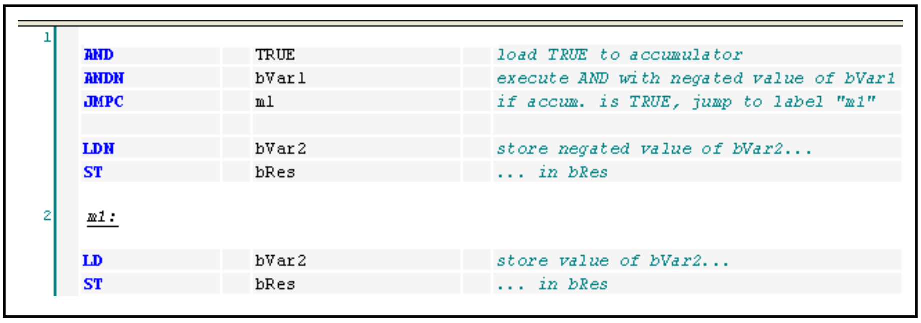
Operator	Modifier	Description	Example
LD	N	Loads the (negated) value of the operand to the accumulator	LD iVar
ST	N	Saves the (negated) contents of the accumulator to the operands	ST iErg
S		Sets the operands (type BOOL) to TRUE if the accumulator content is TRUE	S bVar1
R		Sets the operands (type BOOL) 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 IEC operators, page 569
- Information on using **multiple operands**, **complex operands**, function, method, function block, program, action **calls** and **jumps**; see Working in the IL editor, page 346

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
  JMPC     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 Options dialog, page 207,. 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 FBD/LD/IL menu, page 248, or by dragging and dropping the element from the toolbox, page 355, 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.

- 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 cursor positions in FBD, LD and IL, page 351,.
- Inserting EN/ENO function blocks is handled differently in the FBD editor and in the LD editor. For more information, refer to Add FB call, page 251

  (Adding EN/ENO function blocks is not supported in the IL editor.)

Navigating in the editor:

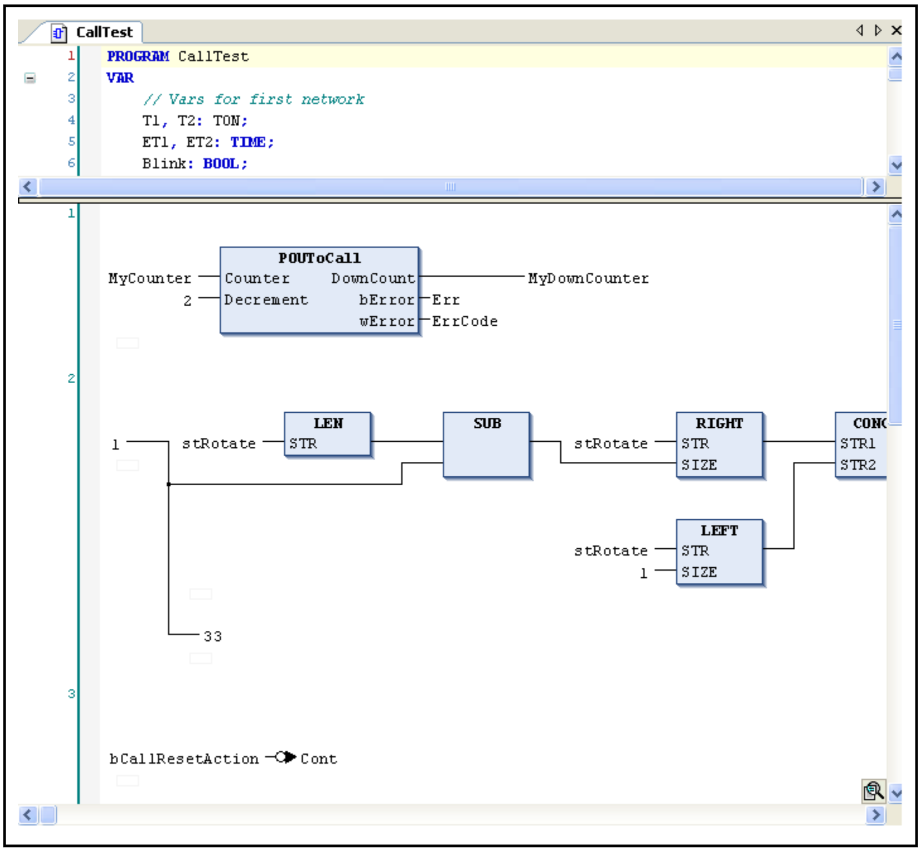
- The **arrow keys** can be used to jump between adjacent cursor positions within and across networks.
- The <Tab> key can be used to jump to the next cursor position within the network.
- <Ctrl>+<Pos1> focuses the start of the document and highlights the first network.
- <Ctrl>+<End> focuses the end 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 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.

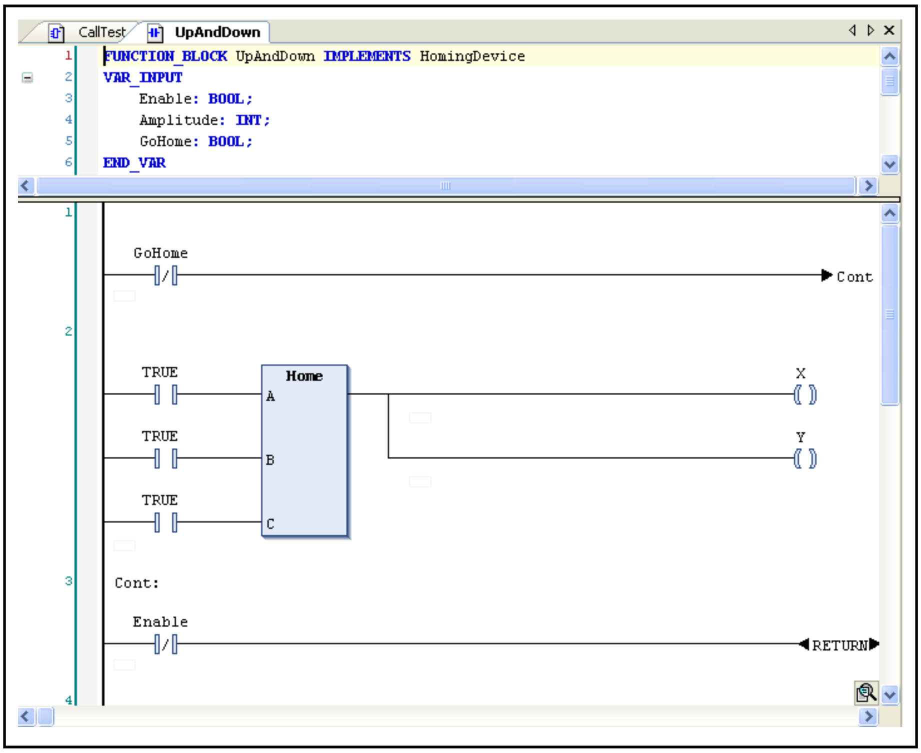
Opening a function block

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.



```
PROGRAM CallTest
VAR
    // Vars for first network
    T1, T2: TON;
    ET1, ET2: TIME;
    Blink: BOOL;
```

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



```
FUNCTION_BLOCK UpAndDown IMPLEMENTS HomingDevice
VAR_INPUT
    Enable: BOOL;
    Amplitude: INT;
    GoHome: BOOL;
END_VAR
```

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

For information on the programming language, see:

- Function block diagram - FBD, page 339

- Ladder diagram - LD, page 340

4.6.5 Working in the IL Editor

The IL (instruction list) 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 also be found in the IL program. However, in IL, one single network is 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 Options dialog, page 207,. 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 FBD/LD/IL menu, page 248, 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 Cursor positions in FBD, LD and IL, page 351,.
- Information on the IL programming language can be found in Instruction list - IL, page 340.

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

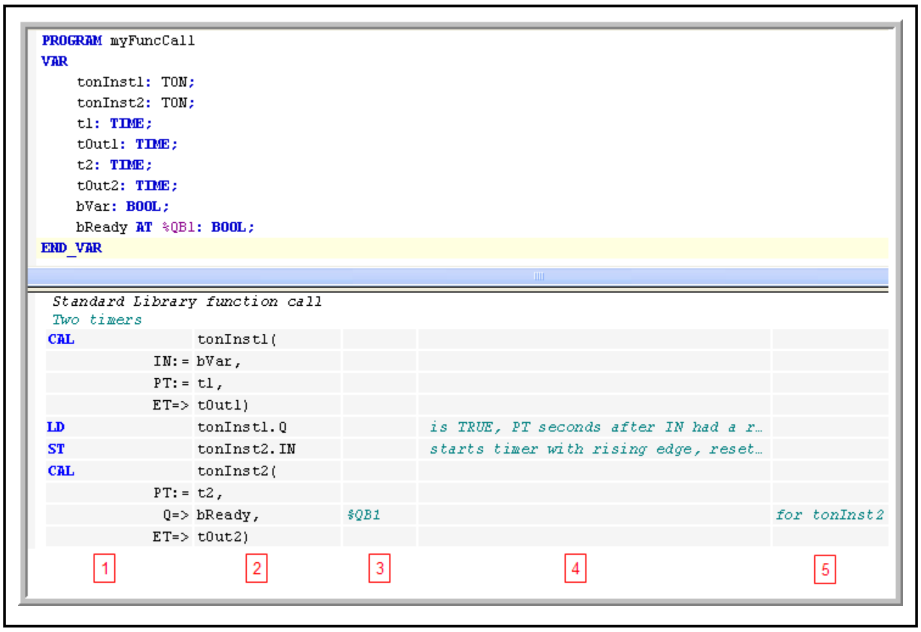
Structure of the IL table editor:

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. Modifiers and operators in IL, page 341
	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 is required (extendable operators "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	If there are multiple operands per operator or in case of function calls, this field contains the separating comma or the opening or closing parenthesis. If a function block, program or action 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 "Show symbol address" option, page 207,.
4	Symbol comment	This field contains the comment that might have been entered for the operand in it declaration. The field cannot be edited and can be shown or hidden in the view using the "Show symbol comment" option, page 207,.
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 "Operand comment" option, page 207,.

Fig.4-47: IL table editor



```
PROGRAM myFuncCall
VAR
    tonInst1: TON;
    tonInst2: TON;
    t1: TIME;
    tOut1: TIME;
    t2: TIME;
    tOut2: TIME;
    bVar: BOOL;
    bReady AT %QB1: BOOL;
END_VAR
```

```
 Standard Library function call
 Two timers
CAL              tonInst1(
           IN:=  bVar,
           PT:=  t1,
           ET=>  tOut1)
LD               tonInst1.Q                is TRUE, PT seconds after IN had a r…
ST               tonInst2.IN               starts timer with rising edge, reset…
CAL              tonInst2(
           PT:=  t2,
            Q=>  bReady,     %QB1                                                  for tonInst2
           ET=>  tOut2)
```

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

Fig.4-48: IL table editor

Navigating in the table:

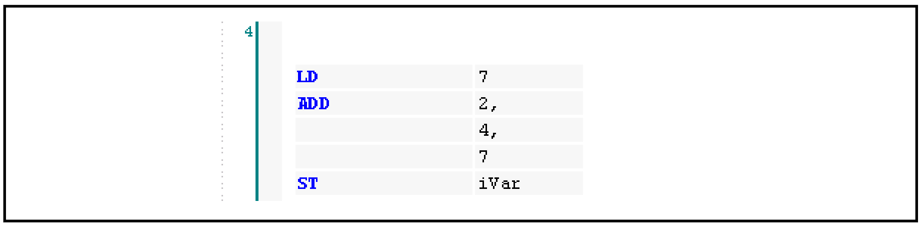
- <Up> and <Down> arrow keys: Jump to the field above or below
- <Tab>: Jump to the next field within the line (to the right)
- <Shift> <+> <Tab>: Jump to the previous field within the line (to the left)
- <Space bar>: Opens the editing frame for the currently selected field. Alternatively, click on 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.

- <Ctrl> <+> <Enter>: Inserts a new line below the current line
- <Del>: 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 operator is used with multiple operands, there are two programming options:

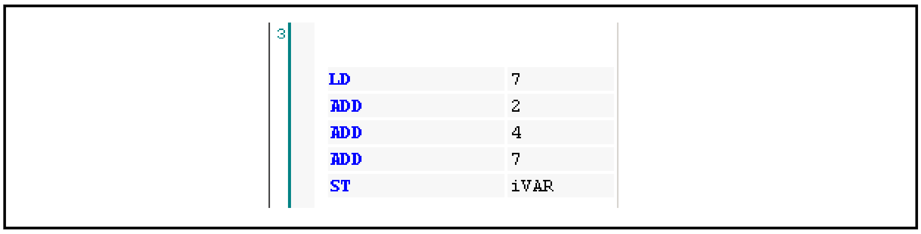
1. The operands are entered in successive lines separated by commas in the postfix field. Example:



```
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:

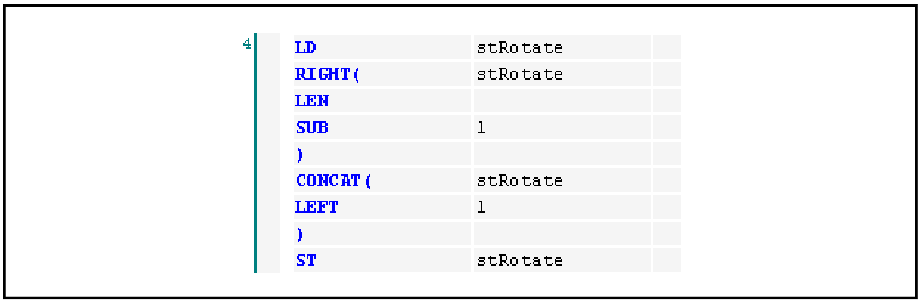
If a complex operand is used, an open parenthesis has to be entered in the prefix field, the further operand entries in the following lines and the closing parenthesis in a separate line, also in the prefix field.

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:



```
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.

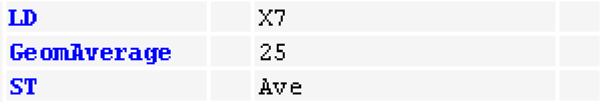
Example:

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:

Use the CAL or CALC operator 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

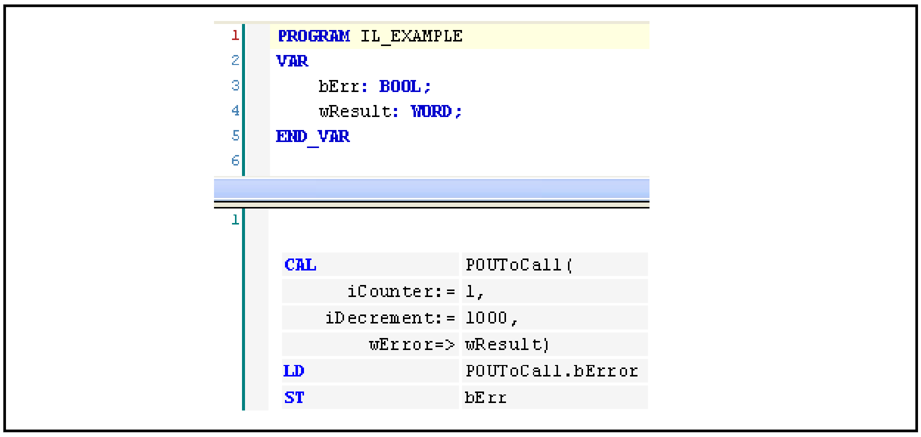
If there are calls without parameters, the postfix field has to contain "()".

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:



```
PROGRAM IL_EXAMPLE
VAR
    bErr: BOOL;
    wResult: WORD;
END_VAR

CAL              POUToCall(
       iCounter:= 1,
     iDecrement:= 1000,
         wError=> wResult)
LD               POUToCall.bError
ST               bErr
```

Fig.4-52: Example for a program call in IL with input/output parameters

Not all parameters of a function block or program have to be configured.

☞ Based on limitations of the IEC standard, complex expressions cannot be used. They have to be assigned to the function block or the program before the call.

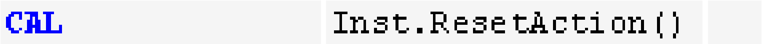
Action call: Execution as with a function block or program call. The action name has to be added to the instance name or program name.

Example: Call of the action "ResetAction"

Code in Structured Text:

```
Inst.ResetAction();
```

Code in IL:



```
CAL    Inst.ResetAction()
```

Method call: Execution as with a function call:

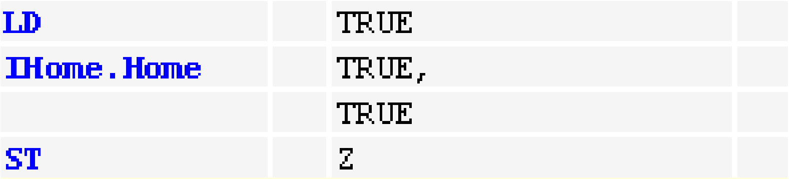
The instance name with the attached method name has to be entered in the operator field.

Example: Call of the "Home" method

Code in Structured Text:

```
Z := IHome.Home(TRUE, TRUE, TRUE);
```

Code in IL:

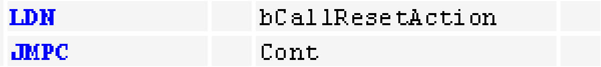


```
LD            TRUE
IHome.Home    TRUE,
              TRUE
ST            Z
```

Jump: A jump is programmed by using "JMP" in the operator field and a jump label name in the operand field. The jump label has to be defined in the target network in the Labels field. Note that the instruction sequence preceding the unconditional jump has to end with the following command: ST, STN, S, R, CAL, RET or another JMP. This does not apply for a conditional jump programmed 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: Working in the IL editor, page 346.

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

- Working in the FBD and LD editor, page 343.

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.

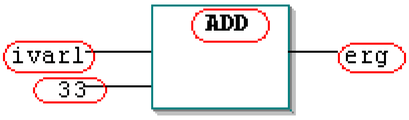
In LD, coil and contact symbols appear in red when the cursor is positioned there.

The cursor position determines which element is provided in the context menu for the Insertion, page 343,.

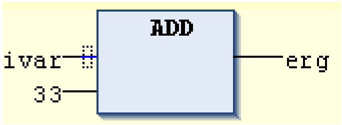
Possible cursor positions: (1) Each text field:

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.

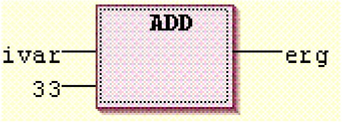
Note that instead of the variable name, the addresses can also be displayed if the corresponding option is enabled in the Options dialog, page 207,.



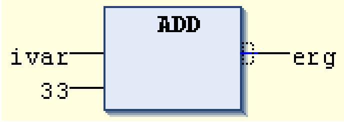
(2) Each input:



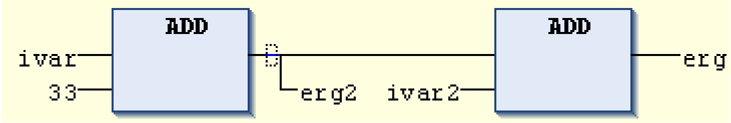
(3) Each operator, function or function block:



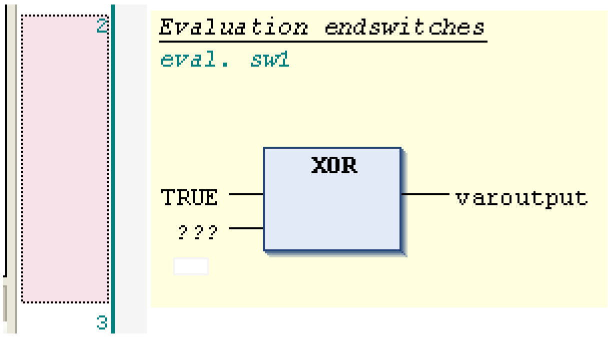
(4) Each output followed by an assignment or a jump:



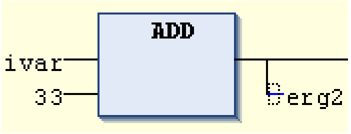
(5) The position in front of a branch to an assignment, a jump or a return instruction:



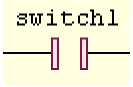
(6) The cursor position located farthest to the right or anywhere in the network where there are no other cursor positioned. This selects the entire network:



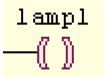
(7) The line cross directly in front of an assignment:



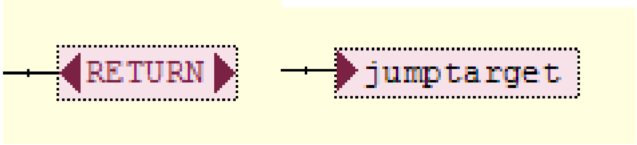
(8) Each contact



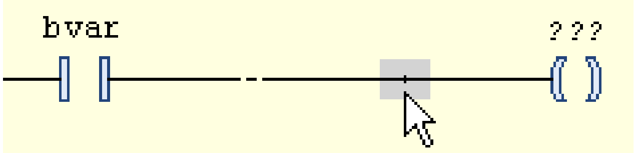
(9) Each coil



(10) Each return and jump:



(11) The connection line between contacts and coils.



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

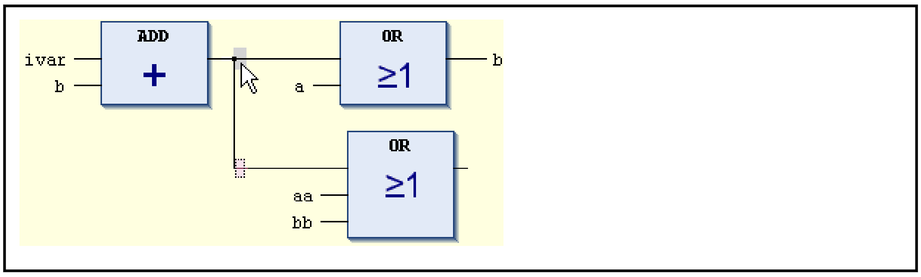


Fig.4-53: Line branches

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

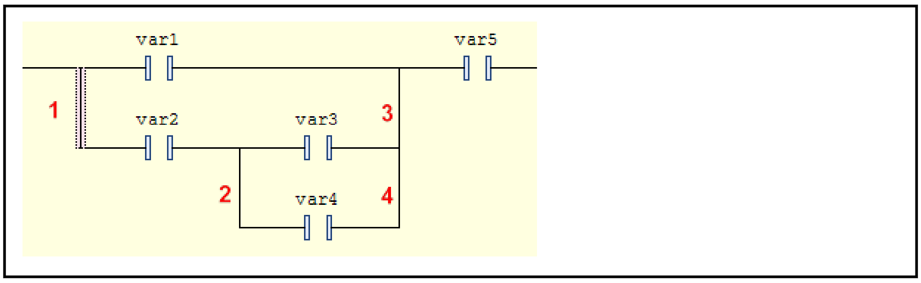


Fig.4-54: Connection line between parallel contacts

(14) In front and behind networks

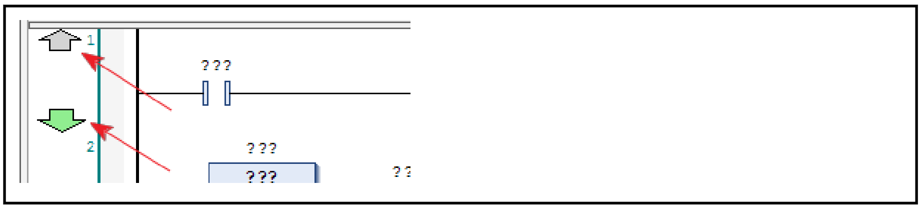


Fig.4-55: In front and behind networks

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

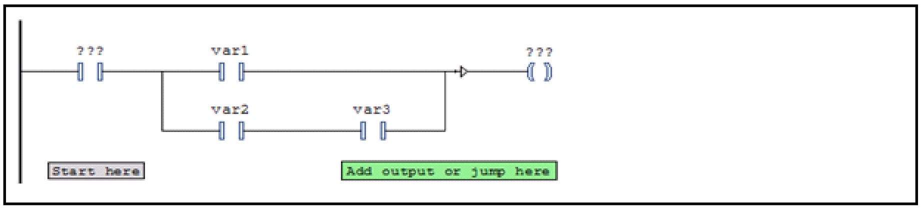


Fig.4-56: Beginning or end of a network

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

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

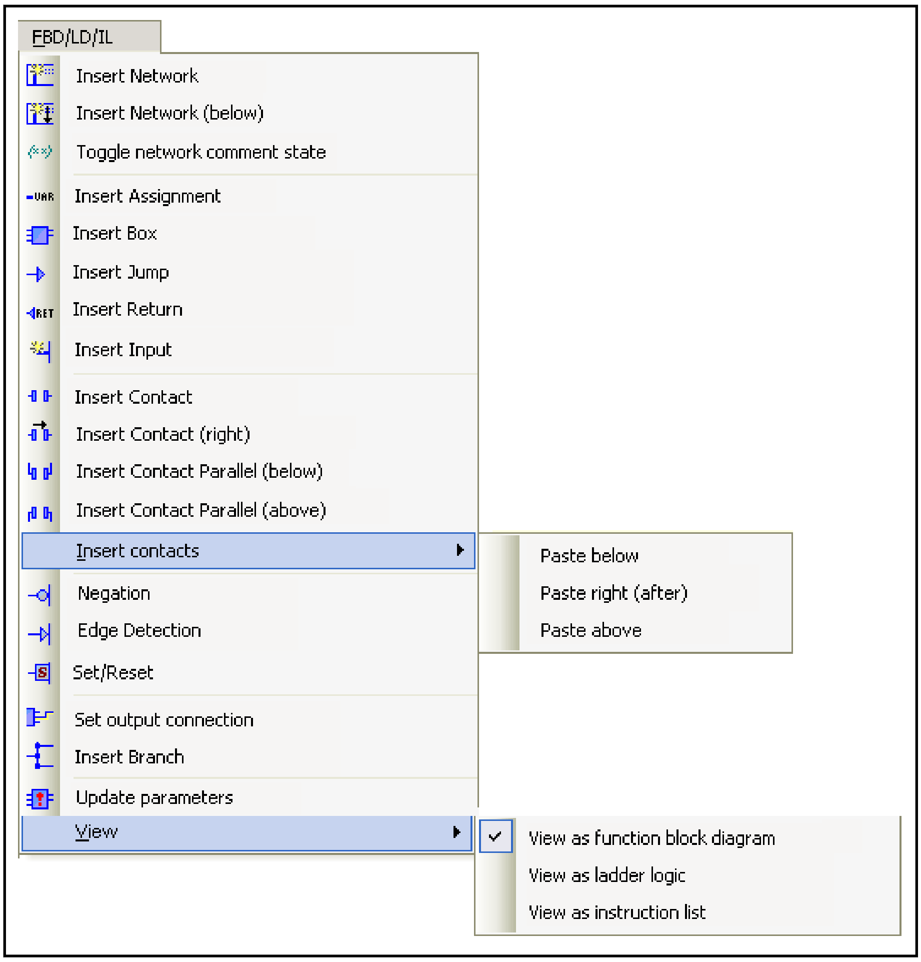


Fig.4-57: FBD/LD/IL menu, here subset of the ladder diagram

A description of the individual commands can be found under FBD/LD/IL commands, menu item, page 248.

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

4.6.8 FBD/LD/IL Elements

FBD/LD/IL tools

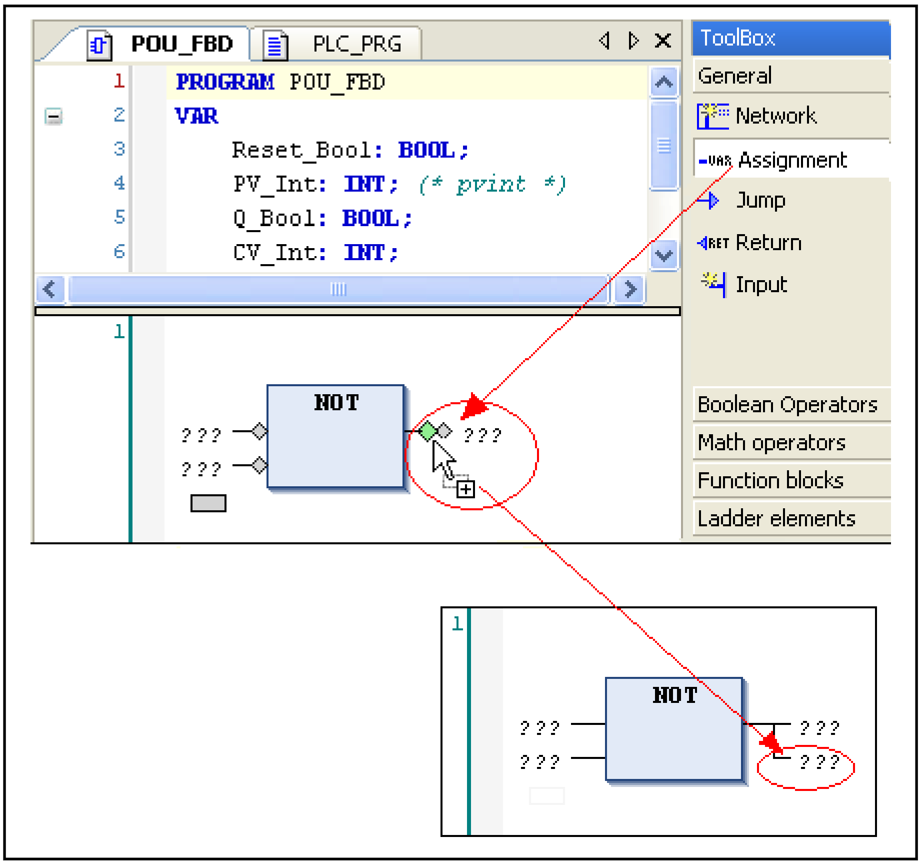
The FBD/LD/IL editor provides a toolbox in a separate view from which the individual programming elements can be inserted into the editor window via drag&drop. By default, the toolbox is opened automatically next to the editor window, but if necessary, it can also be opened explicitly using the Tools, page 106, 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 Insert commands, page 248).

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 POUs.

The "POUs" category provides all POUs 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 POUs 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:



```
PROGRAM POU_FBD
VAR
    Reset_Bool: BOOL;
    PV_Int: INT; (* pvint *)
    Q_Bool: BOOL;
    CV_Int: INT;
```

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 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

A network is the basic unit of an FBD or LD 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.

The IL 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 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 Options dialog, page 207,.

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, use the Insert jump label, page 256 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.

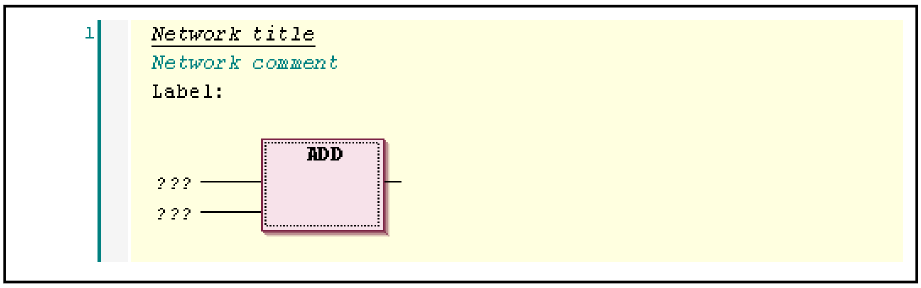


Fig.4-59: Position of title, comment and jump label in a network

A network can be commented out, i.e. the entire network can be redefined as a comment ('Commenting on/off, page 250') and therefore excluded from processing.

When a network is selected (cursor position 6) 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 Add network, page 250,.

Consider the option of "Subnetworks, page 264, 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 cursor position, page 351, 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 coil, page 361,.

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 IL, page 340,, an assignment is programmed using the "LD" and "ST" instructions.

See also the description of operators and modifiers, page 341.

Jump in FBD/LD/IL

In FBD or LD, a jump is inserted as a line depending on the current cursor position, page 351, 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 IL, page 340,, a jump is programmed using a "JMP" instruction.

See also the description of operators and modifiers, page 341.

Label in FBD/LD/IL

Each FBD or LD network has a text input field below the field for the network comment in which a label can be defined. This label is an optional identifier for the network and can be entered as an address for a jump. It can consist of any character string.

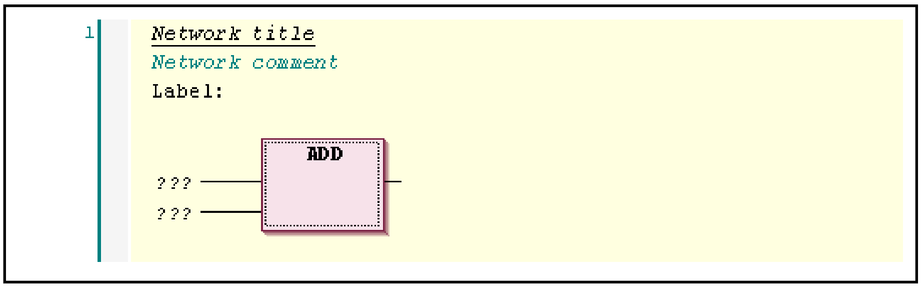


Fig.4-60: Label for network

Function Block Call in FBD/LD/IL

A function block with a call inserted in a FBD-, LD or IL 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 origin 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 toolbox, page 355,.

Usage in IL

In an IL program, a CAL 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 Refresh parameters, page 265; 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 parallely 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 Add Return, page 256.

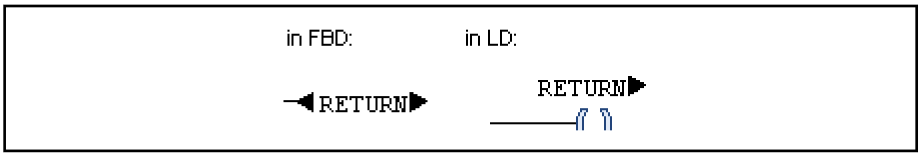


Fig.4-61: RETURN element

In IL, the RET instruction, page 341, 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.

Each subnetwork receives a label icon (rectangle) that can be selected (Cursor position 11, page 351) to execute actions on the subnetwork such as cutting and pasting.

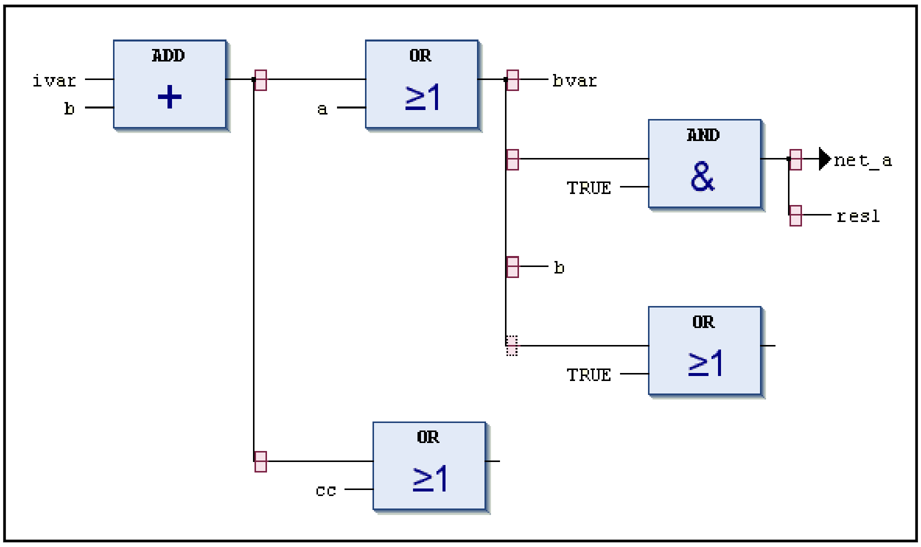


Fig.4-62: Examples for labeling networks within a network.

The "Insert branch" command is used to add a branch to the FBD. Alternatively, the element can also be obtained from the toolbox, page 355,. Possible insertion positions can be found under Inserting Branch, page 264.

☞ 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

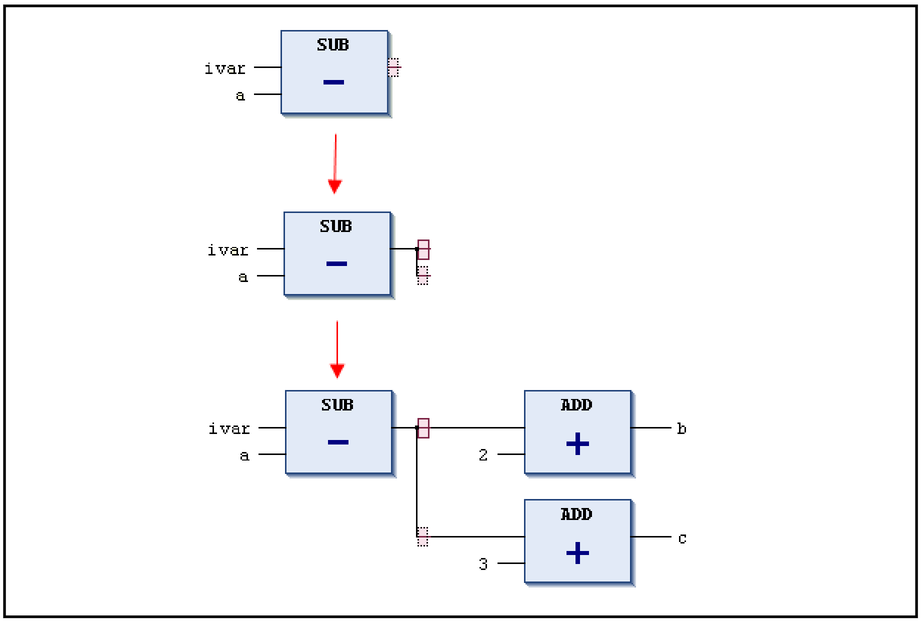


Fig.4-63: Example in FBD, Inserting a branch

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 <Del>. Refer to the following figure: The three-input OR element has to be deleted before the label of the lower subnetwork can be selected and deleted.

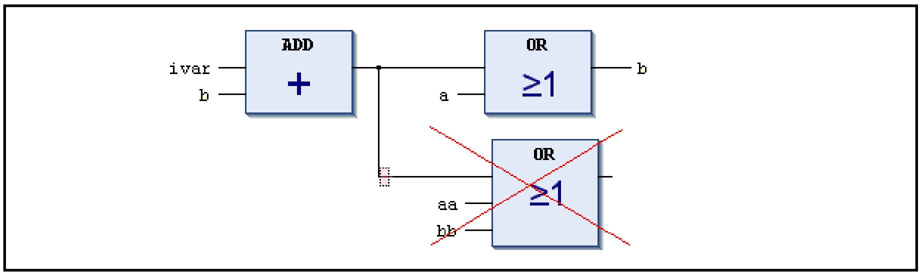


Fig.4-64: Deleting branch or subnetwork

Execution in online mode:

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

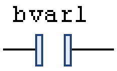
IL (instruction list)

In IL, line branching and post-interconnection are returned by a respective sequence of instructions. See also the description of operators and modifiers, page 341.

Contact in FBD/LD/IL

This is a pure **LD element**.

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

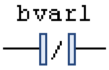


A contact transfers the condition "ON" (TRUE) or "OFF" (FALSE) from left to right until it finally reaches a coil in the right section of the network. For this purpose, a **Boolean variable** that contains the condition is assigned to the contact.

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, page 261,. 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 toolbox, page 355, directly provides negated contact elements.

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

- Add contact, page 257
- Add contact (right), page 259
- Add parallel contact (below), page 259
- Add parallel contact (above), page 259

or directly via drag&drop from the toolbox, page 355

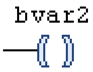
FBD, IL

If you are currently editing in the FBD or 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

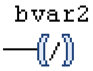
This is a pure **LD element**.

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:



In a negated coil, the negated value, page 261, 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.

A coil can be inserted using the Add assignment, page 250, command located in the FBD/LD/IL menu by default or via drag&drop from the toolbox, the "Ladder diagram elements" category. ( ).

See also Set/reset coils, page 257.

FBD/IL:

If you are currently editing in the FBD or 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

A Boolean output in FBD or the corresponding coil in LD can be set or reset. The output or the coil icon are identified with an S (set) or R (reset).

See also Set/reset, page 263.

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.

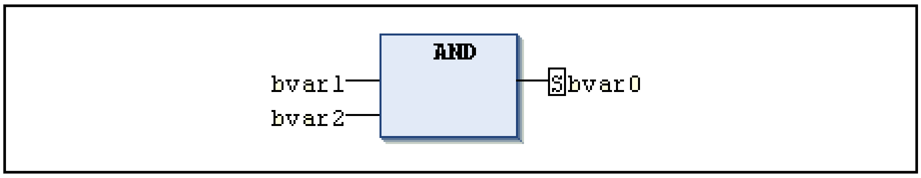


Fig.4-65: Example of a set output in FBD

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

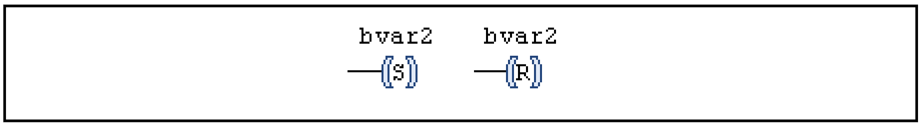


Fig.4-66: Examples of a set coil and reset coil

See Set/reset coil for information on set and reset coils.

IL

In an instruction list, the operators S and R 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": (S), (R).) into the editor.

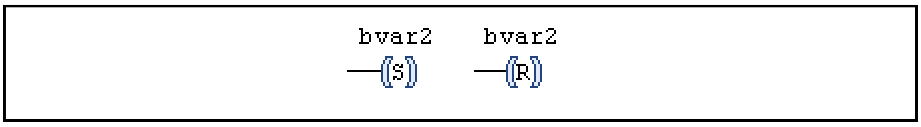


Fig.4-67: Example - Set coil, reset coil

4.6.9 FBD/LD/IL Editor in Online Mode

In the online mode of the FBD/LD/IL editor, there are views for monitoring and for writing, page 144, and forcing page 145, variables and expressions on the control.

Debugging functionality, page 82, (breakpoint, step-by-step execution, etc.) is also available.

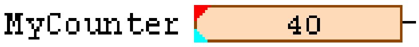
Monitoring

If inline monitoring is not explicitly disabled (Options, page 207), 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.

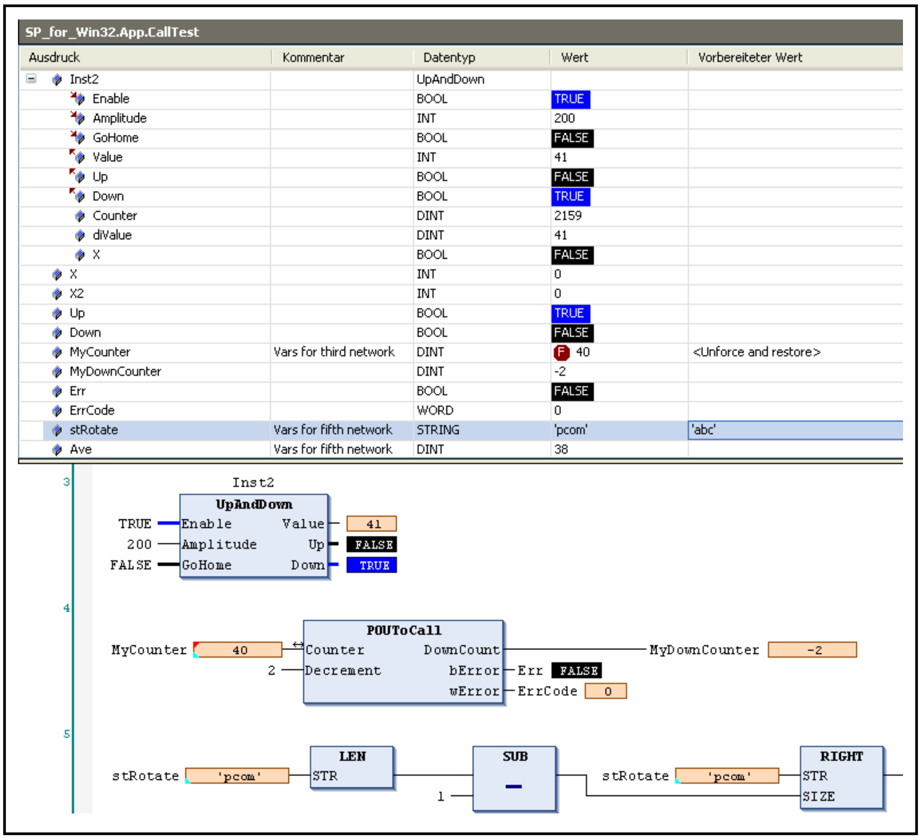
These windows display the respectively current value on the target system (**Inline monitoring**).

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 forced, page 145,,



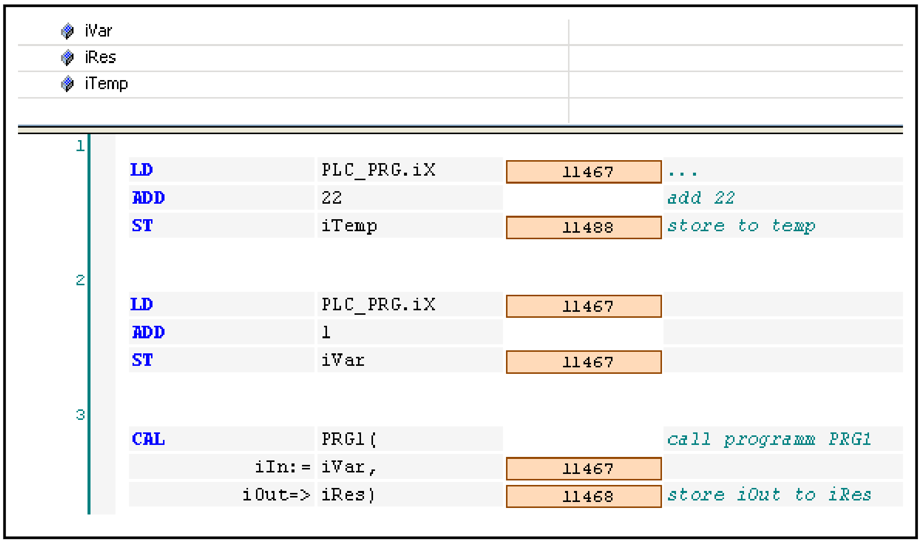
a blue triangle in the left bottom corner if the variable is currently forced and prepared to cancel forcing:



SP_for_Win32.App.CallTest

Ausdruck	Kommentar	Datentyp	Wert	Vorbereiteter Wert
Inst2		UpAndDown		
Enable		BOOL	TRUE	
Amplitude		INT	200	
GoHome		BOOL	FALSE	
Value		INT	41	
Up		BOOL	FALSE	
Down		BOOL	TRUE	
Counter		DINT	2159	
diValue		DINT	41	
X		BOOL	FALSE	
X		INT	0	
X2		INT	0	
Up		BOOL	TRUE	
Down		BOOL	FALSE	
MyCounter	Vars for third network	DINT	F 40	<Unforce and restore>
MyDownCounter		DINT	-2	
Err		BOOL	FALSE	
ErrCode		WORD	0	
stRotate	Vars for fifth network	STRING	'pcom'	'abc'
Ave	Vars for fifth network	DINT	38	

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



iVar	
iRes	
iTemp	

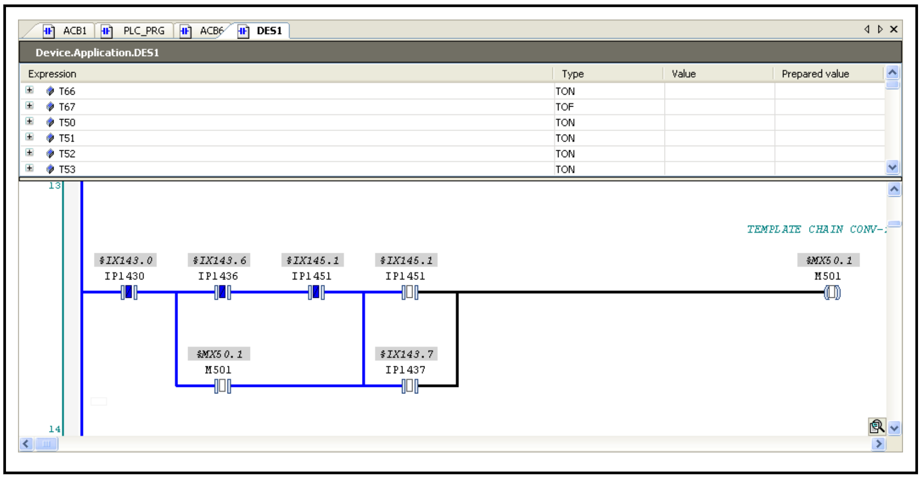
```
1
LD      PLC_PRG.iX    11467   ...
ADD     22                    add 22
ST      iTemp         11488   store to temp

2
LD      PLC_PRG.iX    11467
ADD     1
ST      iVar          11467

3
CAL     PRG1(                 call programm PRG1
   iIn:= iVar,        11467
  iOut=> iRes)        11468   store iOut to iRes
```

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.)



Device.Application.DES1

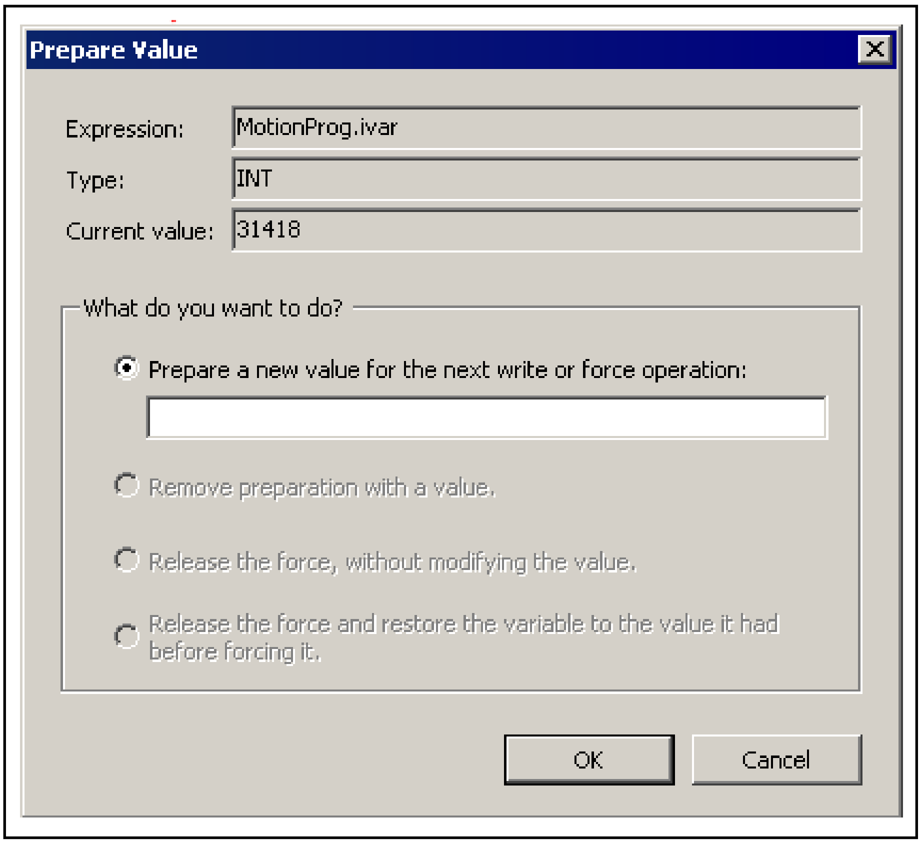
Expression	Type	Value	Prepared value
T66	TON		
T67	TOF		
T50	TON		
T51	TON		
T52	TON		
T53	TON		

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



Expression: MotionProg.ivar

Type: INT

Current value: 31418

What do you want to do?

● Prepare a new value for the next write or force operation:

○ Remove preparation with a value.

○ Release the force, without modifying the value.

○ Release the force and restore the variable to the value it had before forcing it.

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

- 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.

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

Possible positions that can be selected for a breakpoint (Breakpoint, page 136) 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 "Breakpoint" dialog, page 136,. 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.

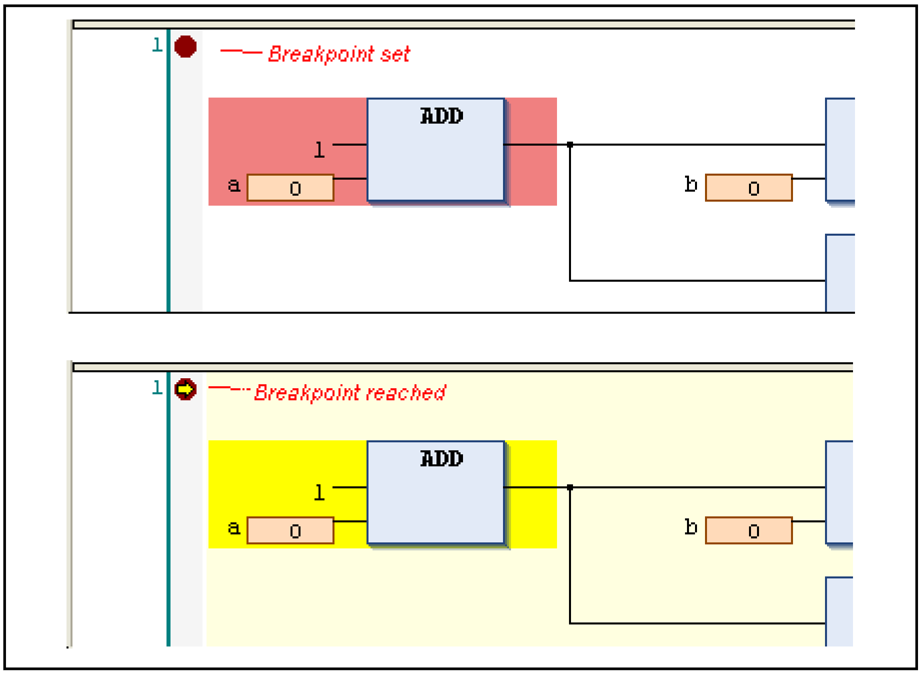


Fig.4-72: Example: Breakpoint set or reached

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.

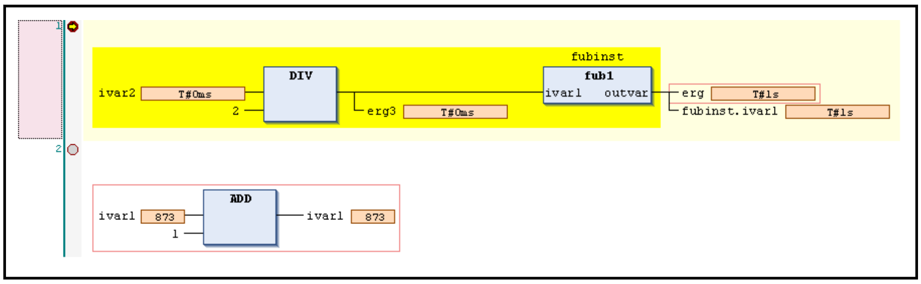
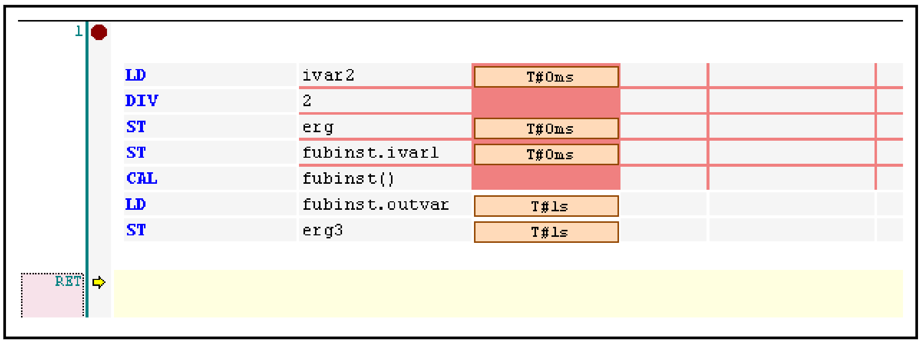


Fig.4-73: Breakpoint positions in FBD



```
1
LD   ivar2           T#0ms
DIV  2
ST   erg             T#0ms
ST   fubinst.ivar1   T#0ms
CAL  fubinst()
LD   fubinst.outvar  T#1s
ST   erg3            T#1s
RET
```

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.

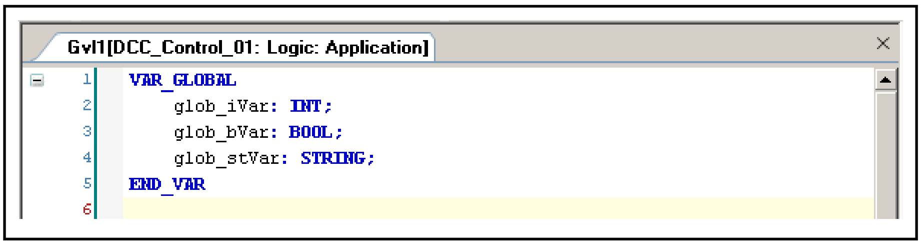
If a method is called by a pointer to a function block, the breakpoints are set in the method of the function block and in all of the derived function blocks that record for the method.

4.7 GVL Editor

The editor for global variable lists (GVL editor) is a declaration editor to create and edit global variable lists, page 52. It functions based on the current settings in the text editor options, page 212, and is displayed in online mode as described for the declaration editor, page 330,.

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

Valid declarations, page 503, of global variables are inserted in between.



```
VAR_GLOBAL
    glob_iVar: INT;
    glob_bVar: BOOL;
    glob_stVar: STRING;
END_VAR
```

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 Library repository dialog, page 185,.

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.

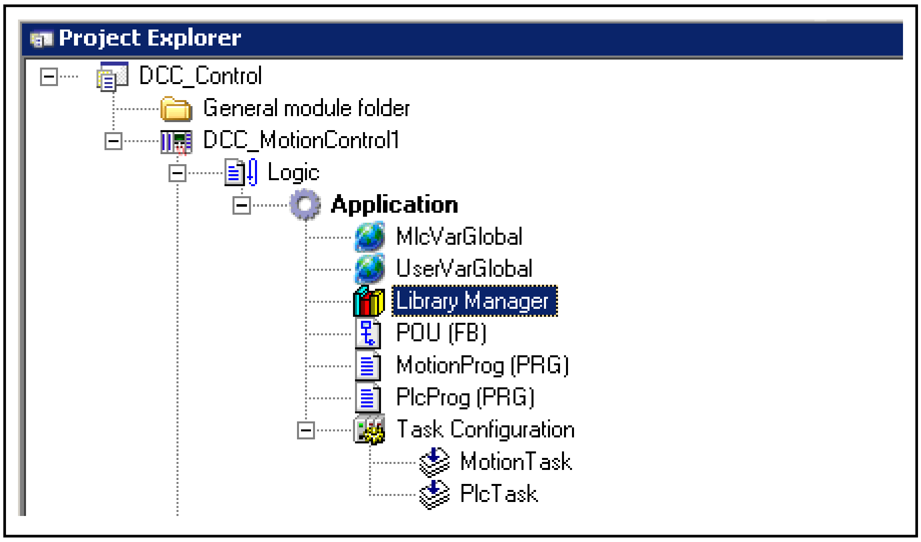


Fig.4-76: Library manager in the Project Explorer

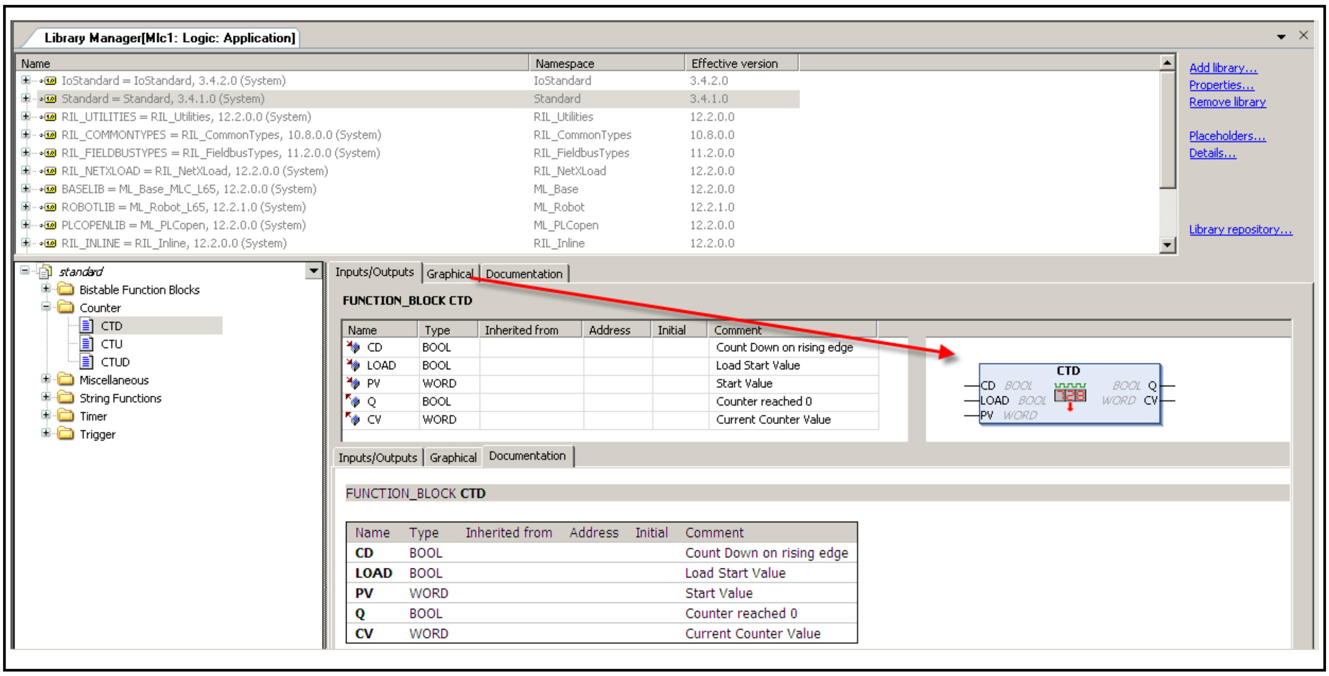
The library manager opens via "Open" or a double-click.

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

For general information on the IndraLogic 2G library management, see page 83.

4.8.2 Editor Window of the Library Manager

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


Library Manager[Mlc1: Logic: Application]

Name	Namespace	Effective version
IoStandard = IoStandard, 3.4.2.0 (System)	IoStandard	3.4.2.0
Standard = Standard, 3.4.1.0 (System)	Standard	3.4.1.0
RIL_UTILITIES = RIL_Utilities, 12.2.0.0 (System)	RIL_Utilities	12.2.0.0
RIL_COMMONTYPES = RIL_CommonTypes, 10.8.0.0 (System)	RIL_CommonTypes	10.8.0.0
RIL_FIELDBUSTYPES = RIL_FieldbusTypes, 11.2.0.0 (System)	RIL_FieldbusTypes	11.2.0.0
RIL_NETXLOAD = RIL_NetXLoad, 12.2.0.0 (System)	RIL_NetXLoad	12.2.0.0
BASELIB = ML_Base_MLC_L65, 12.2.0.0 (System)	ML_Base	12.2.0.0
ROBOTLIB = ML_Robot_L65, 12.2.1.0 (System)	ML_Robot	12.2.1.0
PLCOPENLIB = ML_PLCopen, 12.2.0.0 (System)	ML_PLCopen	12.2.0.0
RIL_INLINE = RIL_Inline, 12.2.0.0 (System)	RIL_Inline	12.2.0.0

FUNCTION_BLOCK CTD

Name	Type	Inherited from	Address	Initial	Comment
CD	BOOL				Count Down on rising edge
LOAD	BOOL				Load Start Value
PV	WORD				Start Value
Q	BOOL				Counter reached 0
CV	WORD				Current Counter Value

FUNCTION_BLOCK **CTD**

Name	Type	Inherited from	Address	Initial	Comment
CD	BOOL				Count Down on rising edge
LOAD	BOOL				Load Start Value
PV	WORD				Start Value
Q	BOOL				Counter reached 0
CV	WORD				Current Counter Value

Fig.4-77: Library manager editor window

Structure of the editor window

The upper section of the library manager displays the libraries, page 83 currently included in the project. The following information is given:

Name: Title, version and the company name (optional) as defined in the "Summary" dialog in the library information, page 371, 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.

The namespace, page 503, 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 buttons and commands in the library editor window, page 370.

Further information on library namespaces, page 83.

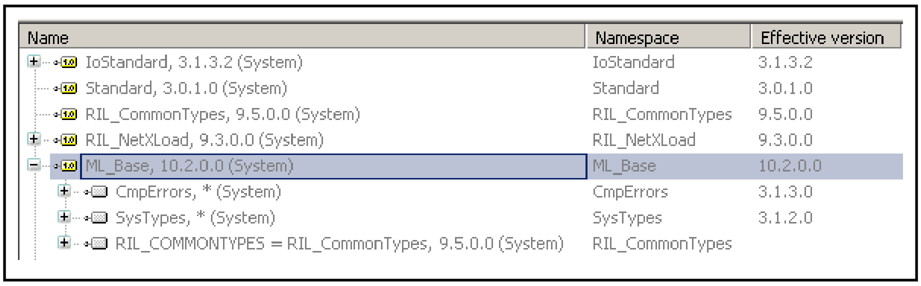
Effective version: indicates the version of the library currently used in the project according to the definition in the library properties, page 230,.

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:

1.0	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: Reloading library, page 231.

If a library has **dependencies** on other libraries (referenced libraries, page 83), 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.



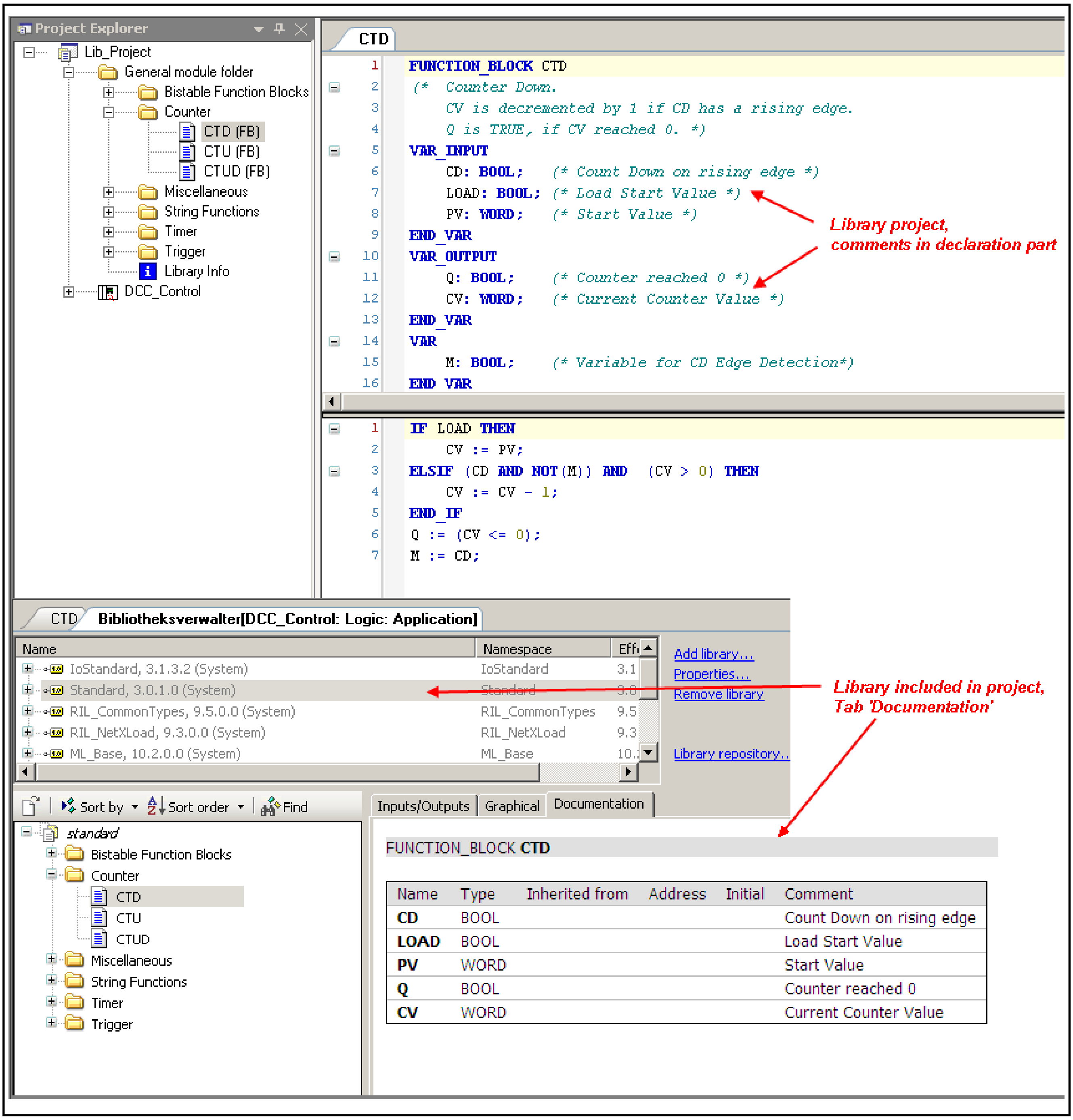
Name	Namespace	Effective version
+ 1.0 IoStandard, 3.1.3.2 (System)	IoStandard	3.1.3.2
1.0 Standard, 3.0.1.0 (System)	Standard	3.0.1.0
1.0 RIL_CommonTypes, 9.5.0.0 (System)	RIL_CommonTypes	9.5.0.0
+ 1.0 RIL_NetXLoad, 9.3.0.0 (System)	RIL_NetXLoad	9.3.0.0
− 1.0 ML_Base, 10.2.0.0 (System)	ML_Base	10.2.0.0
+ ▭ 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 library was created, page 83. Adding such a comment is an easy way to automatically provide the user with documentation of a function block.



```
FUNCTION_BLOCK CTD
(*  Counter Down.
    CV is decremented by 1 if CD has a rising edge.
    Q is TRUE, if CV reached 0. *)
VAR_INPUT
    CD: BOOL;    (* Count Down on rising edge *)
    LOAD: BOOL;  (* Load Start Value *)
    PV: WORD;    (* Start Value *)
END_VAR
VAR_OUTPUT
    Q: BOOL;     (* Counter reached 0 *)
    CV: WORD;    (* Current Counter Value *)
END_VAR
VAR
    M: BOOL;     (* Variable for CD Edge Detection*)
END_VAR
```

Library project, comments in declaration part

```
IF LOAD THEN
    CV := PV;
ELSIF (CD AND NOT(M)) AND  (CV > 0) THEN
    CV := CV - 1;
END_IF
Q := (CV <= 0);
M := CD;
```

Name	Namespace	Effi
IoStandard, 3.1.3.2 (System)	IoStandard	3.1
Standard, 3.0.1.0 (System)	Standard	3.0
RIL_CommonTypes, 9.5.0.0 (System)	RIL_CommonTypes	9.5
RIL_NetXLoad, 9.3.0.0 (System)	RIL_NetXLoad	9.3
ML_Base, 10.2.0.0 (System)	ML_Base	10.:

Library included in project, Tab 'Documentation'

FUNCTION_BLOCK **CTD**

Name	Type	Inherited from	Address	Initial	Comment
CD	BOOL				Count Down on rising edge
LOAD	BOOL				Load Start Value
PV	WORD				Start Value
Q	BOOL				Counter reached 0
CV	WORD				Current Counter Value

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 be found in the "Libraries" 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.

When the Library Manager, page 367, is highlighted in the Project Explorer, the "Library Manager" menu appears in the menu bar by default.

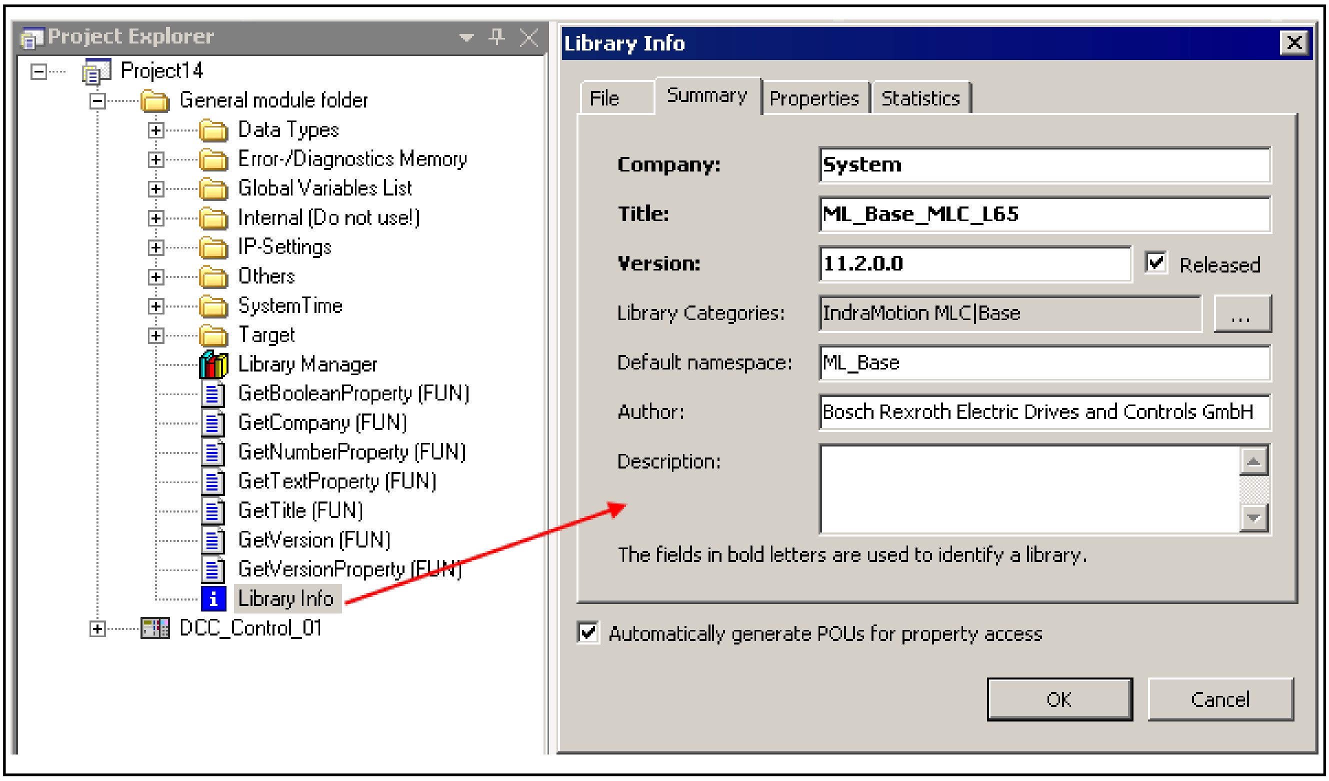
Installing libraries, like defining library storage locations (repositories), is done in the Library Repository dialog, page 185,.

4.8.4 Library Information

Icon: ℹ

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 create a library, page 191,.



Company: System

Title: ML_Base_MLC_L65

Version: 11.2.0.0 ☑ Released

Library Categories: IndraMotion MLC|Base

Default namespace: ML_Base

Author: Bosch Rexroth Electric Drives and Controls GmbH

Description:

The fields in bold letters are used to identify a library.

☑ Automatically generate POUs for property access

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 property keys 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 Properties tab) as input so that the property value is returned.

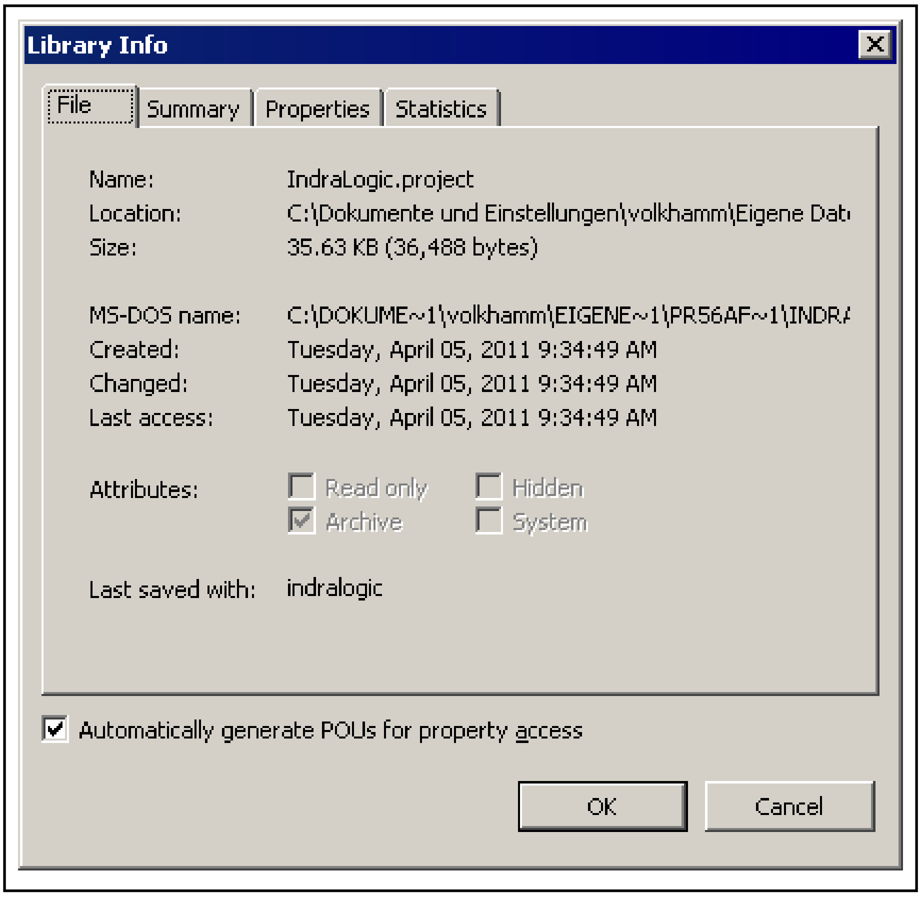
For an example, refer to Library Info and Access Functions, page 377.

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



Name: IndraLogic.project
Location: C:\Dokumente und Einstellungen\volkhamm\Eigene Dat
Size: 35.63 KB (36,488 bytes)

MS-DOS name: C:\DOKUME~1\volkhamm\EIGENE~1\PR56AF~1\INDR
Created: Tuesday, April 05, 2011 9:34:49 AM
Changed: Tuesday, April 05, 2011 9:34:49 AM
Last access: Tuesday, April 05, 2011 9:34:49 AM

Attributes: □ Read only □ Hidden
☑ Archive □ System

Last saved with: indralogic

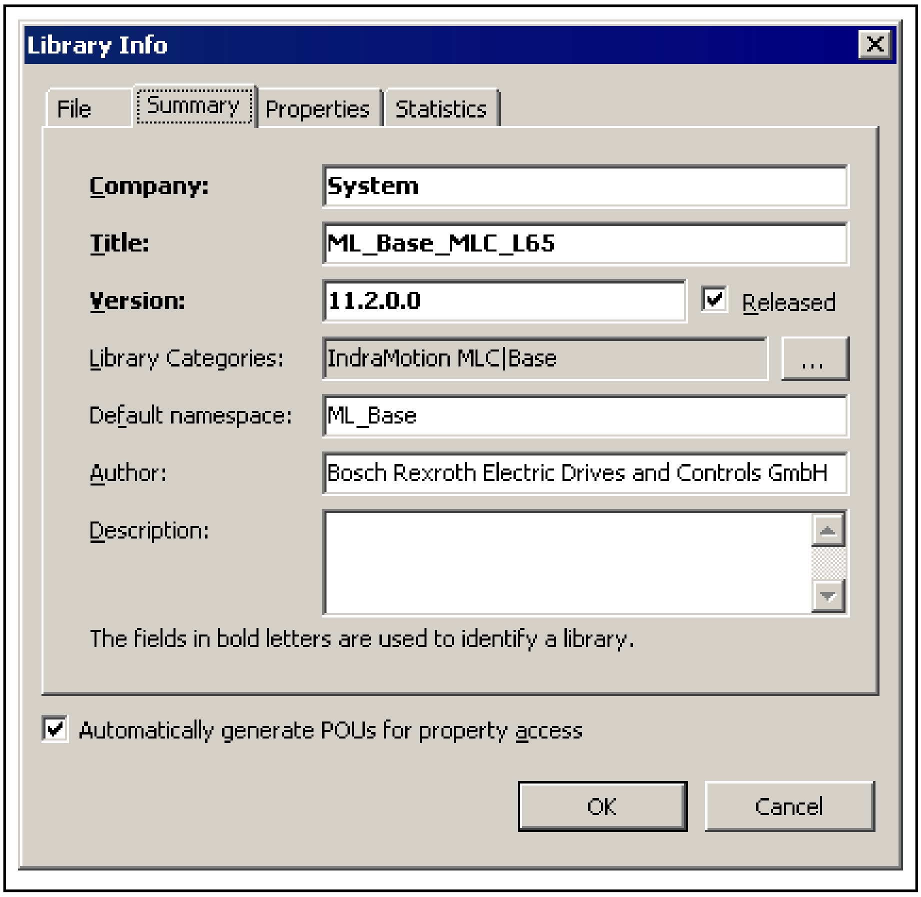
☑ Automatically generate POUs for property access

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).

2. Summary



Company: System

Title: ML_Base_MLC_L65

Version: 11.2.0.0 ☑ Released

Library Categories: IndraMotion MLC|Base ...

Default namespace: ML_Base

Author: Bosch Rexroth Electric Drives and Controls GmbH

Description:

The fields in bold letters are used to identify a library.

☑ Automatically generate POUs for property access

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 installed, page 185, on the system and included in 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:

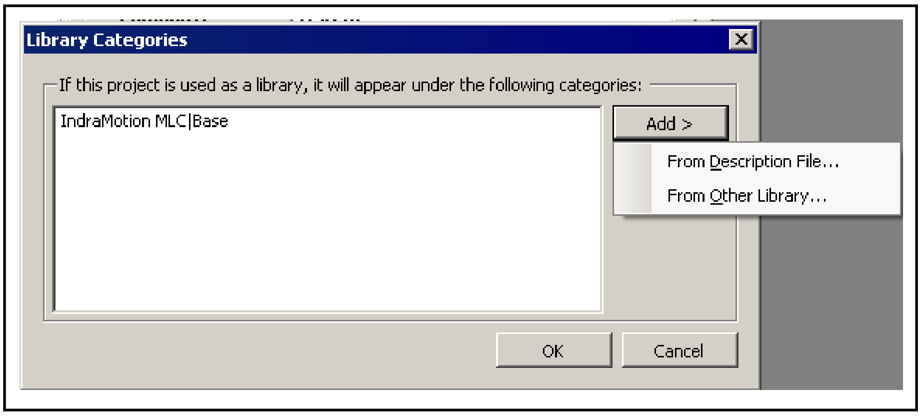


Fig.4-83: Dialog to select library categories

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

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

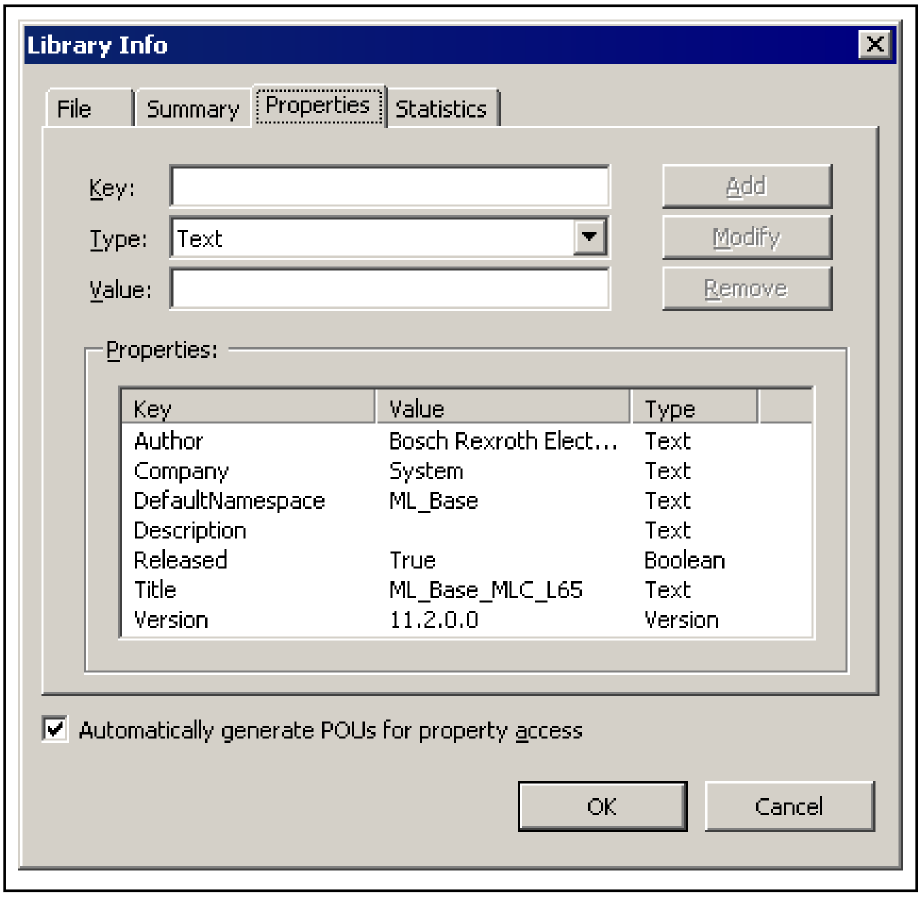
In both cases, the default dialog for browsing for a file appears. The filter is set to *.libcat.xml or *.library.

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.

3. Properties



Key:

Type: Text

Value:

Properties:

Key	Value	Type
Author	Bosch Rexroth Elect...	Text
Company	System	Text
DefaultNamespace	ML_Base	Text
Description		Text
Released	True	Boolean
Title	ML_Base_MLC_L65	Text
Version	11.2.0.0	Version

☑ Automatically generate POUs for property access

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.

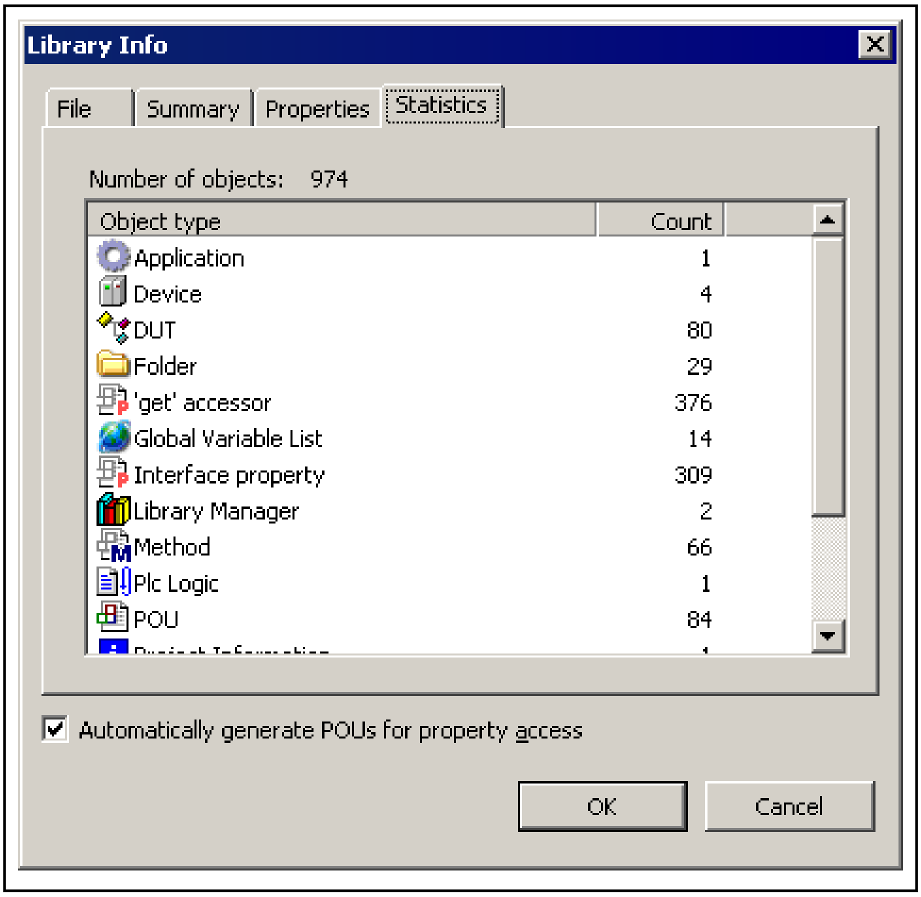
The information in the Summary tab is provided as "key". The property names are used as key names, "Text" is automatically set as the data type and the "Values" consist of the texts entered in the "Summary" tab. However, other keys can be added explicitly (see the following).

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:



Number of objects: 974

Object type	Count
Application	1
Device	4
DUT	80
Folder	29
'get' accessor	376
Global Variable List	14
Interface property	309
Library Manager	2
Method	66
Plc Logic	1
POU	84
Project Information	1

☑ Automatically generate POUs for property access

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:

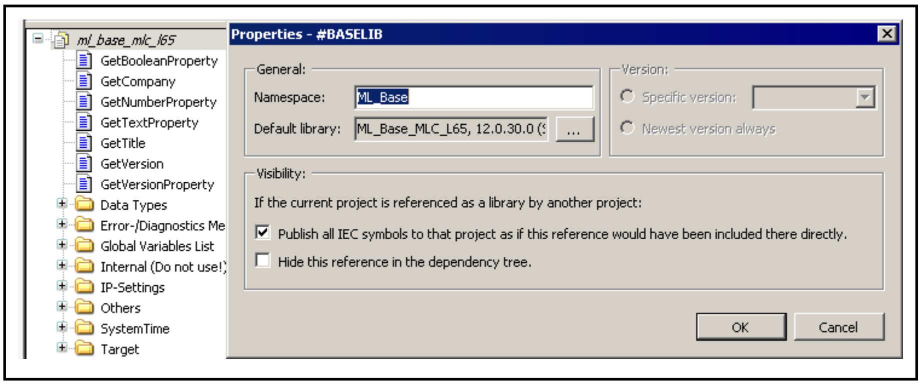
Functions without input parameters

- GetCompany, page 378
- GetTitle, page 378
- GetVersion, page 379

Functions with input parameters (key)

- GetBooleanProperty, page 379,
- GetNumberProperty, page 380,

- GetTextProperty, page 381, and
- GetVersionProperty, page 380



ml_base_mlc_l65
- GetBooleanProperty
- GetCompany
- GetNumberProperty
- GetTextProperty
- GetTitle
- GetVersion
- GetVersionProperty
- Data Types
- Error-/Diagnostics Me
- Global Variables List
- Internal (Do not use!)
- IP-Settings
- Others
- SystemTime
- Target

Properties - #BASELIB

General:

Namespace: ML_Base

Default library: ML_Base_MLC_L65, 12.0.30.0 (

Version:

○ Specific version:

○ Newest version always

Visibility:

If the current project is referenced as a library by another project:

☑ Publish all IEC symbols to that project as if this reference would have been included there directly.

□ Hide this reference in the dependency tree.

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

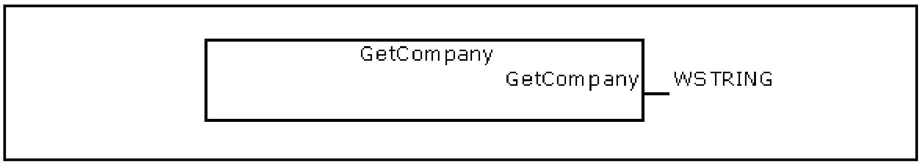


Fig.4-88: "GetCompany" function

	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

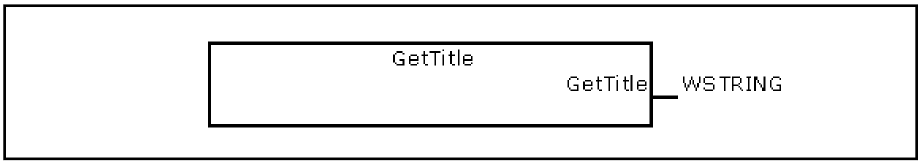


Fig.4-91: "GetTitle" function

	Name	Type	Comment
Function value	GetTitle	WSTRING	Library title

Fig.4-92: "FUN GetTitle" interface

Implementation example:

Example in ST:

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

GetVersion

Brief description

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

Library	Range
ML_Base	

Fig.4-93: Library assignment

Interface description

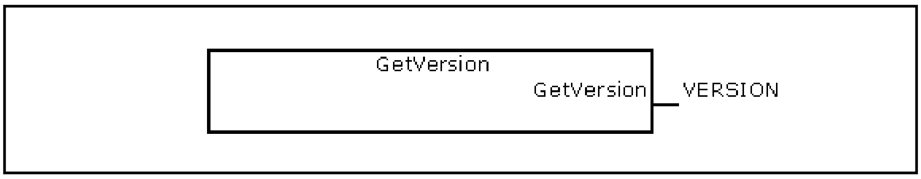


Fig.4-94: "GetVersion" function

	Name	Type	Comment
Function value	GetTitle	VERSION	Exact library version

Fig.4-95: "FUN GetVersion" interface

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.

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

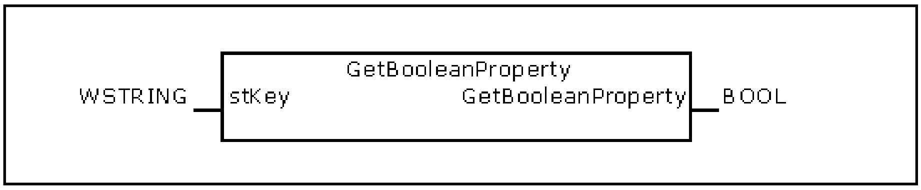


Fig.4-97: "GetBooleanProperty" function

	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

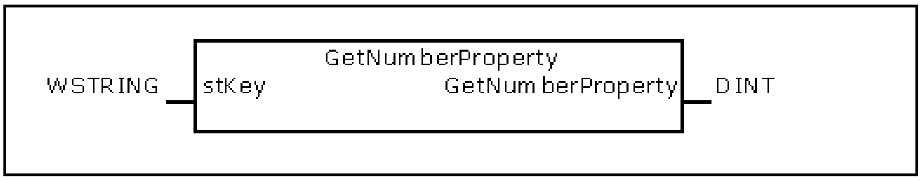


Fig.4-100: "GetNumberProperty" function

	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

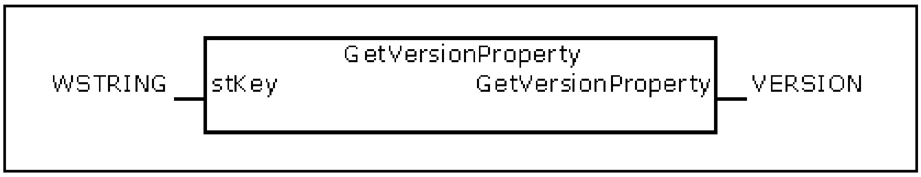


Fig.4-103: "GetVersionProperty" function

	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

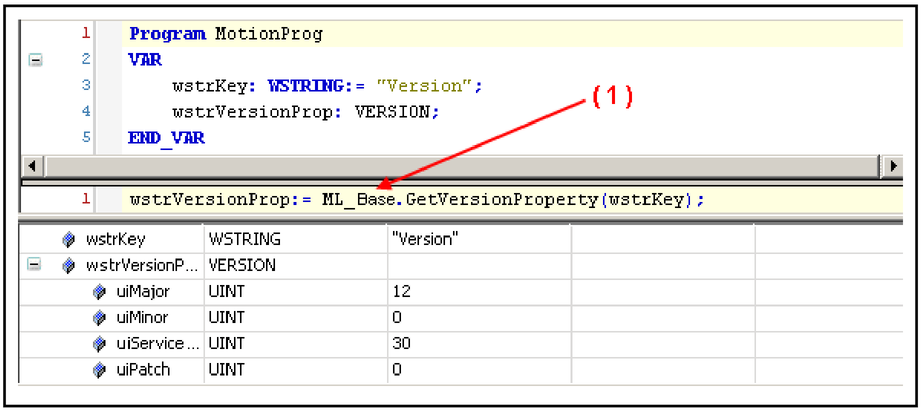
of the library are assigned.

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.



```
Program MotionProg
VAR
    wstrKey: WSTRING:= "Version";
    wstrVersionProp: VERSION;
END_VAR
```

```
wstrVersionProp:= ML_Base.GetVersionProperty(wstrKey);
```

wstrKey	WSTRING	"Version"
wstrVersionP...	VERSION	
uiMajor	UINT	12
uiMinor	UINT	0
uiService...	UINT	30
uiPatch	UINT	0

(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

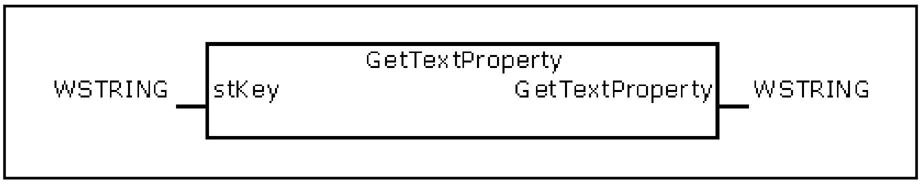


Fig.4-107: "GetTextProperty" function

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

Fig.4-108: "FUN GetTextProperty" interface

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

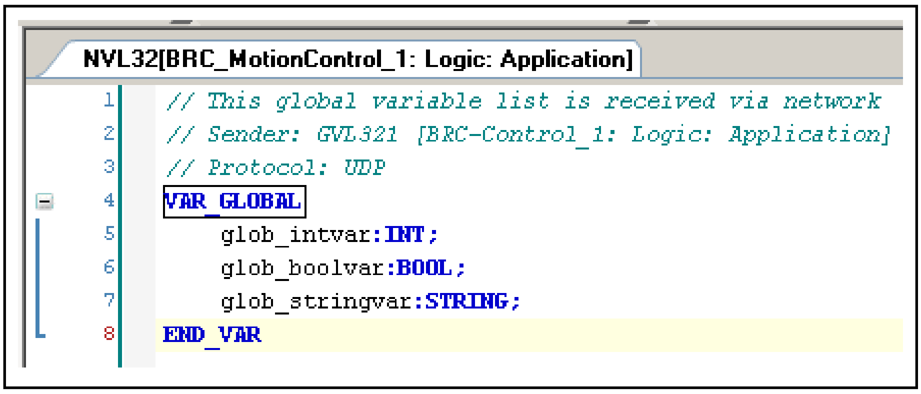
Fig.4-109: Keywords and function values

4.9 Network Variable List Editor

The NVL editor is a declaration editor to create network variable lists, page 53. It works based on the current text editor options, page 212, and in online mode and generally described for the declaration editor, page 330,.

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 declarations, page 503, of global variables are inserted in between.



NVL32[BRC_MotionControl_1: Logic: Application]

```
// This global variable list is received via network
// Sender: GVL321 [BRC-Control_1: Logic: Application]
// Protocol: UDP
VAR_GLOBAL
    glob_intvar:INT;
    glob_boolvar:BOOL;
    glob_stringvar:STRING;
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.

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.

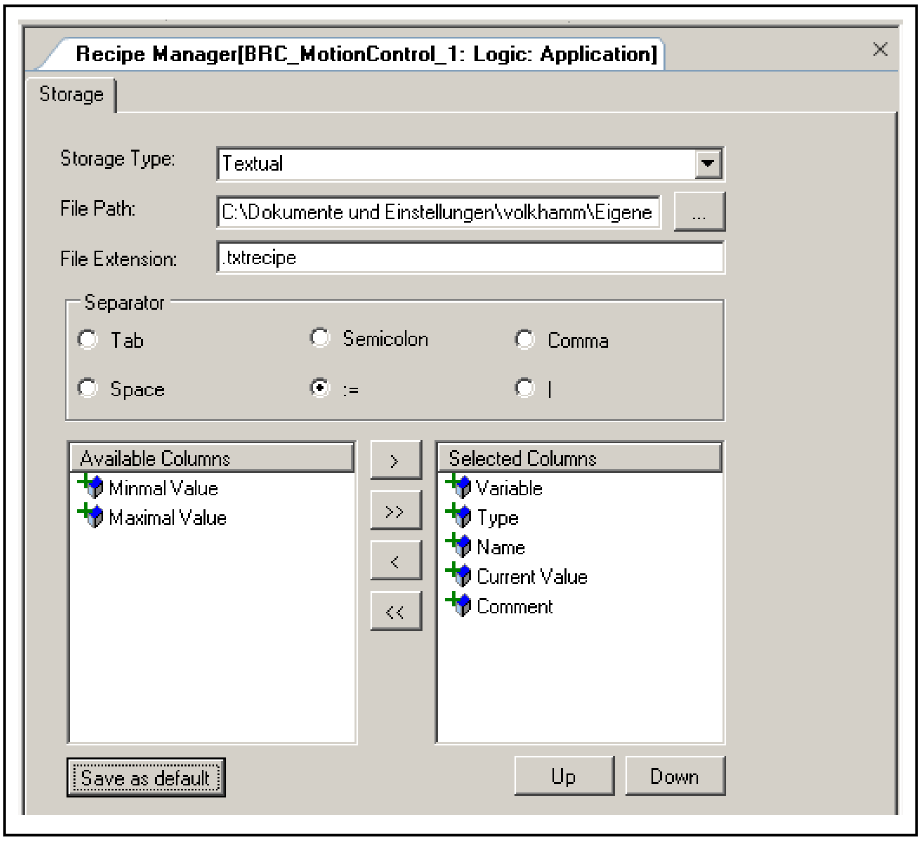
Call `g_RecipeManager.LastError` after reading and writing are completed to check whether the transfer was successful (g_RecipeManager.LastError=0 in this case).

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.



Recipe Manager[BRC_MotionControl_1: Logic: Application]

Storage

Storage Type: Textual

File Path: C:\Dokumente und Einstellungen\volkhamm\Eigene

File Extension: .txtrecipe

Separator

○ Tab ○ Semicolon ○ Comma

○ Space ● := ○ |

Available Columns	Selected Columns
Minmal Value	Variable
Maximal Value	Type
	Name
	Current Value
	Comment

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.

A recipe definition along with the recipes internally defined is displayed as table in the recipe manager editor, page 233, and it can be edited there.

4.10.2 Recipe Definition

The **recipe manager** manager 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.

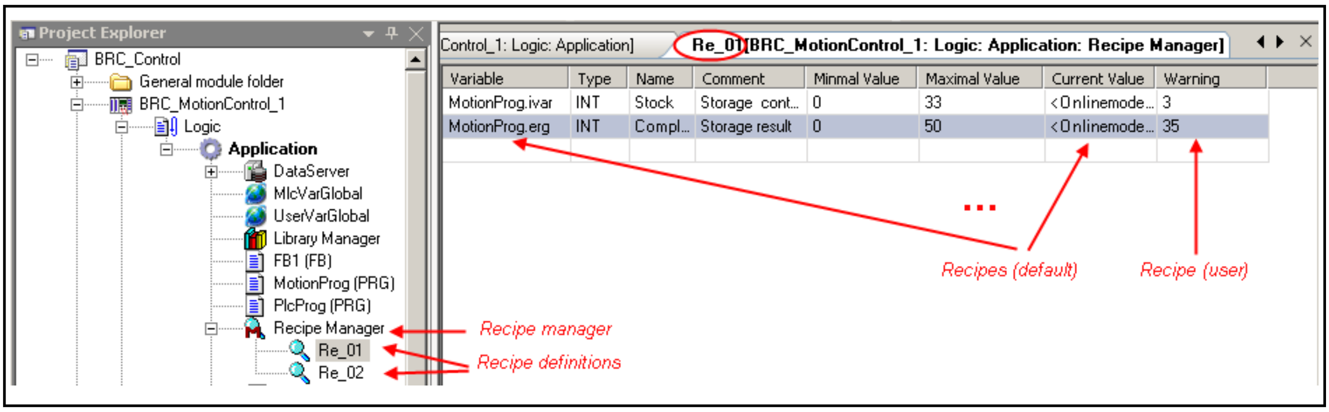
By using different recipes, one set of variables on the control can be assigned in a single action using another set of values.

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.


Project Explorer
Control_1: Logic: Application]
Re_01[BRC_MotionControl_1: Logic: Application: Recipe Manager]
...
Recipes (default)
Recipe (user)
Recipe manager
Recipe definitions

BRC_Control
General module folder
BRC_MotionControl_1
Logic
Application
DataServer
MlcVarGlobal
UserVarGlobal
Library Manager
FB1 (FB)
MotionProg (PRG)
PlcProg (PRG)
Recipe Manager
Re_01
Re_02

Variable	Type	Name	Comment	Minmal Value	Maximal Value	Current Value	Warning
MotionProg.ivar	INT	Stock	Storage cont...	0	33	<Onlinemode...	3
MotionProg.erg	INT	Compl...	Storage result	0	50	<Onlinemode...	35

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 Recipe manager, page 233.

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 name of the recipe, see Recipe definition, editor window, page 384. 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 Visualization editor, page 451.

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 ###

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 text editor options, page 212. 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 declaration editor, page 326, 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 monitoring, i.e. displays and for writing, page 144. and forcing, page 145, values onto variables and expressions.

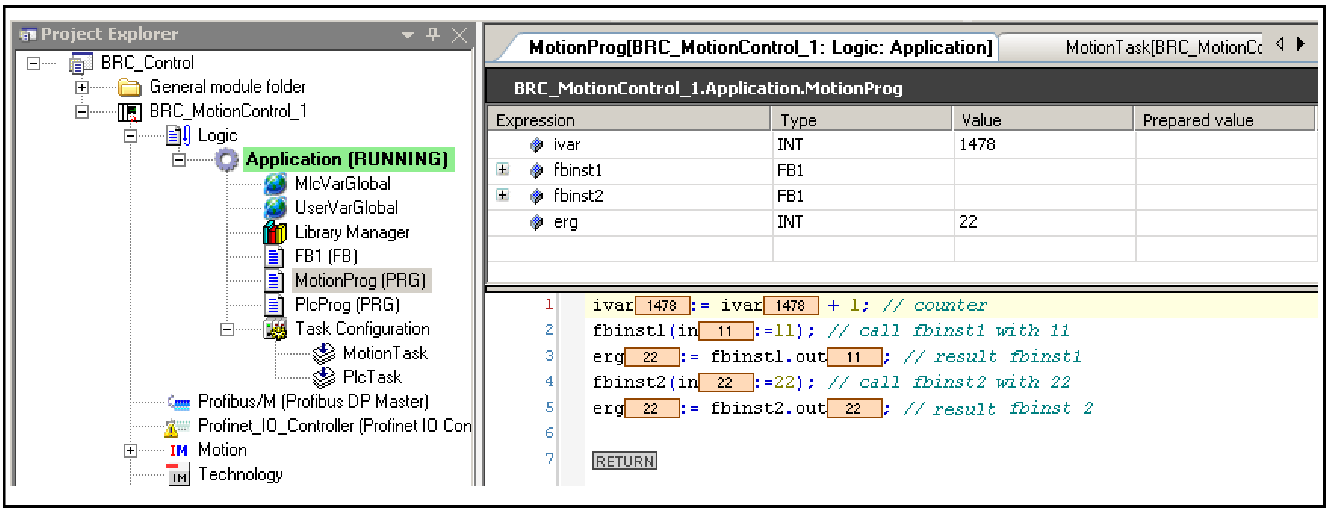
debugging functions, page 82, (breakpoints, single step processing, etc.) are available. See Breakpoint positions in the ST editor, page 388.

- See the description in "Forcing variables", page 387, 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 declaration editor, page 330.

Monitoring

If the **Inline monitoring** function is not explicitly disabled (Options Dialogs, page 212), the current variable value is displayed in a small window after the variable.



BRC_MotionControl_1.Application.MotionProg

Expression	Type	Value	Prepared value
ivar	INT	1478	
fbinst1	FB1		
fbinst2	FB1		
erg	INT	22	

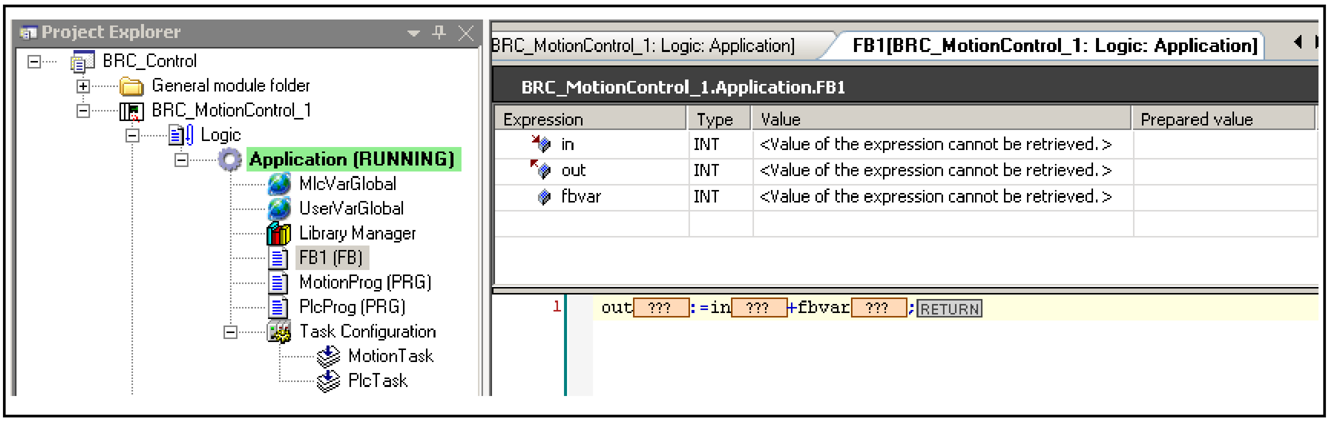
```
ivar 1478 := ivar 1478 + 1; // counter
fbinst1(in 11 :=11); // call fbinst1 with 11
erg 22 := fbinst1.out 11 ; // result fbinst1
fbinst2(in 22 :=22); // call fbinst2 with 22
erg 22 := fbinst2.out 22 ; // result fbinst 2

RETURN
```

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 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.



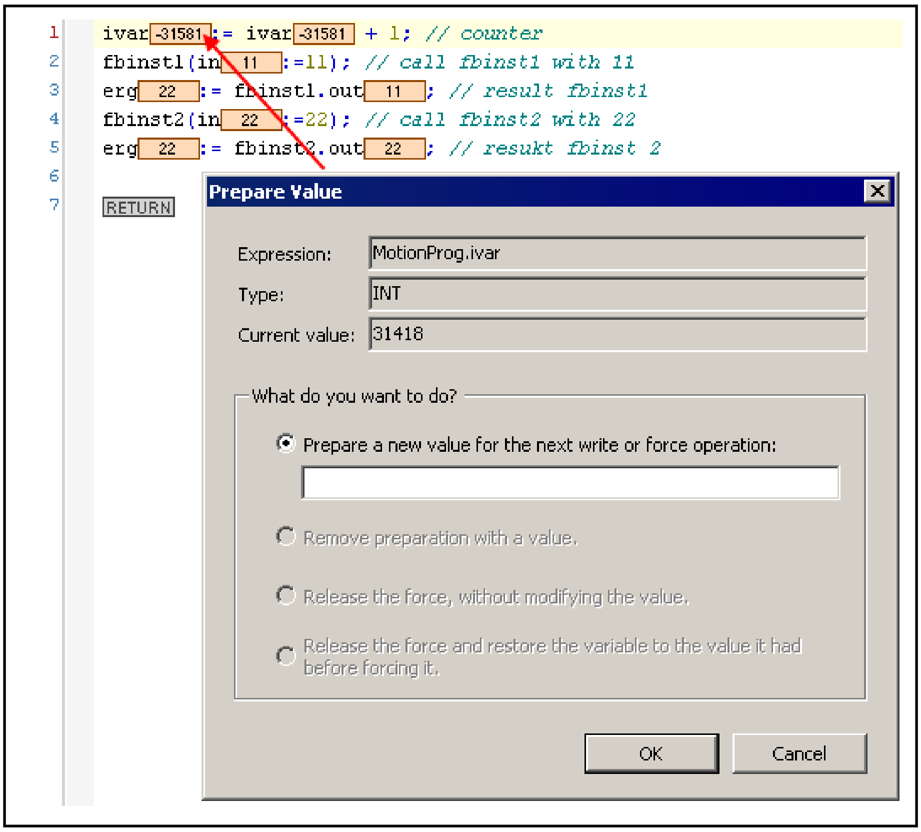
Expression	Type	Value	Prepared value
in	INT	<Value of the expression cannot be retrieved.>	
out	INT	<Value of the expression cannot be retrieved.>	
fbvar	INT	<Value of the expression cannot be retrieved.>	

```
out ??? :=in ??? +fbvar ??? ;RETURN
```

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.



```
ivar -31581 := ivar -31581 + 1; // counter
fbinst1(in 11 :=11); // call fbinst1 with 11
erg 22 := fbinst1.out 11 ; // result fbinst1
fbinst2(in 22 :=22); // call fbinst2 with 22
erg 22 := fbinst2.out 22 ; // resukt fbinst 2

RETURN
```

Prepare Value

Expression: MotionProg.ivar

Type: INT

Current value: 31418

What do you want to do?

- ● Prepare a new value for the next write or force operation:
- ○ Remove preparation with a value.
- ○ Release the force, without modifying the value.
- ○ Release the force and restore the variable to the value it had before forcing it.

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

- 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:

Users can set a breakpoint, page 82, 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 assignments as expressions, page 389, 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:

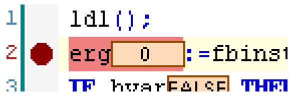
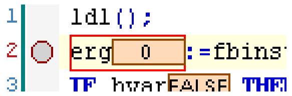
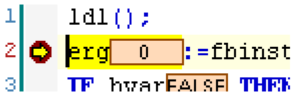

ldl();
erg 0 :=fbinst
IF bvar FALSE THE
	
	ldl();
	erg 0 :=fbins
	IF bvar FALSE THE
		
		ldl();
		erg 0 :=fbinst
		IF bvar FALSE THEN

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 expressions, page 389, and instructions, page 390 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:

- Assignment as expression, page 391
- Set/reset operators, page 391

Expressions

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

Expressions consist of operators, page 569, and operands, page 615.

Assignments, page 391, 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

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,
                                      // real_var1, real_var2 will get the value of int_var
int_var := real_var1 := 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 expressions are to be processed. The following instructions can be used in ST:

Instruction	Example
Assignment (Assignment operators, page 391)	A:=B; CV := CV + 1; C:=SIN(X);

Function block call, page 35,
and using the function block output
	CMD_TMR(IN := %IX5, PT := 300);
	A:=CMD_TMR.Q;

RETURN, page 393	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, page 394	CASE INT1 OF
	1: BOOL1 := TRUE;
	2: BOOL2 := TRUE;
	ELSE
	BOOL1 := FALSE;
	BOOL2 := FALSE;
	END_CASE;

FOR, page 394	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I;
	EXIT;
	END_IF;
	END_FOR;

WHILE, page 395	J:=1;
	WHILE J<= 100 AND ARR[J] <> 70 DO
	J:=J+2;
	END_WHILE;

REPEAT, page 396	J:=-1;
	REPEAT
	J:=J+2;
	UNTIL J= 101 OR ARR[J] = 70
	END_REPEAT;

EXIT, page 396	EXIT;
CONTINUE, page 396	CONTINUE;
JMP, page 397	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 :=.

Also refer to

- MOVE operator, page 573 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 expression, page 389,.

Calling a Function Block in ST

A function block, page 33 (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.

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:

- Function block, page 33,
- FB instance, page 34.

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

IF Instruction

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;
  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>.

- 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):

The CONTINUE instruction, page 396, can be used in a FOR loop.

WHILE 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.

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 CONTINUE instruction, page 396, 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 CONTINUE instruction, page 396, 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

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
 *)
```

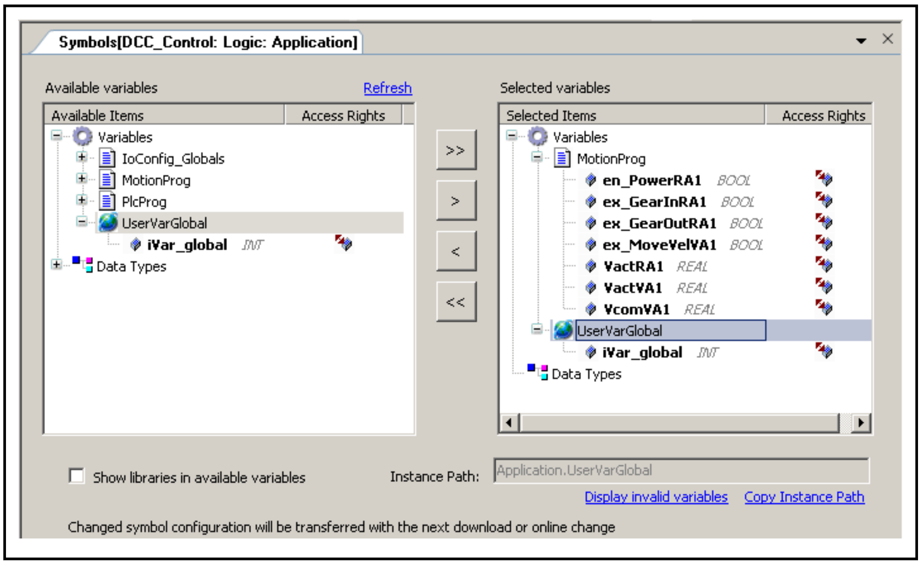
In this example, the comment that begins with the first open parenthesis does not end with the closing parenthesis after "checked", but with the parenthesis in the last line instead.

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 general description of the symbol configuration, page 306.

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.



☐ Show libraries in available variables

Instance Path: Application.UserVarGlobal

Changed symbol configuration will be transferred with the next download or online change

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.

Via POU property "Link Always", page 241,, a non-compiled object can be explicitly compiled and loaded to the control. Thus, all variables declared in the object are available. Furthermore, the pragma {attribute 'linkalways'}, page 536, 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

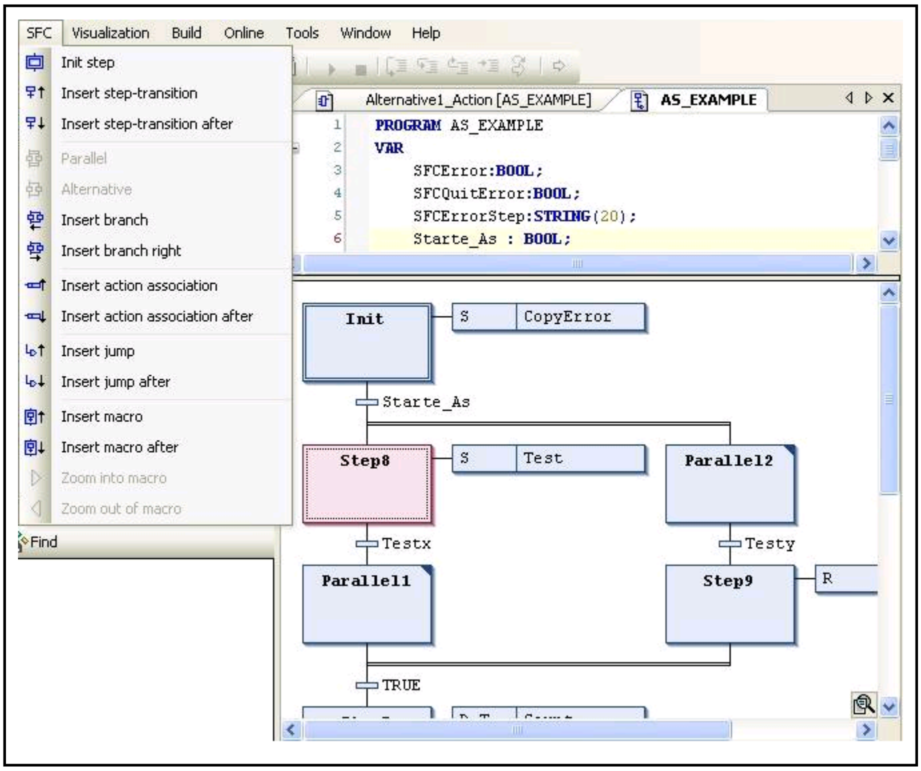
The SFC editor is used for programming POUs in the IEC 61131-3 Sequential Function Chart (SFC). The language is selected when a new POU object is created in the project using the dialog under **... ▸ Application ▸ Add ▸ POU**.

The SFC editor is a graphical editor.

General settings on behavior and appearance are made in the SFC editor options dialog, page 219,.

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 declaration editor, page 326,.



```
PROGRAM AS_EXAMPLE
VAR
    SFCError:BOOL;
    SFCQuitError:BOOL;
    SFCErrorStep:STRING(20);
    Starte_As : BOOL;
```

Fig.4-119: SFC editor

The elements 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: Working in the SFC editor, page 402

The properties of steps can be edited in a separate Properties window. Among other properties, the minimum and maximum activity time for a step can be defined there.

Implicit variables, page 414, 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 Working in the SFC editor, page 402.

  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 IndraLogic 1.x. Actions can now only be provided as POU objects and are always assigned to a step in the step properties.
- Macros can be used to structure an SFC diagram.

4.13.2 SFC - Sequential Function Chart

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 Description of the step processing sequence in online mode, page 420).

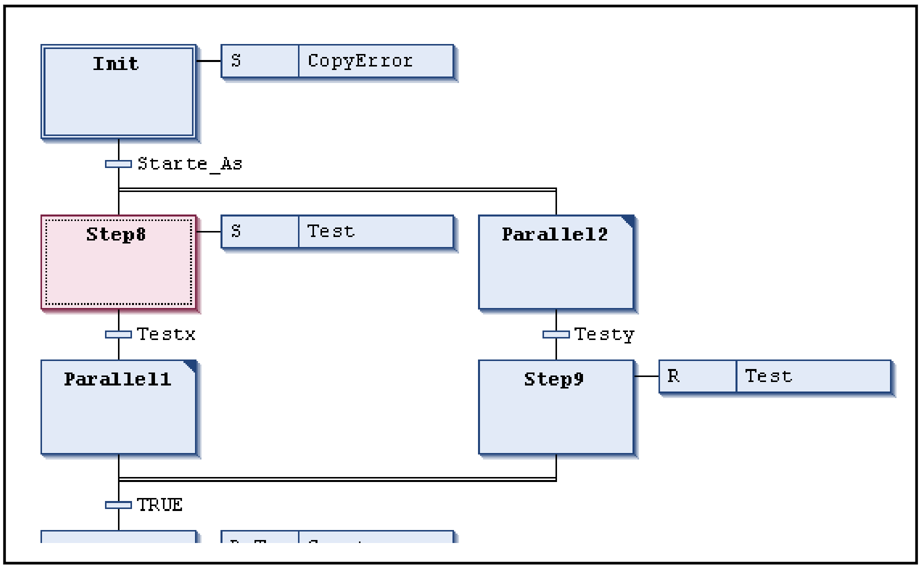


Fig.4-120: Example of a step sequence in an SFC module

4.13.3 Cursor Positions in SFC

A possible cursor position in an SFC diagram is displayed by default with gray shading in the SFC editor 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

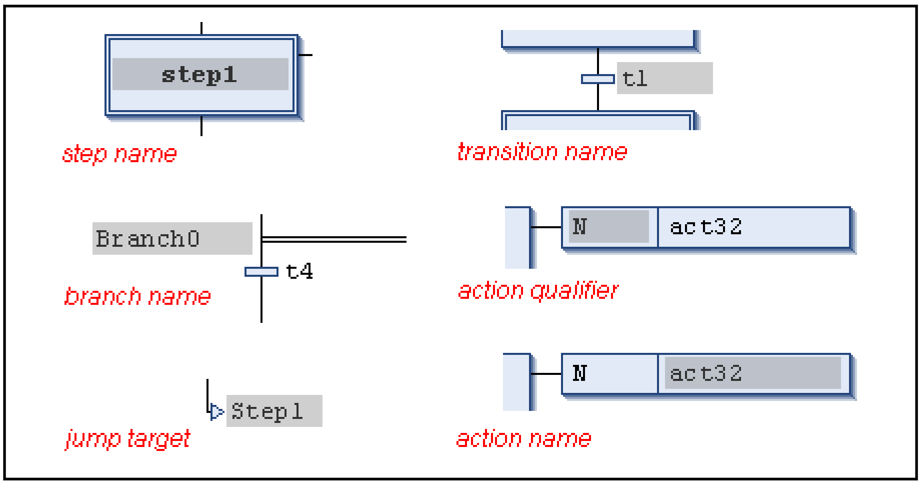


Fig.4-121: Possible cursor positions, texts:

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

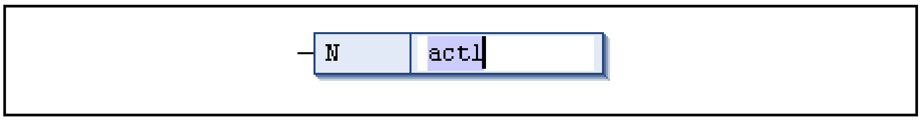


Fig.4-122: The action name was selected for editing

2. Element bodies

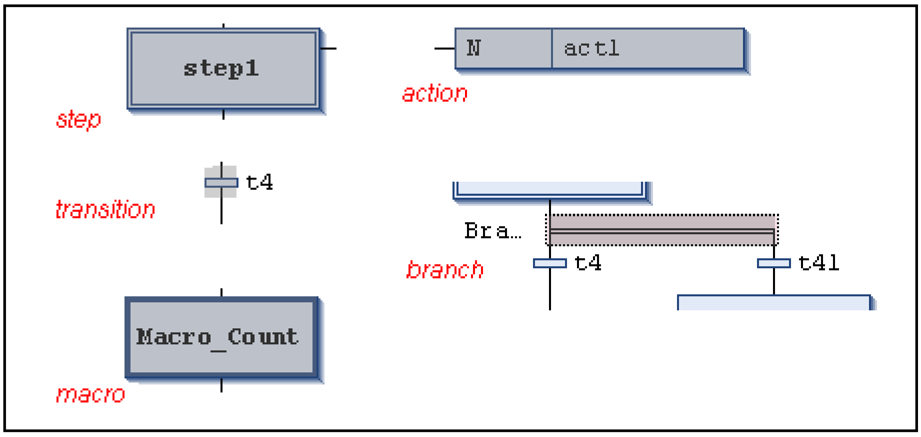


Fig.4-123: Possible cursor positions, element bodies:

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.

(Multiple selection, page 402):

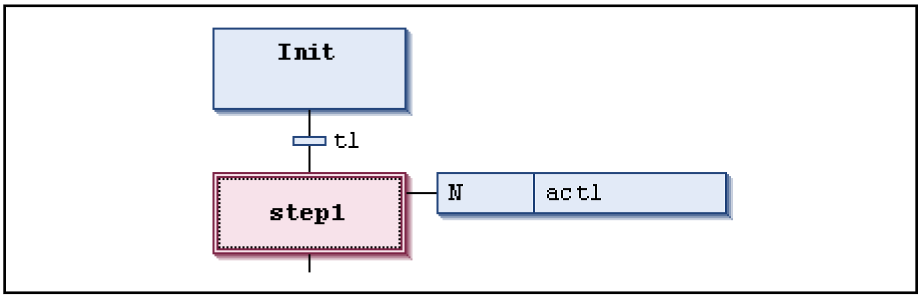


Fig.4-124: Selected step element

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 Cursor positions in SFC, page 401.

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

Adding elements

The individual SFC elements, page 405, 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.

Selecting elements

An element or text area can be selected by clicking on a possible cursor position. Use the arrow keys to switch to the adjacent element. The selected element changes its color to red. (Examples, page 401).

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

Click on a text cursor position, page 401, 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 <Del> key.

- 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 Element properties, page 108, 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 SFC editor options dialog, page 219, 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	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. 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: ● **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. By default, nothing is entered (empty field in the selection list). That means that no symbol is exported to the symbol configuration.

Fig.4-125: General properties

Initial step	This option is only enabled for the step currently defined as initial step. 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. Also note the Initial step, page 279 command in this context.
Times:	Timeouts in steps can be monitored using the implicit variable "SFCError".
● Minimum active	Minimum time period that the step should be active. Possible values: Time specifications based on IEC syntax (e.g. t#8s) or Variable of type TIME with initial value: t#0s.

● Maximum active	Maximum time period that the step should be active. Possible values: Time specifications based on IEC syntax (e.g. t#8s) or Variable of type TIME with initial value: t#0s.
Actions:	Define the actions to be executed when the step is active. Note the processing sequence, page 408.
● Step entry	This action is executed after the step was enabled ("input action").
● Step active	This action is executed when the step is active and any possible input actions have already been processed.
● Step exit	If the step is disabled, this action is executed in the following cycle ("output action").

Fig.4-126: Special features

Refer to the status of a step or an action from the respective implicit SFC variables, page 414 for more information.

Furthermore, you also obtain information on timeouts from SFC flags, page 414.

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 SFC menu, page 278, when the editor is active.

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

In addition, refer to the help page for working in the SFC editor.

The following elements are available and described below:

- Step, page 405,
- Transition, page 406,
- Action, page 408,
- Branching (alternative), page 412,
- Branching (Parallel), page 411,
- Jump, page 412,
- Macro, page 413.

Step, initial step and subsequent transition

Icons: ⊤↑⊤↓

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

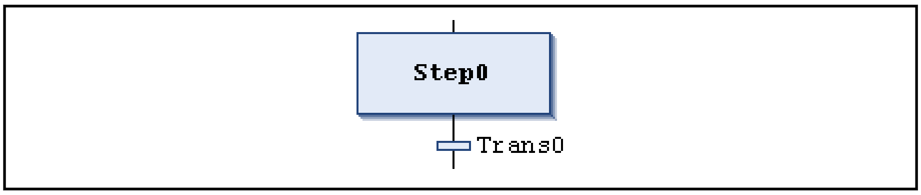


Fig.4-127: Step and subsequent transition:

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

Note that a step can become an initial step using the Initial step, page 279, command or by setting the corresponding step property.

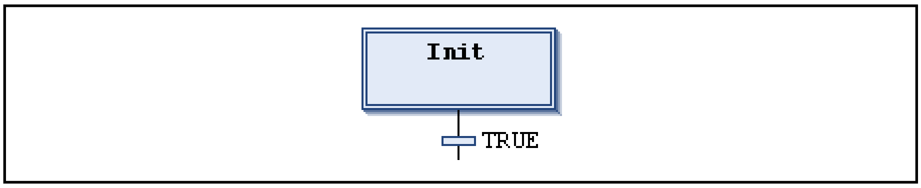


Fig.4-128: Initial step and subsequent transition:

The step name can also be edited inline.

☞ Step names have to be unique within the valid range of the "father" function block.

Especially if actions are used, remember that they have to be programmed in SFC as well.

Each step is defined by Step Properties, page 404.

The actions to be executed when the step is active are added to the step using the "Add action association" command or in its properties (see below, Action, page 408).

First, steps and transitions are always added as a combination using the Insert step transition (after), page 281.

Transition

An SFC transition is represented by a small rectangle connected with the previous or following step via a line.

The transition contains the conditions under which the following step becomes active - as soon is TRUE is returned.

The transition name or transition condition is displayed at the right of this symbol.

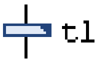
By default, the transition name **trans<n>** is specified when it is inserted, where "n" is a consecutive number.

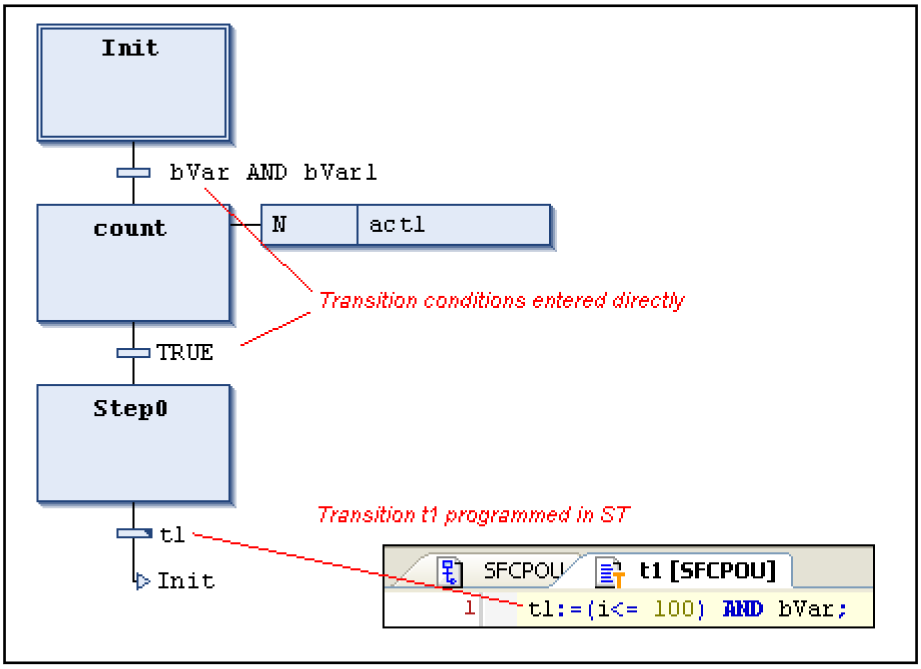
This default name can be modified after selection. The following entries can be made:

- Name of a transition object present in the POU window (see example below, "t1". This allows the multiple usage of transitions)
- Valid expression (e.g. "a AND b", "TRUE")

A transition condition has to have either the value TRUE or FALSE.

☞ Transitions consisting of a "Transition" or "Properties" object are labeled with a **small triangle** in the right upper corner of the transition rectangle.





Transition conditions entered directly

Transition t1 programmed in ST

```
t1:=(i<= 100) AND bVar;
```

Fig.4-129: Examples of transitions

This way, the transition condition can consist of a Boolean variable, address or constant or an expression with a Boolean result.

☞ A transition may not contain programs, function blocks or assignments.

A transition condition is edited like a method call, page 45,.

Input is made according to:

Syntax:

`<Transition name>:=<Transition condition>`

or only

`<Transition condition>`

See the example (t1) above.

In **online mode** ,the following step is only active if the preceding transition is TRUE.

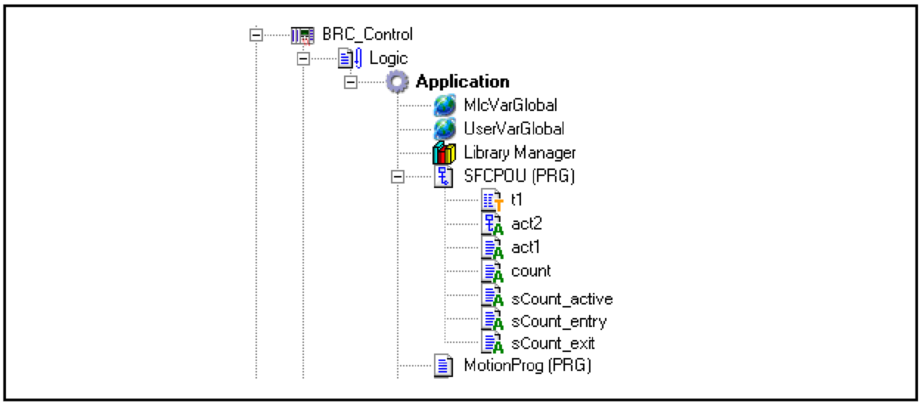


Fig.4-130: Transition t1 in the Project Explorer

Action

Icon:

An action can contain a sequence of instructions written in one of the valid programming languages. It is assigned to a step and executed in online mode according to the defined processing sequence .

All actions used in SFC steps have to be available as objects in the Project Explorer and can be created in the SFC object).

An action can be created below an SFC POU with the Add object, page 234, command.

☞ Step names have to be unique within the valid range of the "father" function block.

An action described in SFC may not contain a step with the same name as the step to which the action is assigned.

The following types of actions exist:

1. **IEC-compliant step actions ("IEC action"):** These actions correspond to the IEC61131-3 standard that are executed according to its qualifier when the step is enabled and a second time when it is disabled. If several actions are assigned to one, the action list is processed from top to bottom.

   Each action symbol contains the qualifier in the first column and the action name in the second.

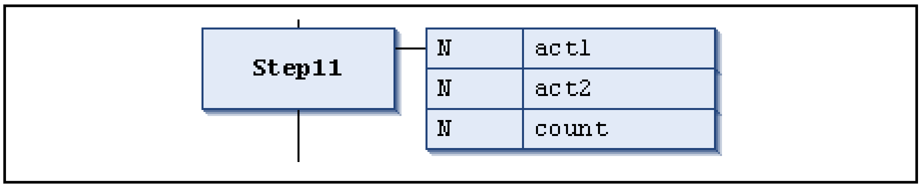
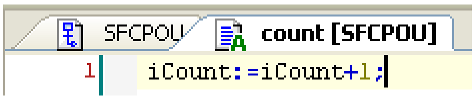


Fig.4-131: IEC-compliant step with action list

- In contrast to "simple" step actions, a variety of qualifiers, page 414 can be used for IEC actions.
- Another difference to "IEC extending step actions" is that each IEC step action has a control flag that ensures that the action is only executed once at a time - even if it is called at the same time by a different step. This cannot be guaranteed for simple step actions.
- An IEC step action is represented by a two-part symbol at the right of the step and is connected via a line. In the left section, it shows

the action "qualifier" and in the right section, the "action names". Both can be edited directly.

- IEC step actions are assigned to a step using the Insert action association (after), page 283, command. One or multiple actions can be assigned to one step. The position of the newly inserted action depends on the current cursor position and the command. The actions have to be available as objects in the project and have to be specified with a unique action name (e.g. plc_prg.a1).



```
iCount:=iCount+1;
```

Fig.4-132: Example of an action written in ST

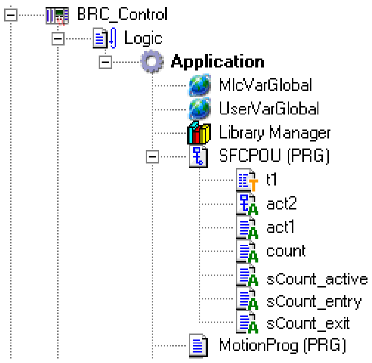


Fig.4-133: IEC actions in the Project Explorer (act1 and count in ST, act2 in SFC)

2. **IEC-extending step actions:**

These are actions that can be used in the extension of the IEC standard:

- **"Step entry" action ("input action")**:

  An input action is executed after the step has been enabled and before the "step action" is executed.

  The action is entered in the property field of its step in "Step enabled". In effect, it is created at the highlighted step with **SFC ▸ Add input action**.

  It is indicated by an "E" in the lower left corner of the step symbol.

- **"Step active" action ("step action")**:

  A step action is executed after the step has been enabled and after any input action, if present, has been executed. However, in contrast to an IEC step action (see above), it is not executed a second time (no postprocessing) if the step is disabled again and is not provided with qualifiers.

  The action is entered in the property field of its step under "Step active". In effect, it is created by double-clicking into the step symbol.

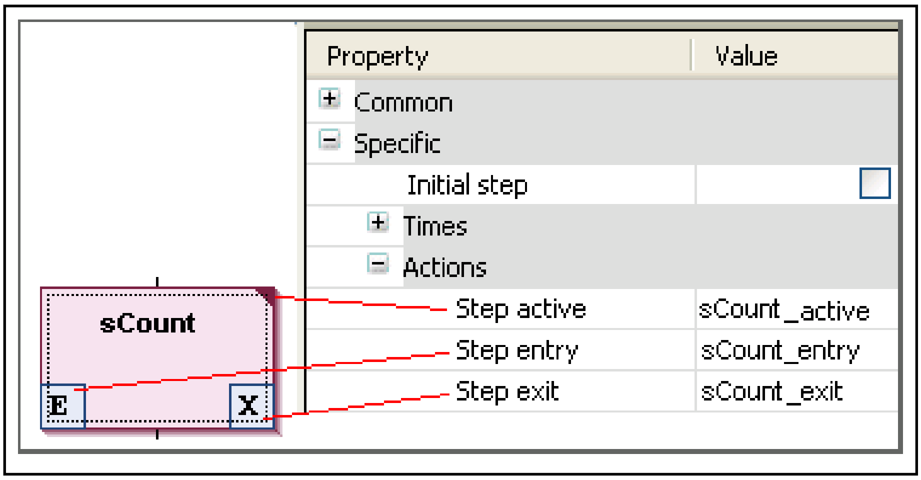
  It is indicated by a red filled triangle in the right upper corner of the step symbol.

- **"Step exit" action (output action)**:

An output action is only executed once before the step is disabled. However, note that the execution does not take place in the same cycle, but at the beginning of the next one.

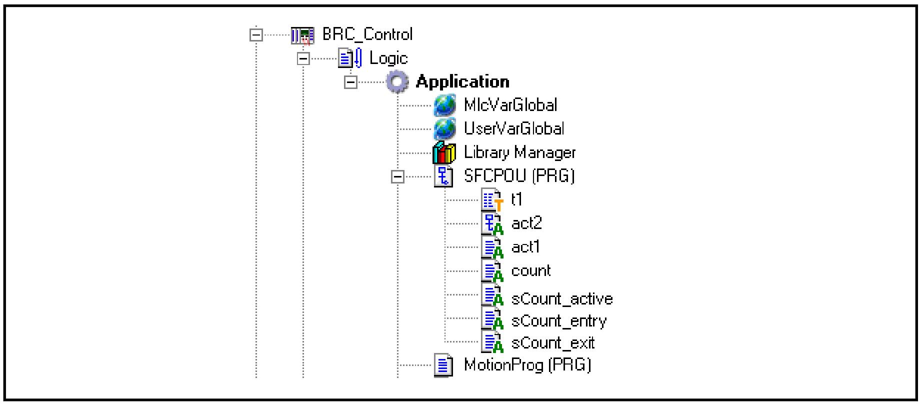
The action is entered in the property field of its step under "Step exit". In effect, it is created at the highlighted step via **SFC ▸ Add output action**.

It is indicated by an "X" in the lower left corner of the step symbol.



Property	Value
⊞ Common	
⊟ Specific	
Initial step	□
⊞ Times	
⊟ Actions	
Step active	sCount_active
Step entry	sCount_entry
Step exit	sCount_exit

Fig.4-134: IEC-extending step actions



```
⊟ BRC_Control
   ⊟ Logic
      ⊟ Application
            MlcVarGlobal
            UserVarGlobal
            Library Manager
         ⊟ SFCPOU (PRG)
               t1
               act2
               act1
               count
               sCount_active
               sCount_entry
               sCount_exit
            MotionProg (PRG)
```

Fig.4-135: Step actions in the Project Explorer (sCount_active, sCount_entry, sCount_exit)

Example:

Difference: IEC-compliant and extending step actions

The essential difference between step actions and IEC actions with the qualifier N consists of the IEC action being executed twice. The first time when the step is active and second time when it is disabled. This is explained in detail using the following example:

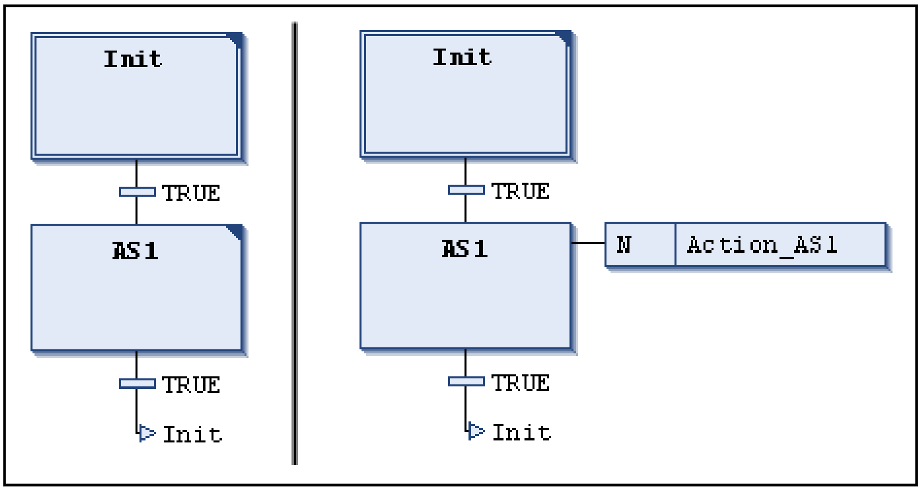


Fig.4-136: Difference: IEC-compliant and extending step actions

The action Action_AS1 is added to the step AS1 once as step action (left) and once as IEC action with the qualifier N. Since two transitions have to be switched in both the cases, it takes two PLC cycles each until the initialization step is reached. It is assumed that a counter variable "iCounter" is initialized with 0 and incremented in the action Action_AS1. Once the Init step has been activated again, iCounter has a value of 1 in the left example. It has a value of 2 in the right example, since the IEC action is executed a second time due to the deactivation of AS1.

Branching

An SFC diagram can contain branches, i.e. the execution line can split into two or more arms. Parallel branches are executed simultaneously. For alternative branches, only one arm is executed depending on the preceding transition condition.

Parallel branching

Icons:

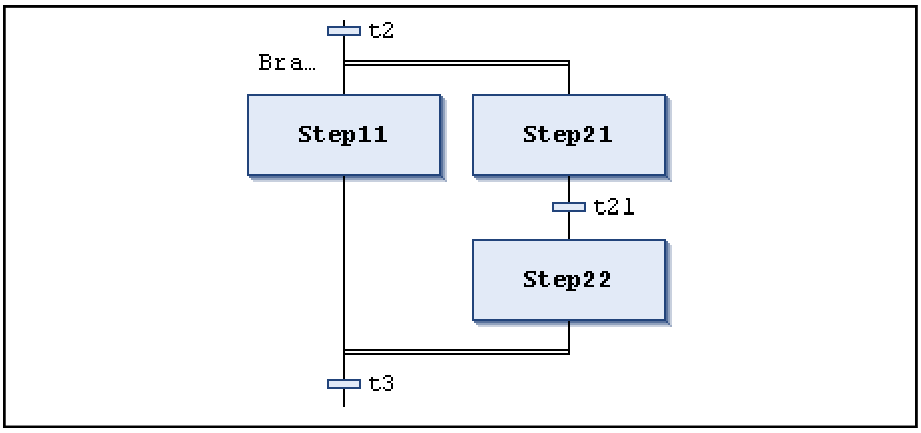


Fig.4-137: Parallel branching

The horizontal line in front of and behind the branch is a double line.

In a "parallel branch" the arms have to start and end with steps. Parallel arms can contain other branches.

Processing in "online mode": If the preceding transition (t2 in the example below) is TRUE, the first steps in all parallel arms become active (Step11 and Step21). The individual arms are processed in parallel and only then is the subsequent transition evaluated (t3).

A parallel branch is inserted with the Insert branch (right), page 282, command if a step is currently selected.

Note that parallel and alternative branches can be transferred into each other using the Parallel, page 281, or Alternative, page 282 commands. This can be useful when structuring a program.

A jump label is automatically added to the horizontal line that forms the beginning of a branch. The label is called "Branch<n>", where n is a consecutive number starting with "0". This label can be specified as jump target, page 412,.

Alternative branch

Icons:

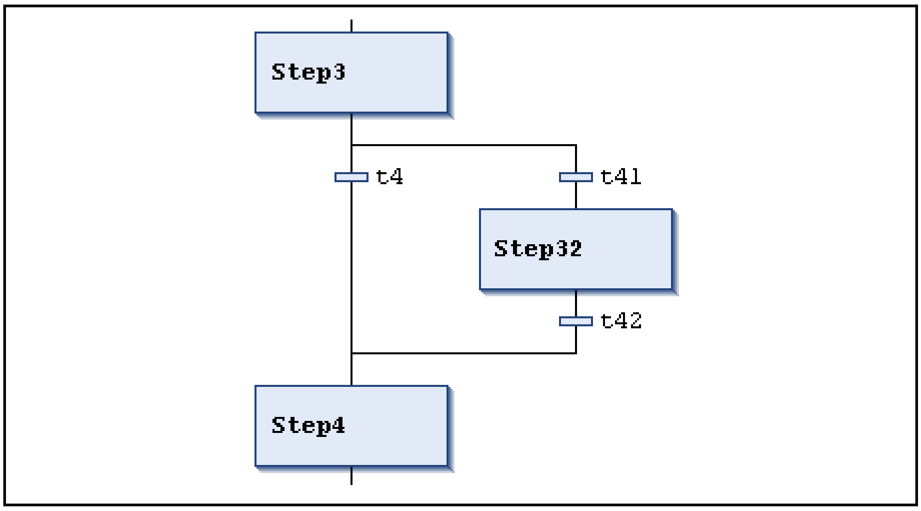


Fig.4-138: Alternative branch

The horizontal lines in front of and behind the branch are single lines.

In an alternative branch, the arms have to start and end with transitions. The arms can contain other branches.

If the step that precedes the branch is active, the first transition of each of the alternative arms is evaluated from left to right. If the first transition returns TRUE, the corresponding arm is "opened", i.e. the step following the transition is enabled.

Alternative branches are inserted with the Insert branch (right), page 282, command if a transition is currently selected.

Note that parallel and alternative branches can be transferred into each other using the Parallel, page 281, or Alternative, page 282 commands. This can be useful when structuring a program.

Jump

Icons:

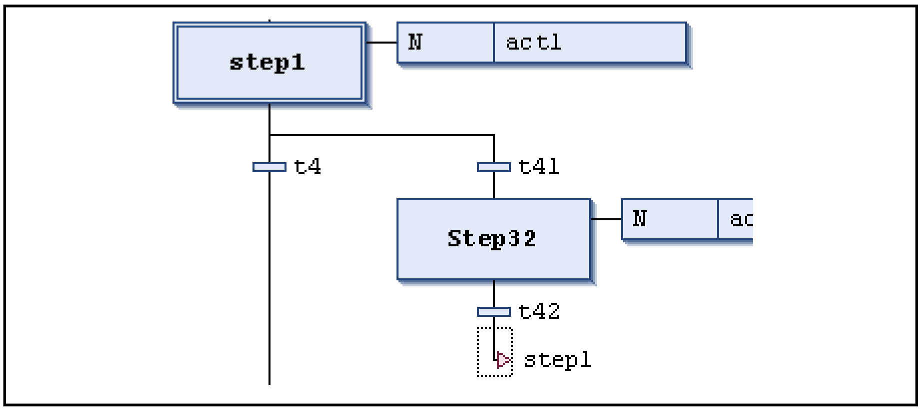


Fig.4-139: Jump

A "jump" is displayed as a vertical connection line with a horizontal arrow and the name of the jump target.

A jump defines which step is to be executed next when the transition preceding the jump becomes TRUE. Jumps can be necessary, since the execution lines cannot cross each other or lead upwards.

In addition to the obligatory jump at the end of the diagram, jumps can only be inserted at the end of branches, page 411, (Insert jump (after), page 284 if the last transition of an arm is currently selected.

The **target** of a jump is defined by the inserted character string which can be edited directly. It can be a step name or the label of a parallel branch, page 411,.

Macro

Icons: ⊡↑⊡↓

A macro is displayed by a rectangle with a thick frame which contains the macro names.

A macro contains a part of the SFC diagram that is not displayed in detail in the main view of the editor.

The processing flow is not affected by the use of macros. They are only used to hide certain parts of the diagram, e.g. to improve clarity.

A macro symbol is inserted using the "Insert macro (after)" command. The macro name can be edited directly.

To open the **macro editor**, double-click on the macro symbol or use the Display macro, page 285 command.

Program as in the main view of the SFC editor and create the desired part of the diagram. To close the macro editor again, use the Exit macro, page 286 command.

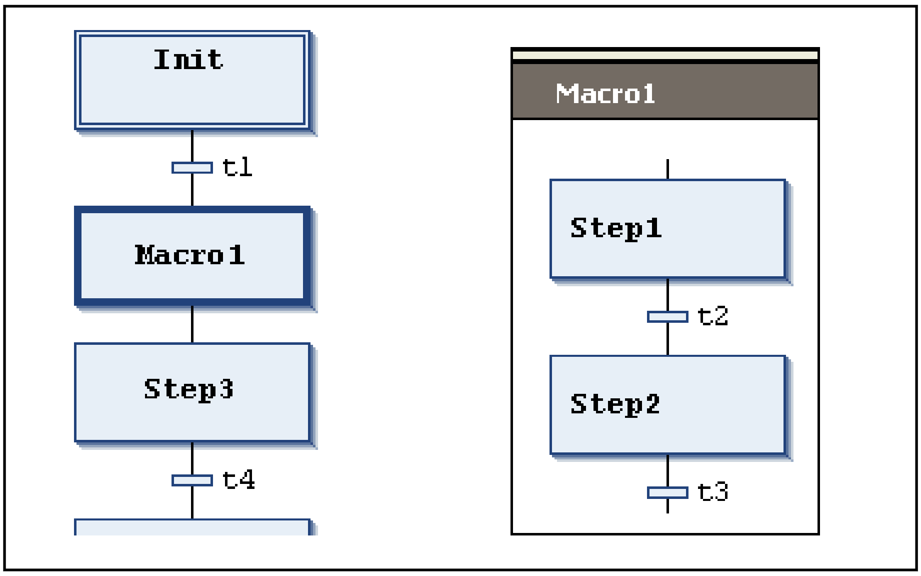


Fig.4-140: Macro1: Main view in the SFC editor and view in the macro editor

Macros can also contain macros. The title line of the macro always shows the path of the macro currently open in the diagram.

Example:

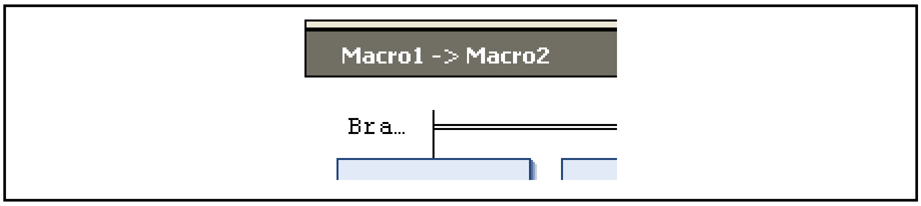


Fig.4-141: Macro in macro

4.13.7 Qualifier for Actions in SFC

To configure how an action, page 408 that is assigned to an IEC step is to be executed, use a variety of qualifiers that can be entered in the "Qualifier" field of an action element.

The qualifiers are processed by the SFCActionControl function block of the **IecSfc.library** which is automatically included in the project.

N	Non-stored	The action is active as long as the step.
R	Overriding Reset	The action is disabled.
S	Set (Store)	The action is executed as soon as the step is entered and continues to be executed even if the step was already disabled until it is reset.
L	Time Limited	The action is executed as soon as the step is enabled and is executed at least during the given time period, but at most only until the step is exited.
D	Time Delayed	The execution of the action begins only if the given delay time is expired after the step is enabled and the step is still active. The action is executed until the step is disabled.
P	Pulse	The action is executed exactly once as soon as the step is entered.
SD	Store and Time Delayed	The execution of the action begins if the given delay time is expired after the step is enabled. It is executed until it is reset.
DS	Delayed and Store	The execution of the action begins only if the given delay time is expired after the step is enabled and the step is still active. The action is then executed until it is reset.
SL	Store and Time Limited	The action is executed as soon as the step is enabled. It is executed until the given time period has expired or it is reset.

Fig.4-142: Qualifier for SFC actions

The time specifications for the qualifiers L, D, SD, DS and SL have to be given in the format of a TIME constant.

☞ If an IEC action is disabled, it is executed a second time. This means that such an action is executed at least twice.

4.13.8 Implicit Variables - SFC Flags

Each SFC object provides implicit variables to check the Status of steps and IEC actions at runtime.

In addition, variables can be defined to affect and control the processing of an SFC diagram (timeouts, reset, jog mode). These variables can also be implicitly generated by an SFC object.

In principle, for each step and each IEC action, an implicit variable is created: A structure instance with the same name as the element, e.g. "step1" for a step with step name "step1". Note that it can be specified in the element properties whether a symbol definition should be exported to the symbol configuration for this flag and how this symbol should be accessed in the control.

The data types for these implicit variables are defined in the **IecSFC.library** library. This library is automatically integrated into the project as soon as an SFC object is created.

Step and action status

For each step and each IEC action, an implicit variable of type **SFCStepType** or **SFCActionType** is created. The structure components ("flags") describe the status of a step or an action or the time period that is currently already expired in an active step.

The syntax for the implicitly executed variable declaration:

<stepname>: SFCSTepType;

or

_<actionname>:SFCActionType;

The following Boolean flags are available for step or action status:

- **For steps:**

  `<StepName>.x` shows the status in the current cycle.

  `<StepName>._x` shows the status for the next cycle.

  If `<StepName>.x = TRUE`, the step is executed in the current cycle.

  If `<StepName>._x = TRUE` and `<Step name>.x = FALSE`, the step is executed in the next cycle, i.e. `<Step name>._x` is copied to `<Step name>.x` at the beginning of a cycle.

- **For actions (implicit variables for actions always written with an underscore):**

  `_<ActionName>.x` is TRUE when the action is executed.

  `_<ActionName>._x` is TRUE when the action is active.

Access to the status flags:

... within an action or transition of the current SFC function block:

- `<StepName>.<flag>` or `_<ActionName>.<flag>`

  Example: "`status:=step1._x;`"

... from another function block:

- `<SFC POU>.<StepName>.<flag>` or

  `<SFC POU>._<ActionName>.<flag>`

  Example: "`status:=SFC_PRG.step1._x;`"

Symbol generation

In the element properties of a step or an action can be defined whether a symbol definition for the symbol application that might be generated and load onto the control is to be added for the step or action name flag. To do this, the desired access rights in the "Symbol" column have to be enabled for the step or action in the "Properties" view.

☞ The flags described above could be used to force a certain status value for a step, i.e. to set a step to active. However, note that this causes an uncontrolled status of the SFC.

Time query using TIME variables

The "t" flag returns the current time period that has elapsed since the step was enabled. This applies only to steps irrespective of whether a minimum time has been defined in the step properties (see below: SFCError)

- **For steps:**

  `<StepName>.<t>` (<StepName>._t cannot be used externally)
- **For actions:**

  The implicit time variables are not used.

Checking the execution of an SFC diagram (timeouts, reset, jog mode)

Implicitly generated variables with specific names (SFC flags) can be used to check the processing of an SFC diagram, e.g. for displaying timeouts or for using the jog mode so that specific switches to transitions can be made.

In order to use these variables, they have to be **declared and enabled**. This **has to be carried out in the SFC settings** dialog, a subdialog of the "Object Properties" dialog.

Manual declaration as in IndraLogic 1.x is only necessary now to allow write-access from a different function block (see below Access to implicit variables, page 414). In this case, observe the following: If the flag is declared globally, the "Declare " settings in the "SFC Settings" dialog has to be disabled in order to not implicitly declare a local flag variable that is then used instead of a global one. Always consider that the SFC settings of the SFC function block are first specified by the current definitions in the SFC Options dialog, page 202,.

☞ Note that the declaration of a flag variable that is only present in the SFC settings dialog is only visible in the online view of the SFC function block.

Example for using SFCError

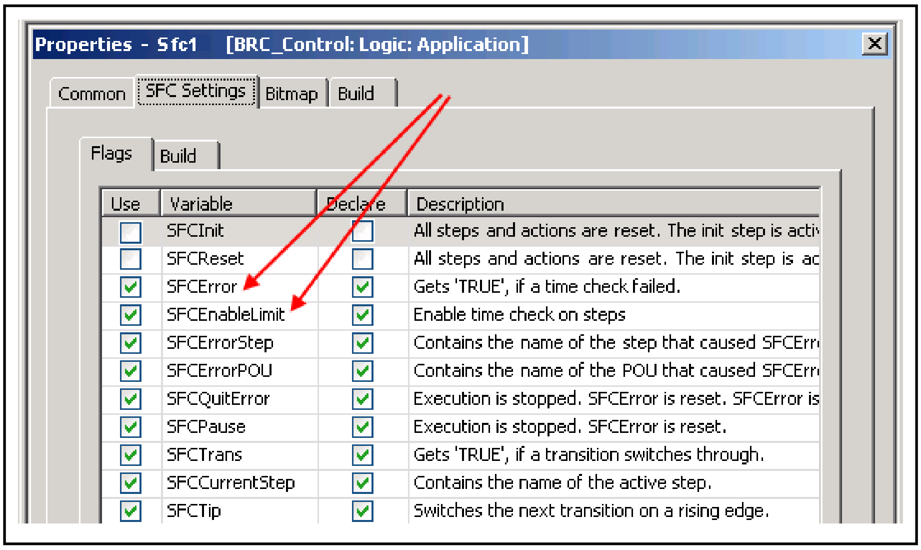
The SFC function block called "Sfc1" contains a step "s2". The time limits are defined in the step attributes for this step. See these below in the figure Online view from SFC function block sfc1, page 418.

If step s2 remains longer active than allowed in its time properties (timeout), an SFC flag should be set which the application can access.

To allow this access, the variables "SFCEnableLimit" and "SFCError" have to be declared in the SFC settings.

To do this, select "Sfc1" in the Project Explorer and select the "Properties" command from the context menu dialog. Open the "SFC Settings" subdialog and place a checkmark in the "Declare" and "Use" columns on the "Variables" tab for the variables "SFCEnableLimit" and "SFCError". (If you only declare them, the variable is displayed in the online view of sfc1 in the declaration, but it has no function.)

SFC settings

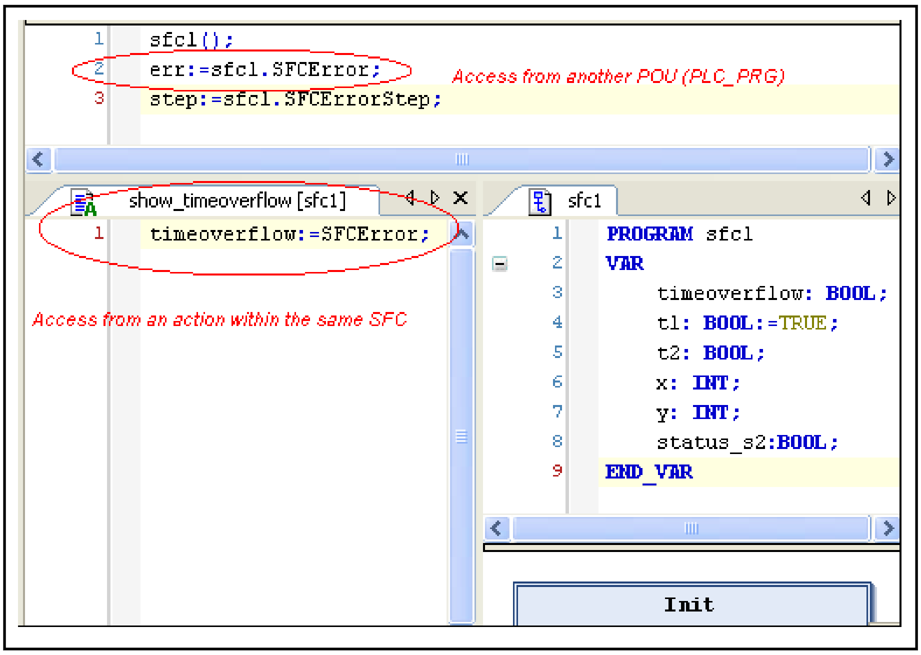


Use	Variable	Declare	Description
□	SFCInit	□	All steps and actions are reset. The init step is acti
□	SFCReset	□	All steps and actions are reset. The init step is ac
☑	SFCError	☑	Gets 'TRUE', if a time check failed.
☑	SFCEnableLimit	☑	Enable time check on steps
☑	SFCErrorStep	☑	Contains the name of the step that caused SFCErr
☑	SFCErrorPOU	☑	Contains the name of the POU that caused SFCErr
☑	SFCQuitError	☑	Execution is stopped. SFCError is reset. SFCError is
☑	SFCPause	☑	Execution is stopped. SFCError is reset.
☑	SFCTrans	☑	Gets 'TRUE', if a transition switches through.
☑	SFCCurrentStep	☑	Contains the name of the active step.
☑	SFCTip	☑	Switches the next transition on a rising edge.

Fig.4-143: SFC settings

Now, address SFCError within SFC e.g. in an action with "SFCError" or from outside the function block with "SFCPOU.SFCError".

Access to SFCError



```
sfc1();
err:=sfc1.SFCError;
step:=sfc1.SFCErrorStep;
```

Access from another POU (PLC_PRG)

```
timeoverflow:=SFCError;
```

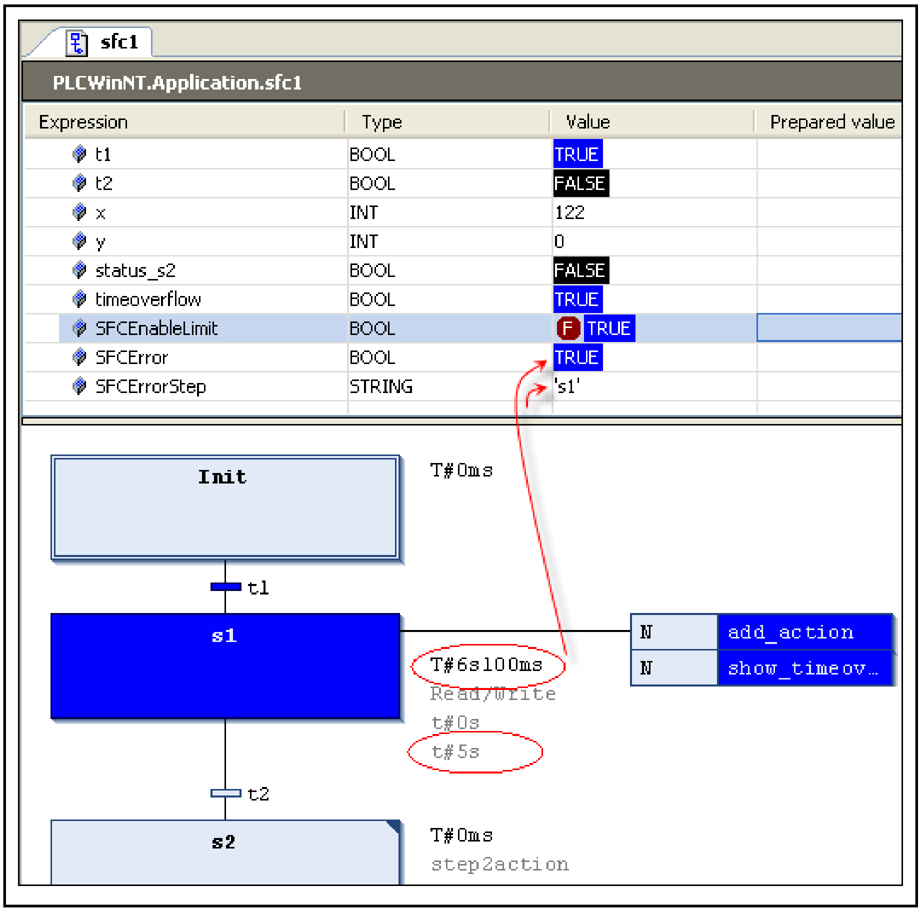
Access from an action within the same SFC

```
PROGRAM sfc1
VAR
    timeoverflow: BOOL;
    t1: BOOL:=TRUE;
    t2: BOOL;
    x: INT;
    y: INT;
    status_s2:BOOL;
END_VAR
```

Fig.4-144: Access to SFCError

SFCError becomes TRUE as soon as a timeout occurs within sfc1.

Online view of the SFC function block sfc1



sfc1

PLCWinNT.Application.sfc1

Expression	Type	Value	Prepared value
t1	BOOL	TRUE	
t2	BOOL	FALSE	
x	INT	122	
y	INT	0	
status_s2	BOOL	FALSE	
timeoverflow	BOOL	TRUE	
SFCEnableLimit	BOOL	F TRUE	
SFCError	BOOL	TRUE	
SFCErrorStep	STRING	's1'	

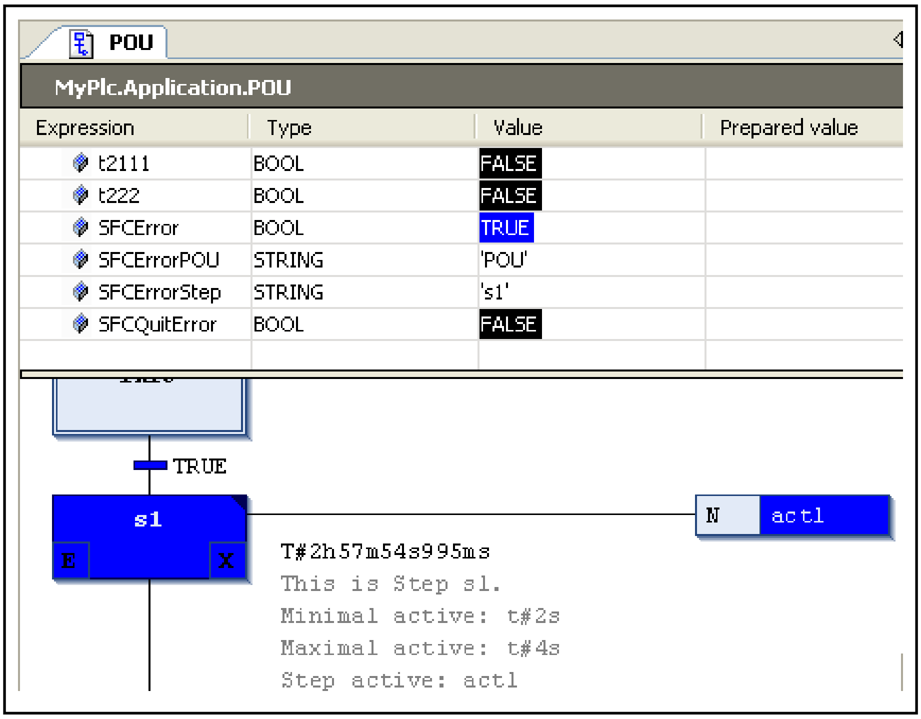
Fig.4-145: Online view of the SFC function block sfc1

The following flag variables can be used. For this purpose, they have to be declared and enabled in the SFC settings, page 202,:

SFCInit: BOOL;	If this variable is TRUE, the sequence is reset to the initial step, page 405,. The other SFC flags are also reset (initialized). As long as the variable is TRUE, the initial step remains set (active), but is not executed. Only when SFCInit is set to FALSE again, the function block continue to be processed normally.
SFCReset: BOOL;	This variable acts similar to SFCInit. In contrast to that variable,processing continues after the initialization of the initial step. This way, the SFCReset flag can be set right back to FALSE again in the Init step, for example.
SFCError: BOOL;	This variable becomes TRUE if a timeout has occurred in an SFC diagram. If another timeout occurs in the program after the first one, the second one is no longer registered if the variable SFCError was not reset before. The declaration of SFCError is required for the functioning of the other flag variables to check the time sequence (SFCErrorStep, SFCErrorPOU, SFCQuitError).
SFCEnableLimit: BOOL;	This variable is used for switching the check for timeouts in the steps performed by SFCError either on (TRUE) or off (FALSE). This means that if this variable is declared and active (SFC settings), it has to be set to TRUE for SFCError to run. Otherwise, timeouts are ignored. This can be useful during commissioning or manual operation for example. If the variable is not declared, SFCError works automatically. The prerequisite is that SFCError is declared.
SFCErrorStep: STRING;	This variable stores the name of the step that caused a timeout registered by SFCError. The prerequisite is that SFCError is declared.

SFCErrorPOU: STRING;	In case of a timeout, this variable stores the name of the function block in which a timeout occurred and was registered by SFCError. The prerequisite is that SFCError is declared.
SFCQuitError: BOOL;	As long as this Boolean variable is TRUE, the processing of the SFC diagram is stopped and any timeout in the SFCError variable is reset. If the variable is set to FALSE again, all of the previous time periods in the active steps are reset. The prerequisite is the declaration of the SFCError flag which registers the timeout.
SFCPause: BOOL;	As long as this variable is TRUE, the processing of the SFC diagrams is stopped.
SFCTrans: BOOL;	This variable becomes TRUE if a transition becomes active.
SFCCurrentStep: STRING;	This variable displays the name of the currently active step irrespective of the time monitoring. In case of a parallel branch, the name of the step of the outermost branch at the right is always saved.
SFCTip, SFCTipMode: BOOL;	These variables allow the "jog mode" in an SFC function block. If it is activated by `SFCTipMode=TRUE` SFCTip has to be set to TRUE to switch to the next step. As long as SFCTipMode is set to FALSE, transitions can also be used to switch to the next step.

Fig.4-146: Flag variables



MyPlc.Application.POU

Expression	Type	Value	Prepared value
t2111	BOOL	FALSE	
t222	BOOL	FALSE	
SFCError	BOOL	TRUE	
SFCErrorPOU	STRING	'POU'	
SFCErrorStep	STRING	's1'	
SFCQuitError	BOOL	FALSE	

T#2h57m54s995ms
This is Step sl.
Minimal active: t#2s
Maximal active: t#4s
Step active: actl

Fig.4-147: Example of an SFC flag in case of a timeout in the online mode of the editor

Access to the flags: To access the flags to check the SFC processing (timeout, reset, jog mode), it has to be written on the flag variables. They have to be declared and enabled as described above (checking execution of an SFC diagram). Syntax for the access: ... from an action or transition in the current SFC function block:

<Step name>.<Flag> or _<Action name>.<Flag>.

```
status:=step1._x;
```

```
checkerror:=SFCerror;
```

... from another function block:

<SFC POU><Step name>.<Flag>

or

<SFC POU>_<Action name>.<Flag>.

```
status:=SFC_prog.step1._x;
```

```
checkerror:=SFC_prog.SFCerror;
```

Please note: If there is **write-protection** from another function block, the implicit variable has to be additionally and explicitly declared as VAR_INPUT variable in the SFC function block or globally in a GVL for example.

Local declaration:

```
PROGRAM SFC_prog
VAR_INPUT
 SFCinit:BOOL;
END_VAR
```

or global declaration in a global variable list:

```
VAR_GLOBAL
 SFCinit:BOOL;
END_VAR
```

Access to the flag in PLC_PRG:

```
PROGRAM PLC_PRG
VAR
 setinit: BOOL;
END_VAR
       //Write access to SFCinit in SFC_prog
SFC_prog.SFCinit:=setinit;
```

4.13.9 Processing Sequence in SFC

In online mode, the various action types are executed in a defined sequence, see the table below.

The following terms are used:

- **Active step**: An active step is a step in which the actions are currently executed. In online mode, the symbols for active steps are shown filled in blue.
- **Initial step**: In the first cycle after an SFC-POU is called, the initial step automatically becomes active and the step action, page 408, is executed.
- **IEC actions**: IEC actions are executed at least twice: The first time after the step has been enabled, the second time - but only in the next cycle **after the step has been disabled** (postprocessing).
- **Alternative branches**: If the step that precedes the branch (before the horizontal line) is active, the first transition of each of the alternative arms is evaluated from left to right. In the first branch in which a transition is found that returns TRUE, is next step then enabled.
- **Parallel branches**: If the step that precedes the branch (before the horizontal double line) is active and the transition that precedes the branch returns TRUE, the first steps are enabled in all branches. Then, the branches are processed simultaneously. The step following the end of the branch (after the horizontal double line) is enabled if all the final steps in the branches are active and the transition after the double line returns TRUE.

The individual steps of the processing correspond with those in IndraLogic 1.x:

1. Resetting IEC actions	The check flags for the IEC actions, page 408, are reset (not the flags for actions that are called within actions).
2. Executing output actions (step exit)	All steps are checked in the order in which they are arranged in the SFC diagram (from top to bottom and from left to right) to see if they fulfill the condition for the execution of the step exit action (output action, page 408,) and - if this is the case - they are executed. An output action is executed when the step is exited, i.e. if its input and step actions (if present) were executed in the preceding cycle and the condition for the following step returns TRUE.
3. Executing input actions (step entry)	All steps are checked in the order in which they are arranged in the SFC diagram (from top to bottom and from left to right) to see if they fulfill the condition for the execution of the step entry action (input action, page 408,. If this is the case, they are executed. An input action (step entry) is executed if the processing of the preceding transition returns TRUE.
4. Executing time check, step actions ("step active")	The following is checked for all steps in the order in which they are arranged in the SFC diagram (from top to bottom and from left to right): ● **(### in preparation###)** The time elapsed for the active step is copied to the corresponding implicit step variable <StepName>.t, page 414,. ● **(### in preparation###)** If a timeout has been determined, the corresponding error flags, page 414, are used. ● For steps outside the IEC standard, the step action ("Step active") is now executed.
5. Executing IEC actions	The IEC actions, page 408, are executed in alphabetical order. This is carried out in two cycles by the list of actions. 1. In the first cycle, the IEC actions for all steps disabled in the preceding cycle are executed (postprocessing). 2. In the second cycle, the IEC actions for all active steps are executed.
6. Transition check, enabling the following steps	The transitions, page 405, are evaluated: If a step in the current cycle is active and the following transition returns TRUE (and any minimum time that might have been defined for the step has expired), the following step is **enabled**.

Fig.4-148: Processing sequence in SFC

☞ Note the following when **implementing actions**:

It can happen that an action is executed multiple times within the same cycle, since it is used in several SFC diagrams.

Example: an SFC contains two IEC actions A and B, both programmed in the SFC language and both calling an IEC action C. In this case, C would be called twice. If the same IEC action is used simultaneously on different levels of an SFC diagram, this can cause unpredictable effects during processing. For this reason, a corresponding error message is output.

☞ Note that implicit variables, page 414, can be used to monitor the processing status of steps and actions or to check the processing.

4.13.10 SFC Editor in Online Mode

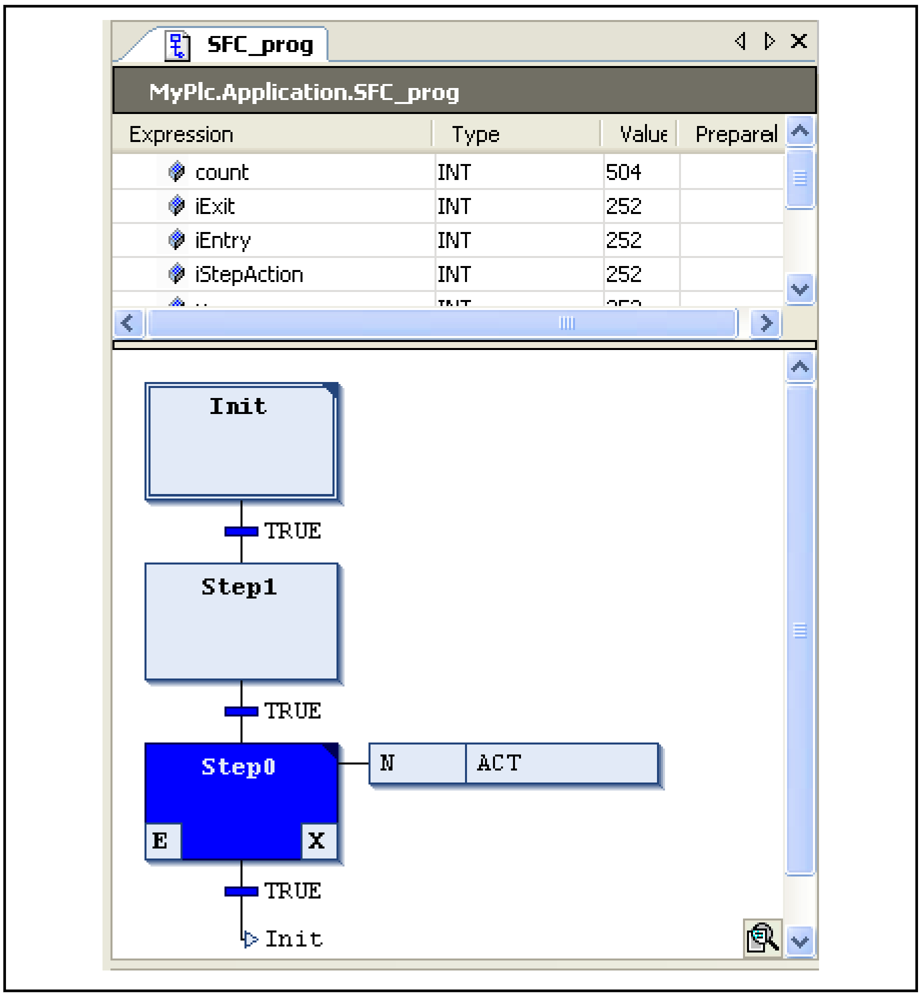
In online mode, the SFC editor allows the displaying (monitoring) and the writing, page 144, and forcing, page 145, of the variables and expressions on the control used in the SFC diagram. debugging functions, page 82, (breakpoints, step-by-step processing, etc.) are not yet available. See below.

- Open the POU in online mode.

- Note that the editor window for an SFC object also contains the declaration editor, page 330 in the upper section. If implicit variables (SFC flags) were declared using the SFC settings dialog, they are inserted here, but they cannot be seen in the declaration part in offline mode.
- Please also note the processing sequence, page 420, for the elements in an SFC diagram.
- Settings with regard to compilation and online display of the SFC elements and attributes can be made in the dialogs of the Object Properties, page 108, or SFC Editor Options, page 219, and SFC Object Options, page 202, .
- The processing of an SFC diagram can be monitored and checked using implicit variables (SFC flags, page 414,).

Monitoring

Active steps are shown filled in blue. The display of the step attributes in the SFC diagram depends on the current settings in the SFC Editor Options, page 219,.



SFC_prog

MyPlc.Application.SFC_prog

Expression	Type	Value	Preparel
count	INT	504	
iExit	INT	252	
iEntry	INT	252	
iStepAction	INT	252	

Fig.4-149: Example: Online view of a program SFC_prog

Breakpoint positions in the SFC editor:

The possible breakpoint positions are always the positions of a POU where the variable values can change or where the program sequence is branched or another POU is called.

For possible positions, refer to the following figure: Fig.: Breakpoint positions in the SFC editor **(### in preparation ###)**

☞ For breakpoints in methods applies:

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

- If an interface method is called, breakpoints are set in all methods of the function block implementing the interface and in all derived function blocks supporting this method.
- If a method is called via 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 supporting this method.

4.14 Task Management

4.14.1 Task Management, General Information

The control can execute several tasks simultaneously. A "job" is called a task. For instance, it is possible to control an axis by a task while another task is performing complex calculations. A PLC task can process one or multiple programs. These are then processed successively by the task in the order of their definition. The PLC programs of a task can be located below the application node of the control as well as below the "General module".

Task types

Tasks differ in their type of control

- **Cyclic:** The task is processed cyclically in a fixedly defined time interval (e.g. every 10 ms). This is the usual case for PLC tasks.
- **Free running:** The task runs permanently. Once the attached PLC program has ended, it is immediately restarted. Such a task runs at lowest priority. It is interrupted by all other tasks.
- **Event-controlled:** The task is always started when a specific Boolean PLC goes from FALSE to TRUE (rising edge).
- **Externally event-controlled:** The task is always started when a specific event task occurs (e.g. point in time within a field bus cycle).

Interval

Cyclic tasks are started at a defined interval. The runtime of the tasks should be smaller than the set interval. If the task has not yet been completed after the interval, it is only started again at the next interval cycle to continue the processing. If this case should be avoided due to technical reasons, monitoring of the task runtime is possible with a watchdog, page 425,.

☞ In order not to unnecessarily increase the CPU load, the cycle times should be selected to fulfill the application requirements but without unnecessary waiting times.

Events

An event is either a changing PLC variable (for event-controlled tasks) or an external event (for externally event-controlled tasks).

The following external events can occur in an IndraMotion MLC / IndraLogic XLC system:

- **EVENT_OP_MODE_MOTION_CYCLIC:** This event is triggered at each Motion cycle.

  It is used to synchronize a PLC program with the calculation of axis motions. It can be used for synchronization with SERCOS I/Os.

  For controls with a SERCOS interface, the "MotionTask" created by default uses this event (for IndraLogic XLC controls only for variants with a SERCOS bus).
- **EVENT_LOCAL_INPUT_BITX:** This event is triggered by the onboard inputs of the control.

Every input is mapped on Bitx according to its number. A rising edge at the input triggers the respective event. This event is only available on the IndraMotion MLC / IndraLogic L65 and L45 controls (controls with onboard inputs and outputs).

Task priority

Only one task can run as only one processor is available on the control.

Task priority decides about the processing order if several tasks are ready for processing. A low number in the task priority means that this task has a high priority and is thus preferred.

If a task with a higher priority is ready for processing, any other low-priority tasks running are interrupted until the task with the higher priority is completed.

The following figure shows the processing of tasks with different priorities:

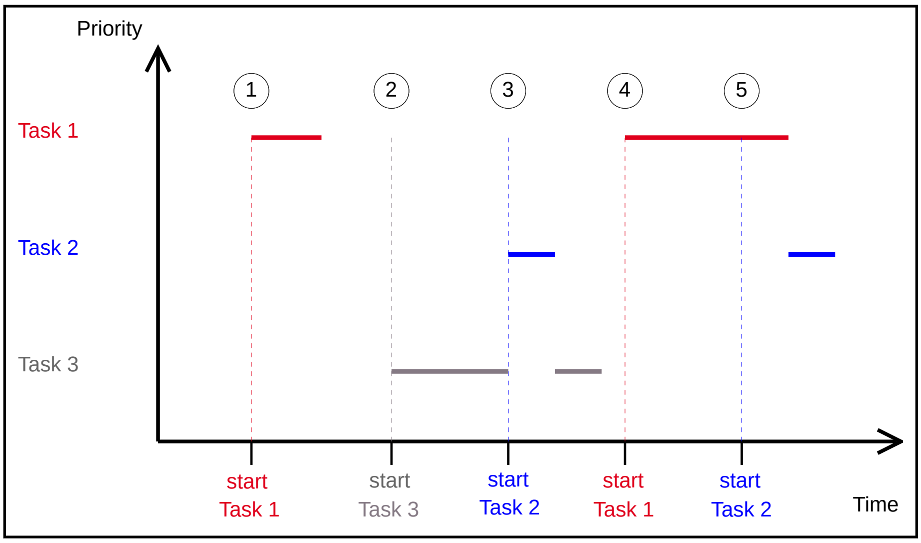


Fig.4-150: Processing different task priorities

The horizontal lines in the figure above represent the times at which the individual tasks are running.

① Task 1 is started and starts to run immediately as it is the only task at this point ready for processing.

② Task 3 is started and starts to run immediately as no other task is running.

③ If task 2 is started, task 3 interrupts as task 2 has a higher priority.

Subsequently, task 3 runs until completion.

④ Task 1 is started and starts to run as it is the only executable task at this point.

⑤ Task 2 is started and cannot start to run as the higher priority task 1 is still running. Once task 1 is completed, task 2 can run.

☞ It should also be ensured that PLC tasks are adjusted to this application. The priority of the different tasks should be selected according to their processing relevance. This prevents that tasks whose job is irrelevant with regard to time block important tasks that need to be processed in specified cycles.

If several competing tasks have the same priority, the so-called round-robin method is applied. A running task is interrupted after 1 ms and the next task with the same priority becomes active. This takes place for all tasks for all tasks with the same priority. This method is applied providing no higher-priority task is started or the starting condition of such a task is met.

The following figure shows the processing of several tasks with identical priority:

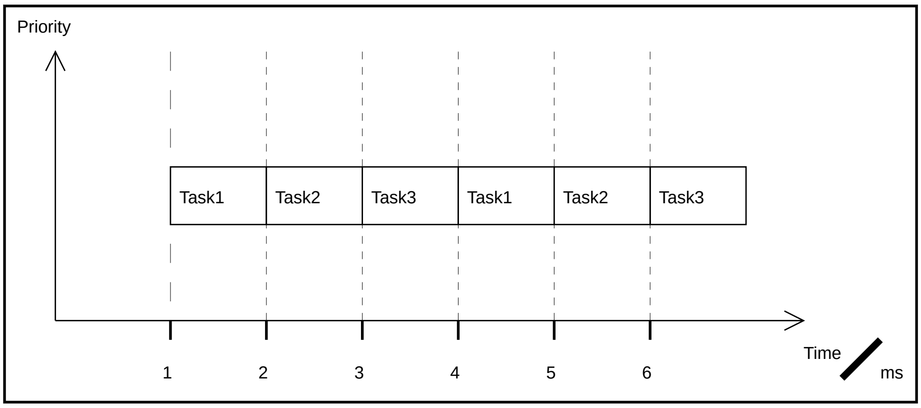


Fig.4-151: Round-robin method

Tasks 1, 2 and 3 have the same priority. If a tasks needs more than one millisecond, it is interrupted and the next task is started. This is done until all tasks have been processed.

☞ Assigning the same priorities is only required for a few applications. It is difficult to predict the behavior of the system and should thus be avoided.

For tasks with the same priority and the same starting condition, programs should be called in one task if possible.

Watchdog

The watchdog is used to monitor the computing time of a task. It reacts if a task runs longer than defined in the watchdog. Triggering the watchdog always leads to a PLC exception error and thus to the stop of the corresponding task. The watchdog is to be set in such a way that the job of the tasks is still completed on time. If the value is too small, the system stops unnecessarily. If the value is too high, the watchdog only starts at extreme events such as endless loops in the PLC program.

☞ The watchdog should always be activated and set to the respective time for time-critical tasks.

Alternating memory access

Consistency problems can also occur in multi-tasking systems when other global objects (global variables, function blocks) are accessed by several tasks if the objects exceed the data width of the processor (structures or arrays that form a logical unit). Access to resources has thus to be protected and blocked for simultaneous access by other tasks. The functions and function blocks of the PLC library system are available as a solution.

4.14.2 Task Configuration

Task Configuration, General Information

Task configuration defines one or multiple tasks to check and execute the application program on the control.

the task configuration is automatically created when adding the control.

If the task configuration was deleted by mistake, i can be added again to the application via the context menu under **Add ▸ PLC Object ▸ Task Configuration** or by dragging it from the library under **PLC Objects ▸ General Objects**.

The uppermost node of a task configuration is always called task configuration. Defined tasks are attached below. Each is represented by its name.

The program calls of the individual tasks are not shown in the tree.

Certain types of objects, such as a visualization object create their individual task automatically (VISU_Task).

In the task configuration, the usual commands to work in the Project Explorer can be used to add, copy, cut or delete tasks.

To add a new task using the context menu, the **Add ▸ PLC object ▸ Task** command is used.

Alternatively, add a task by dragging it from the library **PLC Objects ▸ General Objects.**

The individual tasks are configured in the task editor, page 428, which also provides an online view. The configuration options depend on the target system.

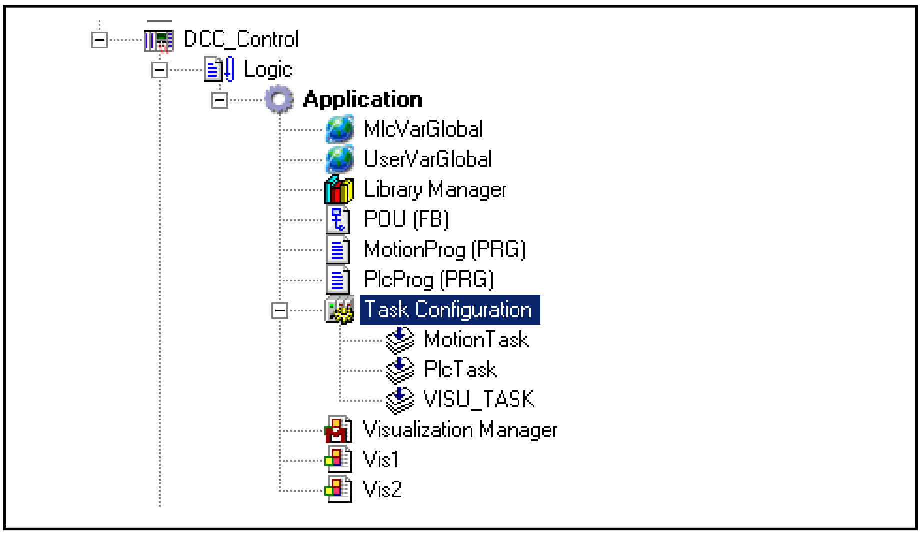


Fig.4-152: Task configuration below an application in the Project Explorer

⚠ WARNING **Inappropriate modification of tasks that were preset by the respective system.**

In order to run, individual systems need the preset values that they bring with them, e.g. IndraMotion systems requires a high(est) priority MotionTask triggered by an external event, EVENT_OP_MOTION_CYCLIC.

Please refer to the information in the respective system descriptions.

In online mode, the "task editor" provides a monitoring view with information on cycles, cycle times and task status.

Default tasks

Two PLC tasks are set up automatically when creating the control:

- **PlcTask:** This task is called cyclically every 10 ms. Time-critical PLC program processes cannot be implemented in this task.

The cycle time and priority of this task can be adapted to the needs of the PLC program.

- **MotionTask:** The MotionTask is coupled to the calculation of the axis motion. It has very high priority and is therefore only to be used for time-critical tasks that require synchronization with axis motions.

  The Motion task provides dead time-optimum behavior for axis-synchronous processes, that is all dead times related to axis motions are of a minimum.

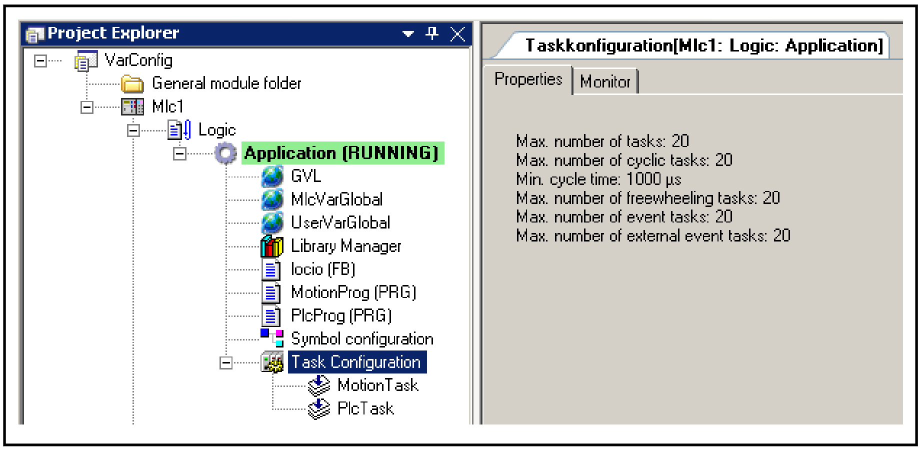
  A task runtime that is too high can lead to the error "Cycle time exceedance"

  The settings of this task are not to be changed.

Other PLC tasks can be created any time if required.

Task configuration, "Properties" Tab

When the uppermost node is selected in the tree of the task configuration, page 428,, the **Properties** dialog opens in the task editor 428, window.



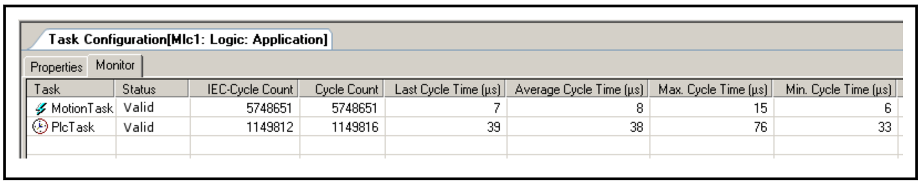
Max. number of tasks: 20
Max. number of cyclic tasks: 20
Min. cycle time: 1000 µs
Max. number of freewheeling tasks: 20
Max. number of event tasks: 20
Max. number of external event tasks: 20

Fig.4-153: Task configuration, "Properties" dialog, IndraMotion MLC L65

The general settings of the task configuration are displayed as specified by the target system. For example, the maximum possible number of tasks per task type.

Task Configuration, "Monitoring" Tab

In online mode, the tab displays the status and some current measurements on cycles and cycle times for the tasks in the task configuration in a table view. The values are updated in the same time interval as when monitoring values from the control.



Task	Status	IEC-Cycle Count	Cycle Count	Last Cycle Time (µs)	Average Cycle Time (µs)	Max. Cycle Time (µs)	Min. Cycle Time (µs)
MotionTask	Valid	5748651	5748651	7	8	15	6
PlcTask	Valid	1149812	1149816	39	38	76	33

Fig.4-154: Task configuration, monitoring

The following information is displayed for each task:

Task	Task name as defined in the task configuration
Status	**Not created:** Task was not started since most recent update; especially for event tasks **Created:** Task is known in the runtime system, but not yet set up for operation **Valid:** Task is normal in operation **Exception:** Task has exceptional state
Number of cycles	Number of cycles already executed
Last cycle time	Last measured cycle time in µs
Average cycle time	Average cycle time across all cycles in µs
Max. cycle time	Maximum cycle time across all cycles in µs
Min. cycle time	Minimum cycle time across all cycles in µs
Jitter:	Most recently measured jitter[1] in µs
Min. Jitter	Minimum measured jitter in µs
Max. Jitter	Maximum measured jitter in µs

If the cursor is positioned on a task name field, the values displayed can be reset to 0 using the command **Reset** in the context menu.

4.14.3 Task Editor

Task Editor, General Information

Individual tasks belonging to the application are located below the task configuration.

Open the task editor window by double-clicking on the task object in the Project Explorer or by using the task editor window.

In contrast to IndraLogic 1.x, the tree structure of the task configuration, page 423, is integrated into the object tree of the Project Explorer. The higher level "task configuration" object and the respective task objects below can be inserted using the **Add ▸ PLC object** command.

Default tasks

Two PLC tasks are set up automatically when creating the control:

PlcTask: This task is called cyclically every 10 ms. Non-critical PLC program sequences can be implemented in this task. Its cycle time and priority can be adjusted according the requirements of the PLC program (Configuration, page 429)

MotionTask: The MotionTask is coupled to the calculation of the axis motion. It has very high priority and is therefore only to be used for time-critical tasks that require synchronization with axis motions.

The Motion task provides dead time-optimum behavior for axis-synchronous processes, that is all dead times related to axis motions are of a minimum. A task runtime that is too high can cause the error "Cycle time exceedance" (see system task). The settings of this task are not to be changed. Other PLC tasks can be created any time if required.

[1] *Jitter: Period between the beginning of the task and the point in time after which the operation system indicates that it is running.*

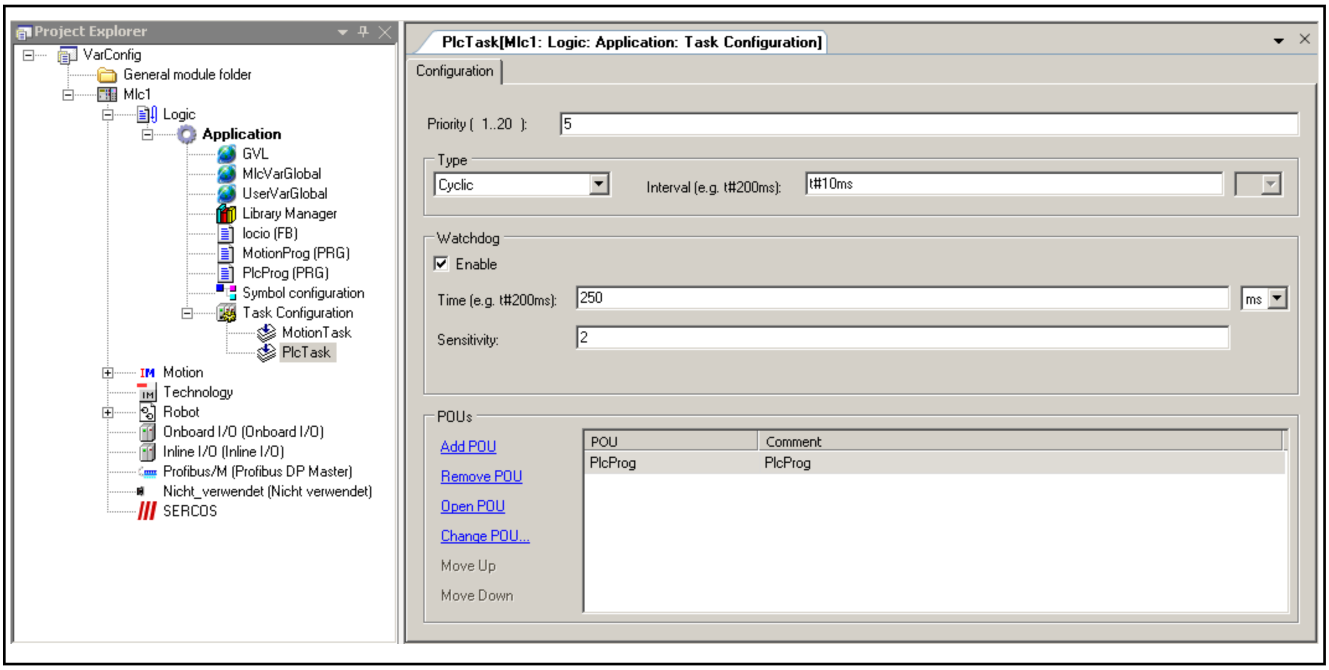
The maximum number of tasks depends on the control used.

For more details on the "Tasks", refer to

- Rexroth IndraMotion MLC 12VRS, Functional Description, DOK-MLC***-FUNC****V12-APxx-EN-P, R911333848.
- Rexroth IndraMotion MTX 12VRS, Functional Description, DOK-MTX***-NC*FUNC*V12-RE01-EN-P, R911334355.

Task Editor, "Configuration" Tab

When a task is added to the task configuration, page 423, (using **Add ▸ PLC object**), the "Configuration" task editor dialog opens to set task properties:



Priority (1..20): 5

Type: Cyclic — Interval (e.g. t#200ms): t#10ms

Watchdog
☑ Enable
Time (e.g. t#200ms): 250 ms
Sensitivity: 2

POUs

POU	Comment
PlcProg	PlcProg

Fig.4-155: Configuration dialog for a task

Define the parameters as desired:

A task can be renamed by editing its name in the Project Explorer.

Priority (1-20): (Number between 1 and 20; 1 is the highest priority, 20 is the lowest)

Type: The selection list provides the following task types:

- **Cyclic**: The task is processed in cycles with a cycle time as defined in the "Interval" field (see below):
- **Free running (FreeWheeling)**: the task processing begins at program start. At the end of the run, automatic restart occurs in a continuous loop. No cycle time is defined.
- **Event-controlled**: The task processing begins as soon as the variable defined in the "Event" field receives a rising edge.
- **Externally event-controlled**: The task processing begins as soon as the event defined in the "Event" field occurs. The events supported and provided in the selection list depend on the target system. (Do not confuse them with the system events)

Obligatory entries depending on the task type:

- **Interval** (obligatory entry for "cyclic" task types or for an "external event" if the event requires a time specification): Time period after which the task is to be restarted. If a number is entered here, the desired unit can be set in the selection box behind the input field: milliseconds [ms] or microseconds [µs]. If the [ms] format is selected, an entry is automatically displayed in TIME format (e.g. "t#200ms") as soon as the focus is returned to the window. However, the entry can also be made directly in TIME format. Entries in [ms] format is always displayed as an abstract number (e.g. "250").
- **Event** (for the task type "Event" or "External event"): Global variable triggering the task processing to start as soon as a rising edge is received.

  Use the ... button or the input assistance <F2> to get a list of all available global variables.

Watchdog:

For each task, a watchdog can be configured. If the target system supports an extended watchdog configuration, it is possible that upper and lower limits and a default watchdog time are preset as well as a time specification in percent.

- **Enable**: If this option is enabled (☑, see below), the task is aborted with error status as soon as the processing is longer than the time period defined in the "Time" field (see below)

  If the option Update I/O in Stop state, page 334, was enabled, the outputs are reset to the defined default values.
- **Time (e.g.: t#200ms)**: Watchdog time; after this time period has been expired, the watchdog is enabled irrespective of whether the task has been completely processed. Depending on the target system, the monitoring interval might have to be entered in percent of the task interval. In this case, the selection box for the unit is grayed out and displays "%".
- **Sensitivity:** Number of acceptable times the watchdog time can be exceeded without outputting an error.

Note that a watchdog can be deactivated for specific PLC cycles by using the CmplecTask.library library functions.This is useful for cycles that can take longer due to initializations.

After declaring a suitable variable for the task handle (of type RTS_IEC_HANDLE),

`hIecTask : RTS_IEC_HANDLE;`,

the watchdog deactivation (and following reactivation) can be carried out using the interface functions:

Disabling the watchdog

```
hIecTask := IecTaskGetCurrent(0);
IecTaskDisableWatchdog(hIecTask);
//Code that is protected against
//watchdog IecTaskEnableWatchdog(hIecTask);
```

POUs:

The function blocks controlled by the task are listed in a table with the "POU" columns (function block name) and an optional "comment". There are commands for editing on the left of the table:

- To define a new program call, use the **Add POU** command to open the input assistance dialog and select the desired program from those available in the project.
- To replace one call with another, highlight the entry in the table, open the input assistance with the **Input assistance** command and select another program.
- To delete an entry, highlight it in the table and use the **Remove POU** command.
- The **Open POU** command opens the currently selected program in the respective editor.
- The **Change POU** command exchanges the currently selected program for a program (POU) provided in the selection window.
- The arrangement of the function blocks in the list from top to bottom specifies the **execution sequence** of the programs in online mode. Use the **Move Up** and **Move Down** commands to move the currently selected entry in the list.

4.15 Trace

4.15.1 Trace Editor

Trace Editor, General Information

This section includes:

- Trace basics, page 431
- Creating a trace object, page 432
- Trace configuration, page 432
- Trace functionality, page 433

Trace basics

The trace editor is used to configure and display trace records.

It allows to detect the value characteristic of the variables on the PLC over a specified period. This is similar to a digital sampling oscilloscope. A trigger can additionally be set to control data acquisition via input/trigger signals. Trace variable values are continuously written to a trace buffer of specified size and can then be seen as two-dimensional graph as function of time.

The trace manager is provided with an extended functionality. The following is possible:

- Configuring and tracing control parameters (such as temperature curve of the CPU or accumulators), see
  - Setting and Changing Variable Settings, page 434, and
  - Determining and Changing Recording (Trigger) Settings, page 436.
- Reading device traces (such as the trace of the current path of a drive, see Updating Trace, page 299.
- Tracing system variables of other runtime components.

Additionally, the command Online list..., page 298, is provided.

The configuration as well as the display settings of the traces are set in the trace editor, page 431,. Commands to access the configuration dialogs are provided. Any number of variables can be simultaneously traced and displayed possibly in different views such as for the multi-channel operation.

Traces of variables with different trigger settings have to be recorded with an own trace object. Any number of trace objects can be created.

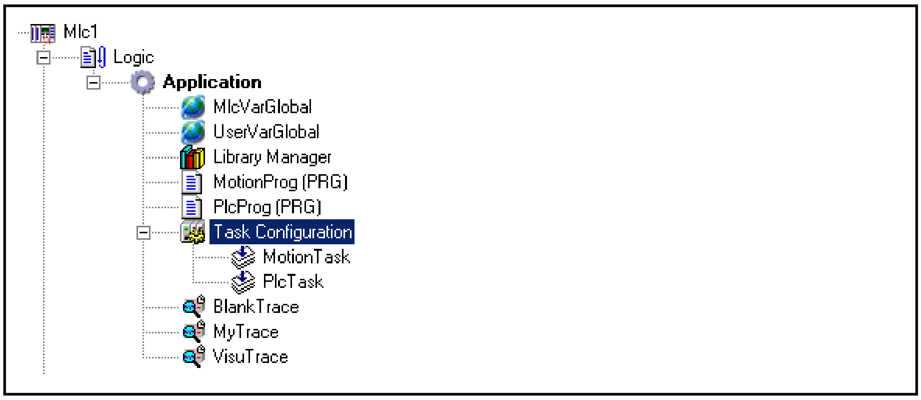


Fig.4-156: Example: Project with traces

Commands to change the display settings are provided in the trace menu, page 294. Thus, the graph is compressed or stretched, zoom functionalities and a cursor as well as commands to operate the trace are available. Reading a trace can also be integrated into a visualization by using a ↘ **special "trace" visualization element.**

Creating a trace object

Add a trace object in the Project Explorer below an application using the Add Object, page 234, command and edit it with the "Edit Object" command or double-click on the object to open the trace editor.

At first, the trace editor window, called <Trace object name> displays an empty area on the left side where the trace curves are shown later (trace window). In the right section is the trace tree as specified in the trace configuration, variable settings, page 434,.

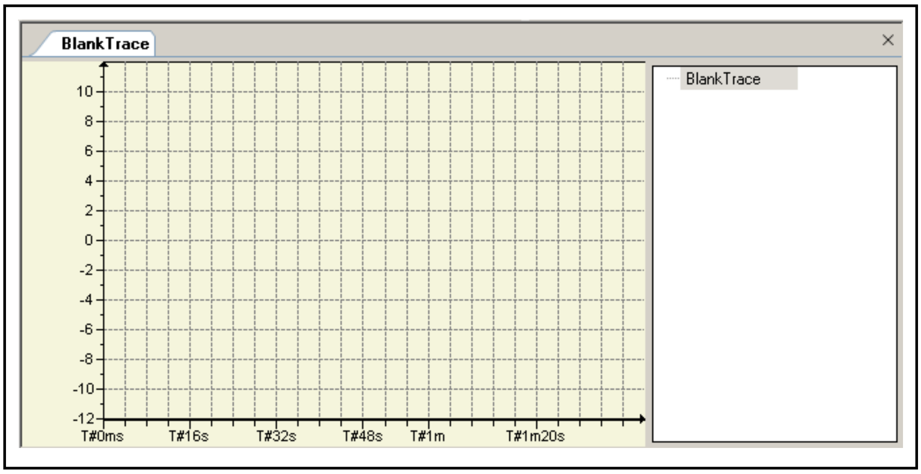


Fig.4-157: Dialog: Trace editor window, still without user-specified configuration

Trace configuration

In the trace, at least one variable is specified that is recorded.

For this purpose, general settings are defined that specify the recording start, sampling, trigger conditions, buffer size, responsible task, etc.

These have to be set first. The **Configuration...** command can be used.

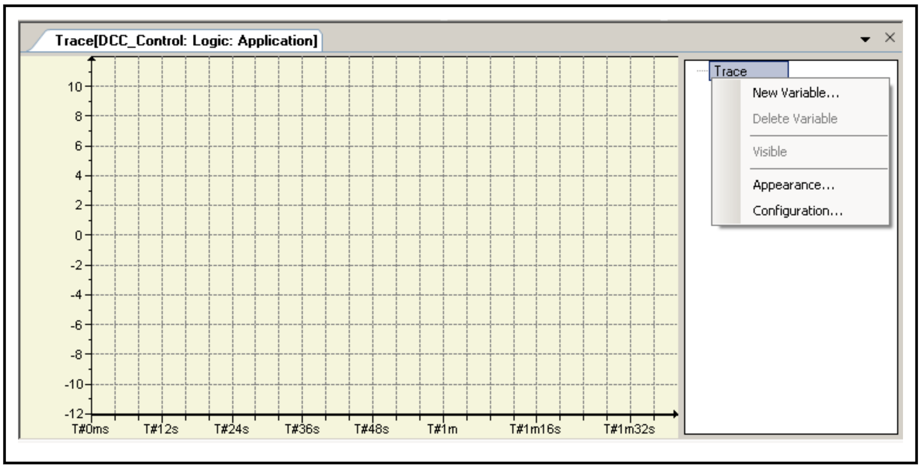


Fig.4-158: Context menu in the trace tree of the trace editor

Additionally, the display settings (display parameters of the coordinate system in which the graph is displayed) are set for each of these trace variables.

To configure a trace, use ...

1. New variable..., page 434 to add a variable and to set some of the display settings.
2. The commands in the context menu in the right section of the trace window are used to delete a selected variable.
3. The commands in the context menu in the right section of the trace window are used to make a selected variable visible.
4. Display settings..., page 439 to specify the display settings for the graphs. (If no configuration is loaded, this command is grayed out.)
5. Configuration, page 434 with its special trace variables to define the trace graphs (display settings, conditions, curve types).

Trace functionality

For offline and online operation of the trace editor, the commands in the trace menu, page 294, are available

Reset trigger.

The following commands have to be used when operating the trace:

- Login trace (download), page 294
- Start/stop trace, page 294
- Reset trigger, page 295

The following commands have to be used when arranging the graph:

- Cursor, page 295
- Scrolling with mouse, page 296
- Zooming with mouse, page 297
- Resetting display, page 297
- Compressing, page 297
- Expanding, page 298
- Multi-channel, page 298

The following commands allow to access traces saved on the runtime system:

- Online list, page 298
- Upload trace..., page 299

The following commands allow to access traces saved on the runtime system:

- Load trace..., page 299
- Save trace..., page 299

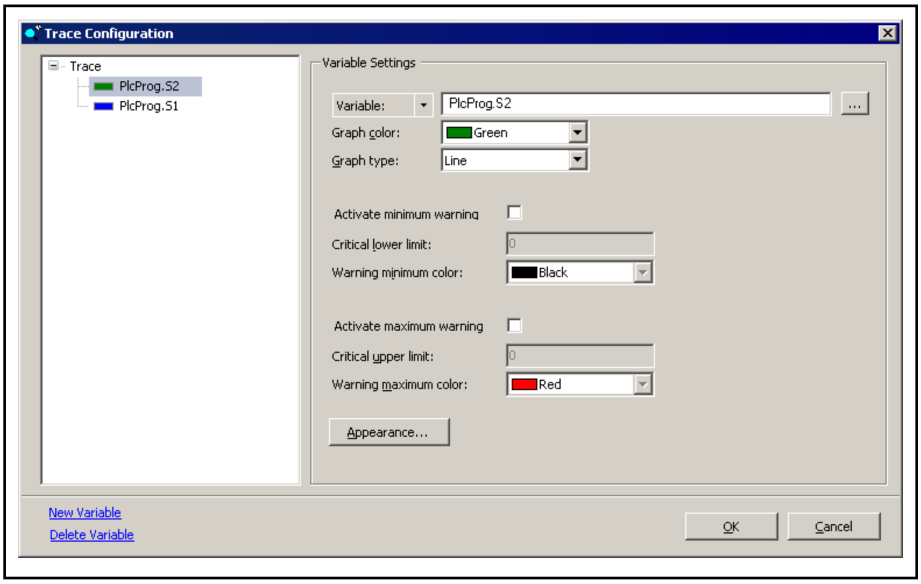
... More commands:

- Shortcuts in the trace diagram, page 444.

Trace Configuration, Variable Settings

The dialog "Trace Configuration, Variable Settings" is opened with the new "New variable..." command or if a new variable is selected in the trace tree and the "Configuration..." command is used. These commands are available in the context menu of the trace tree in the right part of the trace editor.

The variables to be traced can be configured here in their display format:



Variable: PlcProg.S2

Graph color: 🟩 Green

Graph type: Line

Activate minimum warning □

Critical lower limit: 0

Warning minimum color: ■ Black

Activate maximum warning □

Critical upper limit: 0

Warning maximum color: 🟥 Red

Fig.4-159: "Trace Configuration" dialog with "Variable Settings"

All trace variables are displayed with a tree structure in the left part of the window and the upper node is named with the trace name.

Adding and deleting a trace variable

To add a variable to the trace tree or to delete it, use the commands below the trace tree:

New Variable: A nameless entry appears in the trace tree and the settings for the new variable for the configuration are provided in the right part of the dialog.

Delete Variable: The selected variable and its configuration are deleted.

The trace tree is then immediately refreshed.

Setting and changing variable settings

If a variable should be configured, it has to be selected in the trace tree. The settings are then displayed on the right. To be able to change the settings later on, select the variable again and use the **Variable Settings** dialog.

Variable: Enter a variable name (and path) to specify the signal to be traced.

A valid signal is an IEC variable, a property, a reference, the content of a pointer or an array element of the application. Permitted types are all IEC-based types except STRING, WSTRING or ARRAY. Enumerations are also permitted if the basic type is no STRING, WSTRING or ARRAY. The input assistance ... can be used to get valid entries.

If a device parameter should be traced, select the parameter in the drop-down list ⌄. Then, a **parameter** can be selected using the input assistance. Then edit or check variable settings. Device parameters are only supported if the component "CmpTraceMgr" exists in the runtime system. Then, it is also impossible to enable the **Generate trace POU for visualization** option.

If "CmpTraceMgr" is used to trace on the runtime system, a property, page 46, with the "Monitoring" attribute, page 536, has to be provided if it should be used as trigger.

Graph color: Select the color in which the trace graph should be displayed from the shown color selection list.

Graph type: Determine how the value points of the trace curve have to be displayed for the variable. It is recommended to use **Line** for a high amount of data.

- **Line:** The values are interconnected with a line of points.
- **Steps:** The values are connected like a stair made of points (a vertical line up to the y-value of the next point and a horizontal line to the x-value from there)
- **Point:** The individual values are displayed as points.
- **Cross:** The values are displayed as crosses.

Activate minimum warning: If this option is enabled, the trace graph is displayed in the specified color as soon as the variable value falls below the lower limit of the defined value.

Critical lower limit: If the variable value falls below the value entered here and the option **Enable lower limit** is set, the color of the graph changes to the values specified below.

Graph color: Color value of the color below the lower limit.

Activate maximum warning: If this option is enabled, the trace graph is displayed in the specified color as soon as the variable value exceeds the upper limit of the defined value.

Critical upper limit: If the variable value exceeds the value entered here and the option **Enable upper limit** is set, the color of the graph changes to the values specified below.

Graph color: Color value of the color above the upper limit.

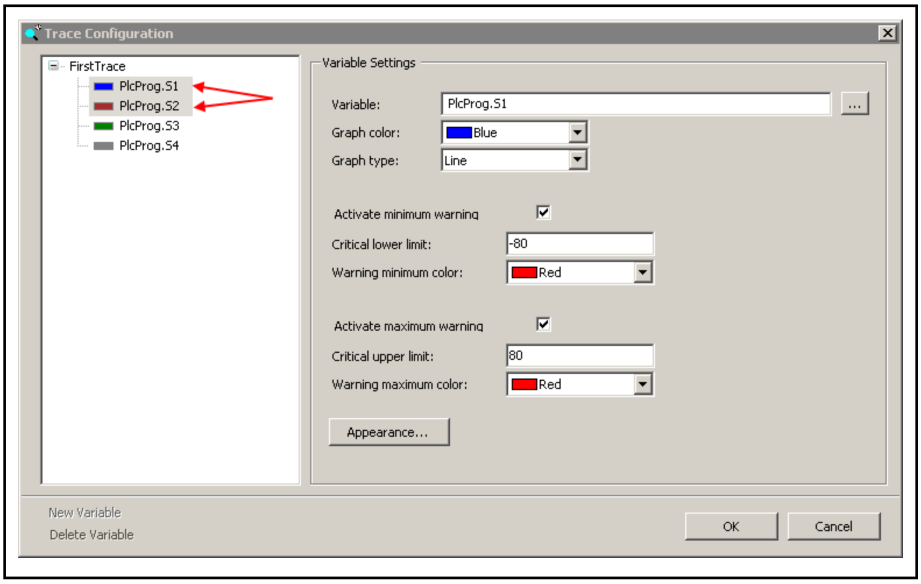
Appearance...: This button opens the dialog Appearance of the Y-axis, page 442. There, the display settings of the currently configured variable for the y-axis has been made in its own style (color and scroll behavior).

Multiple selection of the variable

In addition, several variables can be selected with <Shift + Click> or <Ctrl + Click>.

In this case, the change in the "Variable Settings" dialog is carried out for all variables selected.

The keyboard/mouse commands correspond to the following keyboard commands <Shift> + <Arrow up>/<Arrow down> or <Ctrl> +<Arrow up>/<Arrow down>.



- FirstTrace
  - PlcProg.S1
  - PlcProg.S2
  - PlcProg.S3
  - PlcProg.S4

Variable: PlcProg.S1
Graph color: Blue
Graph type: Line

Activate minimum warning ☑
Critical lower limit: -80
Warning minimum color: Red

Activate maximum warning ☑
Critical upper limit: 80
Warning maximum color: Red

Fig.4-160: Dialog: Multiple selection of variables

Trace Configuration, Record Settings

The dialog "Trace Configuration, Record Settings" is opened via the command "Configuration..." located in the context menu of the trace tree in the right part of the trace editor window or by double-clicking on the trace name (upper node in the trace tree).

The settings made in the "Record Settings" dialog apply for all variables of the trace graph.

Trigger basics

It is not suitable in many cases that tracing and displaying input signals are started at any point in time like immediately after the preceding message or when commanded by the user ▶ for example. In most cases, tracing is preferred. Its displaying starts when certain predefined conditions apply to one or several signals. This is called triggering and can be defined here.

Conditions how input signals can be triggered:

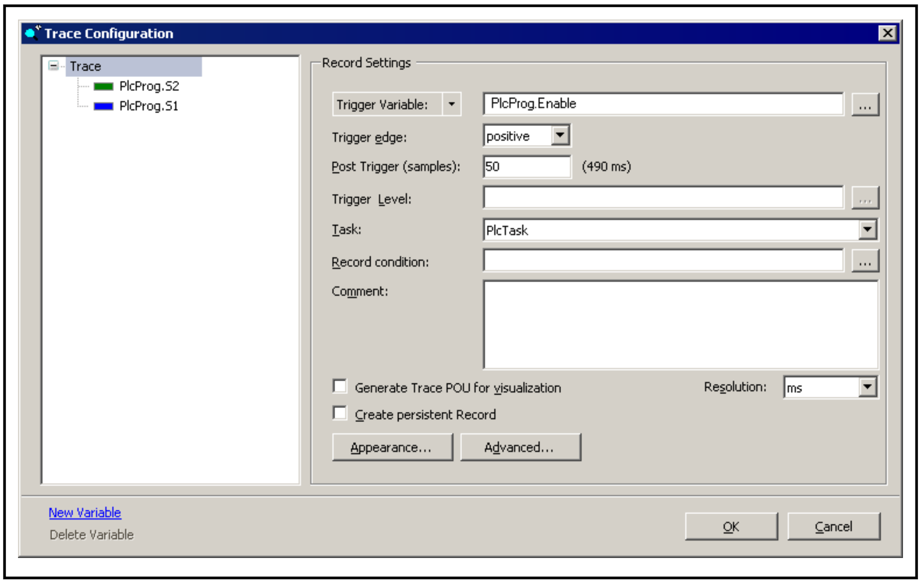
- By configuring trigger variables
- By configuring a recording condition
- or by both

If no trigger condition is defined, the trigger system is switched off and the trace runs independently: The measurement of the values starts directly.

If a trigger variable is defined, it is used to trigger the input signal when a specified level is reached. A trigger is "fired" and the trigger system is enabled. A trigger level can be reached with rising and/or falling edge.

If the recording condition is defined, data acquisition is only enabled if the recording condition is TRUE.

Determining and changing the record (trigger) settings


Trace Configuration

- Trace
  - PlcProg.S2
  - PlcProg.S1

Record Settings

Trigger Variable: PlcProg.Enable ...

Trigger edge: positive

Post Trigger (samples): 50 (490 ms)

Trigger Level:

Task: PlcTask

Record condition:

Comment:

□ Generate Trace POU for visualization Resolution: ms

□ Create persistent Record

Fig.4-161: "Trace Configuration" dialog with "Record Settings"

Trigger Variable: Determines the signal to be used as trigger by entering the name (and path) of a signal.

A valid trigger signal is an IEC variable, a property, a reference, a pointer, an array element of the application or an expression. Permitted types are all IEC-based types **except** STRING, WSTRING or ARRAY.

Enumerations are permitted if the basic type is no STRING, WSTRING or ARRAY.

The content of the pointer is also an invalid signal. The input assistance ... can be used to get valid entries.

If a device parameter should be used as trigger, use ▼ and select the trigger parameter in the drop-down list. Then, a parameter can be selected using the input assistance ....

Device parameters are only supported if the component "CmpTraceMgr" exists in the runtime system. Then, it is impossible to enable the **Generate trace POU for visualization** option.

> If "CmpTraceMgr" is used to trace on the runtime system, a property, page 46, with the "Monitoring" attribute, page 536, has to be provided if it should be used as trigger.

Trigger edge:

- **None:** No trigger effect (default).
- **Positive:**Trigger event at a rising edge on the Boolean trigger variable - or in case of an analog trigger variable - as soon as the value defined in the trigger level is reached "from below".
- **Negative:**Trigger event at a falling edge on the Boolean trigger variable - or in case of an analog trigger variable - as soon as the value defined in the trigger level is reached "from top".
- **Both:**Trigger event for the conditions described for positive and negative (see above).

Post-trigger records: Number of records per input signal after the trigger has been "fired". Default value is 50, the value range is between 0 and (2^{32}-1).

Trigger Level: This has to be set exactly when an analog variable (variable with numeric type, e.g. LREAL or INT) is used as trigger variable. Then, the threshold value at which the trigger is fired has to be defined. A value has to be entered immediately. Entering a GVL constant or an ENUM value is only allowed if the type can be converted to the trigger variable. If it is traced via IEC code in the runtime system, any IEC variable whose type can be converted to the trigger variable can be entered (default: empty.)

Task: Select the task from which data should be acquired from the list of available tasks.

Record condition: If "CmpTraceMgr" is used, the recording condition has to be either a variable of type BOOL or a bit access. The content of a pointer is also an invalid entry. Properties are supported. If the IEC code is used to trace in the runtime system, any Boolean IEC expression can also be entered. Data is acquired of the condition is TRUE.

Comment: Enter a comment about the current recording conditions for example.

Generate trace POU for visualization: Enabling the checkbox explicitly creates the component <Trace name><Task name>_VISU. This is reasonable if the trace values should be integrated into a visualization. If device parameters are used as trigger (or also as variable), it is not possible to enable this option.

Create persistent record: If this option is selected, the trace configuration and the latest content of the trace buffer of the runtime system are saved persistently on the target system. This option is only available if a trace manager traces in the runtime system.

Resolution: The resolution of the trace time stamp can be set to "ms" or "µs". For each signal to be detected, data pairs consisting of value and time stamp are saved and transferred to the programming system. The time stamps to be transferred are relative and relate to the tracing start. If the task cycle time specified is smaller than 1 ms, it is recommended to set the resolution of the time stamp to "1µs". This option is only available if a trace manager exists in the runtime system.

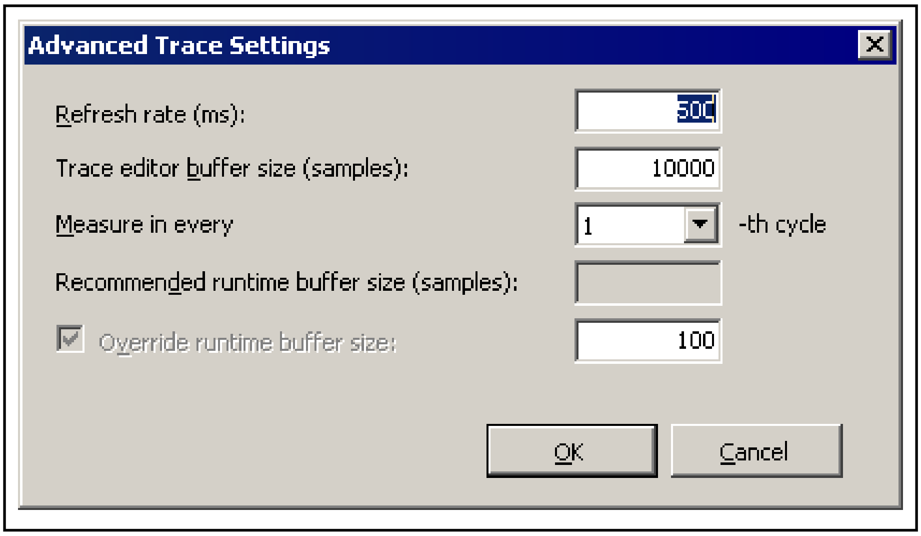
Appearance...: This button opens the dialog Edit Display Settings, page 439. The display in the trace window such as axis scaling, color and scroll behavior can be configured for the currently configured data set.

Advanced...: This button opens the dialog Advanced Trace Settings, page 438. Further settings for the trace trigger can be made here.

☞ If trace signals with different time basis (that is with different trigger) should be detected and displayed, the required recording settings have to be configured in the individual trace objects.

Advanced Trace Settings

The dialog "Advanced Trace Settings" opens when clicking on "Advanced..." in the dialog "Trace Configuration, Record Settings". The desired values can be set here. Complete the dialog with <OK>.



Refresh rate (ms): 500

Trace editor buffer size (samples): 10000

Measure in every 1 -th cycle

Recommended runtime buffer size (samples):

☑ Override runtime buffer size: 100

Fig.4-162: "Advance Trace Settings" dialog

For all values specified in the recordings or cycles, the respective time interval (e.g. 3 s 200 ms) is displayed on the right next to the input field. The time period of the buffer includes the complete buffer. If the task cycle time is unknown (e.g. since the task is not set, it is not cyclic or it is a system task), the time cannot be calculated.

Refresh rate (ms): At this point in time, the data pairs (values with time stamp) detected while tracing are saved in the buffer of the trace editor. The valid range is between 150 ms and 10000 ms. Preset are 500 ms. If a trace manager is used, the data pairs with this time frame are transferred from the runtime system to the programming system. If no trace manager is used, data is transferred every 200 ms to the programming system.

Trace editor buffer size (recordings): Specify the buffer size of the trace editor in the recordings. The buffer has to be higher by a factor of 2 or equal to the runtime system buffer size.

Additionally, the value has to be between 1 and 10^7.

Measure in every nth cycle: The interval (in task cycles) between the data acquisitions of the trace signal is specified. (preset and smallest value of 1 means that a measurement is made in each cycle). To set a value, use the pull-down menu.

Recommended runtime buffer size (samples): The recommended number of records in the runtime buffer for each signal is displayed here. The value is based on the task cycle time, the refresh rate and the value in **Measure in every nth cycle** is calculated. This allocates an individual buffer for each trace variable.

☞ The buffer size is specified in the recordings and a buffer is created for each trace variable.

Override runtime buffer size: if this option is selected, the value specified is used for the runtime buffer size instead of the recommended value. The value has to be set to 10 at least and may not be higher that the trace editor buffer size.

Editing Display Settings (Appearance)

The "Edit appearance" dialog opens using the command "Show..." in the dialog Trace Configuration, Record Settings, page 436. Enter the desired values and complete the dialog with <OK>.

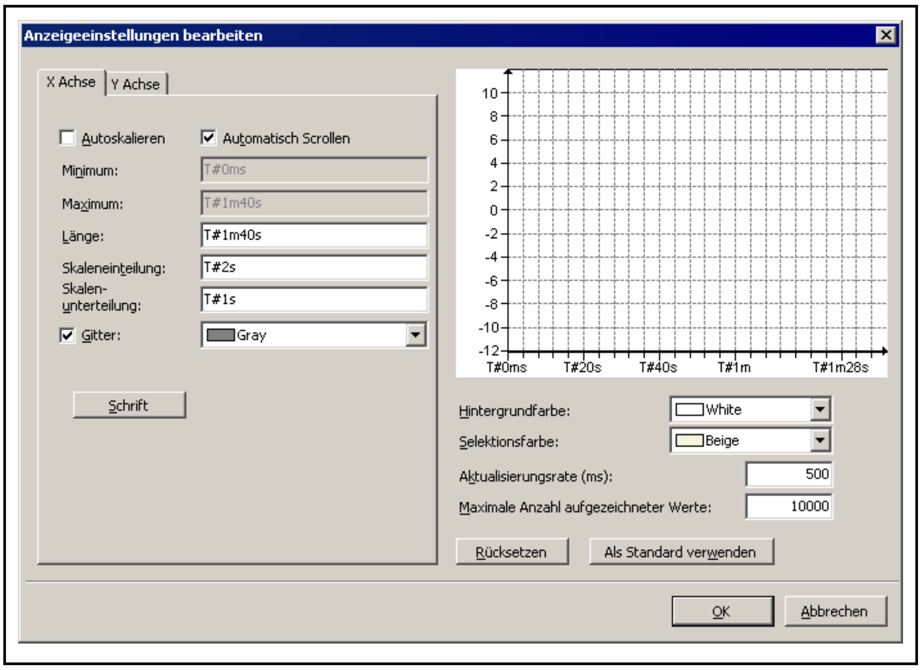
The following settings define the display of the coordinate system or its x-/y-axes. The settings of the y-axis are used if the trace diagram is displayed in **single channel view**. In multi-channel view, the settings of the Appearance of the Y-axis, page 442, is used.

The settings for the x/y-axis (to be set in the left section) and the coordinate system are immediately applied in the coordinate system.

The configuration can be managed using the following commands:

- **Reset:** This command resets the configuration of the display to the default value.
- **Use as default:** This command saves the current configuration of the display as default. If a new trace/trace variable is configured, this setting is used.

X-axis



Anzeigeeinstellungen bearbeiten

X Achse | Y Achse

☐ Autoskalieren ☑ Automatisch Scrollen

Minimum: T#0ms

Maximum: T#1m40s

Länge: T#1m40s

Skaleneinteilung: T#2s

Skalen-unterteilung: T#1s

☑ Gitter: Gray

Schrift

Hintergrundfarbe: White

Selektionsfarbe: Beige

Aktualisierungsrate (ms): 500

Maximale Anzahl aufgezeichneter Werte: 10000

Rücksetzen | Als Standard verwenden

OK | Abbrechen

Fig.4-163: Dialog: Appearance of the X-axis

Selecting the display mode:

- **Auto:** If this option is enabled, the time axis is automatically scaled according to the trace editor buffer, page 438,. The current content of the trace buffer is displayed in the diagram. No further entries have to be made.
-

Fixed Length: If this option is enabled, the displayed interval of the time axis has a fixed length. It has to be set in the input field **Length**. Scaling the time axis is also performed according to the length and the diagram is automatically scrolled to a visible range. Thus, an interval of determined length with the respective latest data is displayed in the diagram. However, only the amount recorded is displayed. If new data is available, the display of the graph scrolls.

Fixed: If this option is enabled, the displayed interval of the x-axis is fixedly determined by a maximum and a minimum.

Minimum: This value defines the smallest value of the time axis that is displayed.

Maximum: This value defines the highest value of the time axis that is displayed.

Length: This value defines the length of the displayed interval of the time axis.

The time entries do not require the prefix "#" as it is the case for the IEC code.

Possible time specifications are for example "2s", "1ms" or "1m20s14ms".

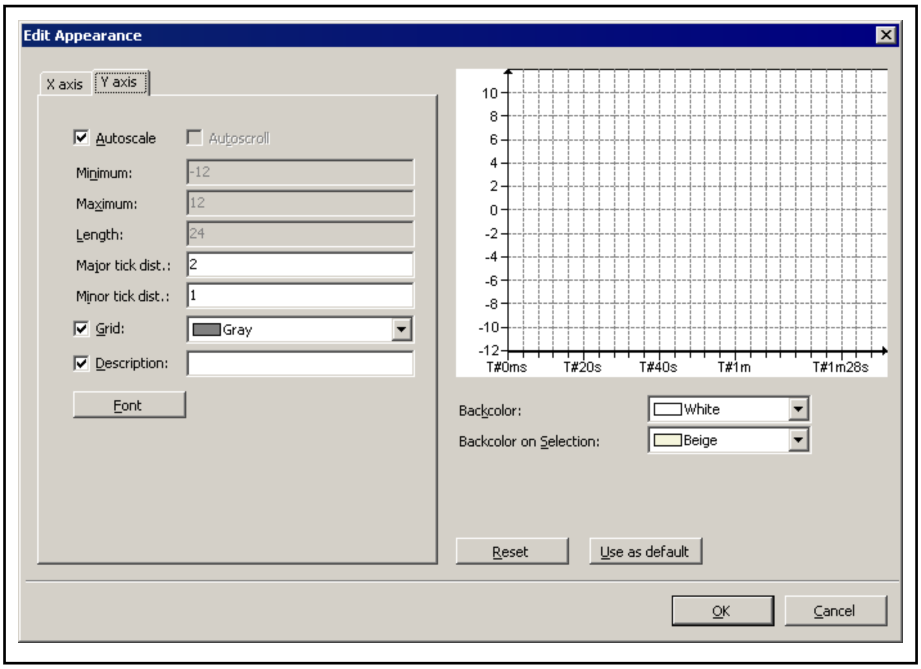
Use "us" for microseconds, e.g. "122ms500us".

Reasonable values also depend on the resolution of the time axis.

Grid: If this option is enabled, a grid is displayed. The color of the grid lines can be selected from the color selection list.

Fonts: This button opens a standard dialog for the font definition to determine a font in the trace display.

Y-axis



☑ Autoscale

□ Autoscroll

Minimum: -12

Maximum: 12

Length: 24

Major tick dist.: 2

Minor tick dist.: 1

☑ Grid: Gray

☑ Description:

Backcolor: White

Backcolor on Selection: Beige

Fig.4-164: "Edit Appearance" dialog, y-axis

The display mode has to be selected:

- **Auto:** If this option is enabled, the y-axis is automatically scaled according to the data acquired. No further entries have to be made.
- **Fixed:** If this option is enabled, the displayed section of the y-axis is fixedly defined by specifying minimum and maximum.

Minimum: This value defines the smallest value of the y-axis displayed.

Maximum: This value defines the highest value of the y-axis displayed.

Grid: If this option is enabled, a grid is displayed. The color of the grid lines can be selected from the color selection list.

Description: If this option is enabled, the y-axis is labeled with the text entered in the **Description** field.

Fonts: This button opens a standard dialog for the font definition to determine a font in the trace display.

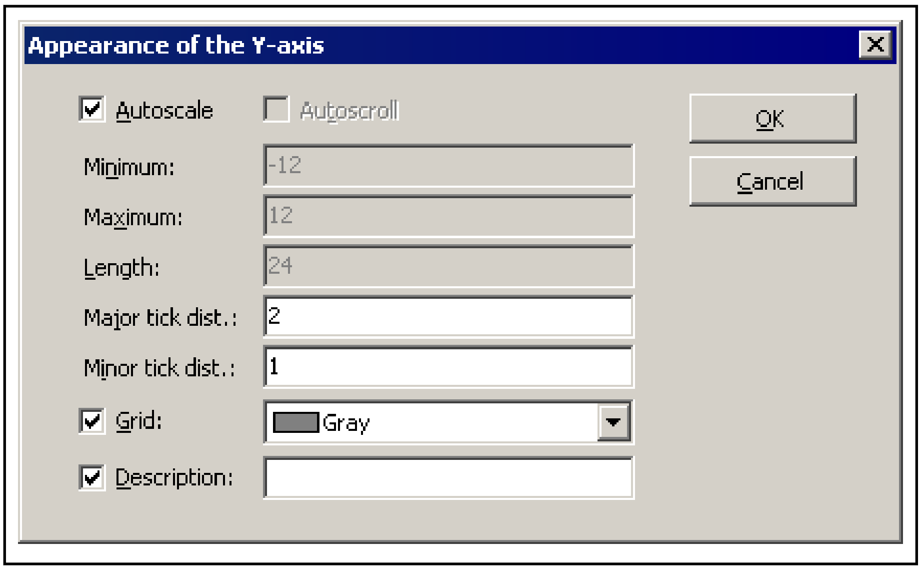
Coordinate system

Background color: Select the background color for the coordinate system from the given color selection list. It is used if the diagram in the trace window is **not** selected.

Selection color: From the given color selection list

Appearance of the Y-Axis

The dialog "Appearance of the Y-axis" opens when clicking on "Show..." in the dialog "Trace Configuration, Variable Settings". Enter the desired values and complete the dialog with OK. The following settings configure the appearance of the y-axis. It is used if the trace diagram is displayed in **multi-channel view**.



☑ Autoscale □ Autoscroll

Minimum: -12

Maximum: 12

Length: 24

Major tick dist.: 2

Minor tick dist.: 1

☑ Grid: Gray

☑ Description:

Fig.4-165: "Edit Appearance" dialog, y-axis

Selecting the display mode:

- **Auto:** If this option is enabled, the y-axis is automatically scaled according to the values acquired. No further entries have to be made.
- **Fixed:** If this option is enabled, the displayed section of the y-axis is fixedly defined by specifying minimum and maximum.

Minimum: This value defines the smallest value of the y-axis displayed.

Maximum: This value defines the highest value of the y-axis displayed.

Grid: If this option is enabled, a grid is displayed. The **color** of the grid lines can be selected from the color selection list.

Description: If this option is enabled, the y-axis is labeled with the text entered in the **Description** field.

Fonts: This button opens a standard dialog for the font definition to determine a font in the trace display.

Trace Editor in Online Mode

If a trace is running on the control, it is displayed in the dialog Online list..., page 298,.

Changes in the application

To go to online mode with a trace and its configuration, the trace has to be **explicitly** loaded to the runtime system at a logged in application using the menu item Trace download, page 294,. The graphs of the trace signal are immediately displayed in the trace editor window. The trace has to be reloaded when changes are made in the application.

If there are only logins and logouts without any changes on the application in the runtime system, the traces continue running on the control without a new download.

If the code of the application is changed, it depends on the login mode how the traces are handled on the control:

- **Log in with online change** or **Log in without changes:** Traces continue running.
- **Login with download:** The traces are deleted on the runtime system and a new download is required.

Online configuration of the trace diagram

The dialogs Trace Configuration, Record Settings, page 436, and Trace Configuration, Variable Settings, page 434, are also available in online mode and changes of the trace configuration can be made while the traces are running.

It depends on the type of changes whether the trace can continue running. If it is not possible since the name of the trace signal was actually changed, the trace stops and a new download is required.

Arranging the trace graph

By default, the commands of the trace, page 294, are available to control the currently displayed graph.

The displayed section of the acquired data depends on the trace configuration, but can be newly arranged by the scroll and zoom functionalities. These are located in the trace menu, the toolbar or can be used via shortcuts.

The following types of zooming/scrolling are supported.

- Go to **single channel view**, click into the trace diagram. Then you can
  - zoom the time axis using the mouse wheel or

    <+> or <-> or

    compress or **expand** or

    →← or ↔

  - zoom the y-axis using the mouse wheel or <+> or <->, **while** <Ctrl> is pressed
  - scroll the y-axis up and down using <arrow up> and <arrow down>.
  - zoom on a rectangle window opened with the mouse. See Zooming with mouse, page 297
- Go to **multi-channel view** (if the multi-channel option is enabled), click into the trace diagram. Then you can
  - zoom the time axis for **all diagrams** using the mouse wheel or

    <+> or <-> or

    compress or **expand** or

    →← or ↔

  - zoom the y-axis **only for the selected diagram** using the mouse wheel or <+> or <-> **while** <Ctrl> is pressed
  - scroll the y-axis up and down **only for the selected diagram** using <arrow up> and <arrow down>.

All trace shortcuts are listed in the Shortcuts in the trace diagram, page 444,.

Please note that the cursor, page 295 is available in online mode.

Keyboard Operation (Shortcuts) in the Trace Diagram

The trace diagram graphs change after longer intervals if the <Ctrl> key is pressed in addition to the following shortcuts.

The trace editor, page 431, provides the following shortcuts by default:

Shortcuts	Action in the trace diagram	Respective control with the mouse
<left/right> arrow keys	If there are no cursors, the time axis scrolls (horizontal). If there is a cursor, it is moved to the left/ right. A cursor is created by clicking once into the diagram.	If there are cursor(s) available, click into a cursor, page 295 and move it with the pressed left mouse button.
<Shift>+<left/right> arrow keys	If there are two cursors, the one that is **not selected** is moved.	
<Shift>+<left/right> arrow keys Keep <Ctrl> additionally pressed	The cursor is moved in bigger steps.	
<Alt>+<left/right> arrow keys	Horizontal scrolling of the x-axis (time axis)	Enable scrolling with mouse, page 296, click into the diagram and scroll the x-axis with the mouse keeping the left mouse button pressed.
<Up/down> arrow keys With or without <Alt>	Vertical scrolling of the y-axis	Enable scrolling with the mouse, click into the diagram and scroll the y-axis with the mouse keeping the left mouse button pressed.
<Up/down> arrow keys With or without <Alt> Keep <Ctrl> additionally pressed	The graph is moved in bigger steps	
<+/-> key	Zooming along the x-axis (time axis is expanded/compressed.)	• Click on Expand, page 298, or Compress, page 297 • or ↔ or →← • or use the mouse wheel.
<Ctrl>+<+/-> keys	Vertical zooming along the y-axis (y-axis is expanded/compressed.)	• Click on Expand, page 298, or Compress, page 297 • or ↔ or →← • or use the mouse wheel. Keep <Ctrl> pressed.

Fig.4-166: Trace editor shortcuts at a single channel

If multi-channel is enabled, the following actions occur when using the following function keys:

Shortcuts	Action in the trace diagram	Respective control with the mouse
<Alt>+<left/right> arrow keys	Horizontal scrolling of x-axis (time axis) in all diagrams	Enable Scrolling with mouse, page 296, click into the diagram and scroll the x-axis with the mouse keeping the left mouse button pressed.
<Up/down> arrow keys With or without <Alt> pressed Select a diagram with <tab>.	Only horizontal scrolling of the selected diagram	Enable Scrolling with mouse, page 296, click into the diagram and scroll the x-axis with the mouse keeping the left mouse button pressed and keep <Ctrl> pressed.
<Up/down> arrow keys With or without <Alt> pressed Select a diagram with <tab>. Keep <Ctrl> additionally pressed	The graph is moved in bigger steps.	
<+/-> keys	Horizontal scrolling of all diagrams along the x-axis (time axis is expanded/compressed for all diagrams.)	• Click on Expand, page 298, or Compress, page 297 • or ↔ or →← • or use the mouse wheel.
<Ctrl>+<+/-> keys Select a diagram with <tab>.	Vertical scrolling of all diagrams along the y-axis (y-axis is expanded/compressed for the selected graph.)	

Fig.4-167: Trace Editor shortcuts at enabled multi-channel

4.16 Visualization Editor

4.16.1 Visualization Editor, General Information

Icon:

The visualization editor is required to create visualization objects.

> ☞ An overview on the visualization in IndraLogic 2G can be found in Visualization in IndraLogic 2G, page 625.

To open the visualization editor, double-click on the entry in the Project Explorer.

The following editors are used together with the visualization editor to create a visualization:

1. The toolbox, page 446 providing the available visualization elements for insertion in the visualization editor: .
2. The properties editor, in which the attributes of the element that is currently highlighted in the visualization editor are displayed and edited: Visualization elements - properties, page 451,.

If at least one of the following editors is opened using the corresponding command (by default in the visualization menu), the visualization editor is divided into two sections and contains the respective tabs in the upper section it.

3. The interface editor to define placeholders if the visualization is to be referenced (added) in another visualization: Interface editor, page 491.
4. The keyboard operation editor to link actions using keys or shortcuts. Note that the device on which the visualization runs has to support these keys.
5. The element list editor containing a list of all elements in the current visualization in which elements can be selected and deleted and their position in the foreground or background can be changed.

Please note:

The presettings in the Options dialog, visualization, page 199 apply when working with visualizations in the programming system. For example, a grid can be stored in the editor window.

After it is added to the visualization editor, an element can be selected and its position and size can be modified via drag&drop. Its direction and order (background, foreground...) can be defined using the corresponding visualization commands, page 300, see Position, Size, Alignment, Order, page 449.

Which other attributes of an element that can also be configured depend on the element type.

All element properties are displayed in the properties editor and can be edited there; see Visualization elements - properties, page 451.

4.16.2 Visualization Elements, Toolbox

The **Tools** window (ToolBox) used together with the visualization editor provides visualization elements to add to the editor window. The elements are provided in libraries, page 632, in the project; the selection in the tool window depends on the currently active visualization profile, page 633,.

The tool window is opened by default when the visualization editor is opened. It can be opened explicitly using the "Tools" command that can be found in the "View" menu by default. The tool window contains the categories "Visualization", "Complex Controls" and "Windows Controls" and its respective visualization elements listed along with symbol and name. Visualization elements can be added to the editor window via **drag&drop,**, i.e. selecting and dragging them.

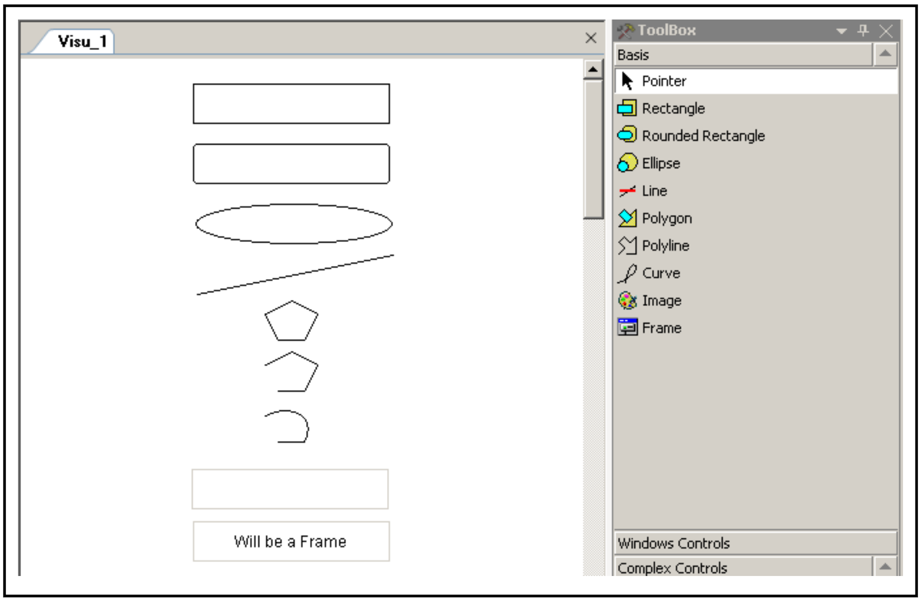


Fig.4-168: Example - Tools in the "Visual" category and added elements

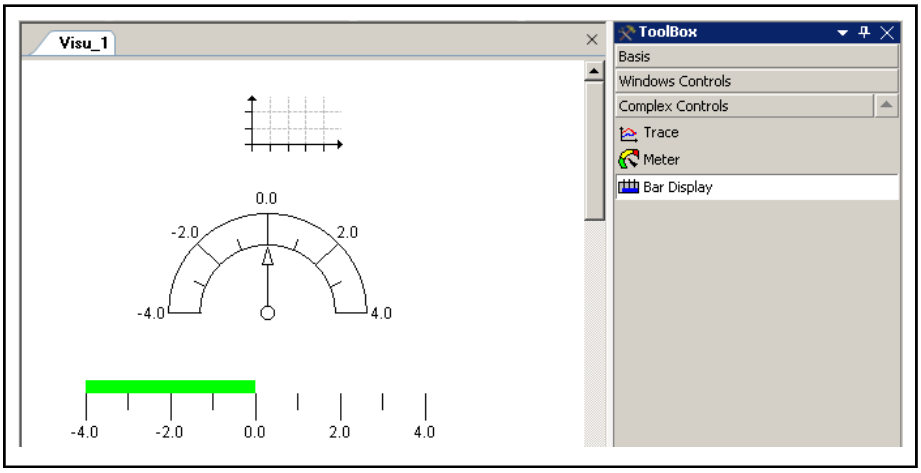


Fig.4-169: Example - Tools in the "Complex Controls" category and added elements

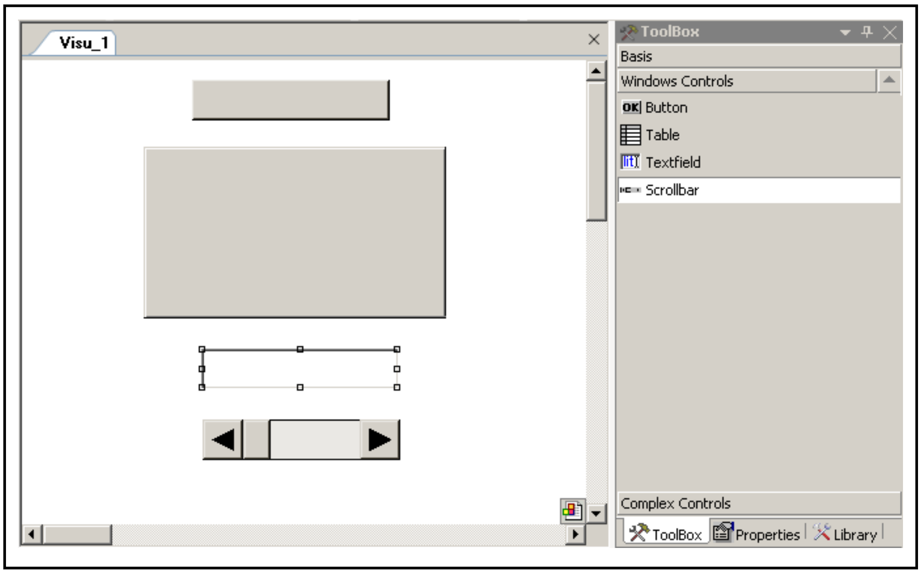


Fig.4-170: Example - Tools in the "Windows Controls" category and added elements

A plus sign at the cursor icon indicates that an element was just dragged into the editor window. After releasing the mouse button, the element is inserted there.

To add other visualizations in the currently highlighted frame element, open the respective selection dialog via **VI-Logic Visualization ▸ Frame Selection**. Refer to the following figure.

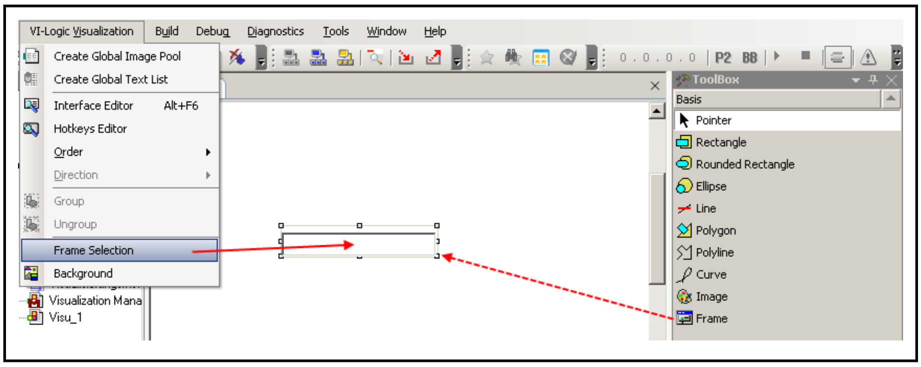


Fig.4-171: Frame Selection

Alternatively, also use "Frame Selection" in the context menu.

"Visualization commands" contain further information on the "Frame Selection"; see Frame Selection, page 302.

When an element is added, highlight it and modify its position, size, order and alignment directly in the editor. See Position, Size, Alignment, Order, page 449.

Further parameters can be edited in the "Properties" window; see Visualization elements - properties, page 451.

The default library for visualization elements is "VisuElems.library". See also Visualization libraries, page 632.

4.16.3 Selecting Elements

Mouse operation: To select a visualization element in the visualization editor, drag the mouse over the element. If the cursor icon displays a hand, press the mouse button.

Multiple selection is possible. To do this, press the <Shift> key and "click on" the individual elements or press the mouse button and drag the mouse around the elements to draw a rectangle.

In the "Visualization commands", page 300,, also refer to the commands "Select all", page 301, and "Deselect all", page 301.

A selected element displays a small black rectangle in a frame that you can click to "select" and move to modify the position or size of the element.

Keyboard operation: If an element is selected, the <Tab> key can be used to move the selection to the next element in the insertion sequence (Z order); <Shift> + <Tab> moves it to the next one.

The <space bar> opens a text input field in the currently selected element. When a character string is entered, <Enter> is used to accept it for the element ("Text" element property).

The properties of a selected element are displayed immediately in the "Properties" window and can be edited there; see Visualization elements - properties, page 451.

If several elements are selected, changes to the properties are applied to all highlighted elements.

4.16.4 Position, Size, Alignment, Direction

After adding a visualization element, page 446,, modify the following properties directly in the editor window. Size and position can also be edited in the "Properties" window:

Position Click on the element to highlight it and without releasing the mouse button drag it to the desired position or click again on an element that is already selected and move it while holding down the mouse button or by using the arrow keys.

Another option is to edit the "Position" values in the Element Properties, page 109,.

Size Highlight the element and then click on one of the small squares in its frame line. The double arrow indicates in which direction the element can be enlarged or reduced.

Another option is to edit the "Size" values in the Visualization elements properties, page 451.

Order This refers to the arrangement of elements on different levels in the editor window between foreground (front) and background (back). To move an element to another level, use **VI Logic Visualization ▸ Order** in the main menu.

Alternatively, these commands are available in the respective context menu.

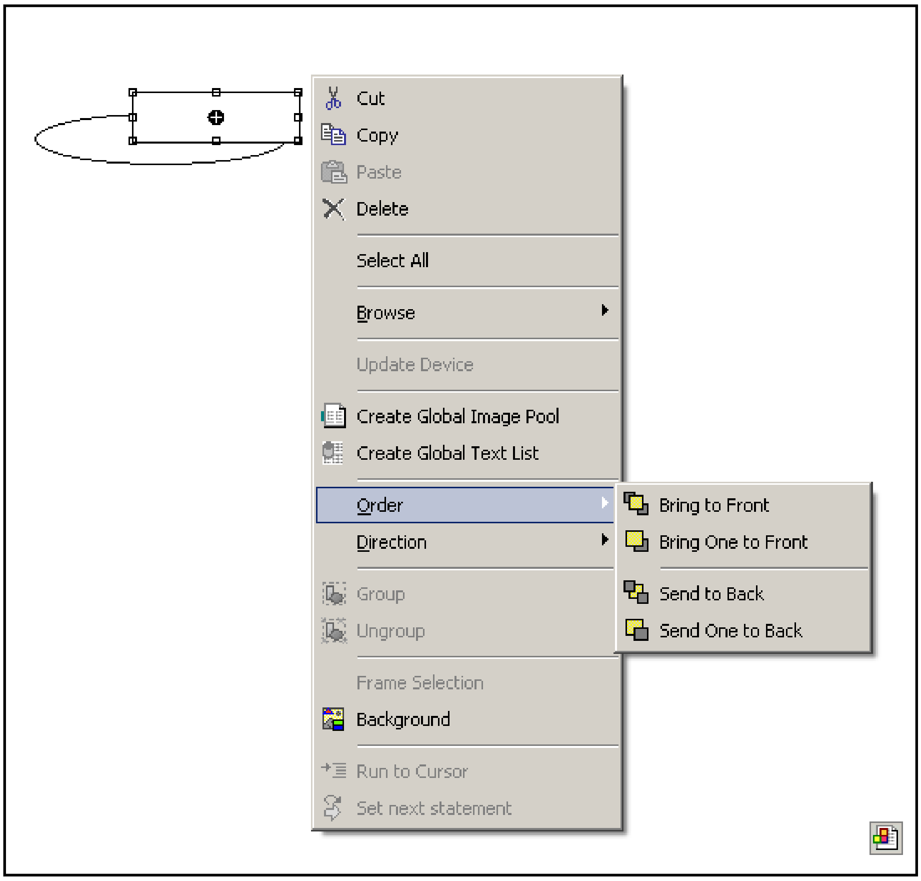


Fig.4-172: Selecting the order

Alignment

In order to align two or more elements with respect to their positions at the right, left, top, bottom or center (horizontal or vertical), use **VI Logic Visualization ▸ Direction**.

Alternatively, these commands are available in the respective context menu:

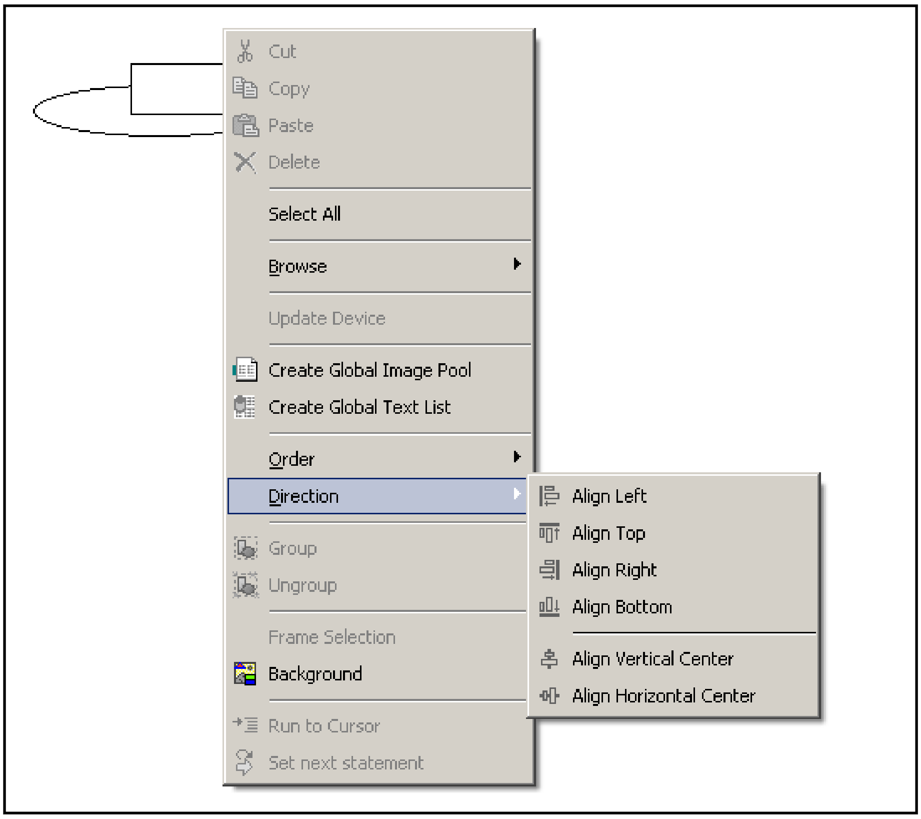


Fig.4-173: Selecting the direction

By default, the commands, page 300, for working in the visualization editor are provided in the main menu "VI Logic Visualization", see visualization commands, page 300.

4.16.5 Grouping Elements

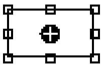
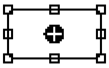
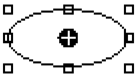
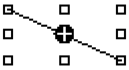
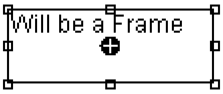
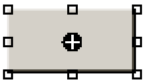
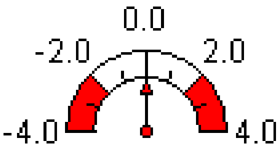
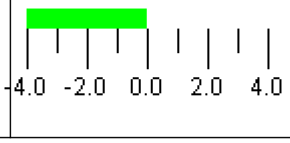
Multiple elements can be selected and combined to a group. To do this, see the description of the visualization commands Group, page 301, and Ungroup, page 301. Subelements of a group can only be selected using the element list.

4.16.6 Visualization Elements - Properties

When the visualization editor is open, visualization elements from the "Tools" window can be added: Toolbox, page 446.

The following table describes the default elements provided with the **VisuElems.library** library.

R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
M= Meter, BD=Bar Display , Tr=Trace
Ta= Table, Te=Text field, Sc=Scrollbar
P=Polygon, pL=Polyline, C=Curve		

Rectangle R		Rectangle, rounded rectangle and ellipse belong to the same type of element. Each shape can be converted into the other by changing the "Type" property.
Rounded Rectangle rR		
Ellipse E		
Line L		Line: The slope of a "line" element can be switched between "upper left - lower right" and "lower left - upper right".
Frame F		Frame: A frame, like the familiar "frames" used on internet pages for example, defines a section of the current visualization to which one or multiple other visualizations are assigned of which one is displayed in online mode. In principle, a frame can be configured as a rectangle element. With the correct configuration, a user input can be used to switch among the visualizations displayed. Assign visualizations to a frame using the configuration of frame visualizations (see frame selection in visualization commands, page 302). All available visualizations are provided for selection. For information on the configuration of visualization switching by user inputs, refer to the possible mouse actions, page 476.
Button Bu		An image and a "height" (for the relief view) can be assigned to a button element.
Image Im		An image element is filled by an image file defined by an ID and name of the image pool in which it is managed. The file specification can also be dynamically configured, i. e. using a project variable. Also refer to , page 61
meter M		The "Meter" element inserts a tachometer into the visualization whose minimum and maximum display value can be entered. The needle position indicates the current value of the linked input variable. Specific background colors can be defined for certain value ranges.
Bar display BD		The "Bar Display" element inserts a bar graph into the visualization for which a minimum and maximum display value can be entered. The length of a the bar indicates the current value of the linked input variable. Specific colors can be defined for certain value ranges. A horizontal bar display is preset. After the element has been inserted, the orientation can be changed to vertical in the element properties.

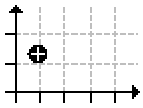
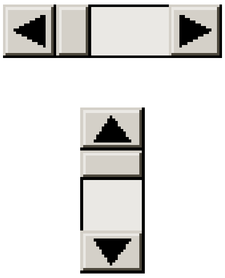
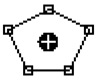
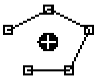
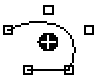
Trace Tr		This element allows a trace to be integrated into a visualization. The name of the trace to be displayed is specified in the element properties. However, the recorded variables are configured in the configuration of the trace. Refer to the Configuration in the Trace Editor, page 437.
Table Ta		The "Table" element displays the values of an array, a structure or the local variables of a function block. A column or line displays the elements of a one-dimensional array or structure or POU. Two-dimensional arrays and arrays with structures or POUs as elements are displayed in a matrix structure (columns and lines). A basic type variable can be understood as a one-dimensional array with an element and thus also displayed using the table element (as table with a single entry).
Text field Te		The "Text field" element is used to display text that is either entered directly into the element properties or that origins from a text input variable. In contrast to the normal rectangle, the frame can be displayed as shadowed.
Scrollbar Sc		The "Scrollbar" element generates a scroll bar whose minimum and maximum value can be defined. The position of the controller is then linked to the value of the input variables. If an output variable is defined, its value can be written by manually positioning of the controller. By default, the scrollbar is horizontal. If the shape of the element changes (by dragging it with the mouse) after the scrollbar has been inserted so that the height of the element is greater than its width, the scrollbar becomes vertical.
Polygon P		Polygon, polyline and curve belong all to the same element type. Each shape can be converted into the other by changing the "Type" property. An additional position point that specifies the segments and the shape of an element can be inserted by clicking on an existing point (small rectangle on the frame line) while holding down the <Ctrl> key. A point can be deleted by clicking on the point while pressing <Alt> + <Ctrl>. In a curve element, the points next to the line are used as "grips" to modify the curve shape.
Polyline pL		
Curve C		

Fig.4-174: Standard elements in the VisuElems.library

After an element has been added to the current visualization, highlight it and modify its position, size, order and alignment in the editor (mouse actions or visualization commands). See Position, Size, Alignment, Order, page 449. Configure other parameters and the visualization behavior in the "Properties" editor. See Visualization elements - properties, page 451.

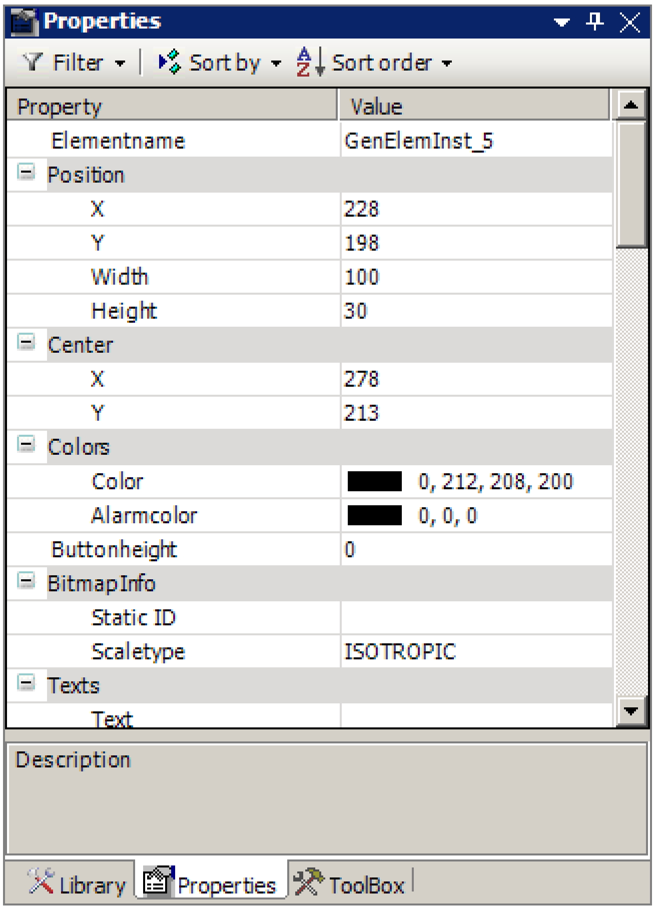
"Properties" editor for visualization elements

The properties of a visualization element - except the "alignment" and "order" - can all be configured in the "Properties" editor.

By default, the "Properties" editor opens next to the visualization editor or explicitly via **View ▸ Other Windows ▸ Element Properties.**

A property can be modified by editing the "Value" field.

To do this, depending on the element type, an input field, a selection list or a dialog is provided in this field which can be open with a double-click or the <space bar> if the field is already selected.

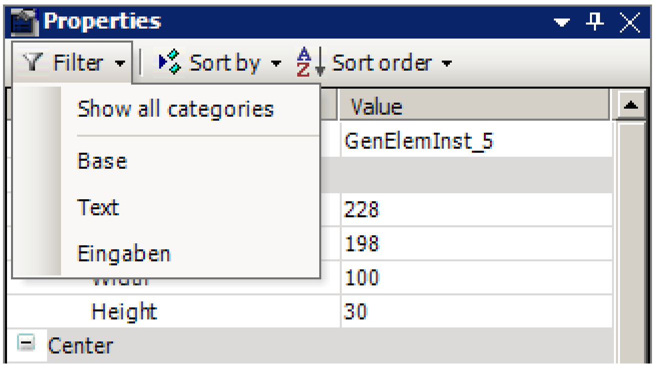


Property	Value
Elementname	GenElemInst_5
⊟ Position	
X	228
Y	198
Width	100
Height	30
⊟ Center	
X	278
Y	213
⊟ Colors	
Color	■ 0, 212, 208, 200
Alarmcolor	■ 0, 0, 0
Buttonheight	0
⊟ BitmapInfo	
Static ID	
Scaletype	ISOTROPIC
⊟ Texts	
Text	

Fig.4-175: Example of the "Properties" editor

For properties in the "Input" category (OnDialogClosed) or "Input" (OnMouse) actions, the current configuration is displayed in one or multiple lines below the "Configure..." field. To edit an individual subconfiguration, click on the corresponding line to open a simplified configuration dialog. To modify the complete configuration for the mouse action, click in the "Configure..." field for the detailed default configuration dialog.

Working in the list of properties can be facilitated using the default "sorting and filtering functions".



	Value
	GenElemInst_5
	228
	198
	100
Height	30
⊟ Center	

Fig.4-176: Example of the "Properties" editor

Select one of the following options to define the properties to be displayed:

Filter menu:

- **All categories:** All properties.
- **Base:** Only the basic properties, e.g. position, center point, colors.
- **Text:** Only text properties.
- **Input:** Only the properties related to input to the element.

Sort menu:

- **Sort by type:** All properties are displayed. The categories are arranged in original order.
- **Sort by name:** All properties are displayed. The categories are arranged alphabetically from top to bottom.

Order menu:

- **Sort in ascending order:** All properties are displayed and the categories are arranged in the original order from top to bottom.
- **Sort in descending order:** All properties are displayed and the categories are arranged in the original order from bottom to top.

Columns (only for the "Table" element)

The "Table" element visualizes an variable of the types array or structure.

The index of the array or the components of the structure are displayed in a column or line.

For two-dimensional arrays or an array of structures in several columns.

The appearance of the table columns, in which the values of the individual array elements are displayed, is defined. An individual configuration is possible for each column that belongs to a specific index or individual component.

Column		Table (Ta)
[index]	Column index of the "Table" element; e.g. "[0]" for the only column that displays the elements of array[2..8] of INT; e.g. "[0]" , "[1]" or "[2]" for the three columns that display the elements of array[1..3][1..10] of INT.	Ta
Column header	By default, the name of the array/structure along with the index/component belonging to the column are used as a header. The column label can be changed by entering a new title here.	Ta
Column width	Column width	Ta
Text in headline	Alignment of column label: • LEFT • CENTER • RIGHT	Ta
Use template	When the checkbox is selected, another visualization element is added to each cell of this table column (element type Rectangle, Rounded Rectangle or Circle). The "Properties" list is automatically extended to include the properties of this element which can be configured as usual.	Ta
Use text alignment in headline	Selecting this checkbox is only effective if the "Use template" checkbox is also selected at the same time: When it is activated, the settings made in the inserted template for font (size) and alignment are also applied to the column header.	Ta
Row header	Switches the display of the line labeling on or off (as line indices of the array displayed)	Ta

Column		Table (Ta)
Column header	Switches the display of the defined column header (under "Column header") on or off	Ta
Row height	Height of the lines in the table	Ta
Row header width	Width of the column with line labeling	Ta
Scrollbar size	Width of the scrollbar	Ta

Fig.4-177: Columns (only for the "Table" element)

Image ID, Show frame, Clipping, Transparent, Transparent color, Scaling type, Trace

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Static ID	Identifier for the image file used for this element (static definition). Enter the image file ID as it is defined in the corresponding image pool (STRING). To make the entry unique, the name of the pool should be added at the beginning (this is not necessary if the image file is managed in the image pool, since this pool is always searched first). Example: "imagepool2.button_image". For information on managing image files in image pools, see "Concepts and basic components", Image Pool, page 61..	`Im`
Show frame	If this option is enabled, the image file is displayed with a frame.	`F, Im`
Clipping	If this option is enabled with the "Scale type" setting as "Fixed" (see below), only the section of the visualization that fits into the frame is displayed in case of a "Frame" element.	`F, Im`
Transparent	If this option is enabled, the color entered in the "Transparent color" property (see below) is displayed as transparent.	`Im`
Transparent color	The ... button opens the color selection dialog to select a color that is to be displayed as transparent if the "Transparent" option (see above) is enabled.	`Im`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Scale type	Definition of how the image file is supposed to react to changes in the element frame size. ISOTROPIC: The image keeps its proportions; i.e. the height-width ratio remains even if the height and width of the element frame are modified separately. ANISOTROPIC: The image adjusts to the size of the element frame; i.e. height and width can be modified independently of each other. FIXED: The image keeps the original size even if the size of the element frame changes. Note whether the "Clipping" option (see above) is also enabled!	`F, Im`
Trace	Name of the trace used as input for the trace visualization element. Enter the complete name of the trace, i.e. including the path of the trace within the device tree, e.g. "PLCWinNT.Application.Trace".	`Tr`

Fig.4-178: Basic properties:

Position

Position, Width, Height, Rotation, ShowCursor, Position points (X, Y, Width, Height, points)

☞ x/y = 0/0 designates the upper left corner of the editor window, x-axis=horizontal axis, y-axis=vertical axis.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

X	Horizontal position in pixels. 0=left window edge	`R, rR, E, L, F, Bu, Im`
		`M, BD, Tr`
		`Ta, Te, Sc`

Y	Vertical position in pixels. 0=upper window edge	`R, rR, E, L, F, Bu, Im`
		`M, BD, Tr`
		`Ta, Te, Sc`

Width	Width of the element in pixels	`R, rR, E, L, F, Bu, Im`
		`M, BD, Tr`
		`Ta, Te, Sc`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Height	Height of the element in pixels	`R, rR, E, L, F, Bu, Im`
		`M, BD, Tr`
		`Ta, Te, Sc`

ShowCursor	If the checkbox is selected, the values of the monitored variables (trace variables) are displayed at the cursor position. See the example in Visualization of a trace, page 643.	`Tr`
Points		
[0], X / Y ..... [n], X / Y	X/Y positions (pixels) of a specific point. [0] is the position of the starting point. The following points are numbered in their order.	`P, pL, C`

Fig.4-179: Position, width, height, rotation, position points

Center

Center, center (X, Y)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

X	X position of the pivot point (center, ⊕) of the element	`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Y	Y position of the pivot point (center, ⊕) of the element	`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Fig.4-180: Center (X, Y)

Colors

Colors (Color, Bar color, Bar background, Framecolor, Alarm value, Condition, AlarmColor, Use color areas, Durable color areas, Begin of area, End of area, Color, Framecolor, Fillcolor, Normalstate, Alarmstate)

Select a color from the selection list or use the color selection dialog (… button).

The definition of color values, page 490, is described below.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Color	Static definition:
	Color of the element in normal state.
		`L, F, Bu, Im`
		`Sc`

Bar color	Color of the bar	`BD`
Frame color	Color of the frame around the bar display if the "Element frame" checkbox was selected	`BD`
AlarmColor		
Alarm value	Limit that indicates an alarm state	`BD`
Condition	The alarm state is triggered if the current value of the input variables exceeds the "Alarm value" (GREATER_THAN) or falls below it (LESS_THAN)	`BD`

AlarmColor	Static definition:
	Color of the element in alarm state, i.e. if the corresponding variable is TRUE.
		`L, F, Bu, Im`
		`BD, Sc`

Use color areas	When this checkbox is selected, the specified color areas are displayed	`BD`
Color areas	Clicking the Create new key creates a new color area whose properties can be edited in the fields that open below	
Durable color areas	By default, the only color area displayed is the one with the current value of the input variable. When this checkbox is selected ,all color areas created are displayed.	`M`
Areas	This folder contains all color areas created.	
[Number]	Ascending numbering of the color area. It cannot be edited. After ✕ Delete is clicked, the related color area is deleted and the numbering is reassigned.	
Begin of area	Value that provides the lower limit for the color area	`M, BD`
End of area	Value that provides the upper limit for the color area	`M, BD`
Color	Color in which the color area is to be displayed	`M, BD`
Normal state		
Frame color	Static definition: Color of the element in normal state.	`R, rR, E`
Fill color	Static definition: Fill color of the element in normal state.	`R, rR, E`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Alarm state		
Frame color	Static definition: Color of the element frame in alarm state.	`R, rR, E`
Fill color	Static definition: Fill color of the element in alarm state.	`R, rR, E`

Fig.4-181: Colors

Element look

Element look (appearance) – Line thickness, line type, slope, fill attributes, frame attributes (line width, line style, FromTopLeft (declination), fill attributes, frame attributes)

These are static definitions with regard to the frame line and element filling.

For dynamic definition of these properties, refer to the "Look variables" category.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Line width	Thickness of the frame line in pixels; "0" corresponds to "1"	`R, rR, E, L, F, Im`
		`Te`
		`P, pL, C`

Line style Y	Line type: PS_SOLID ____ PS_DASH _ _ _ PS_DOT . . . PS_DASHDOT _ . _ . PS_DASHDOTDOT _ . . _ . . PS_HOLLOW (not visible)	`L`
FromTopLeft	If this option is enabled, the line is drawn from upper left to lower right. Otherwise, it ascends from lower left to upper right.	`L`

Fill attributes	Frame line visibility:
	BS_SOLID: Visible
	BS_HOLLOW: Invisible
		`R, rR, E`
		`Te`
		`P, pL, C`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Frame attributes	Type of frame line: PS_SOLID ____ PS_DASH _ _ _ PS_DOT . . . PS_DASHDOT _ . _ . PS_DASHDOTDOT _ . . _ . . PS_HOLLOW (not visible)	R, rR, E, F, Im Te P, pL, C
m_ShadowType	Type of shadowing the outer line: DEEPEND (element appears deepend) NONE (without) RAISED (element appears raised)	Te

Fig.4-182: Appearance

Button height Button height – Button (only for button)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Button height	Determines the relief view of the button element; "height" in pixels ( )	Bu

Fig.4-183: Button

Bitmap info Bitmap info – Image file (only for button: Static ID, scale type)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Static ID	Refer to the table in Visualization elements - properties, page 451.	Bu
Scale type	Refer to the table in Visualization elements - properties, page 451.	Bu

Fig.4-184: Bitmapinfo

Texts Texts – Tooltip (Text, Tooltip)

Static definition of the element text. For a dynamic definition, see the "Text variables" and "Dynamic texts" categories.

For definitions of font and alignment, refer to the "Text properties" (static) and "Font variables" (dynamic) categories.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Text	Text to appear in the element.
	Formatting options are located in Text and language in visualizations, page 628.
	Example: "On/Off"
	Also refer to the "Text variables" and "Dynamic texts" categories.
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te`
		`P, pL, C`

Tooltip	Text to appear as tooltip in online mode when the cursor is positioned on an element.
	Example: "On/Off" button with "Machine 1 on/off" tooltip.
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Fig.4-185: Texts

Text properties Text properties – HorizontalAlignment, VerticalAlignment, Font

These are static definitions for a dynamic definition of font (see the "Font variables" category).

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

HorizontalAlignment	Horizontal alignment of the element text:
	LEFT / HCENTER / RIGHT
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

VerticalAlignment	Vertical alignment of the element text:
	TOP / VCENTER / BOTTOM
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Font	Font type, color and size of the element text. The default dialog for font selection can be opened via the ... button	`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Fig.4-186: Text properties

Arrow (only for Meter: Arrow type, Arrow color, Arrow start, Arrow end, Additional arrow)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Arrow type	The drop-down menu provides four different shapes for the arrow: • Thin needle (THIN_NEEDLE) • Thin arrow (THIN_ARROW) • Normal arrow (NORMAL_ARROW) • Wide arrow (WIDE_ARROW)	M
Arrow color	Arrow color	M
Arrow start	Angle (in degrees) between the left margin of the tachometer and the horizontal axis	M
Arrow end	Angle (in degrees) between the right margin of the tachometer and the horizontal axis	M
Additional arrow	When this checkbox is selected, a second arrow is displayed within the scale opposite the first arrow.	M

Fig.4-187: Arrows

Bar (Diagram type, Orientation, Running direction)

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	
Diagram type	The drop-down-up menu provides three possibilities to position the bar with regard to the scaling: Bar is located within the scaling (BAR_INSIDE_SCALE) Scaling is located within the bar (SCALE_INSIDE_BAR) Scaling is next to the bar (SCALE_BESIDE_BAR)	`BD`
Orientation	Select between horizontal (HORIZONTAL) and vertical (VERTICAL) bar alignment. If there is a scrollbar, the orientation (HORIZONTAL or VERTICAL) results from the ratio between scrollbar length to scrollbar width and cannot be edited.	`BD, SC`
Running direction	Select from these options: LEFT_RIGHT (bar grows from left to right) and RIGHT_LEFT for a horizontal bar or BOTTOM_UP (for a bar that grows from bottom to top) and TOP_DOWN for a vertical bar	`BD, SC`

Fig.4-188: Bars

Scale (Scale start, Scale end, Main Scale, Sub Scale, Frame inside, Frame outside, Element Frame)

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	
Scale start	Value that provides the lower limit for the scaling display	`M, BD`
Scale end	Value that provides the upper limit for the scaling display	`M, BD`
Main scale	Distance between two lines on the large scale.	`M, BD`
Sub scale	Distance between two lines on the fine scale. It can be set to 0 if the large scale should not to be subdivided.	`M, BD`
Frame inside	This checkbox is selected by default. If it is disabled, the lower margin of the curved scaling display is hidden.	`M`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Frame outside	This checkbox is selected by default. If it is disabled, the upper margin of the curved scaling display is hidden.	`M`
Element Frame	By default, this checkbox is disabled. If it is enabled, a frame is drawn around the bar element	`BD`

Fig.4-189: Scaling display

Label (Unit, Font, Scale Format)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Label Position	Select "OUTSIDE" to position the scaling label outside the scale and "INSIDE" to position it on the inside	`M`
Unit	String displayed below the arrow (M) or below the center point of the scaling (BD) (intended for entering scaling unit)	`M, BD`
Font	Font in which unit and scaling label are displayed	`M, BD`
Scale Format (C-syntax)	Use the C-syntax to enter the formatting for the scaling label; e.g. enter the string "%3.2f s" in this field to display the scaling labels with 3 places, 2 of which are after the decimal point followed by the letter "s"	`M, BD`
max. text width of labels	Value that specifies the maximum width of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.	`M`
text height of labels	Value that specifies the maximum height of the scaling label. Usually, this value is automatically specified correctly. Only use this specification if the automatic adjustment does not lead to the desired result.	`M`

Fig.4-190: Scaling label

Bitmap ID variable Bitmap ID variable (image ID) – only for image element

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Bitmap ID	Project variable that defines the image file ID. See also Image Pool, page 61, in "concepts and basic components". Click on the … button to open the "Input assistance" dialog.	`Im`

Fig.4-191: Bitmap ID variable

Absolute movement

Absolute movement – Movement (X, Y) Rotation, Scaling, Interior rotation

Absolute movement: The element can be moved by modifying the x- and y-positions (pixels) of the upper left corner of the element by an integer variable. Absolute coordinate values are used here. (For contrast, see: "Relative movement" for moving specified corners/side lines of an element with respect to a fixed origin).

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Movement		

X	The integer variable entered defines the current x-position of the upper left corner of the element in pixels. It can be used to move the element in x-direction.	`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Y	The integer variable entered defines the current y-position of the upper left corner of the element in pixels. It can be used to move the element in the y-direction.	`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Rotation	The integer variable entered defines the angle (in degrees) around which the element rotates around the rotation point;
	positive values=mathematically positive=clockwise.
	Note: The element itself does not rotate in contrast to the "interior rotation" (see below).
	The point of rotation (center ⊕) becomes visible when clicking with the mouse on the element. It can be moved with the mouse button pressed.
		`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Scaling	The integer variable entered defines the current scaling factor (percent). The element size is modified (linear) with respect to this value. The value is divided implicitly by 1000 so that it is not necessary to use REAL variables to reduce the size of the element. The scaling always relates to the point of rotation (center).
	The point of rotation (center ⊕) becomes visible when clicking with the mouse on the element. It can be moved with the mouse button pressed.
		`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Interior rotation	The integer variable entered defines the angle (in degrees) around which the element is rotated around its rotation point. Positive values=mathematically positive=clockwise.
	In contrast to "Rotation" (see above), the element itself rotates.
	The point of rotation (center ⊕) becomes visible when clicking with the mouse on the element. It can be moved with the mouse button pressed.
		`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Fig.4-192: Absolute movement

Relative movement

Relative movement – Movement top left (X,Y), Movement bottom right (X, Y)

The top, left, bottom or right edge of the element is moved in x- or y-direction based on the value given by an integer variable (pixels). In contrast to absolute movement (see above), a relative position is defined, i.e. the distance to the original position. The shape of the element can thus be modified.

Positive values move the horizontal edges down and the vertical edges to the right.

See above: "Absolute movement" to move the entire element.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Movement topleft		
X	Number of pixels moved by the left edge in x-direction.	`R, rR, E, L, F, Bu, Im`
Y	Number of pixels moved by the top edge in y-direction.	`R, rR, E, L, F, Bu, Im`
Movement bottomright		

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

X	Number of pixels moved by the right edge in x-direction.	`R, rR, E, L, F, Bu, Im`
Y	Number of pixels moved by the bottom edge in y-direction.	`R, rR, E, L, F, Bu, Im`

Fig.4-193: Relative movement

Dynamic values

Dynamic values – For elements with multiple position points: pPointArray, Count)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

pPointArray	The variable entered has to be a pointer to an array of the "VisuStructPoint" structure. The "iX" and "iY" components of "VisuStructPoint" describe the x-/y-coordinates of a point of the element. The array writes all points of the element and the structure can be filled dynamically from the project. Note that the number of element points has to be defined explicitly (see below), since the usage of a point to define the points dynamically does not allow the number of points to be checked. Example: `pPoints : POINTER TO ARRAY[0..100] OF VisuStructPoint;`	`P, pL, C`
Count	An integer variable has to be entered to define the number of element points. Example: `iCount : INT:=24;` This means that the element has 24 individual points. This specification is necessary, since the individual points are defined via a pointer (see "pPointArray" above) which does not allow the number to be checked.	`P, pL, C`

Fig.4-194: Dynamic values

Text variables

Text variables – Text Variable, Tooltip Variable

These are dynamic definitions. See also the "Dynamic texts" category for the usage of text lists.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Text Variable	String variable that defines the element text.
	If a table is used, the text is positioned in the foreground crosswise over the entire table.
	Also refer to
	● Format text, page 628
	● Dynamic texts, page 468
		`R, rR, E, L, F, Bu, Im`
		`Te, Sc`
		`P, pL, C`

Tooltip Variable	String variable that defines the tooltip text for the element	`R, rR, E, L, F, Bu, Im`
		`Te, Sc`
		`P, pL, C`

Fig.4-195: Text variables

Dynamic texts

Dynamic texts –Text list, Text index, Tooltip index

These parameters define dynamic texts generated by test lists (this allows language selection). Further information on the usage of text lists is located in "Concepts and basic components", see Text List, page 55.

Another option of a dynamic text definition is that the text can be provided via a string variable, refer to the "Text variables" category.

For information on text definitions, refer to the "Texts" category.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Text list	Name of the text list (string) as used in the project tree; e.g. "errorlist".	`R, rR, E, L, F, Bu, Im`
		`Te, Sc`
		`P, pL, C`

Text index	Index (ID) of the text as defined in the text list (String);
	Direct specification of the ID (e.g. "M1") or string variable (e.g. "error_22")
		`R, rR, E, L, F, Bu, Im`
		`Te, Sc`
		`P, pL, C`

Tooltip index	Index (ID) of the tooltip text as defined in the text list (string);
	Direct specification of the ID (e.g. "T1") or string variable (e.g. "error_tooltip_22")
		`R, rR, E, L, F, Bu, Im`
		`Te, Sc`
		`P, pL, C`

Fig.4-196: Dynamic texts

Font variables

Font variables – Font name, Height, Flags, CharSet, Color

These are dynamic definitions on the font of the element text via a project variable.

For information on static definitions, see the "Texts" category.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Font name	String variable that indicates the name of the font used for the element text (name specification as in the default font dialog);
	e.g. "PlcProg.font_var" (`font_var := 'Arial';`)
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Height	Integer variable that defines the size of the element text in pixels (specification as in the default font dialog);
	e.g. "prog1.font_height" (`font_height := 16;`)
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Flags	DWORD variable that specifies the font appearance using the flag values listed below. A combined definition can be achieved by adding the respective flag values and entering the sum:
	● 1 italic
	● 2 bold
	● 4 underlined
	● 8 canceled
	e.g. "prog2.font_type" (if `font_type := 6;` the text is displayed in bold and underlined)
		`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

CharSet	The character set to be used for the font can be defined using the default character set number. A DWORD variable can be used to specify this number (see also the "Script" definition in the default font dialog).	`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Color	A DWORD variable can be used to define the color of the element text.	`R, rR, E, L, F, Bu, Im`
		`Ta, Te, Sc`
		`P, pL, C`

Fig.4-197: Font variables

Color variables

Color variables – ToggleColor, Color, Alarm color, Normal state (Frame color, Fill color) Alarm state (Frame color, Fill color)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

ToggleColor	Boolean variable that switches the color between "Color" (FALSE) and "Alarmcolor" (TRUE).	`R, rR, E, L, F, Bu, Im`
		`Ta`
		`P, pL, C`

Color	DWORD variable that defines the fill color of the element (attention: It overwrites the value currently set by "Colors/Normal state"). The color "Color" is used if the variable in "ToggleColor" (see above) has the value FALSE. See also Definition of color values, page 490.	`L, Bu, Im`
Alarm color	DWORD variable that defines the fill color of the element (attention: It overwrites the value currently set by "Colors/Alarm state"). The color "Alarmcolor" is used if the variable in "ToggleColor" (see above) has the value TRUE. See also Definition of color values, page 490.	`L, Bu, Im`
Normal state		

Frame color	Color of the element frame to be defined as described above for "Color".	`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Fill color	Fill color of the element in normal state to be defined as described above for "Color".	`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Alarm state		

Frame color	Color of the element frame to be defined as described above for "Alarmcolor".	`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Fill color	Fill color of the element in alarm state to be defined as described above for "Alarmcolor".	`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Fig.4-198: Font variables

Look variables

Look variables (appearance variables) – Line width, Fill attributes, Frame attributes

These are dynamic definitions for the appearance of the frame lines and filling of the element.

For information on static definitions, refer to the "Elementlook" category.

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	

Line width	Integer variables that define the line thickness for the element in pixels.	`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Fill attributes	DWORD variable that defines the filling of the element. The color currently defined by color variables can be used or not to be used:
	0 = Use color
	>0 = Do not use color
		`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Frame attributes	DWORD variable that defines the element frame lines: Possible types:
	0 = solid, PS_SOLID ____
	1 = dashed, PS_DASH _ _ _
	2 = dotted, PS_DOT . . .
	3 = dashed-dotted, PS_DASHDOT _ . _ .
	4 = dash-dot-dot, PS_DASHDOT _ . . _ . .
	8 = not visible, PS_HOLLOW
		`R, rR, E, L,`
		`Ta`
		`P, pL, C`

Fig.4-199: Look variables, appearance variables

State variables

State variables (Status variables) – Invisible, Deactivate inputs

These are dynamic definitions on the availability of the element in online mode.

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	

Invisible	Boolean variable. If it returns TRUE, the element is not visible in online mode.	`R, rR, E, L, F, Bu`
		`Ta, Sc`
		`P, pL, C`

Deactivate inputs	Boolean variable. If it returns TRUE, the inputs to the element have no effect.	`R, rR, E, L, F, Bu`
		`Ta, Sc`
		`P, pL, C`

Fig.4-200: State variables, Status variables

m_pLineWidthVariable

m_pLineWidthVariable (line thickness variable) (only for line; IntValue)

Dynamic definition of the line thickness:

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

IntValue)	Integer variable that indicates the line thickness in pixels. "0" corresponds to "1" This corresponds to the fixed setting "LineWidth" in the "Elementlook" category.	L

Fig.4-201: m_pLineWidthVariable, line thickness variable

m_pPenStyleVariable

m_pPenStyleVariable (line type variable) – Only for line: IntValue)

Dynamic definition of the line type:

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

IntValue)	Integer variable that defines the line type. Possible values: 0 = solid, PS_SOLID ____ 1 = dashed, PS_DASH _ _ _ 2 = dotted, PS_DOT . . . 3 = dashed-dotted, PS_DASHDOT _ . _ . 4 = dash-dot-dot, PS_DASHDOT _ . . _ . . 8 = not visible, PS_HOLLOW This corresponds to the fixed setting "LineStyle" in the "Elementlook" category.	L

Fig.4-202: m_pPenStyleVariable, Line type variable

m_pLineDirectionVariable

m_pLineDirectionVariable (line slope variable) – Only for SimpleLine: DigitalVar

Dynamic definition of the slope of the line;

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

DigitalVar	Boolean variable that indicates if the line is to be drawn from top left to bottom right (TRUE) or from bottom left to top right (FALSE - default value). This corresponds to the fixed setting "FromTopLeft" in the "Elementlook" category.	L

Fig.4-203: m_pLineDirectionVariable, line slope variable

Selection

(only for Table: Selection color, Apply to columns, Selection type, Frame around selected cells, Variable for selection X, Variable for selection Y, Variable for valid selection X, Variable for valid selection Y)

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	
Selection		
Selection color	Fill color for selected table elements	Ta
Apply to columns	Pressing this key overwrites the "Alarmstate Fillcolor" specified in templates that would normally be displayed at the selection of elements	Ta
Selection type	Defines which section of the table is selected by clicking on one of its cells: SEL_NONE (no selection) SEL_CELL (only the cell is selected) SEL_ROW (selects the line that contains the cell) SEL_COLUMN (selects the column that contains the cell) SEL_ROW_AND_COLUMN (selects the line and column that contain the cell)	Ta
Frame around selected cells	Checkbox for switching the display of a frame around the selected elements on and off	Ta
Variable for selection X	Variable of type INT set on the array index belonging to the row of the selected element	Ta
Variable for selection Y	Variable of type INT set on the array index belonging to the column of the selected element. If there is a structure/POU, the components are counted in sequence starting with 0 (note: that the value only represents the correct position in the array if no columns were removed from the table display)	Ta
Variable for valid selection X	Boolean variable set to TRUE or FALSE based on the validity of the selected column	Ta
Variable for valid selection Y	Boolean variable set to TRUE or FALSE based on the validity of the selected line	Ta

Fig.4-204: Selection

Button state variable

Button state variable – only for button: ButtonState

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	
DigitalVar	Boolean variable that specifies whether the button element is to be displayed as "pressed" (TRUE) or "not pressed" (FALSE).	Bu

Fig.4-205: Button state variable

Isotropic

Isotropic – only for Frame

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Isotropic	See Static ID, page 456	F

Fig.4-206: Isotropic

Clipping

Clipping (only for Frame)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Clipping	See Static ID, page 456	F

Fig.4-207: Clipping

References

References, Frame visualizations (only for Frame: references, visualizations)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

References		
Visualizations	The visualizations assigned to the frame using the "Configuring frame visualizations" dialog are listed; input cannot be made!	F

Fig.4-208: References

Switching Variable

Visualization of a frame is switched with this property.

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Switching variable		
Integer value	Visualizations assigned to a frame can be switched with a variable. The value (ID) of the visualization is determined by the order of these elements in the list of the assigned visualizations in the dialog "Configuration of Frame Visualizations". The first entry in this list results in the value 0 for the integer value, the second value results in 1, etc.	F

Fig.4-209: References

Inputs

User inputs (OnMouse actions: OnMouseDown, OnMouseLeave, OnMouseEnter, OnMouseUp, OnMouseMove, Toggle: Variable, Toggle on up if captured, Tap: Variable, Tap False, Tap on enter if captured).

Mouse actions: Open dialog, close dialog, switch frame visualization, write variable, toggle variable, change language, execute ST code, switch visualization and internal command (execute program, print, recipe management)

Here is defined was should happen if the user enters input on the respective element in online mode using the mouse.

Different possibilities include the "OnDialogClosed" action, "OnMouse" actions and the option to "toggle" or "tap" variables as well as a shortcut via input.

- **OnDialogClosed:** This entry can be made by closing a dialog in the visualization previously open for a user input. One or multiple of the resulting actions, page 479, described below can be specified. See also the descriptions of the resulting actions "Open dialog" and "Close dialog" below used to open and close an input dialog via mouse action on a visualization element.
- **OnMouse actions:** Here, each of the various mouse actions can be assigned to one or multiple of the resulting actions, page 479, defined below: Until a resulting action is defined, "Configure" appears in the "Properties" field. Click on this field to open the "input configuration" dialog to configure the resulting action(s) (see below). Assigned resulting actions are displayed in the "Properties" dialog below the element property.
- **Toggle, Tap:** Boolean project variables are specified which should become TRUE or FALSE when clicking on them or when clicking on and releasing the mouse button on the element.
- **Hotkey:** Here a key can be linked with a specific action (MouseDown, MouseUp) to be executed with the corresponding movement of the key (KeyDown, KeyUp).

  By default, "KeyDown" is used to execute the "MouseDown" action and "KeyUp" for the "MouseUp action". This can be useful if a visualization is to be operated using both mouse and keyboard inputs, since the input actions only have to be configured once. This hotkey for an element is also managed in the hotkey editor, page 494 of the visualization. Changes are always synchronized between this editor and the "element properties" editor.

See the following configuration options:

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

OnDialogClosed	Defines what happens if a dialog that was previously opened via the input to a visualization element is closed again. For example, the option "Execute ST Code" could be used to implement a certain reaction to the "Result" of the most recently closed dialog (this result, e.g. "OK" or "Cancel" is specified in the "Close dialog" input configuration of an element);
	Note:
	The "OnDialogClosed" property is not limited to the element for which it is configured, but applies instead to the entire visualization, i.e. it reacts to each "close dialog action". Currently, there is no way to define such a property for the entire visualization and for this reason, it has be assigned to one of its elements.
	In Input dialogs, page 663, there is further information and an example about the usage and evaluation of user-defined dialogs.
		`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouse actions		

OnMouseClick	If the cursor points to the element, the mouse button is pressed and immediately released. *	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouseDown	Mouse action: The mouse button is pressed if the cursor is pointing to the element *	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouseLeave	Mouse action: The pressed mouse button is released if the cursor is still pointing to the element *	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouseEnter	Mouse action: The mouse cursor is dragged onto the element.	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouseUp	Mouse action: The mouse button is pressed as long as the cursor was not pointing to the element and is released after the cursor was dragged onto the element *	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

OnMouseMove	Mouse action: The mouse is moved *	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

Toggle		

Property	**R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image** **M= Meter, BD=Bar Display , Tr=Trace** **Ta= Table, Te=Text field, Sc=Scrollbar** **P=Polygon, pL=Polyline, C=Curve**	

Variable	Here, enter a Boolean variable that is to switch between TRUE and FALSE each time the element is clicked. The input assistance can be opened using the ... button. The input assistance is available after a double-click on the field.	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

Toggle on up if captured	● Option disabled (default option): If the mouse button is pressed while the cursor is pointing to the element, but then the cursor is removed from the element before the button is released again, the assigned variable (see above) is not toggled. This way, a toggle input can be aborted.
	● Option enabled: The assigned Boolean variable is also toggled if the pressed mouse button is released after the cursor has been removed from the element.
		`R, rR, E, L, F, Bu, Im`
		`P, pL, C`

Tap		

Variable	Here, enter a Boolean variable that is set to TRUE if the mouse button is pressed and the cursor is pointing to the element. The Boolean variable becomes FALSE again when the mouse button is released.	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

Tap False	Enabling this option reverses the tap behavior described above for the variable entered. That means if the mouse button is pressed, the variable is set to FALSE and when the button is released again, the variable it set to TRUE.	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

Tap on enter if captured	If this option is enabled, the variable entered above is also "tapped" if the mouse button is pressed before the cursor was dragged onto the element.	`R, rR, E, L, F, Bu, Im`
		`Te`
		`P, pL, C`

Hotkey		`R, rR, E, L, F, Bu`
		`Te`
		`P, pL, C`

Button	Key name, e.g. "M"; a selection list provides the currently supported keys, page 494.	

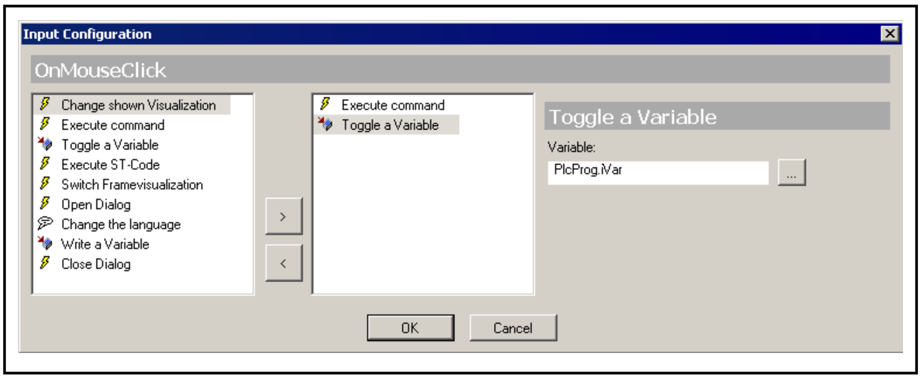
Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Input	Action that is to be executed when the key + any possibly defined modifier (<Shift>, <Ctrl>, <Alt>) is used; possible values provided in a selection list: - No action - MouseDown action (OnMouseDown action when the button is pressed) - MouseUp action (OnMouseUp action when the button is released) - MouseDown/MouseUp action (OnMouseUp and OnMouseDown actions when the button is pressed or released).	
<Shift>	If this option is enabled, the key has to be used together with the <Shift> key.	
<Ctrl>	If this option is enabled, the key has to the used together with the <Ctrl> key.	
<Alt>	If this option is enabled, the key has to be used together with the <Alt> key.	

Fig.4-210: Inputs

*** "OnMouse..." resulting actions:**

The resulting actions described below can be configured in the "Input configuration" dialog for a specific mouse action on the element. The dialog opens when clicking on the "Configure..." field.



Variable: PlcProg.iVar

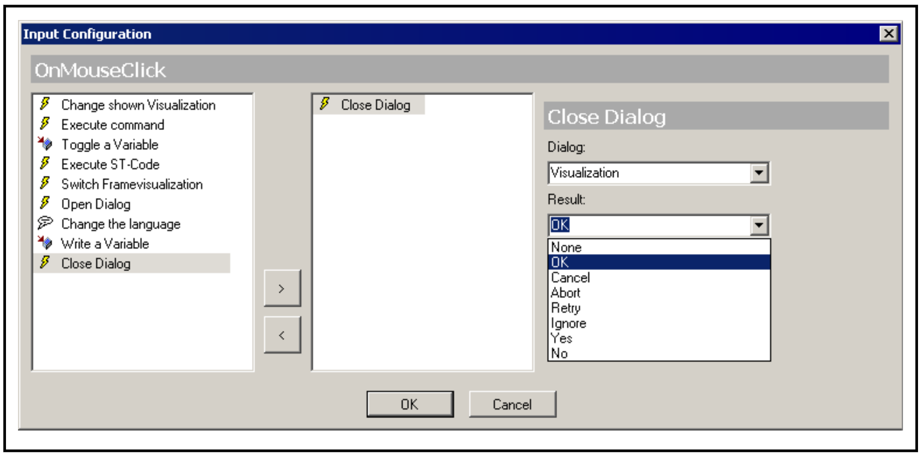
*Fig.4-211: * OnMouse..." resulting actions:*

At complete left is a selection of possible resulting actions. In the center are the resulting actions already defined for the current mouse action. The configuration fields relevant for the action selected in the center field always appear at the complete right.

To define one or multiple of the resulting actions for the current mouse action, select the action from the left field and use the right arrow button to move it into the right field. Then select an action from the center field to make the respective configuration in the fields on the right.

The following include the configuration options for the individual resulting actions:

- Close dialog



Dialog: Visualization

Result: OK

Fig.4-212: Input Configuration, Close Dialog

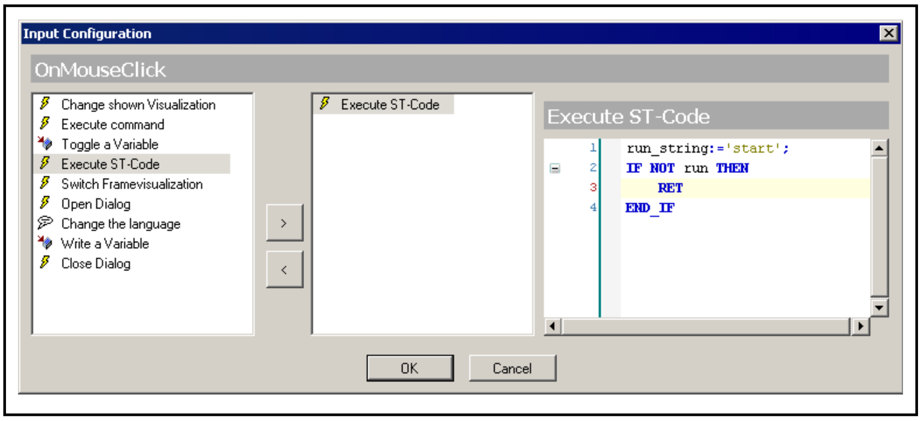
Here, determine that the dialog specified is closed with the give result due to the mouse action. Select the desired dialog from the selection list, which - as in the "Open dialog" configuration (see above) - provides all currently available input dialogs, page 663,.

The "Result" list provides the default options used in dialogs in which user action is required: OK, Cancel, Abort, Retry, Ignore, Yes, No.

Note that the result from the latest dialog closed can be called and that a corresponding reaction to it can be configured in any desired element in the same visualization. To do this, use the "OnDialogClosed" configuration option.

☞ In the input dialogs, page 663,, there is more information on the default and user-defined input dialogs in visualizations.

- Execute ST code

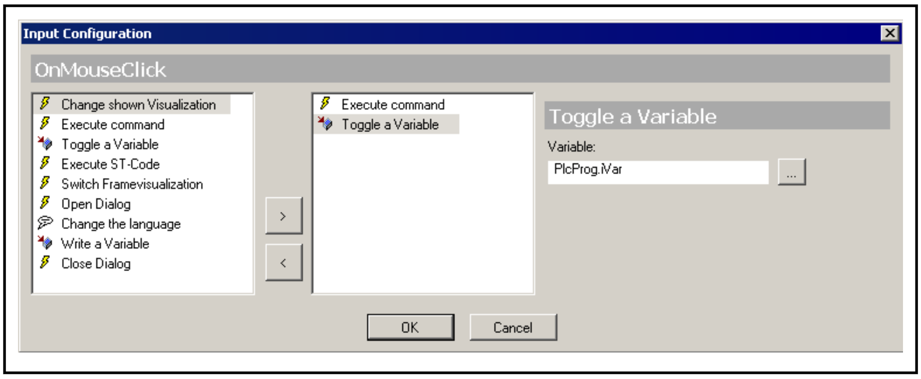


```
run_string:='start';
IF NOT run THEN
    RET
END_IF
```

Fig.4-213: Input Configuration, Execute ST Code

In this input field, enter code in structured text that is to be executed in response to the mouse action.

- Toggle Variable



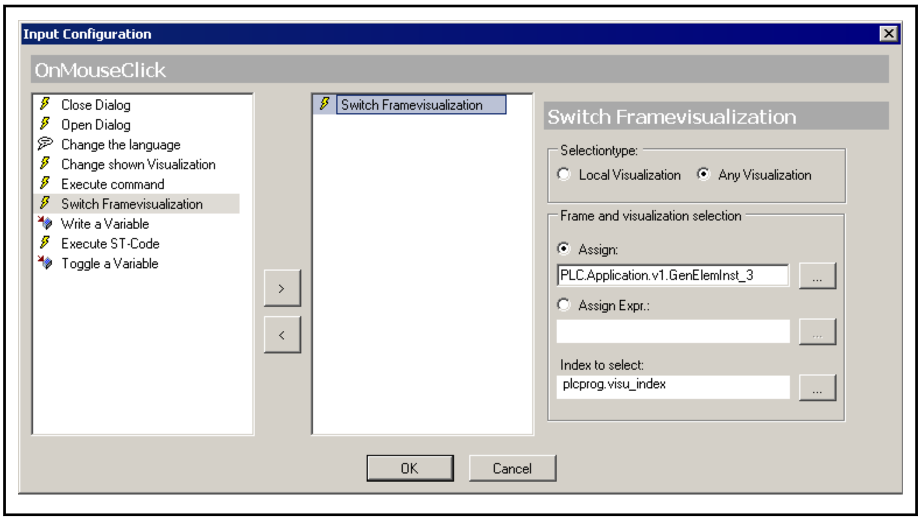
Variable: PlcProg.iVar

Fig.4-214: Input Configuration, Toggle a Variable

Here, enter a Boolean variable that should switch between TRUE and FALSE at repeated mouse actions.

Example: "PlcProg.iVar"

- Switch Frame Visualization



Selectiontype: ○ Local Visualization ● Any Visualization

Frame and visualization selection

● Assign: PLC.Application.v1.GenElemInst_3

○ Assign Expr.:

Index to select: plcprog.visu_index

Fig.4-215: Input Configuration, Switch Frame Visualization

– **Prerequisites**:

In a visualization in the project, there are frame elements to which different visualizations using "Frame selection" and the "Configuring frame visualizations" dialog; see Frame selection, page 302 were assigned.

These visualizations receive an index within the frame (0, 1, 2, etc.). By default, the first of the assigned visualizations (index "0") is displayed online in the frame.

– **Switch frame visualization:**

In this dialog, configure that using the mouse on the current visualization element in a specific frame element, a specific visualization assigned to it is displayed. This way, program a "switch" to display visualizations in a frame using the current element.

– **Selection type**:

The selection of the respective frame element can be limited to the current visualization, which means that the configuration dialog is simpler, but does not have a dynamic configuration option.

⇒ Option: "Local Visualization"

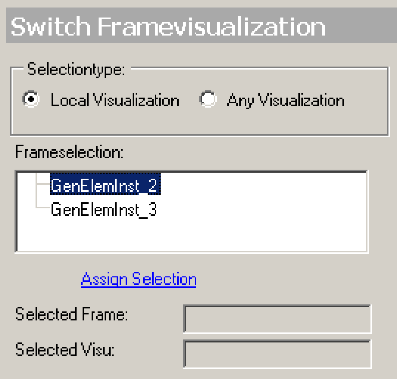
The selection of the respective frame element can integrate all visualizations in the project, which also allows dynamic frame and visualization input.

⇒ Option: "Any Visualization"

Local Visualization

Only frames in the current visualization can be addressed and can be selected here in the "Frame selection" field (see below). This is a simple dialog for quick, direct configuration in the local visualization. However, it does not provide the option to enter the desired visualization using project variables. If this is necessary, select the "Any Visualization" option (see below).

The "Frame selection" field displays the local frame elements with the respectively assigned visualizations indented below.



Switch Framevisualization

Selectiontype:

● Local Visualization ○ Any Visualization

Frameselection:

GenElemInst_2
GenElemInst_3

Selected Frame:

Selected Visu:

Fig.4-216: Frame selection, local

Below the respective frame, select the desired visualization to be displayed in response to the mouse action. Click on "Assign Selection" to save the setting. The current selection will then displayed in the "Frame Selection" and "Visualization Selection" fields.

Any Visualization

All frames in the project and their assigned visualizations can be selected. In this case, frame and visualization can also be specified via project variables or dynamically. Frame and visualization are selected via the "Direct assignment" and "Assign using expression":

The desired frame element can be entered directly or by using an expression:

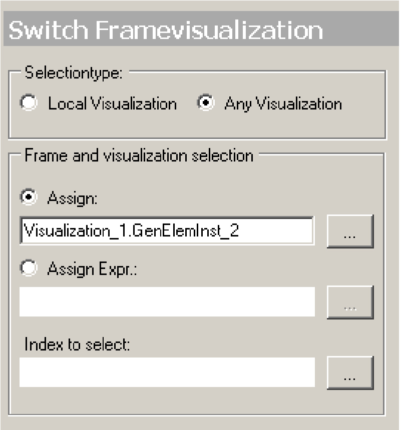
"Direct assignment":

If this option is enabled, the complete path of the respective frame element has to be entered directly. Use the input assistance via the ... button.

Syntax:

```
<DeviceName>.<ApplicationName>.<Visualization-
Name>.<FrameElementName>
```

See the example in the following figure.



Switch Framevisualization

Selectiontype:

○ Local Visualization ● Any Visualization

Frame and visualization selection

● Assign:

Visualization_1.GenElemInst_2 ...

○ Assign Expr.:

...

Index to select:

...

Fig.4-217: Selected Frame, direct assignment

Assign using expression:

Alternatively, this option can be enabled to use a project variable of type STRING to enter the frame element.

Use the ... button to open the input assistance. The variable has to return the entire path of the frame element.

Syntax:

`<DeviceName>.<ApplicationName>.<Visualization-Name>.<FrameElementName>`.

The desired visualization to be displayed in the selected frame in response to the mouse action is specified via its index.

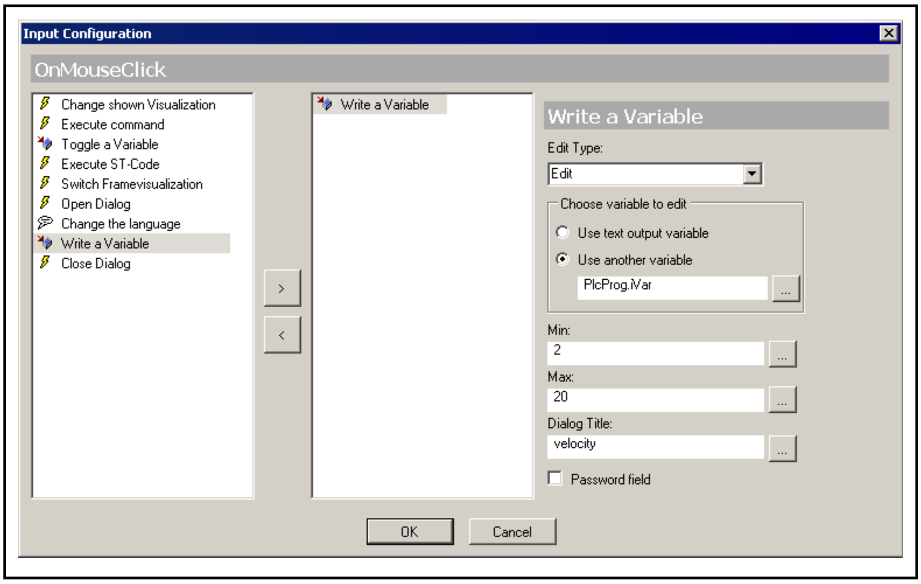
This index is assigned with an ascending, integer numbering of visualizations with integers starting with "0". The numbered visualizations were assigned to a frame in the "Configuration of Frame Visualizations" dialog. This way, the first visualization is always shown in this list by default.

Index to select:

Enter the index of the desired visualization directly or via a project variable used in the application. The ... button opens the input assistance to select a variable.

- **Write variable**

  If there is an input configuration of the "Write a variable" type for a visualization element, the element provides the option to enter a value when the corresponding mouse action is executed. The value can then be entered as a character string or using a numpad or keypad and after the entry is completed, it is written on the project variable specified in the "Input Configuration" dialog. The value is interpreted as text or numerical value based on the data type of the project variable.



Write a Variable

Edit Type: Edit

Choose variable to edit

○ Use text output variable

● Use another variable: PlcProg.iVar

Min: 2

Max: 20

Dialog Title: velocity

□ Password field

Fig.4-218: Input Configuration, Write a Variable

Chose one of the following input types from the selection list in the right section of the dialog:

Using the configuration dialog:

– **Edit:**

The mouse action opens an input field into which a text or numerical value can be typed.

– **VisuDialogs.Keypad:**

The mouse action opens a simulated keypad (keyboard). Click on the corresponding keypads to enter a character string.

– **VisuDialogs.Numpad:**

The mouse action opens a simulated numpad (numerical keypad). Click on the respective keys to enter a numerical character string.

– In addition:

If there is a user-defined visualization in the project that is defined as "numpad/keypad" in its properties:

<device>.<application>.<numpad/keypad-visualization>:

The mouse action opens the numpad or keypad displayed in the selected visualization.

Note that the "Register images without path" option has to be enabled so that the necessary background bitmaps can be loaded; see Register images without path, page 199.

☞ To use the keypad or numpad, the "VisuDialogs.library" has to be integrated into the Library Manager.

– In the box "Choose variable to edit", specify whether the value entered by the user is to be written on the text output variable or on another variable in the project. The text variable is part of the "Text variables" element property and is highlighted via click. To write the value to another project variable, enter its name (including the path). Alternatively, also access the input assistance using the ... key.

– **Min/Max:** A limit range for allowable inputs can be defined using the "minimum" and "maximum value" either as value or via project variables.

To display your input hidden with asterisks ("************"), select the checkbox in front of the password field.

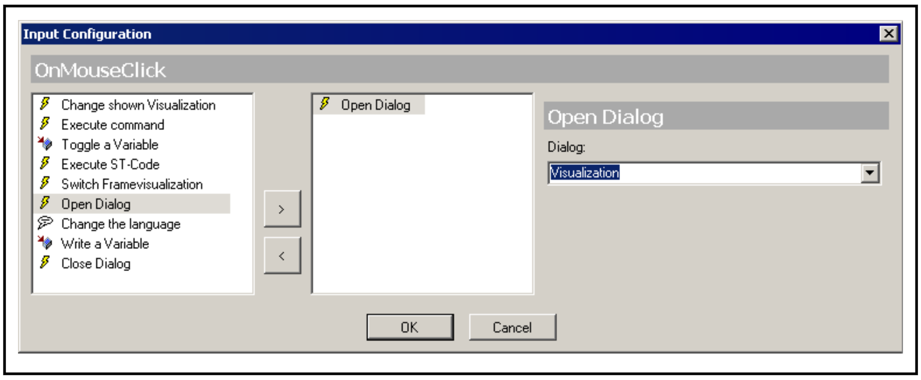
– **Dialog title:** Enter what is to be displayed in the dialog title bar as text or via a text variable.

Example:

See the example in the dialog figure above and assume hat it is the configuration dialog for a rectangle element. If the mouse button is pressed on this rectangle in online mode, a numerical keypad appears with the title of the variable Plc_Main. MyTitle.

Enter the value "123" for example by clicking on 1,2 and 3. This value is displayed in the upper section of the dialog along with the minimum and maximum values for the input entered via Plc_Main.minVal and Plc_Main.maxVal. When the entry is confirmed with "OK", "123" is written to the variable Plc_Main.ivar. If "ivar" is defined as STRING variable, it assumes the value "123". If "ivar" is a numerical variable, it assumes the value "123".

- **Open dialog**



Dialog: Visualization

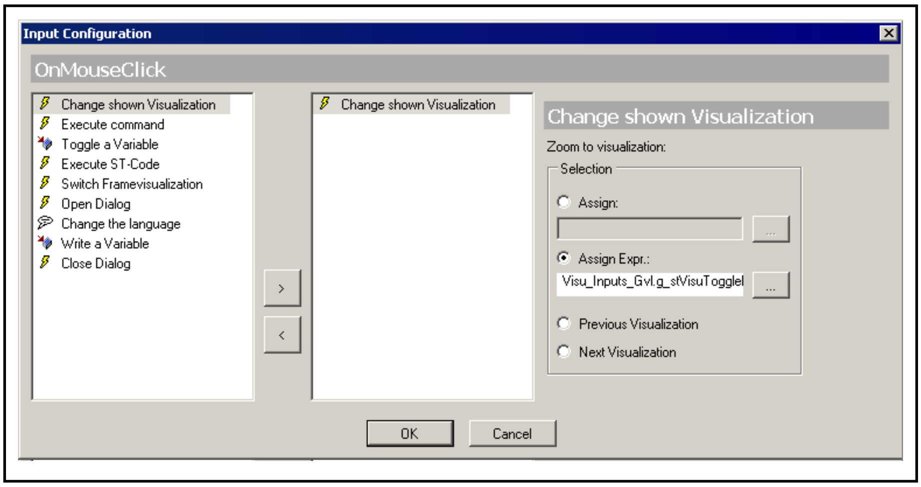
Fig.4-219: Input Configuration, Open Dialog

Here, define that a dialog opens as a result of the mouse action. This dialog is represented by another visualization (default or user-created). The selection list provides all visualizations that entered the purpose "Dialog" in their "Properties" windows (context menu: Properties, Visualization).

This way, a self-created dialog can be used as a user input mask in a visualization.

Further information and an example are located in Input dialogs, page 663.

- **Visualization switch**



Assign Expr.: Visu_Inputs_Gvl.g_stVisuTogglel

Fig.4-220: Input Configuration, Visualization Switch (Change Shown Visualization)

Select one of the following options to specify which visualization is displayed as response to the mouse action in online mode (selection):

– Direct assignment:

The visualization can be input directly. The entire path <DeviceName>.<ApplicationName>.<VisualizationName> has to be entered. To do this, use the input assistance opened via the ... button.

Example: "BRC_Control.Application.Vis1".

– Assign using an expression:

Here, enter a variable of type STRING that is used by the application and that returns the name of the visualization.

Example: "visu_stringvar: STRING := 'MyPlc.Appl1.Visu_xy' "

The sequence in which the visualizations were displayed via user inputs is saved internally. This information can be used with both of the following options:

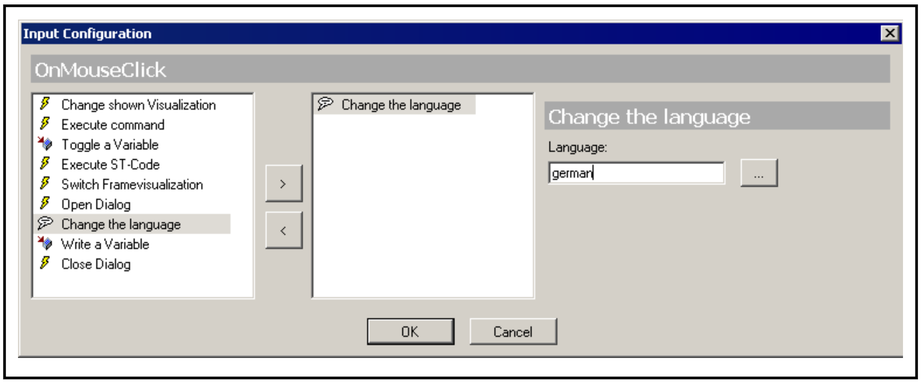
– Previous Visualization:

The previously shown visualization is displayed again. If none was called previously, the current one remains shown.

– Next Visualization:

The visualization that follows the current one in the recorded sequence of changing visualizations is displayed. This is only possible if a visualization switch has already been made for this visualization using "Previous Visualization".

- Language selection



Language: german

Fig.4-221: Input Configuration, Language Selection (Change the language)

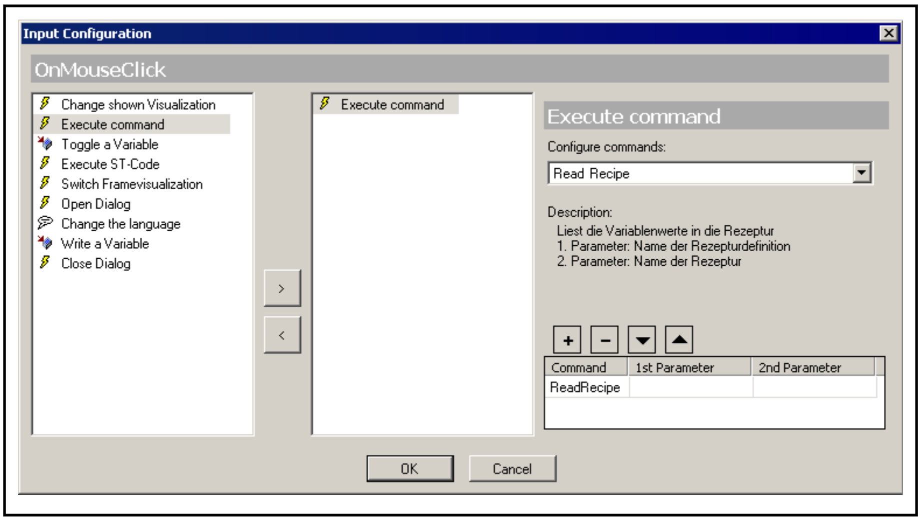
In the "Language" field, a language can be entered that is used in the display of visualization text as response to a mouse action.

Use the language code from the respective text list.

Also refer to

– Text list, page 55

- Internal command (execute programs, print, recipe management)



Configure commands: Read Recipe

Description:
Liest die Variablenwerte in die Rezeptur
1. Parameter: Name der Rezepturdefinition
2. Parameter: Name der Rezeptur

Command	1st Parameter	2nd Parameter
ReadRecipe		

Fig.4-222: Input Configuration, Internal Command (Execute Command)

Here, define one or multiple commands to be executed as response to the mouse action.

Select a command from the "Configure command" list and press the + button to insert it into the table in the lower section of the dialog. This table contains all current commands selected for the current input configuration.

A brief description of the command selected in the list above is always shown in the center section of the dialog. Based on this description, complete the parameters for the command in the table columns "1st parameter" and "2nd parameter" below. The "Command" column displays the internal command names. See the descriptions of the individual commands below and on the help page.

Use the − button to remove an entry selected in the table.

Later, when the user input is carried out in the visualization element, the configured commands are executed according to their arrangement in the table from top to bottom.

To change the order, the entries can be moved using the ▼ and ▲ buttons.

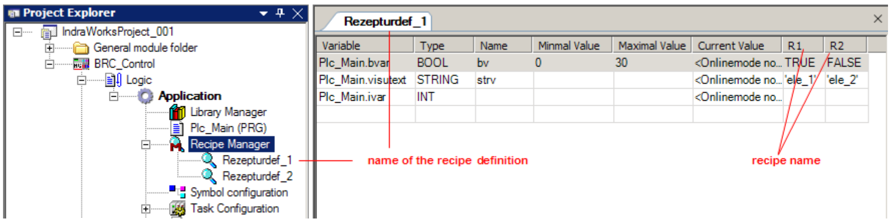
The following commands can be selected:

Execute program on the control	The specified program (*.exe) is executed on the control or on the visualization client. **1st parameter**: Character string, program file path (Example: "C:\programs\notepad.exe") **2nd parameter**: Character string, arguments for the program to be executed, e.g. file name the program has to open. (Example: "copyfile.txt")
Execute program at client	
Print	The default print dialog opens. There, settings to set the print area and the printer parameters can be made. The current visualization can be printed using these settings.

Note the following for commands for the recipe management, page 382,:

The "**recipe definition**" is an object below the recipe manager in the project tree of the project. It consists of a list of variables and the existing recipes for these variables (i.e. variable values). It can be opened and edited in the editor window of the recipe manager.

The "**recipe name**" is the name of a recipe described in the recipe definition for the respective list of variables. The recipe names are entered as table headings. Refer to the following figure.



Variable	Type	Name	Minmal Value	Maximal Value	Current Value	R1	R2
Plc_Main.bvar	BOOL	bv	0	30	<Onlinemode no...	TRUE	FALSE
Plc_Main.visutext	STRING	strv			<Onlinemode no...	'ele_1'	'ele_2'
Plc_Main.ivar	INT				<Onlinemode no...		

Fig.4-223: Recipe definition and recipe name

Create recipe	A new recipe is created in the indicated recipe definition. **1st parameter**: Name of the recipe definition. **2nd parameter**: Name of the new recipe.
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. In other words, the recipe managed in IndraLogic is updated with the values from the control. **1st parameter**: Name of the recipe definition. **2nd parameter**: Name of the recipe for which the values are read from the control and are to be saved as new recipe values.

Write recipe	The values of the indicated recipe - as stored in the recipe manager - are written to the corresponding variables one the control. **1st parameter**: Name of the recipe definition. **2nd parameter**: Name of the recipe from this recipe definition whose values are to be written to the variables on 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, i.e. the name for the new file has to be something other than `<RecipeName>.<RecipeDefinitionName>.txtrecipe`! **1st parameter:** Name of the recipe definition. **2nd parameter:** Name of the recipe to be saved in the specified file.
Load recipe	The recipe saved in a file (see "Save recipe" above) can be loaded from this file. 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. **1st parameter:** Name of the recipe definition. **2nd parameter:** Name of the recipe for which the values from the specified file are to be reloaded to the recipe manager.
Delete recipe	The specified recipe is deleted from the specified recipe definition. **1st parameter:** Name of the recipe definition **2nd parameter:** Name of the recipe

Fig.4-224: Execute Command, (internal command -> execute program, recipe management)

Each input via mouse assigned to an action in the input configuration dialog displays the "Configured" property in the "Properties" window.

Scrollbar-specific inputs

(only for scrollbar: Input Variable, Output Variable, Minimum value, Maximum value)

Property	R=Rectangle, rR=Rounded rect., E=Ellipse, L=Line, F=Frame, Bu=Button, Im=Image
	M= Meter, BD=Bar Display , Tr=Trace
	Ta= Table, Te=Text field, Sc=Scrollbar
	P=Polygon, pL=Polyline, C=Curve	

Input Variable	The scrollbar controller is positioned based on the value of this variable	Sc
Output Variable	The value of this variable is written based on the positioning of the scrollbar controller	Sc
Minimum value	Value corresponding to the positioning of the controller at the left	Sc
Maximum value	Value corresponding to the positioning of the controller at the right	Sc

Fig.4-225: Scrollbar-specific inputs

Defining color values:

A color is defined by a hexadecimal number consisting of the blue/green/red (RGB) components.

For each of these three colors, 256 values (0-255) are available.

Example: 16#00FF00FE

- FF – blue components
- 00 – green components
- FE – red components

For **static definitions** ("Colors" category in the element properties), the color can be selected from a selection list or the default color selection dialog (opened via the ... button).

Example: 🟩 0; 128; 64

For **dynamic definitions** ("Color variables" category), a project variable of the type DWORD has to be entered in the element properties that return the color definition:

e.g. "prog1.dwFillColor".

☞ The variable has to return the color value in hexadecimal format. The first two zeroes after "16#" should always be specified in order to fill the DWORD size.

Example: dwFillColor := 16#00FF00FF;

Menu bar for filters and sorting

The menu bar above in the "Properties" editor provides the following menus and options:

Filter menu: Select one of the following options to define the properties to be displayed:

Simple: Only the basic properties, e.g. position, center point, colors.

Default: Preferably used properties (this selection is based on the properties set used in the programming system of IndraLogic 1.x)

Color: Only the color properties "Animation": Only the properties of an animation of the element (movement, toggle color, dynamic look)

Text: Only text properties

Input: Only the properties that refer to a user input of the element

Show all categories: All properties

Sort menu:

Sort by type: All properties are displayed; the categories are arranged in original order.

Sort by name: All properties are displayed; the categories are arranged alphabetically from top to bottom.

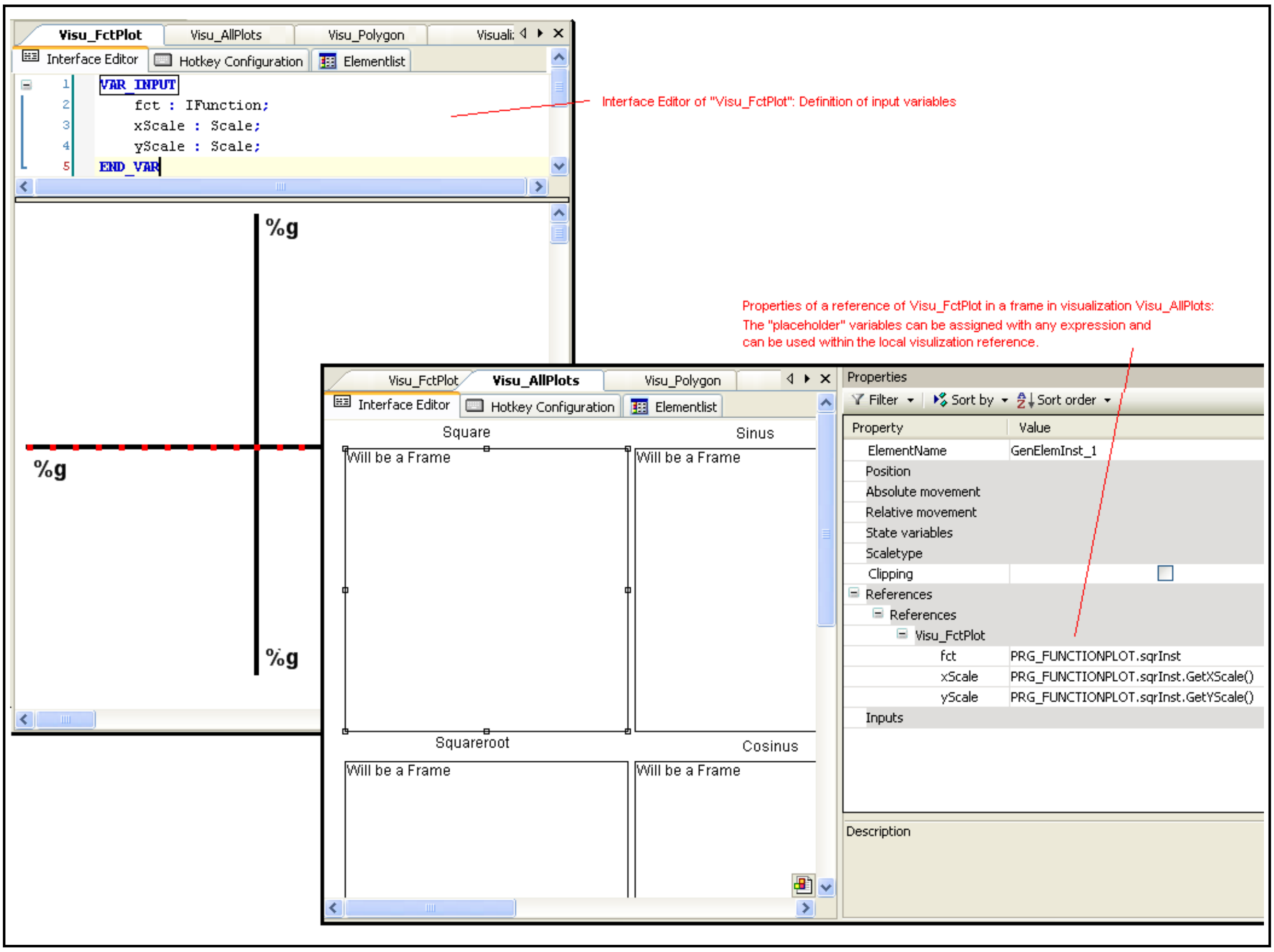
Order menu:

Sort in ascending order: All properties are displayed and the categories are arranged in the original order from top to bottom.

Sort in descending order: All properties are displayed and the categories are arranged in the original order from bottom to top.

4.16.7 Interface Editor

The interface editor for visualizations is part of the visualization editor, page 445, and when the "Interface editor" command is activated (by default in the "Visualization" menu), it can be hidden and made visible. It appears in the upper section of the visualization editor as a separate tab next to other editors, e.g. the editors for keyboard configuration or the element list.



```
VAR_INPUT
    fct : IFunction;
    xScale : Scale;
    yScale : Scale;
END_VAR
```

Interface Editor of "Visu_FctPlot": Definition of input variables

Properties of a reference of Visu_FctPlot in a frame in visualization Visu_AllPlots: The "placeholder" variables can be assigned with any expression and can be used within the local visulization reference.

Property	Value
ElementName	GenElemInst_1
Position	
Absolute movement	
Relative movement	
State variables	
Scaletype	
Clipping	□
References	
References	
Visu_FctPlot	
fct	PRG_FUNCTIONPLOT.sqrInst
xScale	PRG_FUNCTIONPLOT.sqrInst.GetXScale()
yScale	PRG_FUNCTIONPLOT.sqrInst.GetYScale()
Inputs	

Fig.4-226: Interface editor, example

The editor is used to define "**placeholder**" variables in a visualization which are added to another visualization as reference.

See Frames, References..., page 628, for information on "References" and "Frames".

See Visualization, page 625, for general information (prerequisites, overview) on the visualization in IndraLogic 2G.

Since a visualization is handled as a function block, the interface editor appears as a normal **declaration editor** in the upper section of the Visualization editor, page 445, window. Here, declare input variables (VAR_INPUT) that can be replaced later by variables or expressions used locally when the visualization is added to a frame element, page302, in another visualization. These replacements have to be made in the Properties dialog, page 451, of the frame element. The inserted (instantiated) visualization and its input variables are executed there and expressions that can be used in the local configuration of the instance can be assigned to the variables.

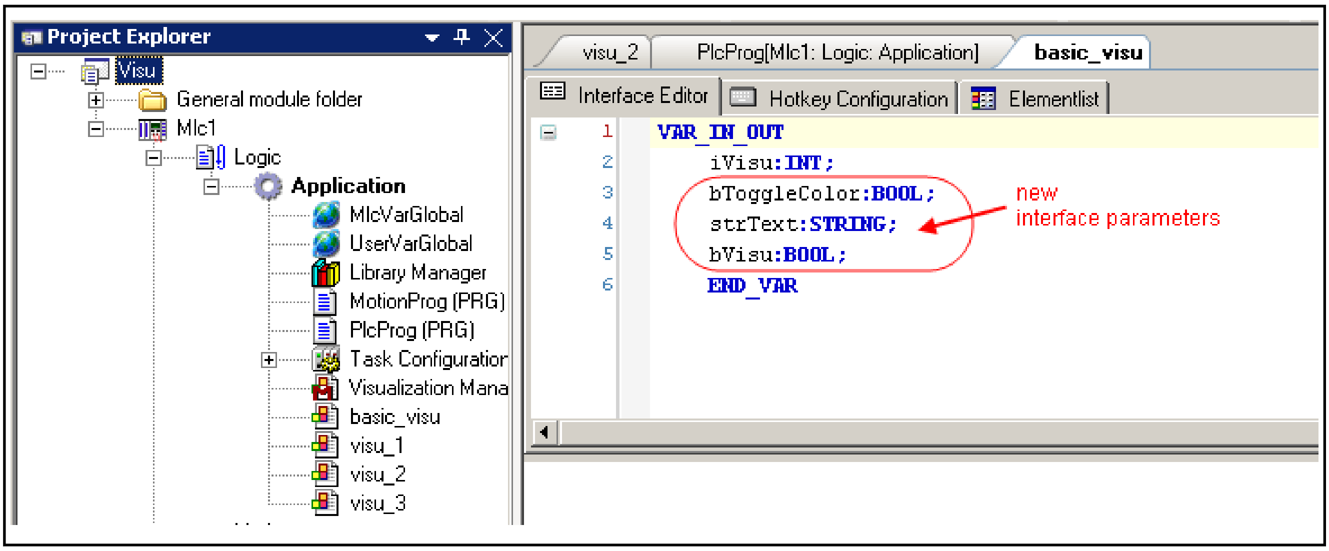
If the interface of a visualization inserted into a frame is modified, the **Updating the frame parameters** dialog opens to define how the modified interface parameters have to be handled when saving or compiling the project or when opening the respective project.

Note the usage of this dialog in the following example:

At first, the interface of the **basic_visu** visualization contains only the parameter **iVisu**. **basic_visu** is inserted into a frame in **visu2** and **visu3** and **iVisu** is assigned to the project variable **PlcProg.ivar** in **visu2** and **visu3**.

Initially, the project is used in this state.

Later, the interface of **basic_visu** is extended to include the parameters **bToggleColor**, **strText** and **bVisu**:

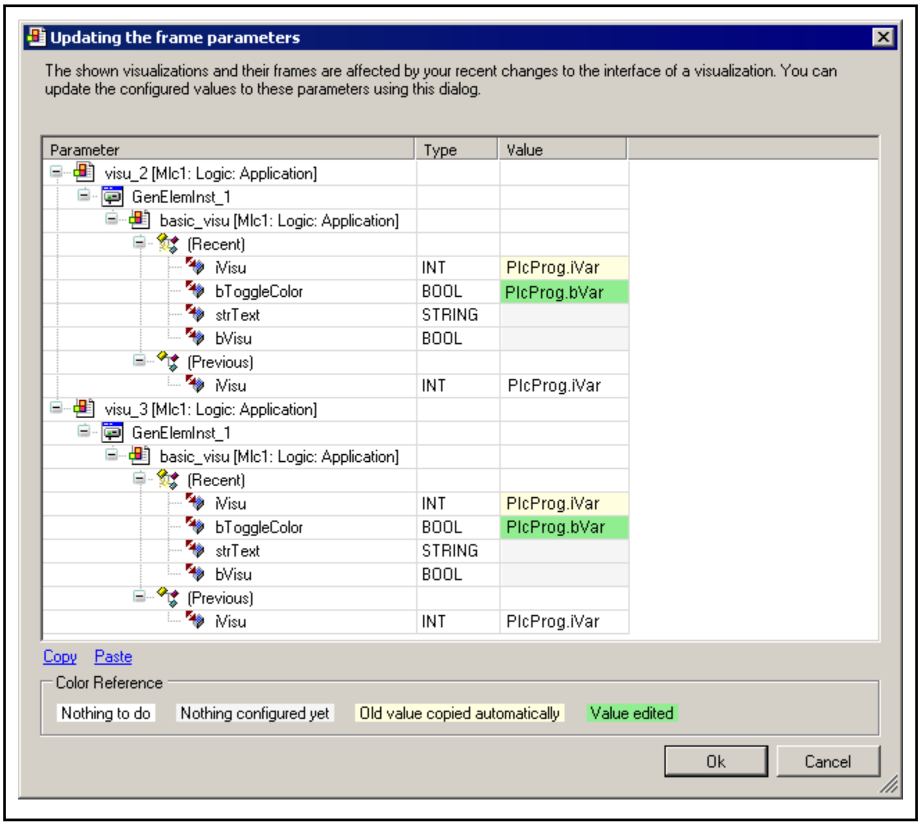


```
VAR_IN_OUT
    iVisu:INT;
    bToggleColor:BOOL;
    strText:STRING;
    bVisu:BOOL;
    END_VAR
```

new
interface parameters

Fig.4-227: Changes in the interface editor

If the project is compiled or saved for example or the language is switched, the dialog to configure the new parameters appears:



Updating the frame parameters

The shown visualizations and their frames are affected by your recent changes to the interface of a visualization. You can update the configured values to these parameters using this dialog.

Parameter	Type	Value
visu_2 [Mlc1: Logic: Application]		
GenElemInst_1		
basic_visu [Mlc1: Logic: Application]		
(Recent)		
iVisu	INT	PlcProg.iVar
bToggleColor	BOOL	PlcProg.bVar
strText	STRING	
bVisu	BOOL	
(Previous)		
iVisu	INT	PlcProg.iVar
visu_3 [Mlc1: Logic: Application]		
GenElemInst_1		
basic_visu [Mlc1: Logic: Application]		
(Recent)		
iVisu	INT	PlcProg.iVar
bToggleColor	BOOL	PlcProg.bVar
strText	STRING	
bVisu	BOOL	
(Previous)		
iVisu	INT	PlcProg.iVar

Copy Paste

Color Reference

Nothing to do Nothing configured yet Old value copied automatically Value edited

Fig.4-228: Dialog: Updating the frame parameters

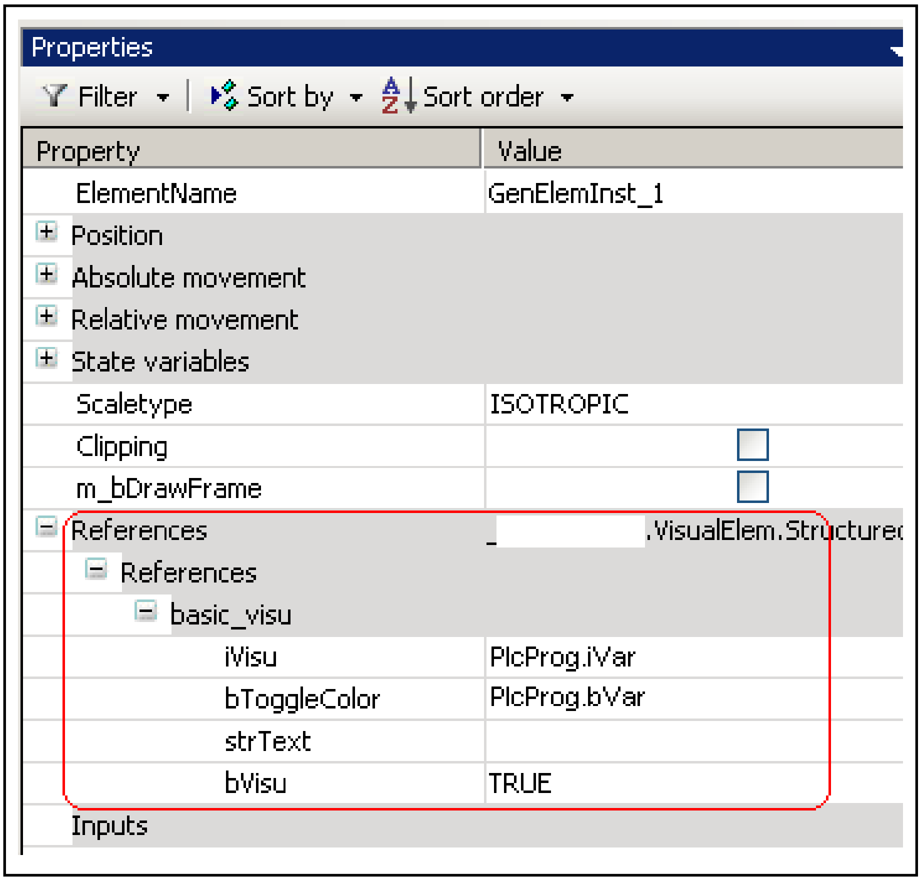
For visualizations that reference the modified **basic_visu**, that is **visu2** and **visu3**, the current and former values of the respective interface parameters are displayed. The current value can be reconfigured here. The colored background of the "Value" fields creates visual support to indicate where input is required or was already made (see also the "Color Reference" legend below the parameter table):

- Beige background: This parameter is automatically accepted from the previous configuration (see the example: **PlcProg.iVar** for **visu2** and **visu3**).
- Gray background: No value has yet been assigned to this new parameter (see the example: **strText** for **visu2** and **visu3**).
- Green background: A value has already been assigned to this new parameter (see the example: **bToggleColor** for **visu2** and **visu3**).
- White background: Nothing needs to be configured here.

To edit a "Value" field, click on the field and press <Enter>.

To copy an existing value, highlight the respective field and press <Copy>, then select the "Value" field into which the value is to be copied and press <Paste>.

Confirm with "OK" and the new parameter assignments appear in the "Properties" dialog (References) of the respective frame element; in this example, for the frame in **visu2**:



Property	Value
ElementName	GenElemInst_1
⊞ Position	
⊞ Absolute movement	
⊞ Relative movement	
⊞ State variables	
Scaletype	ISOTROPIC
Clipping	□
m_bDrawFrame	□
⊟ References	_ .VisualElem.Structurec
⊟ References	
⊟ basic_visu	
iVisu	PlcProg.iVar
bToggleColor	PlcProg.bVar
strText	
bVisu	TRUE
Inputs	

Fig.4-229: "Properties" dialog: New parameter assignments

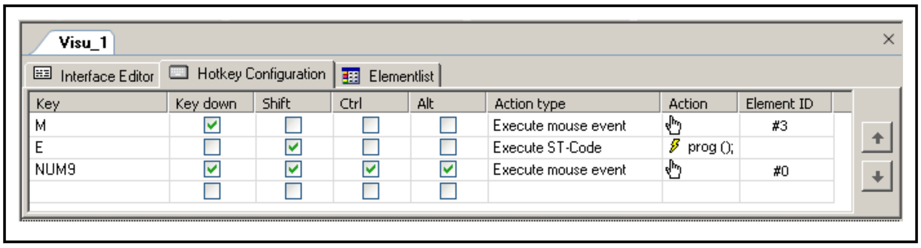
The pragma attribute "parameterstringof", page 543, supports the provision of variable names as strings in the referenced visualization function block (here "basic_visu").

4.16.8 Hotkey Configuration Editor

In addition to the standard hotkeys, page 661, define special hotkeys to operate a visualization in online mode in this section of the visualization editor, page 445,. This means that an action can be assigned to a specific key or shortcut (hotkey). This hotkey applies only to the respective visualization. Hotkeys that are to be used in all visualizations of the application should be defined in the Visualization Manager, page 496,.

In this context, also note the "Hotkey" property of a visualization element which can be configured in the element properties (input), page 451,. Such an element-specific hotkey is also managed in the hotkey configuration editor. It can be edited in both locations and it is updated in the respective other editor.

The hotkey configuration editor - to be opened via the command Hotkey Configuration, page 657, is a separate tab next to the Interface Editor, page 491, and other editors in the upper section of the visualization editor.



Key	Key down	Shift	Ctrl	Alt	Action type	Action	Element ID
M	☑	□	□	□	Execute mouse event		#3
E	□	☑	□	□	Execute ST-Code	prog ();	
NUM9	☑	☑	☑	☑	Execute mouse event		#0
	□	□	□	□			

Fig.4-230: Hotkey configuration editor

In each line of the table editor, a key or a hotkey can be linked with an action. See the following columns:

Key: Name of the key. The key name can be input manually or using a selection list that opens by double-clicking the cell. The list contains all keys defined via the **device description**. In the case of a visualization that is not assigned to any device, all keys are supported by all devices.

Key down: If this option is selected, the action is executed as soon as the key is pressed. Otherwise, it is executed when the key is released. If the action is to be executed both when the key is pressed (KeyDown) and released (KeyUp), two corresponding definitions for the the key have to be present in the table.

Shift, **Ctrl**, **Alt**: If this option is selected, the <Shift> or <Ctrl> or <Alt> key have to be pressed together with the key to execute the action.

Action type: The action type can be defined using a selection list that opens by double-clicking on the cell. The action types correspond with those available in the input configuration dialog, page 451, of a visualization element.

Action: Exact configuration of the action to be executed. It depends on the action type and corresponds to the mouse action as it can be used in the input configuration of a visualization element.

Element ID: ID of the visualization element to which the key is assigned (via the "Hotkey" property, page 451). Unique identification in the current visualization.

If a key is linked with several actions, these are executed in the same sequence as they appear; from top to bottom in the configuration table. This order can be changed by selecting a key definition (click in the table line) and using the **arrow keys** on the right of the table to move up or down.

☞ The **device description** defines which keys are supported in a visualization running on the device!

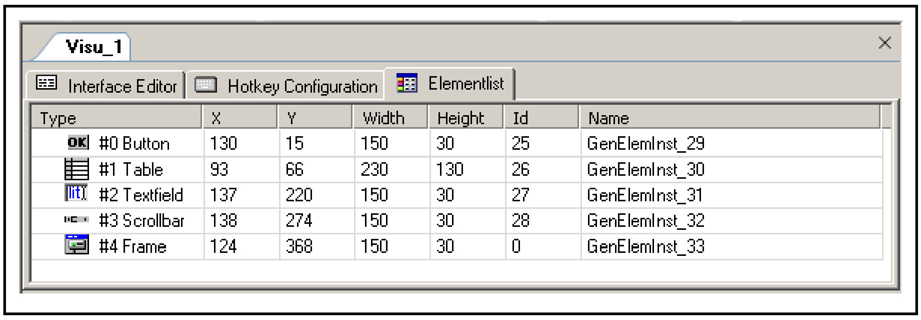
Note the following **call sequence** to process key actions:

1. Event handler of the application if enabled (optional); e.g. g_VisuEventManager, page 658)
2. Hotkey configuration that applies for all visualization in the application; see in the Visualization Manager, page 496
3. Definitions of the default keyboard operation, page 657, in online mode.
4. Special hotkey configurations of individual visualizations (managed in the hotkey configuration editor). The main visualizations are considered before those in the frames.

4.16.9 Element List

This part of the visualization editor, page 445, shows a list of all elements in the current visualization.

The element list can be opened using the "Interface Editor" or "Hotkey Configuration" (by default in the "VI Logic Visualization" menu) and is then located as separate tab next to the interface editor or the editor for hotkey configuration.



Type	X	Y	Width	Height	Id	Name
#0 Button	130	15	150	30	25	GenElemInst_29
#1 Table	93	66	230	130	26	GenElemInst_30
#2 Textfield	137	220	150	30	27	GenElemInst_31
#3 Scrollbar	138	274	150	30	28	GenElemInst_32
#4 Frame	124	368	150	30	0	GenElemInst_33

Fig.4-231: Element list

The elements are listed from top to bottom according to their positions on the z-axis of the visualization. The element furthest away in the background in the first line. The following values are displayed but cannot be edited here:

Type: Element type and symbol as used in the toolbox, page 446, as well as the element number that results from the insertion sequence first.

X, Y: Position of the upper left corner of the element (0,0 = upper left corner of the visualization area)

ID: Internally assigned element identification

Name: Element name as defined in the element properties, page 451,

Element(s) can be selected by selecting the corresponding table line(s). The selection is always synchronized with that in the main window of the visualization editor. However, note that **subelements of a group** can only be selected here in the element list.

The position of a selected element on the z-axis, i.e. on the **background-foreground** axis can be changed using the following commands:

Place on top, page 304

One level up, page 305

Send to back, page 305

One level down, page 305

The following commands can be used to undo **processing actions** in the element list or to remove elements:

Undo, page 89,

Redo, page 89,

Delete, page 91.

4.17 Visualization Manager

4.17.1 Visualization Manager

Symbol:

The visualization manager manages the general settings for all visualizations that belong to an application.

Changes in settings become effective immediately.

☞ An overview on the visualization in IndraLogic can be found in Visualization in IndraLogic 2G, page 625.

The "visualization manager object" is automatically added to the project tree below an application as soon as the first visualization is assigned to this application.

The main dialog contains tabs for

Settings, page 497

Default hotkey configuration, page 498

Visualizations, page 499

Note that starting with V 3.3.2.0 of the programming system, the "Diagnostic Visualization" is automatically used if no client objects have been added below the Visualization Manager. For details, please see: Visualization running in the programming system, page 634.

☞ If a fixed compiler version was set in a project that was originally created with a version < V3.3.2, the project acts according to the visualization functionalities available in that version!

In the following figure, the Visualization Manager below "Application" is responsible for the visualizations Visu_01 and Visu_02.

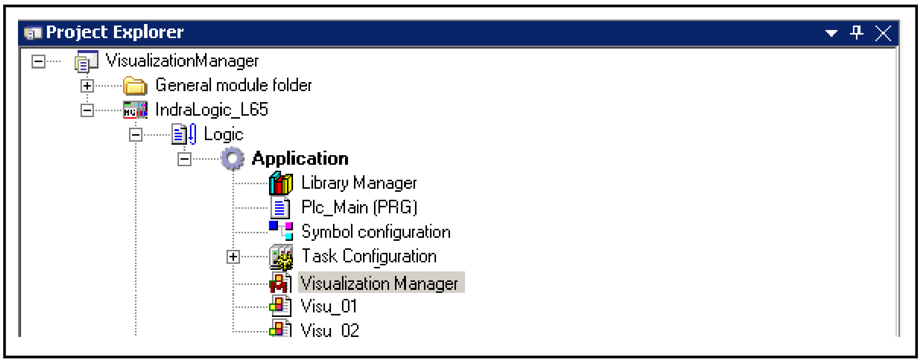
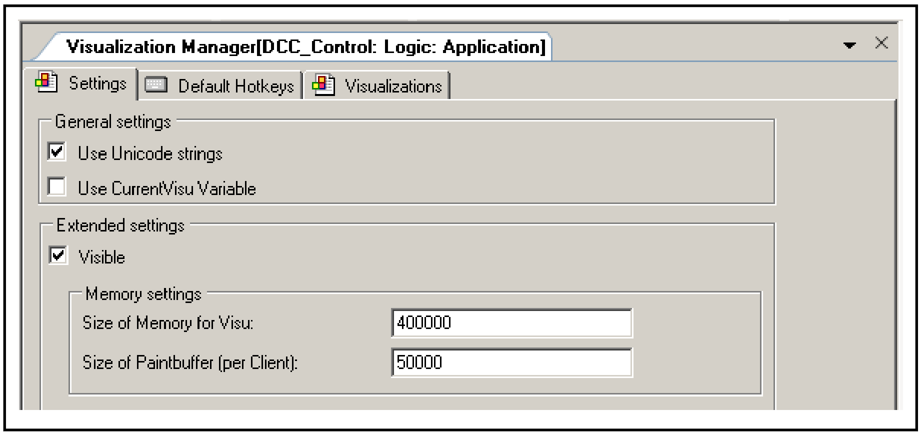


Fig.4-232: Visualization Manager in the Project Explorer

To open the "Visualization Manager Editor", double-click on the entry in the Project Explorer.

The editor opens in a "Visualization Manager" window with the subdialogs Settings, page 497, and default hotkey configuration, page 498.

Settings



Visualization Manager[DCC_Control: Logic: Application]

General settings

☑ Use Unicode strings

□ Use CurrentVisu Variable

Extended settings

☑ Visible

Memory settings

Size of Memory for Visu: 400000

Size of Paintbuffer (per Client): 50000

Fig.4-233: "Visualization Manager" editor window

The settings on this tab apply for all application-related visualizations:

General settings:

- **Use Unicode strings:**

  If this option is selected, all character strings used in the visualization are processed in Unicode format.

- **Use CurrentVisu variable:**

  If this option is selected, the value of the "CurrentVisu" variable is considered. By assigning the name of the corresponding visualization to this variable as a value, it can be controlled which visualization is displayed in the program.

Extended settings:

- Enable this option if the following settings have to be modified (which should not be required for a default application):

  – **Size of Memory for Visu:**

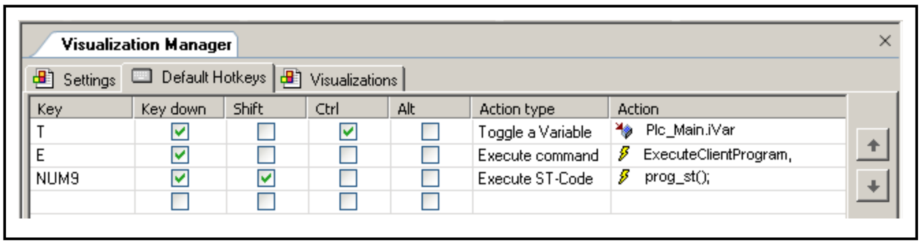
    Size of memory in bytes reserved for the visualization; default value: 400000

  – **Size of character buffer (Paintbuffer) (per client):**

    Buffer size in bytes reserved for character actions on the client system; default value: 50000

☞ If the diagnostic visualization is used, the settings currently not available for this visualization are grayed out.

Default hotkey configuration



Visualization Manager

Key	Key down	Shift	Ctrl	Alt	Action type	Action
T	☑	□	☑	□	Toggle a Variable	Plc_Main.iVar
E	☑	□	□	□	Execute command	ExecuteClientProgram,
NUM9	☑	☑	□	□	Execute ST-Code	prog_st();
	□	□	□	□		

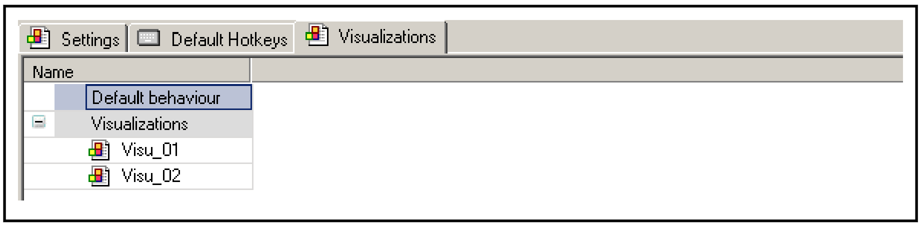
Fig.4-234: "Visualization Manager" editor, Default Hotkeys

This tab of the "Visualization Manager" dialog contains the hotkey configuration that applies for all of the visualizations belonging to the application. That means that the keys (hotkeys) defined can - if the respective device supports them - be used for user inputs in the visualization in online mode. The configurator can be operated as the one used in the visualization editor for a special visualization.

See the description of the hotkey configuration editor, page 494.

In addition, depending on the device, a few specific default hotkeys, page 657, always apply for navigation in a visualization.

Visualizations



Name
Default behaviour
Visualizations
Visu_01
Visu_02

Fig.4-235: "Visualization Manager" editor, Visualizations

This tab of the "Visualization Manager" dialog indicates the currently available visualizations.

By default, the visualizations directly attached below the current application or directly referenced by others are automatically loaded to the respective target system.

If the **Default behavior** is enabled, those in the table are selected in the respective "TargetVisualization" column.

By explicit selection of the checkmark, more visualizations can be added to the download package (which are not included by default, since they are only indirectly used/referenced by a variable in the application for example)

4.18 Watch List Editor

4.18.1 Watch List Editor, General Information

A watch list is a user-defined summary of project variables that is displayed in the watch window as table and is used for monitoring, page 500, their values.

Writing, page 144, and forcing, page 145, variables is also possible in the watch window.

An editor window for defining a watch list can be opened with the Watch r<n>, page 141, command (**Debug ▸ Monitor**).

Four watch windows are available for selection:

Watch 1, **Watch 2**, **Watch 3**, **Watch 4**.

The view **Watch All Forces** is automatically filled in, page 501 with all currently forced values in the active application.

4.18.2 Creating a Watch List

To define a variable for the watch list in one of the watch windows, open an input field in the "Expression" column.

To do this, select the field and then press the <space bar>. Enter the entire path of the desired variable. The input assistance is available via the ... button.

Syntax for a watch expression:

```
<DeviceName>.<ApplicationName>.<ObjectName>.<VariableName>
```

Example:

"DCC_Control_01.Application.Plc_Main.iVar_01", (see the figure below).

The symbol for the expression indicates whether it is a(n)

- input variable
- output variable or
- "normal" variable

When the input for the expression is complete, the variable type appears automatically in the "Data Type" column.

If a comment was added to the declaration of the variables, this is displayed in the "Comment" column.

The Value column then displays the current variable value in online mode.

To prepare a value for a variable, click on the respective field of the "Prepared Value" column and enter the desired value.

Handling a Boolean variable is even easier:

Switch back and forth between the Boolean values by pressing <Return> or the <space bar> according to the following sequence:

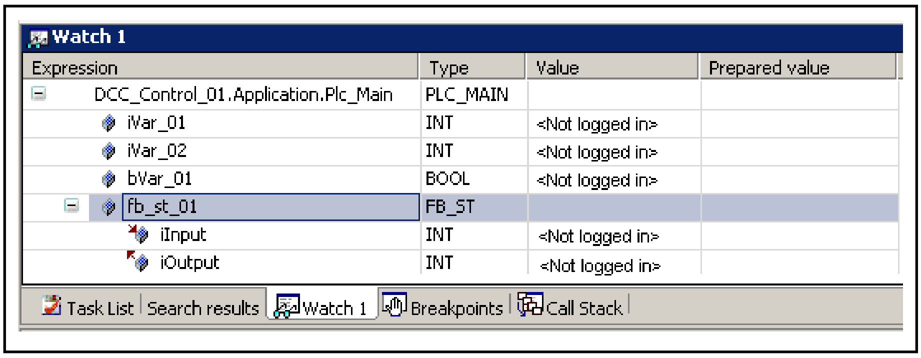
If the previous value is TRUE, the switch occurs as follows: FALSE -> TRUE -> no entry; otherwise.

if the previous value is FALSE, the switch occurs in the sequence TRUE -> FALSE -> no entry.

Enter all desired variables in the lines in the window this way. See the figure below with an example of a watch list in offline mode: The list contains expressions from the "Plc_Main" object.

In case of a structure variable, such as the function block instance, note that the individual components are automatically added to other lines when the instance name is entered (example: "DCC_Control_01.Application.Plc_Main.fb_st_01").

The components can also be hidden or shown using the plus or minus sign (parentheses, page 212).



Expression	Type	Value	Prepared value
⊟ DCC_Control_01.Application.Plc_Main	PLC_MAIN		
iVar_01	INT	<Not logged in>	
iVar_02	INT	<Not logged in>	
bVar_01	BOOL	<Not logged in>	
⊟ fb_st_01	FB_ST		
iInput	INT	<Not logged in>	
iOutput	INT	<Not logged in>	

Fig.4-236: Example, watch window in offline mode

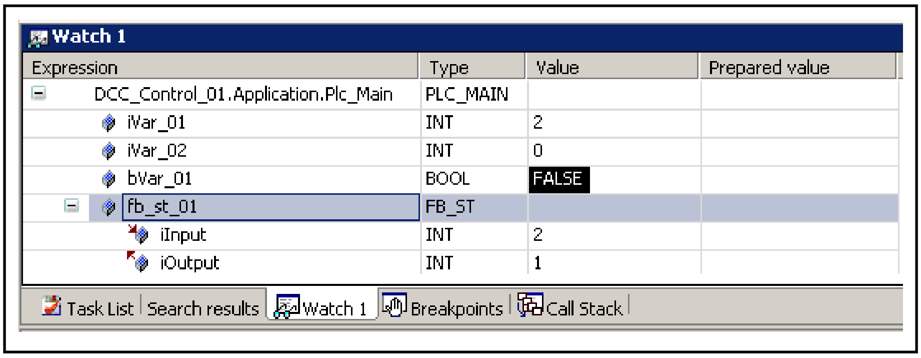
The list for monitoring the variable values can be used in online mode.

4.18.3 Watch List in Online Mode

Monitoring

In online mode, the value of the variables or the expression is displayed in the "Value" column of a watch list, page 499,.

The parentheses in case of structure variables are also displayed in Create watch list, page 499,.



Expression	Type	Value	Prepared value
⊟ DCC_Control_01.Application.Plc_Main	PLC_MAIN		
iVar_01	INT	2	
iVar_02	INT	0	
bVar_01	BOOL	FALSE	
⊟ fb_st_01	FB_ST		
iInput	INT	2	
iOutput	INT	1	

Fig.4-237: Example of a watch list in online mode

Writing or forcing values

In the "Prepared Value" column, a value which can be written to the variable on the control is entered and with which the variable can be "forced".

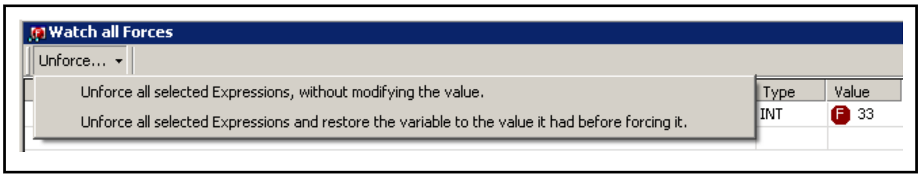
Refer to the descriptions for writing, page 144, and forcing, page 145, variables. This can also be done in other monitoring views as in the declaration editor.

Watch all forces

This is a special watch view, which is automatically filled at runtime with all currently forced values for the active application. "Expression", "Data Type", "Value" and "Prepared Value" are displayed as in the "Watch <n>" lists.

Below the "Unforce..." button, the following commands are available to unforce values:

- Unforce all selected expressions without modifying the value.
- Unforce all selected expressions and restore the variable to its original value.



Unforce all selected Expressions, without modifying the value.

Unforce all selected Expressions and restore the variable to the value it had before forcing it.

Type	Value
INT	F 33

Fig.4-238: Example – Watch all forces

5 Programming Reference

5.1 Declaration

5.1.1 Variable Declaration

A variable can be declared in the Declarations of a POU, page 326, in the statements for a POU using the Auto declare dialog, page 510,, in a DUT editor, page 338, or in a GVL editor, page 367.

The data type of the variables to be declared is specified by the keywords, page 513,.

For example, the declaration of simple variables begins with "VAR" and ends with "END_VAR".

For other types, refer to

- Input variables - VAR_INPUT, page 517,
- Output variables - VAR_OUTPUT, page 517,
- Input/output variables – VAR_IN_OUT, page 517,
- Global variables - VAR_GLOBAL, page 518,
- Temporary variables - VAR_TEMP, page 518,
- Static variables - VAR_STAT, page 518,
- External variables - VAR_EXTERNAL, page 519,
- Variable configuration - VAR_CONFIG, page 523.

The variable type keywords can be supplemented by attributes, page 519,, which also consist of keywords.

Example: "RETAIN" (VAR RETAIN).

A variable declaration has to follow these rules:

Syntax:

<Identifier> {AT<Address>}: <DataType> {:=<Initialization>};

The sections in curly brackets {} are optional.

Identifier

The identifier is the name of the variable. The points listed below have to be considered when assigning an identifier. For further recommendations for assigning an identifier, see Recommendations for assigning an identifier, page 505.

- An identifier may not contain any spaces or special characters.
- Case is not taken into consideration for identifiers, i.e. "VAR1", "Var1" and "var1" identify the same variable.
- Underscores are recognized (i.e. "A_BCD" and "AB_CD" are treated as two different identifiers), but a sequence of multiple underscores is not permitted.
- The length of an identifier and its significant parts are unlimited.
- Also note the rules for "Multiple use of identifiers (namespace, validity ranges)".

  Multiple use of identifiers (namespaces, validity ranges):

  - An identifier may not be locally used twice.
  - An identifier may not be identical with a keyword.

- An identifier may be used globally more than once. Thus, a local variable may have the same name as a global variable. In this case, the local variable takes precedence locally.
- A variable defined in a global variable list, page 52, can have the same name as a variable defined in another GVL.

  In this context, please note the following functionalities that extend the standard with regard to the **namespace**/validity range of variables that were not available in IndraLogic 1.x:

  1. Operators that apply globally, "global scope":

     An instance path that begins with "." always opens a global namespace. If a local variable (e.g. "`ivar`") has the same name as a global variable (e.g. "`.ivar`"), the global variable is addressed.
  2. The name of a "global variable list" can provide a unique definition for the namespace for the variables it contains. Thus, variables with the same name can be declared in different global variable lists and still be addressed uniquely by using the list name as prefix.

     Example:

     ```
     globlist1.ivar := globlist2.ivar;
     //ivar from GVL globlist2 is copied to ivar in GVL globlist1
     ```
  3. Variables defined in the global variable list of a library integrated into the project can be addressed uniquely based on the syntax

     "`<NameSpace library>.<GVLname>.<VariableName>`"

     The next point includes further information on the namespace for libraries.

     Example:

     ```
     globlist1.ivar := lib1.globlist1.ivar
     //ivar from GVL globlist1 in library lib1 is copied to ivar in GVL globlist1.
     ```
- When a library manager is added to a library, a namespace, page 367, is also defined.

  Thus, a library manager or a library variable can be uniquely addressed with

  "`<NameSpace library>.<BlockName|VariableName>`"

  If libraries are nested, note that the namespaces for all participating libraries have to be entered in sequence.

  Example: If Lib1 is referenced by Lib0, the function block "fun" in Lib1 is addressed using

  "`Lib0.Lib1.fun`"

  :

  ```
  ivar := Lib1.fun(4, 5);//return value from fun is copied to the variable ivar in the project
  ```

  If "Publish all IEC symbols in the referencing project" has been enabled in the Properties, page 230, of the referenced library "Lib1", fun can also be addressed directly using "Lib0.fun".

AT <Address>: Using the keyword"AT", the variable can be linked directly to a specific address, page 512,.

In function blocks, variables with incomplete address information can also be declared. To use such a variable in a local instance, a corresponding entry for this variable has to be in the "variable configuration".

Data type: valid data type, page 552, optionally extended by an

":=<Initialization>", page 509.

☞ Note that automatic declaration, page 510 is possible. To enter declarations more quickly, use the short form mode, page 511,.

Note that pragma statements, page 526, can be used to affect code generation in the declarations of an object.

5.1.2 Recommendations for Assigning an Identifier

Recommendations for Assigning an Identifier, General Information

Identifiers are assigned during the declaration, page 503, of variables (variable names, page 503), user-defined data types, page 560, and when POUs and visualizations are created (function blocks: functions, function blocks, programs).

In addition to the allocation rules for identifiers, page 503, the following rule is recommended for achieving the greatest possible consistency when assigning names:

- Variable names <Identifiers>, page 505
- Variable names in IndraLogic 2G libraries, page 507
- User-defined data types (DUT), page 507,
- User-defined data types (DUT) in IndraLogic 2G libraries, page 508
- Functions, function blocks, programs (POU), actions, page 509
- POUs in IndraLogic 2G libraries, page 509
- Identifiers for visualization, page 509

Variable Names <Identifiers>

The naming of variables should be related to the **Hungarian notation**.

A short, meaningful description should accompany each variable, the "basic name". The first letter of each respective word in a basic name should be written in upper case, the others in lower case (example: FileSize).

If necessary, the compilation file can also be generated in other languages.

"Prefix(es)" corresponding to the variable data type are attached to the front of the basic name in lower case letters.

Data type	Lower limit	Upper limit	Information content	Prefix	Comment
BOOL	FALSE	TRUE	1 bits	x[1]	
				b	Reserved

[1] *For Boolean variables, x was purposely chosen as a prefix to provide a separation from BYTE on one hand, and on the other, to accommodate the perspective of the IEC programmer (cf. address %IX0.0).*

Data type	Lower limit	Upper limit	Information content	Prefix	Comment
BYTE	16#00	16#FF	8 bits	by	Bit string, not for arithmetic operations
WORD	16#0000	16#FFFF	16 bits	w	Bit string, not for arithmetic operations
DWORD	16#00000000	16#FFFFFFFF	32 bits	dw	Bit string, not for arithmetic operations
LWORD	16#00000000 00000000	16#FFFFFFFF FF FFFFFF	64 bits	lw	Bit string, not for arithmetic operations
SINT	-128	127	8 bits	si	
USINT	0	255	8 bits	usi	
INT	-32.768	32.767	16 bits	i	
UINT	0	65.535	16 bits	ui	
DINT	-2.147.483.648	2.147.483.647	32 bits	di	
UDINT	0	4.294.967.295	32 bits	udi	
LINT	-2^{63}	2^{63} - 1	64 bits	li	
ULINT	0	2^{64} - 1	64 bits	uli	
REAL			32 bits	r	
LREAL			64 bits	lr	
STRING				s	
WSTRING				ws	
TIME				tim	
TIME_OF_DAY				tod	
DATE_AND_TIME				dt	
DATE				date	
ENUM			16 bits/32 bits	e	0...32767
POINTER				p	
ARRAY				a	

Example

```
bySubIndex: BYTE;
sFileName:  STRING;
udiCounter: UDINT;
```

In case of **nested declarations**, the prefixes are attached to each other in the order of the declaration:

Example

```
 pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

Function block instances and **variables** of user-defined data types have a short identifier for the FB or data type names as prefix.

Example

```
cansdoReceivedTelegram: CAN_SDOTelegram;

TYPE CAN_SDOTelegram :     (* prefix: sdo *)
STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Local constants (c) begin with the constant prefix "c" and an additional underscore "_", followed by a type prefix and the variable name.

Example

```
VAR CONSTANT
 c_uiSyncID: UINT := 16#80;
END_VAR
```

For **global variables** (g) and **global constants** (gc), an additional prefix and underscore are added to the library prefix.

Examples:

```
VAR_GLOBAL
    CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
    CAN_gc_dwExample: DWORD;
END_VAR
```

Variable Names in IndraLogic 2G Libraries

Variable names in IndraLogic 2G are formed as in the description above, except that global variables and constants do not require library prefixes, since this function is replaced by the namespace.

Example

```
g_iTest: INT; // Declaration
CAN.g_iTest   // Usage, call in program
```

User-Defined Data Types (DUT)

Structures:

The name of each structure data type consists of the library prefix (in the example: "CAN"), an underscore and a short, meaningful description of the structure (in the example: "SDOTelegram").

The corresponding prefix for variables created with this structure should follow as a comment directly after the colon.

Syntax:

`<Libraryprefix>_<Identifier>`

Example

```
TYPE CAN_SDOTelegram :     (* prefix: sdo *)
STRUCT
    wIndex:WORD;
    bySubIndex:BYTE;
    byLen:BYTE;
    abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Enumeration values:

Begin with the library prefix (in the example: "CAL") followed by an underscore and the identifier in upper case letters.

Syntax:

`<Libraryprefix>_<Identifier>`

☞ In past versions of IndraLogic, ENUM values > 16#7FFF have led to errors, since they were not automatically converted to INT. For this reason, ENUMs should always be defined with the correct INT values.

Example

```
TYPE CAL_Day :(
 CAL_MONDAY,
 CAL_TUESDAY,
 CAL_WEDNESDAY,
 CAL_THIRSDAY,
 CAL_FRIDAY,
 CAL_SATURDAY,
 CAL_SUNDAY);
END_TYPE

// Declaration:
VAR
 eToday: CAL_Day;
END_VAR
```

User-Defined Data Types (DUT) in IndraLogic 2G Libraries

The library prefix is not used for DUT names in IndraLogic 2G libraries, since its function is replaced by the "namespace".

Likewise, enumeration values are also defined without a library prefix:

Example (from a **library with namespace CAL**):

Type definition

```
TYPE Day :(
  MONDAY,
  TUESDAY,
  WEDNESDAY,
  THIRSDAY,
  FRIDAY,
  SATURDAY,
  SUNDAY);
END_TYPE
```

Declaration:

```
  eToday: CAL.Day;
```

Use in the application:

```
 IF eToday = CAL.Day.MONDAY THEN
```

POUs (Functions, Function Blocks, Programs, Actions...)

Functions, function blocks and programs consist of the library prefix (in the example: "CAN"), an underscore and a short, meaningful name of the POU (in the example: "SendTelegram").

As with the variables, the first letter of each respective word in a basic name should be written in upper case, the others in lower case. It is recommended to use a combination of a verb and a noun in the POU name.

Example

```
FUNCTION_BLOCK CAN_SendTelegram //prefix: can
```

The declaration contains a **short description** of the function block as a comment.

In addition, all **inputs** and **outputs** have comments.

For function blocks, the corresponding prefix for created instances should come right after the name as a comment.

Actions do not contain a prefix; only actions to be called internally only by the function block itself begin with prv_ (private).

For compatibility reasons to previous versions of IndraLogic, each **function** has to have at least one transfer parameter.

External functions may not use structures as return values.

POUs in IndraLogic 2G Libraries

The library prefix is not used for POU names in IndraLogic 2G libraries, since its function is replaced by the namespace.

Method names are created as action names.

Possible inputs for methods have to include comments. Likewise, the declaration should contain a short description of the method.

For methods, there are no limitations with regard to their implementation among external and internal libraries.

Interfaces should begin with the letter "I", e.g. "ICANDevice"

Identifiers for Visualizations

☞ Note that a visualization does currently not have the same name as another function block in the project, since this would lead to problems when switching visualizations.

5.1.3 Variable Initialization

The default initialization value for all declarations is 0. User-defined initialization values can also be entered in the declarations for each variable and each data type.

The user-defined initialization starts with the assignment operator ":=" and can consist of any valid ST expression, refer to ST expression, page 389.

Thus, the initial value can be defined using constants, other variables or functions.

The programmer has to ensure that a variable "x" used to initialize another variable, "y" is also declared.

Examples for valid variable initializations:

```
VAR
 var1:INT := 12;              // Integer variable with initial value 12
 x : INT := 13 + 8;           // Initial value is defined by a term of constants
 y : INT := x + fun(4);       // Initial value is defined by a term
                              // that contains a function call;
                              // in these cases please observe the order!

                              // Not described in the IEC61131-3
 z : POINTER TO INT := ADR(y); // standard: Initial value is defined by an address function;
                              // However, in this case the pointer is not initialized
                              // during an online change!
END_VAR
```

A description on the initialization can be found under

- Arrays, page 560,
- Structures, page 563,
- Unions, page 564
- Subrange Types, page 566,
- Expressions for variable initialization

☞ Starting from compiler version 3.3.2.0, variables from thr global variable lists, page 367, are always initialized before the local variables of a POU.

5.1.4 Any Expressions for Variable Initialization

A variable can be initialized with any desired ST expression, page 389,.

Other variables from the same namespace as well as function calls can also be used.

If a variable is initialized with another variable, the other variable should also be initialized.

Examples for valid variable initializations:

```
VAR
 x : INT := 13 + 8;
 y : INT := x + fun(4);

 //Warning: The pointer is not initialized in the case of an online change!
 z : POINTER TO INT := ADR(y);
END_VAR
```

5.1.5 Declaration Editor

The declaration editor is a text editor for the variable declaration.

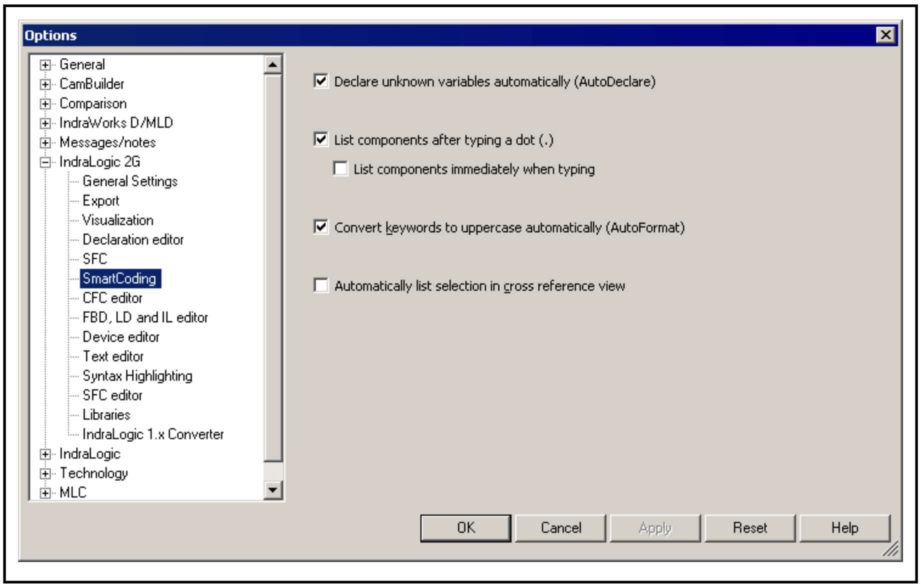
Usually, it is used with language editors.

Also refer to

- Declaration Editor, page 326.

5.1.6 "Auto Declare" Function

Under **Tools** ▸ **Options** ▸ **IndraLogic2G** ▸ **Smart coding** a setting can be made to open the "Auto Declare" dialog automatically as soon as a variable that is not declared is entered in the statements into an editor and <Enter> is pressed.



☑ Declare unknown variables automatically (AutoDeclare)

☑ List components after typing a dot (.)

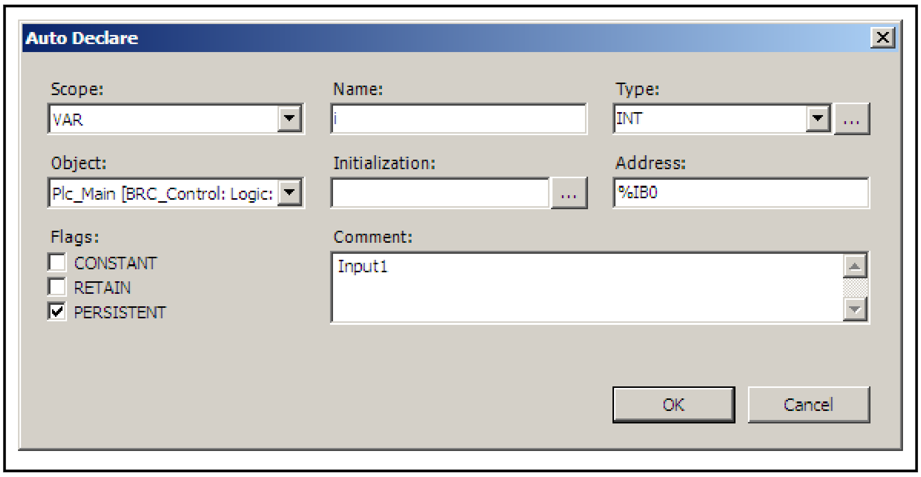
☐ List components immediately when typing

☑ Convert keywords to uppercase automatically (AutoFormat)

☐ Automatically list selection in cross reference view

Fig.5-1: "SmartCoding" options

The "Auto Declare" dialog supports the variable declaration.



Scope: VAR

Name: i

Type: INT

Object: Plc_Main [BRC_Control: Logic:

Initialization:

Address: %IB0

Flags:

☐ CONSTANT

☐ RETAIN

☑ PERSISTENT

Comment: Input1

Fig.5-2: "Auto Declare" dialog

The dialog to declare a variable can also be opened explicitly using the Declare variables, page 100, command (usually found in the "Edit" menu).

This command or the shortcut <Shift>+<F2> also opens the dialog, even if a variable that is already declared is selected in the statements of an editor.

A description of automatic declaration is found under Auto Declare, page 100.

5.1.7 Short Form Mode

In the declaration editor, as in other text editors for variable declaration, the "short form mode" for the input is supported.

☞ The statement in the editor does not support the "short form mode".

This mode is enabled if a declaration line is completed with the shortcut <Ctrl>+<Enter>.

In the short form mode, use short forms to enter a declaration instead of typing in all the details.

The following short forms are supported:

- All identifiers except the last identifier in a line become variable identifiers of a declaration.
- The declaration data type is specified by the last identifier in the line. The following applies here:

B or BOOL	results in	BOOL
I or INT	results in	INT
R or REAL	results in	REAL
S or string	results in	STRING

- If a data type cannot be specified using these rules, BOOL is automatically used as the data type and the last identifier in the line is not used as data type (see below: example)1).
- Depending on the type of declaration, each constant entered becomes an initialization or a string length specification (see below, examples (2) and (3)).
- An address (as in %MD12) is automatically extended with the AT attribute (see below, example (4)).
- A text following a semicolon (;) becomes a comment (see below, example (4)).
- All other characters in the line are ignored (see below, exclamation point in example (5)).

Examples:

	Short form	Resulting declaration
(1)	O	A: BOOL;
(2)	A B I 2	A, B: INT := 2;
(3)	ST S 2; A string	ST:STRING(2); // A string
(4)	X %MD12 R 5; Real Number	X AT %MD12: REAL := 5.0; // Real Number
(5)	B !	B: BOOL;

5.1.8 AT Declaration

A project variable in the declaration can be linked to a specific input, output or memory address of the control configured in the Project Explorer.

Syntax:

```
<Identifier> AT <Address>: <DataType>;
```

The keyword "AT" has to be followed by a valid address, page 623 that corresponds to the currently active control configuration in the Project Explorer.

This allows to give a meaningful name to the address. Modifications with regard to the incoming or outgoing signal may be carried out only at one position (e.g. in the declaration).

Note the following when assigning a variable to an address:

- Write access to variables to which an input was assigned is not possible.

- AT declarations can only be carried out for local and global variables, not for input and output function block variables.
- If AT declarations are used with structure or function block components, all instances use the same memory which is as using static variables in classic programming languages such as "C".
- The memory layout of structures depends on the target system.

Examples:

```
counter_heat7 AT %QX0.0: BOOL;
lightcabinetimpulse AT %IX7.2: BOOL;
download AT %MX2.2: BOOL;
```

☞ If Boolean variables are assigned to a BYTE, WORD or DWORD address, they assign TRUE or FALSE to an entire byte, not only to the first bit after the offset!

☞ I/O modules of the same type on the respective slot and in the same sequence are required for automatic address generation resulting in the same assignment and thus to the connection of the correct inputs and outputs.

Please check this carefully if the control is replaced.

Note that a variable can also be assigned to an address when configuring the modules.

5.1.9 Keywords

Keywords can be entered in upper case, lower case and mixed.

The following character strings are reserved as keywords, i.e. they cannot be used as identifiers for variables or POUs:

ABS, page 602

ACOS, page 607

ACTION (only used in export format)

ADD, page 570

ADR, page 587

AND, page 575

ANDN, page 575

ARRAY, page 560

ASIN, page 606

AT, page 512

ATAN, page 607

BITADR, page 588

BOOL, page 553

BY

BYTE, page 554

CAL, page 589

CALC, page 589

CALCN, page 589

CASE, page 394

CONSTANT, page 522

CONTINUE, page 396
COS, page 605
Date, page 555
DINT, page 554
DIV, page 571
DO, page 394
DT, page 555
DWORD, page 554
ELSE, page 393
ELSIF, page 393
END_ACTION (only used in export format)
END_CASE, page 394
END_FOR, page 394
END_FUNCTION (only used in export format)
END_FUNCTION_BLOCK (only used in export format)
END_IF, page 393
END_PROGRAM (only used in export format)
END_REPEAT, page 396
END_STRUCT
END_TYPE
END_VAR, page 516
END_WHILE, page 395
EQ, page 586
EXIT, page 396
EXP, page 604
EXPT, page 608
FALSE, page 553
FOR, page 394
FUNCTION, page 31
FUNCTION_BLOCK, page 33
GE, page 586
GT, page 584
IF, page 393
INDEXOF, page 574
INT, page 554
INTERFACE, page 38,
JMP, page 397
JMPC
JMPCN
LD
LDN
LE, page 585
LINT, page 554

LN, page 603
LOG, page 604
LREAL, page 554
LT, page 584
LTIME, page 555
LWORD, page 554
MAX, page 582
METHOD, page 45,
MIN, page 582
MOD, page 573
MOVE, page 573
MUL, page 570
MUX, page 583
NE, page 587
NOT, page 587
OF
OR, page 576
ORN, page 576
PARAMS, page 524
PERSISTENT, page 519
POINTER, page 557
PROGRAM, page 29
PROPERTY, page 46,
R
REFERENCE, page 556
READ_ONLY
READ_WRITE
REAL, page 554
REPEAT, page 396
RET
RETAIN, page 519
RETC
RETCN
RETURN, page 393
ROL, page 579
ROR, page 580
S
SEL, page 581
SHL, page 577
SHR, page 578
SIN, page 605
SINT, page 554
SIZEOF

SQRT, page 603

ST

STN

STRING, page 555

STRUCT, page 563

SUPER, page 36

SUB, page 570

TAN, page 606

THEN, page 393

TIME, page 555

TO

TOD, page 555

TRUE, page 553

TRUNC, page 601

TYPE

UDINT, page 554

UINT, page 554

ULINT, page 554

UNTIL, page 396

USINT, page 554

VAR

VAR_ACCESS (only used in special circumstances, depending on hardware)

VAR_CONFIG, page 523

VAR_EXTERNAL, page 519

VAR_GLOBAL, page 518

VAR_IN_OUT, page 517

VAR_INPUT, page 517

VAR_OUTPUT, page 517

VAR_STAT, page 518

VAR_TEMP, page 518

WHILE, page395

WORD, page 554

WSTRING, page 555

XOR

XORN

In addition, all conversion operators as listed in the input assistance are treated as keywords.

5.1.10 Local Variables (VAR)

All local variables of a function block are declared between the keywords VAR and END_VAR. External access to local variables is not possible.

VAR can be extended by an attribute, page 519,.

Example

```
VAR
 iLoc1:INT;
END_VAR
```

5.1.11 Input Variables (VAR_INPUT)

Variables used as input variables for a function block are declared between the keywords VAR_INPUT and END_VAR.

This means that when calling the function block, a value can be transferred to these variables.

VAR_INPUT can be extended by an attribute, page 519,.

Example

```
VAR_INPUT
 iIn1:INT;
END_VAR
```

5.1.12 Output Variables (VAR_OUTPUT)

All output variables of the function block are declared between the keywords VAR_OUTPUT and END_VAR.

This means that the values of these variables can be returned to the function block called. They can be queried and used there.

VAR_OUTPUT can be extended by an attribute, page 519,.

Example

```
VAR_OUTPUT
 iOut1:INT;
END_VAR
```

Output variables in functions and methods:

According to the IEC 61131-3 standard, functions and methods can have additional outputs. These have to be assigned when the function is called:

Example

```
fun(iIn1 := 1, iIn2 := 2, iOut1 => iLoc1, iOut2 => iLoc2);
```

The return value of the function "fun" is additionally calculated and transferred to its outputs.

5.1.13 Input/Output Variables (VAR_IN_OUT)

Variables used as input/output variables for a function block are declared between the keywords VAR_IN_OUT and END_VAR.

☞ For input/output variables, the value of the transferred variable is changed directly ("Transfer as pointer", Call by reference).

That means that the input value for such variables may not be a constant.

Thus, the VAR_IN_OUT variables for a function block cannot be read or described using

`<FBinstance>.<InputOutputVariable>`

from an external location.

Example

```
VAR_IN_OUT
 iInOut1:INT;
END_VAR
```

5.1.14 Global Variables (VAR_GLOBAL)

Variables, constants or remanent variables that should be known across the entire project can be declared as global variables.

☞ A variable declared locally in a function block and with the same name as a global variable has priority in the function block.

☞ Starting from compiler version 3.3.2.0, variables from thr global variable lists, page 367, are always initialized before the local variables of a POU.

The variables are declared locally between the keywords **VAR_GLOBAL** and **END_VAR**.

VAR_GLOBAL can be extended by an attribute, page 519,.

A variable is detected as a global variable if a dot is placed in fron of the variable name, e.g. ".iGlobVar1".

☞ For more detailed information on the multiple usage of variable names, on the operator for the global namespace "." and namespaces, see Global namespace operators, page 609.

Global variable lists can be used to manage global variables in a project.

A "GVL" can be added as an object in the Project Explorer using the Add, page 234 command.

5.1.15 Temporary Variables (VAR_TEMP)

The temporary variable functionality is an extension with regard to the IEC 61131-3 standard.

Temporary variables **are re-initialized every time the function block is called**. **VAR_TEMP** declarations can only be made in programs and function blocks.

The variables can only be accessed in the statement of the program or function block.

The variables have to be declared between the keywords **VAR_TEMP** and **END_VAR**.

5.1.16 Static Variables (VAR_STAT)

The static variable functionality is an extension with regard to the IEC 61131-3 standard.

Static variables can be used in function blocks, functions and methods. They have to be declared between the keywords **VAR_STAT** and **END_VAR** and **are initialized the first time the respective function block is called**.

Static variables can only be accessed in the namespace in which they are declared (as with static variables in C), but like global variables they keep their value even after the function block is exited again. They can be used as counters for function calls for example.

VAR_STAT can be extended by an attribute, page 519,.

5.1.17 External Variables (VAR_EXTERNAL)

External variables are global variables imported into a function block.

The variables have to be declared locally between the keywords **VAR_EXTERNAL** and **END_VAR**.

☞ If a VAR_EXTERNAL is not declared in a GVL, an error message is output.

☞ It is not required in IndraLogic to declare variables as external. The keyword in intended to ensure compliance with IEC 61131-3.

Example

```
VAR_EXTERNAL
 iVarExt1: INT :=12;
END_VAR
```

5.1.18 Attribute Keywords for Variable Types

The following keywords can be used to add the corresponding attributes in the declaration, page 503, of variable types.

RETAIN: See remanent variables, page 519, of type RETAIN.

PERSISTENT: See remanent variables, page 519, of type PERSISTENT.

CONSTANT: See constants, page 522,

5.1.19 Access Variables

### In preparation ###

5.1.20 Remanent Variables (VAR RETAIN, VAR PERSISTENT)

Remanent variables can retain their value for the entire program runtime. They are declared as pure "retain variables" or "persistent variable" or as a combination of retain and persistent.

Each has its own memory area used for management.

The type of declaration chosen specifies the degree of "resistance" a remanent variable has in case of a reset, download or computer reboot.

In practice, the combination of both types is most often requested (PERSISTENT).

☞ When an **IndraLogic1.x project** is opened, the declarations of retain variables remain effective and are not changed, but the declarations of persistent variables have to be revised or recreated.

An individual global variable list has to be created for the project! Refer to Persistent variables page 520.

For further information, refer to

- Retain Variables, page 519
- Persistent Variables, page 520
- Overview table on the behavior, page 521

Retain variables

The management of variables declared as retain variables depends on the target system: Usually, their are managed in their **own memory space**. They are identified by the keyword **RETAIN** in a function block or in a global variable list (GVL) in the IndraLogic project.

Example

```
VAR RETAIN
 iRem1 : INT;
END_VAR
```

Retain variables keep their value after an uncontrolled shutdown or in response to the online command Warm reset <Application>, page 135 as well as after switching the control off and on normally (reboot).

When the program restarts, the saved values are used for further processing. All other variables are re-initialized in this case, either with their initialized values or with default initializations.

Use case:

A counter in a production facility that is to continue counting after power failure.

However, retain variables are re-initialized at a Reset (origin), page 136 a Reset (cold), page 135 or a new program download.

The retain property can be combined with the persistent property. To do this, refer to the overview table, page 521 below.

☞ If a local variable is declared as RETAIN in a **program**, **this exact variable** is saved in the retain area (like a global retain variable).

If a local variable is declared as RETAIN in a **function block**, the **entire instance** of this function block is saved in the retain area (all function block data), although only the declared retain variable is treated as such.

If a local variable is declared in a **function** as RETAIN, it has no effect! The variable is not saved in the retain area! If a local variable is declared as PERSISTENT in a function, it has also no effect!

☞ The memory space for retain and persistent variables is limited to 127 KB each.

Persistent variables

Currently, persistent variables are treated as persistent retain variables. In this case, the values are kept even at a control reboot. See below.

Persistent variables are identified by the keyword "PERSISTENT" (**VAR_GLOBAL PERSISTENT**). They are only re-initialized during a reboot or reset (origin), page 136, of the control.

In contrast to the retain variables, they keep their value after a download. A use case for "persistent retain variables" might be a counter for operating hours which is supposed to continue counting after a power failure or download. See below Overview table on the behavior, page 521

Persistent variables are treated as follows and thus different as in IndraLogic1.x:

Persistent variables can ONLY be declared in a **special global variable list** of the object type persistent variables, page 54, that is part of an application. There is only "ONE" such list per application.

☞ From V3.3.0.1, a declaration with "VAR_GLOBAL PERSISTENT" has the same effect as a declaration with "VAR_GLOBAL PERSISTENT RETAIN" or "VAR_GLOBAL RETAIN PERSISTENT".

Like retain variables, persistent variables are managed in their own memory space

Example

```
VAR GLOBAL PERSISTENT RETAIN
 iVarPers1 : DINT;
 bVarPers : BOOL;
END_VAR
```

☞ Currently, only **global** persistent variables can be created

The target system has to provide one separate memory space per application for the persistent variable list.

Each time the **application is loaded**, the persistent variable list on the control is compared with that in the project. The variable list on the control is identified by the application name among others. In case of **inconsistencies**, the user is prompted to initialize all persistent variables before the download. Inconsistencies occur due to renaming, deletion or other modifications to existing persistent variable declarations.

☞ Thus, carefully consider **each change in the declaration part** of the persistent variable list and the effects on a re-initialization subsequently asked.

New declarations can only be added to the end of the list, but they are identified as new while loading. Thus, re-initializing the entire list is not necessary.

Overview table on the behavior of remanent variables

After online command	VAR	VAR RETAIN	VAR PERSISTENT VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset warm	-	x	x
Reset cold	-	-	x
Reset origin	-	-	-
Loading (= download)	-	-	x
Online change	x	x	x
Reboot control	-	x	x

x	Value is kept
-	Value is re-initialized

Fig.5-3: Overview table on the behavior of remanent variables

⚠ WARNING **Dangerous state due to PERSISTENT data in the remanent data memory**

The device description can define that the PERSISTENT data be mapped in the remanent data memory. In this case, the values are kept even at control reboot!

Make absolutely sure that the PERSISTENT data cannot cause damage at a reboot.

☞ The maximum memory space for retain and persistent variables is limited to 127 KB each.

5.1.21 Constants (VAR CONSTANT), Typed Constants

CONSTANT

Constants are identified by the keyword **CONSTANT**. They can be declared locally or globally.

Syntax:

```
VAR CONSTANT
 <Identifier> : <Type> := <initialization>;
END_VAR
```

Example

```
VAR CONSTANT
 c_iCon1:INT:=12;
END_VAR
```

In the description of the operands, page 615, a list of possible constants, page 616 can be found.

Typed constants can also be used:

Typed constants (typed literals)

Normally, the smallest possible data type is used when calculating with IEC constants. If another data type is to be used, this can be done with constants of a specific data type (typed literals) to which a specific type is assigned.

In this case, the constants do not have to be declared explicitly as "VAR CONSTANT" in the declaration. The constants are provided with a prefix that specifies the type:

It is written as follows:

Syntax:

```
 <Type>#<Literal>
```

<Type> indicates the desired data type.

Possible inputs:

BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, LINT, ULINT, LWORD, REAL, LREAL.

The type has to be written in upper case letters.

<Literal> indicates the constant. The input has to match the data types specified under <Type>.

Example

```
 iVar1:= DINT#34 ;
```

If the constant cannot be transferred to the target type without loss of data, an error message is output.

Constants of a specific data type can be used anywhere normal constants can be used.

☞ Furthermore, the system is also provided with time literals, page 555, and corresponding constants: TIME#,T#, DATE#, D#, TIME_OF_DAY#, TOD#, DATE_AND_TIME#, DT#

and character string literals, page 555, and corresponding constants CHAR#, WCHAR#, STRING#, WSTRING#.

Constants in online mode

If the default setting "Replace constants" is selected, constants in online mode in the declaration or monitoring window are indicated by a preceding symbol C in the value column. In this case, they cannot be accessed by forcing or writing for example.

Replace constants: This option is selected by default. This means: Its value is directly loaded for each constant, page 522, of scalar type (not for strings, arrays and structures). In online mode, constants are labeled in the declaration editor, page 326, or in the monitoring window, page 141, by a symbol preceding the value. In this case, access via an ADR operator, forcing and writing is not possible. If the option is disabled, the constant can be accessed, but the computing time increases.

5.1.22 Variable Configuration (VAR_CONFIG)

The variable configuration can be used to "map" function block variables to the process image, i.e. to assign the variables to the device I/Os without having to specify the exact address, page 622, at the variable declaration in the function block. In this case, the address is assigned centrally in a **global VAR_CONFIG list** below the application for all function block instances of the application.

Incomplete addresses are assigned to the function block variables in the declaration between the keywords "VAR" and "END_VAR" for this purpose.

These addresses are identified by a "*".

Syntax:

```
<Name> AT %<I|Q>* : <Data type>;
```

Example for the assignment of incompletely defined addresses:

```
FUNCTION_BLOCK locio
VAR
 xLocIn AT %I*: BOOL := TRUE;
 xLocOut AT %Q*: BOOL;
END_VAR
```

Two local I/O variables - a local input variable (%I*) and a local output variable (%Q*) - are defined here.

The **final definition** of the addresses is then made in the "variable configuration" in a global variable list:

To do this, use the Add, page 234 command to insert an object of the type "global variable list" (GVL) into the Project Explorer below the desired application.

In this GVL, enter the declarations of the instance variables with the exact addresses between the keywords **VAR_CONFIG** and **END_VAR**.

The keyword **VAR_GLOBAL** is replaced by the keyword **VAR_CONFIG**.

The instance variables have to be specified with the complete instance path in which the individual POU and instance names are both separated by a dot each.

The declaration has to contain an address, page 622, whise class (input/output) matches with that of the incomplete specification (%I*, %Q*) in the function block. The data type has to match as well.

Syntax:

```
<instance variable path> AT %<I|Q><location> : <data type>;
```

Configuration variables whose instance paths are invalid since the instance does not exist, are reported as errors.

But an error is also output if there is no address configuration present for an instance variable declared with an incomplete address.

Example for a variable configuration:

The following usage of the function block "locio" exists in a program (see example above).

Declaration:

```
PROGRAM PlcProg
VAR
  locioVar1: locio;
  locioVar2: locio;
END_VAR
```

Then, a correct variable configuration would look as follows for example:

Configuration:

```
VAR_CONFIG
  PlcProg.locioVar1.xLocIn AT %IX1.0 : BOOL;
  PlcProg.locioVar1.xLocOut AT %QX0.0 : BOOL;
  PlcProg.locioVar2.xLocIn AT %IX1.0 : BOOL;
  PlcProg.locioVar2.xLocOut AT %QX0.3 : BOOL;
END_VAR
```

☞ Modifications of variables directly assigned to the I/O addresses are displayed immediately in the process image, while modifications of variables "mapped" using a variable configuration are displayed when the task responsible is completed.

5.1.23 User-Defined Data Types

In addition to the standard data types, user-defined data types can also be used.

On declaration and initialization, refer to the description of User-Defined Data Types, General Information , page 560, (Data Unit Type).

5.1.24 Extendable Functions, PARAMS

### In preparation ###

5.1.25 FB_init, FB_reinit Methods

FB_init

The "FB_init" method replaces the INI operator used in IndraLogic 1.x.

A method named FB_init is a special method, page 45, for a function block.

It can be declared explicitly but is always created implicitly as well. Thus, it can be controlled for every function block.

The FB_init method contains an **initialization code** for the function block based on the declarations in the declaration part of the function block.

If the init method is also explicitly declared, the implicit initialization code is added to the explicitly created method.

The programmer can then add further initialization code.

☞ When the execution reaches the user-defined initialization code, the function block has already been completely initialized using the implicit initialization code.

The FB_init method is called for all declared instances after a download. **Attention:** For online changes, the most recent values overwrite the initialization values.

To call in sequence in case of inheritance, refer to: See FB_exit, call sequence, page 526.

Also refer to the option to call a function block method automatically after the initialization via FB_init: Attribute call_after_init, page 530.

Interface of the FB_init method:

```
METHOD fb_init : BOOL
VAR_INPUT
 bInitRetains: BOOL; // if TRUE, the Retain variables are initialized
                     // (warm start / cold start)
 bInCopyCode : BOOL; // if TRUE, the instance is copied
                     // into the Copy-Code (Online Change)
END_VAR
```

The return value is not used.

☞ An "fb_exit" method and the resulting processing sequence can also be used. See FB_exit method, page 526.

User-defined input:

Additional inputs can be defined in an FB_init method. These have to be assigned in the declaration of the function block instance.

Example of an init method for a function block called "serialdevice":

```
METHOD fb_init : BOOL
VAR_INPUT
  bInitRetains : BOOL; // Initialization of Retain variables
  bInCopyCode : BOOL;  // Instance is copied into the  Copy-Code
  nCOMnum : INT;       // additional input: Number of the COM
                       // interfaces that is leading
END_VAR
```

Declaration of the function block "serialdevice":

```
COM1 : serialdevice(nCOMnum:=1);
COM0 : serialdevice(nCOMnum:=0);
```

FB_reinit

If a method is declared with the name "FB_reinit" for a function block instance, it is called if the **instance is copied** (for example at an online change after changes were made in the function block declaration).

The method re-initializes the instance module generated by the copy code. A re-initialization may be desirable, since after copying, the original instance data is written on the newly created instance, but the original values are the desired ones. Note that the FB_reinit method **has to be explicitly declared** in contrast to the FB_init method. If the basic implementation of the function block should be re-initialized, the FB_reinit has to be called explicitly for this function block.

The FB_reinit method has no inputs.

To call in sequence in case of inheritance, refer to: FB_exit method, page 526.

5.1.26 FB_exit Method

The method named "FB_exit" is a special method for a function block. It has to be declared explicitly. There is no implicit declaration.

The "exit" method - if available - is called for all declared instances of the function block before a new download or during online changes for all new or deleted instances.

Interface of the FB_exit method: There is only one obligatory parameter:

Interface of the FB_exit method:

```
METHOD fb_exit : BOOL
VAR_INPUT
 bInCopyCode : BOOL; // if TRUE, the exit method is called
                     // to leave the instance, which is copied
                     // afterwards (Online Change).
END_VAR
```

See also the FB_init method, page 524 and the following execution sequence:

1. exit method: exit old instance

   `old_inst.fb_exit(bInCopyCode := TRUE);`

2. init method: initialize new instance:

   `new_inst.fb_init(bInitRetains:=   FALSE,bInCopyCode:= TRUE);`

3. Copying the function block values (copy code):

   `copy_fub(&old_inst, &new_inst);`

In case of inheritance, the following call sequence applies additionally (the following is assumed for the POUs named in this list as example).

`SubFB EXTENDS MainFB` and `SubSubFB EXTENDS SubFB`):

Call sequence:

```
fbSubSubFb.FB_Exit(...);
fbSubFb.FB_Exit(...);
fbMainFb.FB_Exit(...);
fbMainFb.FB_Init(...);
fbSubFb.FB_Init(...);
fbSubSubFb.FB_Init(...);
```

for FB_reinit:

```
fbMainFb.FB_reinit(...);
fbSubFb.FB_reinit(...);
fbSubSubFb.FB_Init(...);
```

5.1.27 Pragma Statements

Pragma Statements, Overview

A pragma statement is used to affect the properties of one or more variables with regard to compiling or precompiling (preprocessor). That means that a pragma affects the code generation.

For example, a pragma statement can specify if a variable is initialized, displayed in online mode, added to the symbol list or kept invisible in the library manager. During the compilation, message outputs can be forced. Conditional pragmas that specify how a variable is to be interpreted under certain conditions can be used. These conditional pragmas can also be entered as "compiler definitions" in the compiler properties, page 238, of an object.

A pragma can be inserted in a separate line or together with the code in an implementation or declaration line. In the FBD/LD/IL editor, call the "Add jump label" command first and then the "Label:" entry in the label text field has to be replaced by the corresponding pragma statement.

☞

1. If a label and a pragma should be used, enter the pragma first and then the jump label.
2. A pragma statement is given in curly brackets.
3. **Lower-case letters** are currently required for pragma statements.
4. If the compiler cannot interpret the statement text, the entire pragma is treated as a comment and is not read.

   However, a warning is output.

Syntax:

```
{ <instruction text> }
```

The opening bracket may be placed directly after a variable name. Opening and closing brackets always have to be placed in the same line.

Depending on the type and content of a pragma, it affects either:

- the line in which it is located
- or all following lines until it is canceled by a corresponding pragma or
- until the same pragma is executed with other parameters or
- the end of the file is reached.

A file is a: declaration part, statement part, global variable list, type declaration.

Possible pragma types:

Message pragma, page 527

Obsolete pragma, page 542

Attribute pragma, page 528

Conditional pragma, page 546

Pragma for symbol configuration, page 545

### In preparation ###: Pragmas as in IndraLogic 1.x.

Message Pragmas

Message pragmas can be used to force messages to be output in the message window during the compilation (project build).

The pragma statement can be inserted into a separate or an existing line in the text editor of a POU and considered when the project is compiled.

There are four types of message pragmas:

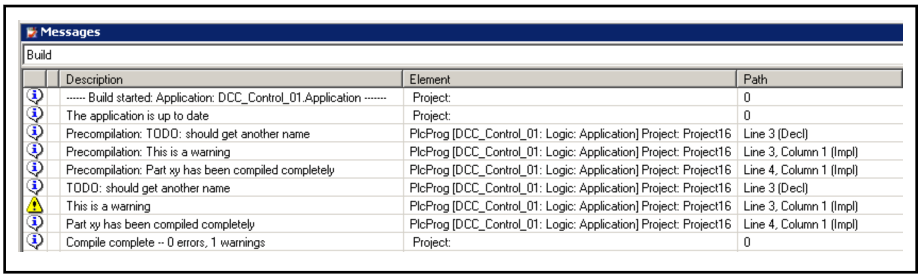
Pragma	Message type
{text 'textstring'}	**Text** ⓘ: The text specified, "Textstring", is output.
{info 'textstring'}	**Information** ⓘ: The text specified, "Textstring", is output.
{warning digit 'textstring'}	**Warning** ⚠: The text specified, "Textstring", is output. ### In preparation ###: The specified number indicates the warning level (between 1 and 5). In contrast to a "data type global " obsolete pragma, the warning is locally defined for the current position.
{error 'textstring'}	**Error** ⊗: The text specified, "Textstring", is output.

☞ For the message types "information", "warning" or "error message", use the Next message, page105, command to move to the source position of the message, i.e. the position at which the pragma is positioned in the POU.

Declaration and implementation in the ST editor:

```
VAR
 ivar : INT; {info 'TODO: should get another name'}
 bvar : BOOL;
 arrTest : ARRAY [0..10] OF INT;
 i: INT;
END_VAR

 arrTest[i] := arrTest[i]+1;
 ivar:=ivar+1;
 {warning 'This is a warning'}
 {text 'Part xy has been compiled completely'}
```



Messages

Build

	Description	Element	Path
ⓘ	------ Build started: Application: DCC_Control_01.Application -------	Project:	0
ⓘ	The application is up to date	Project:	0
ⓘ	Precompilation: TODO: should get another name	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 3 (Decl)
ⓘ	Precompilation: This is a warning	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 3, Column 1 (Impl)
ⓘ	Precompilation: Part xy has been compiled completely	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 4, Column 1 (Impl)
ⓘ	TODO: should get another name	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 3 (Decl)
⚠	This is a warning	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 3, Column 1 (Impl)
ⓘ	Part xy has been compiled completely	PlcProg [DCC_Control_01: Logic: Application] Project: Project16	Line 4, Column 1 (Impl)
ⓘ	Compile complete -- 0 errors, 1 warnings	Project:	0

Fig.5-4: Example for an output in the message window

Attribute Pragmas

Attribute Pragmas, General Information

Attribute pragmas can be added to a signature to influence the compilation or precompilation. All variables in the declaration is one signature.

There are user-defined attributes used with conditional pragmas, page 546,, see User-defined attributes, page 529, for a description, but also refer to the predefined attribute pragmas listed below:

☞

1. A pragma statement is given in curly brackets.
2. **Lower-case letters** are currently required for pragma statements.
3. If the compiler cannot interpret the statement text, the entire pragma is treated as a comment and is not read.

   However, a warning is output.

- Attribute 'call_after_init', page 530,
- Attribute 'displaymode', page 531,
- Attribute 'enable_dynamic_creation', page 531,
- Attribute ''external_name', page 531,
- Attribute 'expandfully', page 531,
- Attribute 'global_init_slot', page 532,
- Attribute 'hide', page 533,
- Attribute 'hide_all_locals', page 534,
- Attribute 'initialize_on_call', page 534,
- Attribute 'init_namespace', page 534,
- Attribute 'init_on_onlchange', page 535,
- Attribute 'instance-path', page 535,
- Attribute 'linkalways'}, page 536,
- Attribute 'monitoring', page 536,
- Attribute 'no_check', page 538,
- Attribute 'no_copy', page 538,
- Attribute 'no-exit', page 539,
- Attribute 'no_init', page 539,
- Attribute 'no_virtual_actions', page 540,
- Attribute 'obsolete', page 542,
- Attribute 'pack_mode', page 542,
- Attribute 'parameterstringof', page 543,
- Attribute 'qualified_only', page 544,
- Attribute 'reflection', page 544,
- Attribute 'relative_offset', page 545,
- Attribute 'symbol', page 545.

User-Defined Attributes

Any desired user-defined or application-defined attribute can be assigned to a function block, a type definition or a variable. This attribute can be prompted before compiling the project using conditional pragmas, page 546,.

Syntax:

```
{attribute 'attribute'}
```

This pragma always affects the current line or, if it is located in a separate line, the following line.

An attribute can be assigned to the following objects:

Example: POUs, actions

Attribute 'vision' for function fun1:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
 i : INT;
END_VAR
VAR

END_VAR
```

Example: variables

Attribute 'docount' for variable ivar:

```
PROGRAM Plc_Main
VAR
 ivar:INT; {attribute 'docount'};
 bvar:BOOL;
END_VAR
```

Example: types

Attribute 'atype' for data type DUT_1:

```
{attribute 'atype'}
TYPE DUT_1 :
STRUCT
 a:INT;
 b:BOOL;
END_STRUCT
END_TYPE
```

To use conditional pragmas, page 546.

Attribute 'call_after_init'

This pragma can be used to define a method to be implicitly called after the initialization of a function block instance.

For this purpose, the attribute has to be added to the function block as well as to the method (due to reasons of performance).

The method has to be called after FB_Init, page 524 and after the variable values of an initialization expression became valid in the instance declaration. This functionality is supported from the compiler version >= 3.4.1.0.

Syntax:

```
{attribute 'call_after_init'}
```

Example

With the following function block definition:

```
{attribute 'call_after_init'}
FUNCTION_BLOCK FB
... <functionblock definition>
```

and the method definition:

```
{attribute 'call_after_init'}
METHOD FB_AfterInit
... <method definition>
```

... a declaration such as:

```
inst : FB := (in1 := 99);
```

... is implemented in the following code processing:

```
inst.FB_Init();
inst.in1 := 99;
inst.FB_AfterInit();
```

Thus, the FB_AfterInit can react on the user-defined initialization.

Attribute 'displaymode'

The display mode of an individual variable can be defined using this pragma.

This specification overwrites the global setting for the display of all monitoring variables made using the commands in the display mode submenu (located in the debug menu by default).

The pragma has to be located in the line above the line which contains the variable declaration.

Syntax:

```
 {attribute 'displaymode':= '<displaymode>'}
```

The following definitions are possible:

for a display in binary format:

```
 {attribute 'displaymode':='bin'}
 {attribute 'displaymode':='binary'}
```

for a display in decimal format:

```
 {attribute 'displaymode':='dec'}
 {attribute 'displaymode':='decimal'}
```

for a display in hexadecimal format:

```
 {attribute 'displaymode':='hex'}
 {attribute 'displaymode':='hexadecimal'}
```

Example

```
VAR
 {attribute 'displaymode':='hex'}
 dwVar1: DWORD;
END_VAR
```

Attribute 'enable_dynamic_creation'

### In preparation ###

Attribute 'external_name'

The pragma specifies the name of an externally implemented function or function block in the runtime environment.

It can only be used for functions and function blocks.

Syntax:

```
{attribute 'external_name':='<implementation_name>'}
```

Example

```
{attribute 'external_name':='myFunctionBlockImplementationName'}
FUNCTION_BLOCK MyFunctionBlock
...
```

Attribute 'expandfully'

The components of an array used as input variable for referenced visualizations in the Properties dialog, page 451, of the visualization can become visible using this pragma.

Syntax:

```
{attribute 'expandfully'}
```

Example:

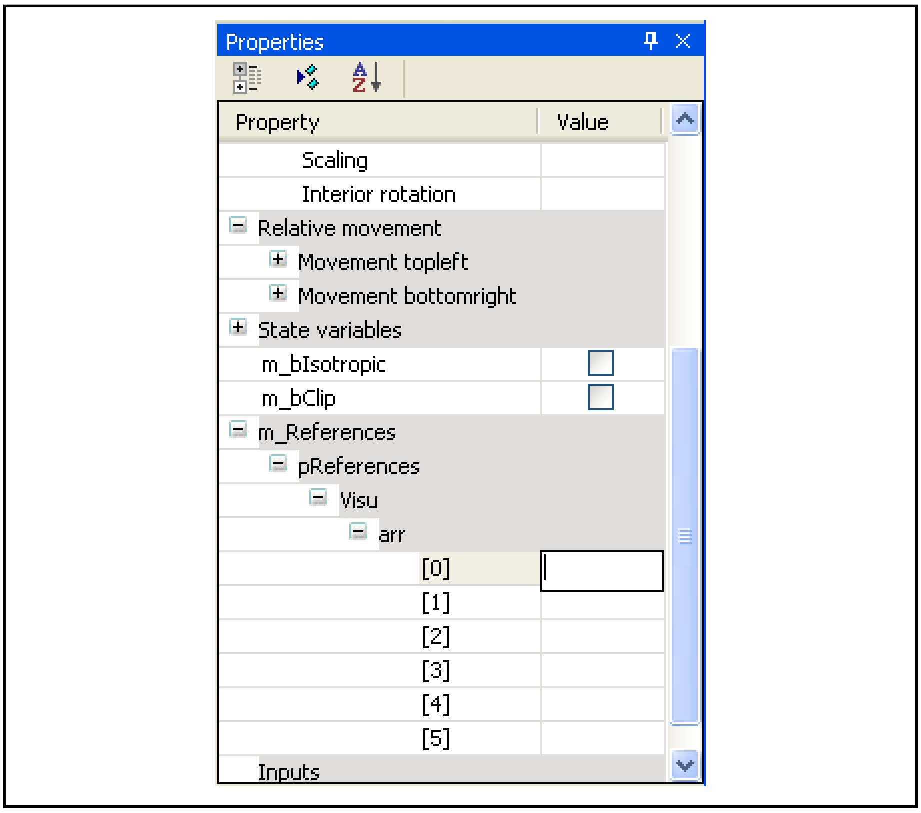
Visualization **visu** is to be inserted into a frame in the visualization **visu_main**.

arr is defined as input variable in the "visu" interface editor and is thus available for for assignments in the property dialog of the frame in **visu_main**.

To provide the individual components of the array in this "Property" dialog, the 'ExpandFully' attribute has to be added to the interface editor of **visu** directly in front of **arr**.

Declaration in the interface editor of "visu":

```
VAR_INPUT
{attribute 'expandfully'}
arr : ARRAY[0..5] OF INT;
END_VAR
```



Properties

Property	Value
Scaling	
Interior rotation	
⊟ Relative movement	
⊞ Movement topleft	
⊞ Movement bottomright	
⊞ State variables	
m_bIsotropic	□
m_bClip	□
⊟ m_References	
⊟ pReferences	
⊟ Visu	
⊟ arr	
[0]	
[1]	
[2]	
[3]	
[4]	
[5]	
Inputs	

Fig.5-5: "Properties" dialog for the frame in "visu_main"

Attribute 'global_init_slot'

This pragma can only be used for signatures.

The sequence for the initialization of variables from global variable lists is not specified from the very beginning. Sometimes, however, specifiying such a sequence is necessary for example if the variables of a list of variables depend on another list.

In this case, the pragma for specifying the sequence during the global initialization can be used.

Syntax:

```
{attribute 'global_init_slot' := '<value>'}
```

The placeholder `<value>` has to be replaced by an integer value determining the significance in the initialization sequence.

The default value is 5000.

A lower value causes an earlier initialization. If several signatures have the same significance for the attribute 'global_init_slot', the sequence of their initialization remains unspecified.

Example: The example project contains two global variable lists, GVL_1 and GVL_2:

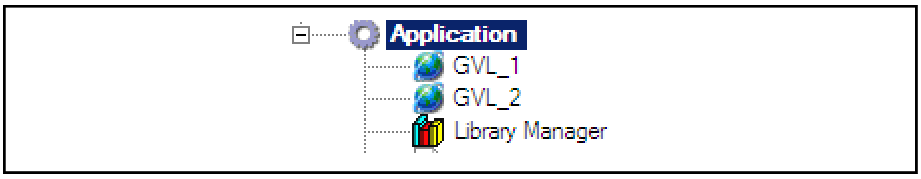


Fig.5-6: Two global variable lists

The global variable "A" is part of the global variable list GVL_1:

```
{attribute 'global_init_slot' := '300'}
VAR_GLOBAL
      A: INT:= 1000;
END_VAR
```

The initialization values of the variables "B" and "C" from GVL_2 depend on the variable "A".

Initialization values of the variables "B" and "C":

```
{attribute 'global_init_slot' := '350'}
VAR_GLOBAL
      B : INT:=A+1;
      C : INT:=A-1;
END_VAR
```

If the attribute 'global_init_slot' of the global variable list GVL_1 is set to 300 - that is the lowest initialization value in the example - it is ensured that the expression "A+1" is well defined when "B" is initialized.

Attribute 'hide'

Use the pragma to prevent that variables or even entire signatures become visible in the "List component" functionality, in the input assistance or in the declaration part in online mode.

Only the variable directly following the pragma becomes invisible.

Syntax:

```
{attribute 'hide'}
```

To hide all local variables of a signature, use the attribute Attribute 'hide_all_locals' page534.

Example:

Function block myPOU with attribute "hide":

```
FUNCTION_BLOCK myPOU
VAR_INPUT
      a:INT;
      {attribute 'hide'}
      a_invisible: BOOL;
      a_visible: BOOL;
END_VAR
VAR_OUTPUT
      b:INT;
END_VAR
VAR
END_VAR
```

In the main program, two instances of the function block myPOU are defined:

Two instances of the function block myPOU in the program Plc_Main:

```
PROGRAM Plc_Main
VAR
  POU1, POU2: myPOU;
END_VAR
```

While the input value for POU1 is implemented for example, the "List component" function that opens when jogging "POU1" displays (in the implementation part of PLC_PRG) the variables "a", "a_visible" and "b", but not the hidden variable "a_invisible".

Attribute 'hide_all_locals'

Use this pragma to hide local variables of a signature in the display of the "List component" functionality and the input assistant of the online monitoring in the declaration part.

The effect of this attribute is identical to that of the usage of the attribute 'hide' on each local variable.

Syntax:

```
{attribute 'hide_all_locals'}
```

Example:

The function block myPOU is implemented using the attribute:

Function block myPOU with attribute:

```
{attribute 'hide_all_locals'}
FUNCTION_BLOCK myPOU
VAR_INPUT
      a:INT;
END_VAR
VAR_OUTPUT
      b:BOOL;
END_VAR
VAR
      c,d:INT;
END_VAR
```

In the main program, two instances of the function block myPOU are defined:

Instances of the function block

```
PROGRAM Plc_Main
VAR
      POU1, POU2: myPOU;
END_VAR
```

While an input value for POU1 is implemented, the input assistance (Intellisense), which opens when typing "POU1" (in the statements of Plc_Main), displays the variables "a" and "b", but not the hidden local variables "c" or "d".

Attribute 'initialize_on_call'

This pragma can only be used for input variables.

An input variable of a function block that has this attribute is initialized every time the function block is called. If an input expects a pointer and if it was removed during an online change, this input is set to ZERO.

Syntax:

```
{attribute 'initialize_on_call'}
```

Attribute 'init_namespace'

A STRING or WSTRING variable provided with this pragma is initialized with the current namespace.

To use this pragma, the additional attribute 'noinit', page 539, has to be used for the STRING variable.

Syntax:

```
{attribute 'init_namespace'}
```

Example

```
PROGRAM PLC_PRG
VAR
    {attribute 'init_namespace'}
    {attribute 'noinit'}
    newString: STRING;
END_VAR
```

The variable "newString" is initialized with the current namespace, e.g. "PLC1.app1.PLC_PRG.newString".

Attribute 'init_on_onlchange'

If the pragma is applied to a variable, it is initialized during each online change, page 80,.

Syntax:

```
{attribute 'init_on_onlchange'}
```

Attribute 'instance-path'

The pragma can be applied to a local string variable.

The string variable is initialized with the Project Explorer path of the POU to which it belongs. This can be useful for error messages.

The application of this pragma requires the application of the attribute 'reflection', page 544, to the associated POU and the application of the additional attribute 'noinit', page 539, to the string variable itself.

Syntax:

```
{attribute 'instance-path'}
```

Example: Function block "POU" with all necessary attributes

Declaration of the function block POU:

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
      {attribute 'instance-path'}
      {attribute 'noinit'}
      str: STRING;
END_VAR
```

In the main program, one instance, "myPOU", of the function block POU is defined:

Instance creation and usage:

```
PROGRAM Plc_Main
VAR
     myPOU:POU;
     myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

The initialization of the instance myPOU contains the string variable "str" which contains the path of the instance myPOU.

In the example "DCC_Control.Application.Plc_Main.myPOU" this path is assigned to the variable "myString" in the main program.

Attribute 'linkalways'

Use this pragma to highlight the belonging object at the compiler. Thus, it is always included in the compiler information which means that it is always compiled and loaded to the PLC. The pragma is only effective for POUs and GVLs located below an application or in libraries below an application.

The compiler option POU property 'Link Always', page 241, performs the same.

Syntax

```
{attribute 'linkalways'}
```

When using the symbol configuration editor, page 398,, the highlighted POU is used as basis for the selectable variables of the symbol configuration.

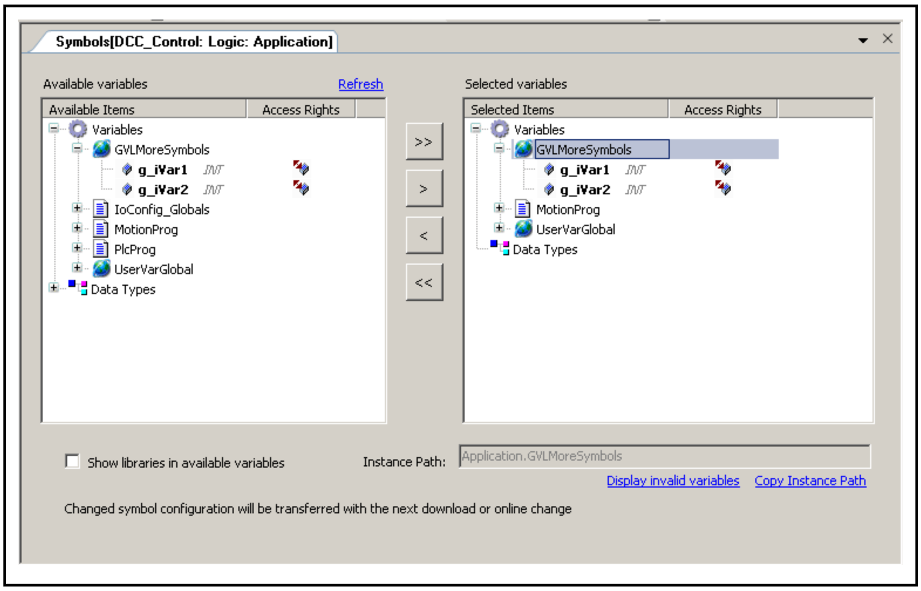
Example

The attribute 'linkalways' is used to implement the global variable list "GVLMoreSymbols":

Global variable list "GVLMoreSymbols"

```
{attribute 'linkalways'}
VAR_GLOBAL
 g_iVar1: INT;
 g_iVar2: INT;
END_VAR
```

This code provides the variables of "GVLMoreSymbols" as selectable symbols.



Symbols[DCC_Control: Logic: Application]

Available variables

Available Items	Access Rights
Variables	
GVLMoreSymbols	
g_iVar1 INT	
g_iVar2 INT	
IoConfig_Globals	
MotionProg	
PlcProg	
UserVarGlobal	
Data Types	

Selected variables

Selected Items	Access Rights
Variables	
GVLMoreSymbols	
g_iVar1 INT	
g_iVar2 INT	
MotionProg	
UserVarGlobal	
Data Types	

☐ Show libraries in available variables

Instance Path: Application.GVLMoreSymbols

Changed symbol configuration will be transferred with the next download or online change

Fig.5-7: Editor of the symbol configuration

Attribute 'monitoring'

In online mode, a property, page 46, can be monitored either using the Inline monitoring, page 386, or a watch list, page 499,.

Monitoring can be enabled by inserting the 'monitoring' attribute pragma into the line above the definition of the property. Subsequently, name, type and value of the variables are displayed in the online view of the function block using the property or in a monitoring list, page 141,. Values to force the property variables can also be prepared there.

There are two different ways to display the current value of the "properties" variables.

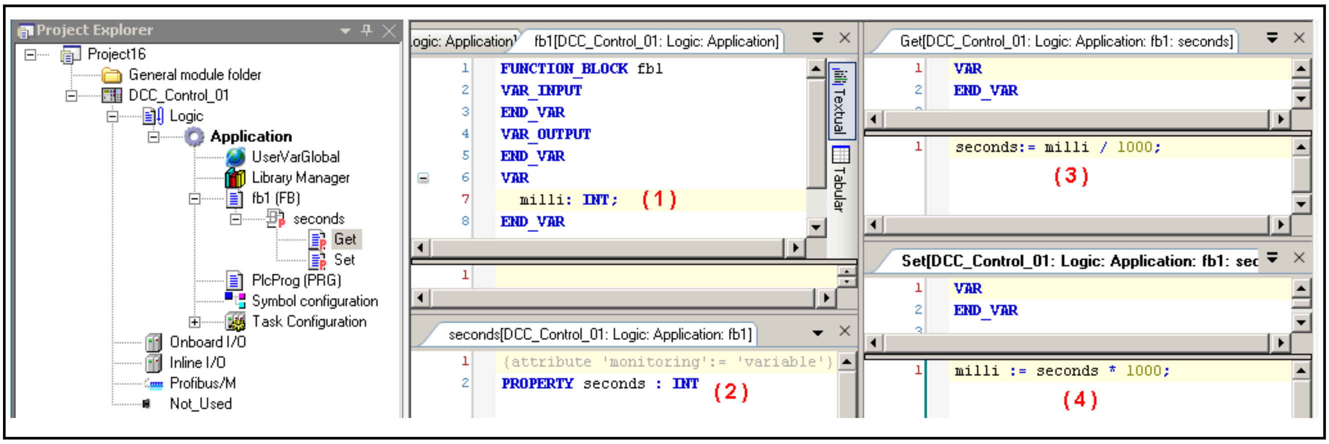
Decide for each case which attribute pragma is suitable to display the desired value. It depends on whether further operations with the variables are implemented in the property.

1. Pragma **{attribute 'monitoring':='variable'}**

   An implicit variable is created for the property that receives the current "properties" value whenever the application calls either the "Set" or "Get" method. The value of these implicit variable is shown in the monitoring.

Syntax:

```
{attribute 'monitoring':='variable'}
```



```
FUNCTION_BLOCK fb1
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
  milli: INT;
END_VAR
```

```
{attribute 'monitoring':= 'variable'}
PROPERTY seconds : INT
```

```
VAR
END_VAR
```

```
seconds:= milli / 1000;
```

```
VAR
END_VAR
```

```
milli := seconds * 1000;
```

(1) Function block "fb1" with the local variable 'milli'
(2) Property 'seconds' with attribute pragma
(3) "Get" of the 'seconds' property
(4) "Set" of the 'seconds' property

Fig.5-8: Example of the 'seconds' property prepared for monitoring

The figure below shows the program with the test variable 'testvar' and the declaration of the function block instance in the upper part.

The implementation includes in line 1: "Set" method and in line 2: "Get" method.

The figure below shows the monitoring including the display of the 'seconds' property.

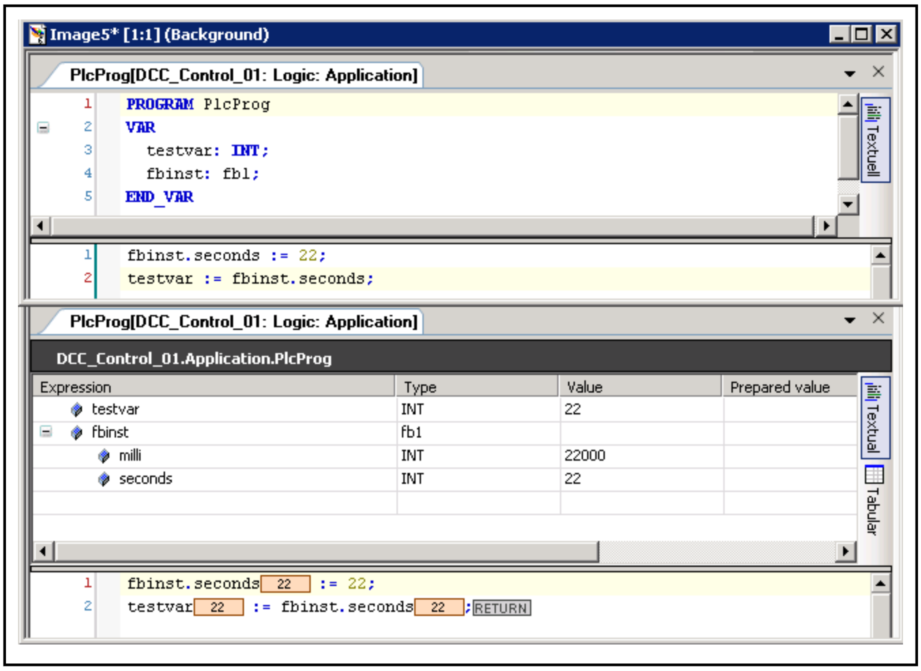


Image5* [1:1] (Background)
PlcProg[DCC_Control_01: Logic: Application]
Textuell
PlcProg[DCC_Control_01: Logic: Application]
DCC_Control_01.Application.PlcProg
Textual
Tabular

```
PROGRAM PlcProg
VAR
  testvar: INT;
  fbinst: fb1;
END_VAR
```

```
fbinst.seconds := 22;
testvar := fbinst.seconds;
```

Expression	Type	Value	Prepared value
testvar	INT	22	
fbinst	fb1		
milli	INT	22000	
seconds	INT	22	

```
fbinst.seconds 22 := 22;
testvar 22 := fbinst.seconds 22 ;RETURN
```

Fig.5-9: Example of a monitoring view with the 'seconds' property

2. Pragma **{attribute 'monitoring':='call'}**

   This attribute can only be used for properties that return only simple data types or pointers, but no structured types:

   The monitored value is obtained by calling the property directly, that means that the monitoring service of the runtime system calls the "Get" method.

   If operations are implemented on the variables in the "property", the value can still change!

Syntax:

```
{attribute 'monitoring':='call'}
```

Attribute 'no_check'

The pragma is added to a POU to impede each call of a check function (POUs for implicit checks, page 63).

Since the check functions can affect the processing velocity of the program, it is reasonable to apply the attribute to function block that have already been checked or are often called.

Syntax:

```
{attribute 'no_check'}
```

Attribute 'no_copy'

In general, an online change, page 80, requires a reallocation of instance, e.g. of a POU.

In this case, the value of the variables in the instance is copied.

However, if the pragma is applied to a variable, a copy of the variable value is not made. Instead, the variable is re-initialized during an online change.

This can be useful for a local pointer variable that points to a variable that was just moved due to an online change (which thus also changed the address).

Syntax:

```
{attribute 'no_copy'}
```

Attribute 'no-exit'

If the function block provides an exit method, page 526,, the call of this method can be suppressed for a special instance of the function block by applying the pragma to the instance.

Syntax:

```
{attribute 'symbol' := 'no-exit'}
```

Example:

The exit method "FB_Exit" is added to the function block called "POU":

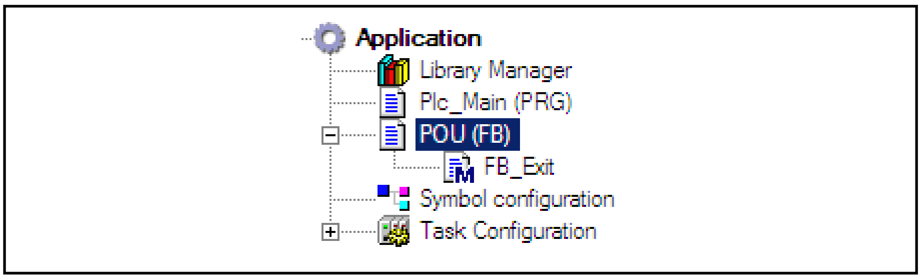


Fig.5-10:

In the main program PLC_PRG, two instances of the function block "POU" are created:

Declaration of the program PLC_PRG:

```
PROGRAM PLC_PRG
VAR
      POU1 : POU;
      {attribute 'symbol' := 'no-exit'}
      POU2 : POU;
END_VAR
```

The method named "FB_exit" is a special method for a function block. It has to be declared explicitly. There is no implicit declaration.

The "exit" method - if available - is called for all declared instances of the function block before a new download or during online changes for all new or deleted instances.

Interface of the FB_exit method: There is only one obligatory parameter:

Interface of the FB_exit method:

```
METHOD fb_exit : BOOL
VAR_INPUT
 bInCopyCode : BOOL; // if TRUE, the exit method is called
                     // to leave the instance, which is copied
                     // afterwards (Online Change).
END_VAR
```

If the variable "bInCopyCode" is assigned to the value TRUE within POU1, the "exit" method FB_Exit is called. In contrast, the value of the variable "bInCopyCode" in POU2 has no effect.

Attribute 'no_init'

Variables that have the pragma are not initialized implicitly.

The pragma refers only to the variables declared in direct succession.

Alternatives in the syntax:

```
{attribute 'no_init'}
{attribute 'no-init'}
{attribute 'noinit'}
```

Example

```
PROGRAM PLC_PRG
VAR
       A : INT;
       {attribute 'no_init'}
       B : INT;
END_VAR
```

When the respective application is reset, the integer variable "A" is initialized with 0 again while the variable "B" keeps its current variable.

Attribute 'no_virtual_actions'

This attribute concerns function blocks derived from a function block implemented in the SFC and that use the SFC procedure of this basic class.

The actions called from there show the same virtual behavior as the methods. That means that the implementations of the actions into the basic class can be replaced by the derived class with individual, specific implementations.

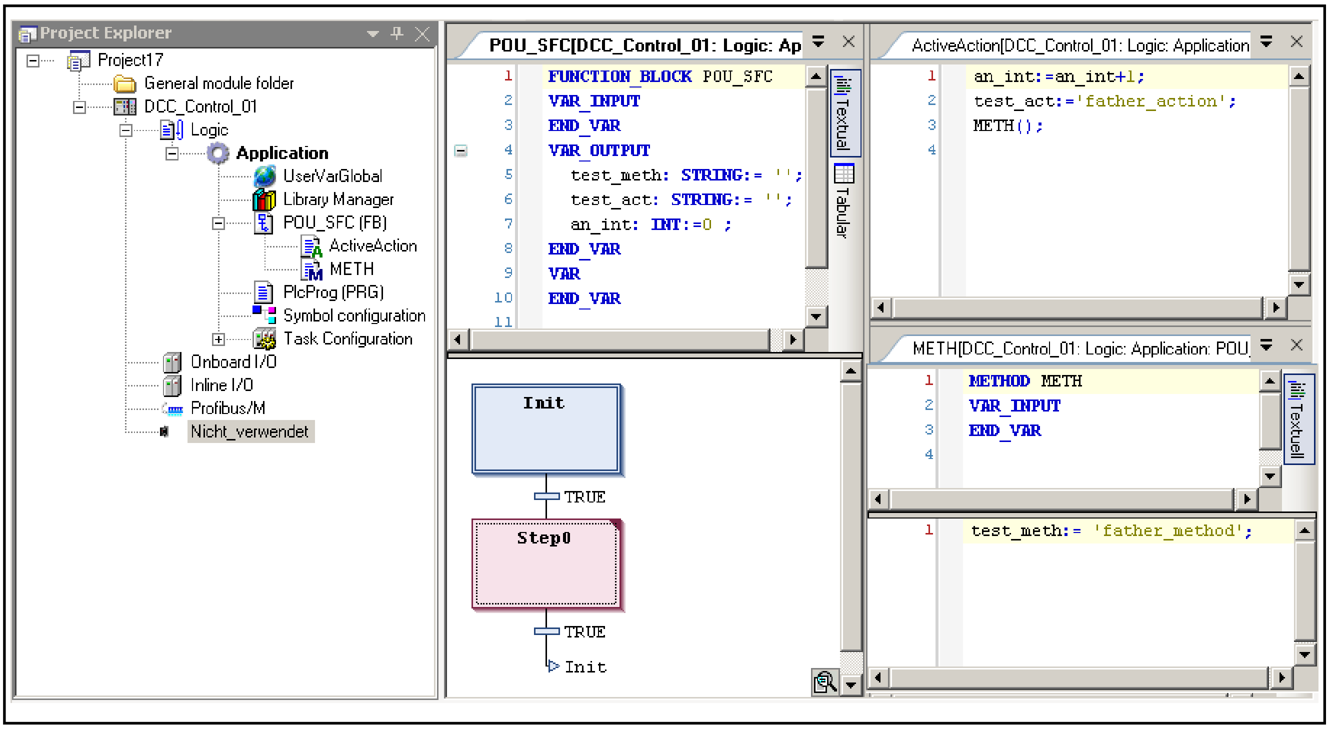
However, if the basic class gets the pragma {attribute 'no_virtual_actions'}, its actions are protected against overload.

Syntax:

```
{attribute 'no_virtual_actions'}
```

Example:

The following example shows the function block POU_SFC that creates the basic class for the derived function block POU_child.



```
FUNCTION_BLOCK POU_SFC
VAR_INPUT
END_VAR
VAR_OUTPUT
  test_meth: STRING:= '';
  test_act: STRING:= '';
  an_int: INT:=0 ;
END_VAR
VAR
END_VAR
```

```
an_int:=an_int+1;
test_act:='father_action';
METH();
```

```
METHOD METH
VAR_INPUT
END_VAR
```

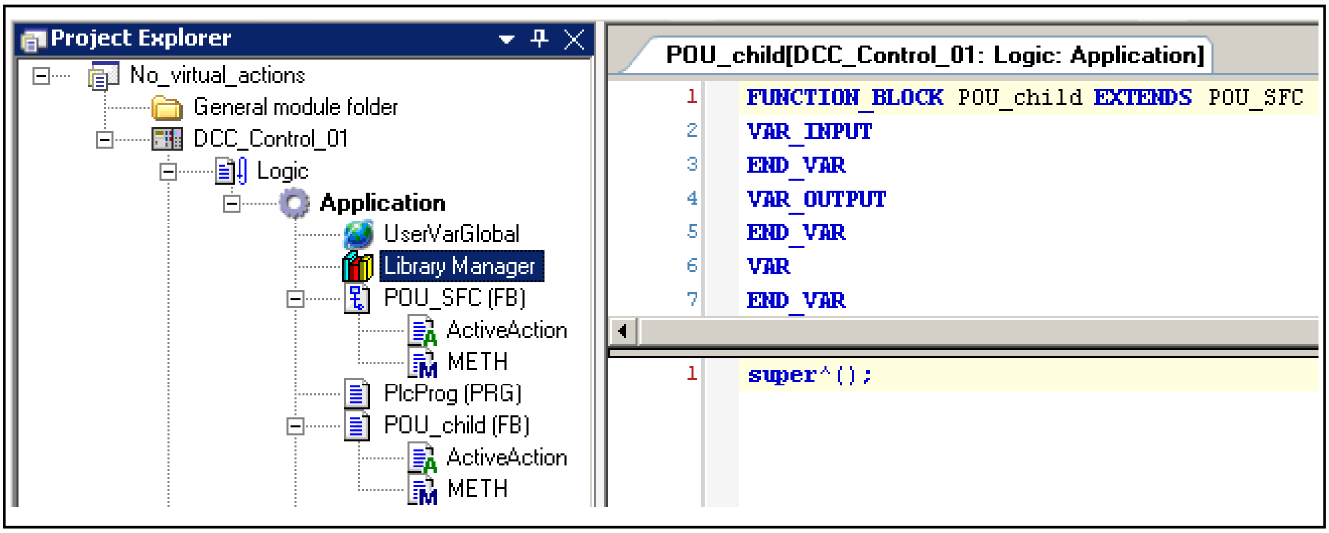
```
test_meth:= 'father_method';
```

Fig.5-11: Function block POU_SFC with the action "ActiveAction" and the method "METH"

The exemplary implementation of this sequence is limited to the initial step followed by only one step with the linked step action "ActiveAction" which assigns the output variables and calls the "METH" method.

The "METH" method assigns the string 'father_method' to the "test_meth" variable in the basic class.

The function block POU_child derives from the function block POU_SFC (basic class).



```
FUNCTION_BLOCK POU_child EXTENDS POU_SFC
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
END_VAR
```

```
super^();
```

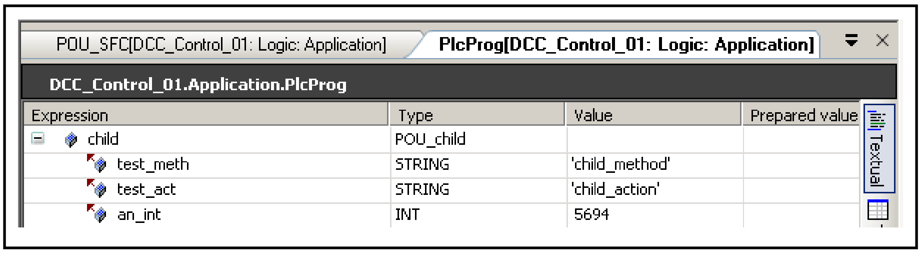
Fig.5-12: Derived function block POU_child

POU_Child uses ST as implementation code and is provided with the "ActiveAction" action and the "METH" method.

For the derived class POU_child, the step action is replaced by a special implementation of "ActiveAction" only differing from the original by assigning the "child_action" string instead of "father_action" to the "test_act" variable.

The METH method, assigning the "father_method" string to the "test_meth" variable in the basic class, is overwritten in such as way that "test_meth" gets the "child_method" value.

The main program "PlcProg" calls an instance of the function block "POU_child" called "Child". As expected, the value of the strings reflect the call of action and method of the derived class:


POU_SFC[DCC_Control_01: Logic: Application]
PlcProg[DCC_Control_01: Logic: Application]
DCC_Control_01.Application.PlcProg
Textual

Expression	Type	Value	Prepared value
⊟ child	POU_child		
test_meth	STRING	'child_method'	
test_act	STRING	'child_action'	
an_int	INT	5694	

Fig.5-13: Main program "PlcProg"

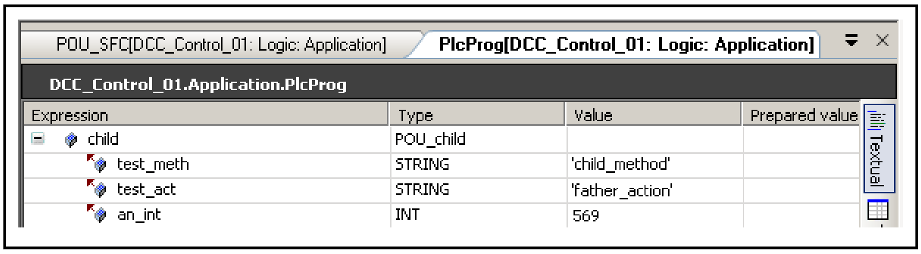
The "no_virtual_actions" attribute precedes the basis

```
{attribute 'no_virtual_actions'}
```

```
FUNCTION_BLOCK POU_SFC...
```

a different behavior can thus be observed: the implementation of the derived class is used for the "METH" method. Calling the step action now results in calling the "ActiveAction" of the basic class.

Thus, "test_act" is now provided with the value "father_action":


POU_SFC[DCC_Control_01: Logic: Application]
PlcProg[DCC_Control_01: Logic: Application]
Textual

DCC_Control_01.Application.PlcProg

Expression	Type	Value	Prepared value
child	POU_child		
test_meth	STRING	'child_method'	
test_act	STRING	'father_action'	
an_int	INT	569	

Fig.5-14: Main program "PlcProg" online, using the attribute "no_virtual_actions" for FUNCTION_BLOCK POU_SFC

Attribute 'obsolete'

An "Obsolete pragma" can be added to a data type definition in order to output a defined warning during compilation if the data type (structure, function block, etc.) is used in the project.

This way, it can be noted that a data type is no longer valid, since an interface has changed for example and that this should be updated in the project.

In contrast to a message pragma, page 527, that is used locally, this warning is defined "centrally" for all instances of a data type.

This pragma always affects the current line or, if it is located in a separate line, the following line.

Syntax:

```
{attribute 'obsolete' := 'user defined text'}
```

The attribute is added to the declaration of the function block "fb1":

Example

```
{attribute 'obsolete' := 'datatype fb1 not valid!'}
FUNCTION_BLOCK fb1
VAR_INPUT
 i:INT;
END_VAR
```

If "fb1" is used as (data) type, e.g. in "fbinst: fb1;", this warning is output when the project is compiled: "datatype fb1 not valid"

Attribute 'pack_mode'

The pragma specifies how a data structure is packed during the allocation.

The attribute has to be inserted above the data structure and affects the zipping of the entire structure.

Syntax:

```
{attribute 'pack_mode' := '<value>'}
```

The placeholder <value>, enclosed in simple apostrophes, has to be replaced by one of the following values:

Value	Associated packing mode
0	Aligned, i.e. there are no memory gaps
1	1-byte aligned (same as aligned)
2	2-byte aligned, i.e. the maximum size of a memory gap is 1 byte
4	4-byte aligned, i.e. the maximum size of a memory gap is 3 bytes
8	8-byte aligned, i.e. the maximum size of a memory gap is 7 bytes

Example

```
{attribute 'pack_mode' := '1'}
TYPE myStruct:
STRUCT
 Enable: BOOL;
 Counter: INT;
 MaxSize: BOOL;
 MaxSizeReached: BOOL;
END_STRUCT
END_TYPE
```

The memory space for a variable of the data type myStruct is assigned "1-byte aligned":

If the memory address of its component is "Enable", e.g. 0x0100, the component "Counter" follows at the address 0x0101, MaxSize at the address 0x0103 and MaxSizeReached at the address 0x0104.

If 'pack_mode'=2, "Counter" is at 0x0102, "MaxSize" is at 0x0104 and "MaxSizeReached" is at 0x0106.

The attribute can also be applied to POUs.

Due to possibly internal pointers, deal carefully with the application of "Attribute Pack_mode".

Attribute 'parameterstringof'

The attribute of the pragma can be used to provide the instance name of a variable to a visualization function block.

Syntax:

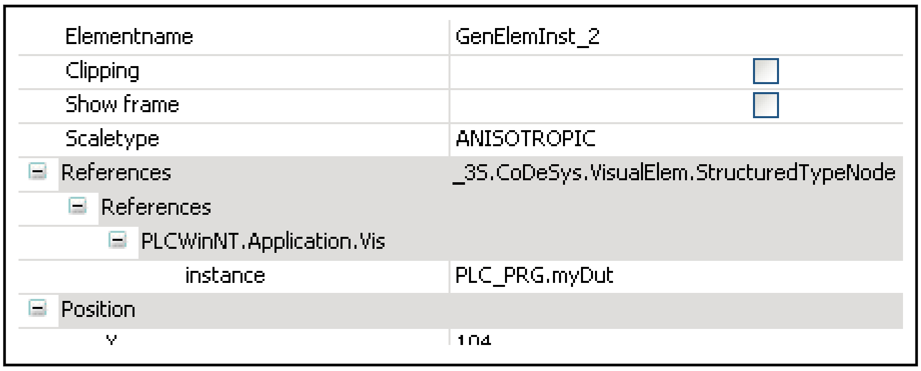
```
{attribute 'parameterstringof' := '<variable>'}
```

In the main program, the instance myDUT of the user-defined structure DUT is created.

Example

```
PROGRAM PLC_PRG
VAR
    myDUT: DUT;
END_VAR
```

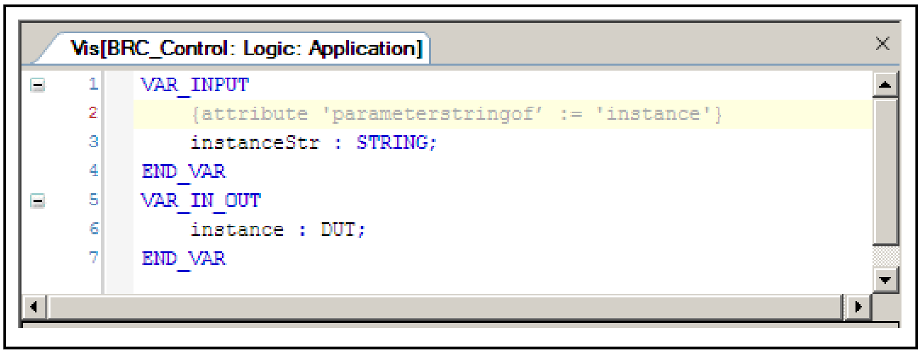
This instance is the input of a visualization block "Vis" (in the input parameter "instance") which is referenced by a frame of another visualization "MainVisu":



Elementname	GenElemInst_2
Clipping	□
Show frame	□
Scaletype	ANISOTROPIC
⊟ References	_3S.CoDeSys.VisualElem.StructuredTypeNode
⊟ References	
⊟ PLCWinNT.Application.Vis	
instance	PLC_PRG.myDut
⊟ Position	
y	104

Fig.5-15: Section from the element properties of a visualization element of MainVisu.

In the interface editor belonging to "Vis", there is the input/output variable "instance" and also another input variable called "instanceStr":



```
VAR_INPUT
    {attribute 'parameterstringof' := 'instance'}
    instanceStr : STRING;
END_VAR
VAR_IN_OUT
    instance : DUT;
END_VAR
```

Fig.5-16: Interface editor

Although "instanceStr" is an input variable, it is not executed as an input in the reference (compare with the figure above!). This is caused since the variable "instanceStr" has the 'parameterstringof' attribute and is therefore automatically initialized with the name of the variables specified with the attribute. In the example, "instance" is the associated variable, so the string variable "instanceStr" is set to "PLC_PRG.myDUT" and can only be used in the visualization "Vis" as text variable for a placeholder %s for example.

Attribute 'qualified_only'

If the pragma precedes a global variable list, the "GVL" variables can only be addressed by specifying the global variable name, e.g.: "gvl.g_var".

This also applies to variables of the type enumerations.

The attribute "Qualified_Only" can be used to prevent confusion with local variables.

Syntax:

```
{attribute 'qualified_only'}
```

The following global variable list "GVL" has the attribute 'qualified_only':

Example

```
{attribute 'qualified_only'}
VAR_GLOBAL
      iVar:INT;
END_VAR
```

Within a POU, e.g. within the program PLC_Main, the global variable "iVar" can only be addressed by using the prefix GVL.

Example:

```
GVL.iVar:=5;
```

However, the incomplete call of the variables generates an error:

```
iVar:=5;
```

Attribute 'reflection'

Signatures are added to the pragma.

For optimization purposes, this attribute has to be specified to function blocks containing the attribute 'instance-path', page 535,.

Syntax:

```
{attribute 'reflection'}
```

Function block "POU" with all necessary attributes:

Example

```
{attribute 'reflection'}
FUNCTION_BLOCK POU
VAR
      {attribute 'instance-path'}
      {attribute 'noinit'}
      str: STRING;
END_VAR
```

In the main program, one instance, "myPOU", of the function block POU is defined:

Application in the program:

```
PROGRAM Plc_Main
VAR
     myPOU:POU;
     myString: STRING;
END_VAR
myPOU();
myString:=myPOU.str;
```

The initialization of the instance myPOU contains the string variable "str" which contains the path of the instance myPOU.

In the example "DCC_Control.Application.Plc_Main.myPOU" this path is assigned to the variable "myString" in the main program.

Attribute 'relative_offset'

The attribute is used in structure definitions to shift structure elements in the memory space allocated by the structure. It is thus for example possible to allocate two structure entries on one memory space.

Syntax:

```
{attribute 'relative_offset' := '<offset>'}
```

Counting the relative offset always starts at the starting address of the structure. The attribute becomes effective for all the following structure entries.

Application in the structure definition:

```
TYPE STRUCT_TEST :
STRUCT
    wTest: WORD;

    {attribute 'relative_offset' := '0'}
    bit_0 : BIT; // Bit 0
    bit_1 : BIT; // Bit 1
    bit_2 : BIT; // Bit 2
    bit_3 : BIT; // Bit 3
    bit_4 : BIT; // Bit 4
    bit_5 : BIT; // Bit 5

END_STRUCT
END_TYPE
```

Attribute 'symbol'

The pragma defines which variables are included in the symbol configuration, page 306, i.e. which variables should be exported to the symbol list as symbols.

These variables are provided for the external access as XML file in the project directory and for users as a hidden file on the target system; e.g. for access by an OPC server.

The variables with the attribute are loaded to the control, even if they are not explicitly configured or visible in the editor of the symbol configuration.

☞ Note that the symbol configuration has to be created as an object below the respective application in the Project Explorer.

☞ Note that only symbols from programs or global variable lists can be accessed. To access a symbol, the symbol name has to be entered completely.

Syntax:

```
{attribute 'symbol' := 'none' | 'read' | 'write' | 'readwrite'}
```

The pragma statement can be added to individual variables or can be assigned collectively to all variables declared in a program.

- To affect just a single variable, the pragma has to be placed in the line in front of the variable declaration.
- To affect all variables in the declaration part of a program, the pragma has to be placed in the first line of the declaration editor. However, even in this case, statements for individual variables can be positioned explicitly in the respective lines.

The parameter determines the possible **access** to a symbol. Possible specifications: 'none', 'read', 'write' or 'readwrite'. If no parameter is specified, the default value 'readwrite' applies.

Example:

The variables "A" and "B" are exported with read-only and write access using the following configuration. Variable "D" is exported with read-only access.

Examples:

```
{attribute 'symbol' := 'readwrite'}
PROGRAM PLC_PRG
VAR
      A : INT;
      B : INT;
      {attribute 'symbol' := 'none'}
      C : INT;
      {attribute 'symbol' := 'read'}
      D : INT;
END_VAR
```

Conditional Pragmas

The extension of the programming language "ST", ExST (Extended ST), page 389, supports a variety of conditional pragma statements, page 526 that affect the code generation in the precompiling or compiling process (compilation of the project).

The compilation of a implementation code becomes conditional to:

- a specific data type or variable declared
- a data type of a variable with a specific attribute
- a variable with a specific data type
- a specific function block or task existing or part of the call tree
- etc.

💡 If is not possible for a POU or a GVL created in the POU window to use a "{define...}" declared in an application.

"defines" in applications become only effective for interfaces located below the application.

{define **identifier string**}	During the precompilation, all subsequent instances of the identifier **Identifier** are replaced by the specified character string **string** if it is not empty (which is permitted). The identifier remains defined and valid until the end of its application area or until it is reset by an {undefine} statement. Required for a conditional compilation; see Conditional compilation operators, page 547.
{undefine identifier} .	The preprocessor definition of the identifier **identifier** (by {define}, see above) is reset. The identifier is now undefined again. If the currently specified identifier is not defined at all, the pragma is ignored.

{IF expr}
...
{ELSIF expr}
...
{ELSE}
...
{END_IF}
	These Pragmas are for the **conditional** compilation. The specified **expr** expressions have to be constant at the time of compilation. They are evaluated in the sequence in which they appear until one of the expressions displays a value not equal to "zero". The text linked with the statement is prepared and then compiled. The other lines are ignored. The sequence of the sections is specified. However, the **elsif** and **else** sections are optional and **elsif** sections can occur unlimitedly.
	Within the **expr** constant, several conditional compilation operators, page 547, can be used, see below.

Fig.5-17: Possible effects of the compilation

Conditional compilation operators

One or more operators can be used in the constant expression **expr** of a conditional compilation pragma **{IF}** or **{ELSIF}**, see above. However, they may not be defined {define} or undefined {undefine}.

Note that like the definition, these expressions can also be entered as "compiler definitions" using a {define} in compilation properties, page 238, of an object.

The following operators are supported:

Operator	Effect and example
defined (**identifier**)	This operator causes that the expression gets the value TRUE if the identifier "identifier" was defined using a {define} statement and was not undefined afterwards with an {undefine} statement. Otherwise, FALSE is returned. defined (identifier), page 548
defined (variable: **variable**)	This operator causes that the expression gets the value TRUE if the variable "variable" is declared in the current validity range. Otherwise, FALSE is returned. defined (variable: variable), page 548
defined (type: **identifier**)	This operator causes that the expression gets the value TRUE if a data type with an identifier "identifier" is declared. Otherwise, FALSE is returned. defined (type: identifier), page 549
defined (pou: **pou-name**)	This operator causes that the expression gets the value TRUE if a function block or an action with the name "pou-name" is present. Otherwise, FALSE is returned. defined (pou: pou-name), page 549
not yet implemented: defined (task: **identifier**)	This operator causes that the expression gets the value TRUE if a task with the name "identifier" is defined. Otherwise, FALSE is returned. not yet implemented: defined (task: identifier), page 549
not yet implemented: defined (resource: **identifier**)	This operator causes that the expression gets the value TRUE if a resource object with the name "identifier" is present for the application. Otherwise, FALSE is returned. not yet implemented: defined (resource: identifier), page 549

Operator	Effect and example
hasattribute (pou: pou-name, attribute)	This operator causes that the expression gets the value TRUE if the attribute "attribute" is specified in the first line of the declaration part of the function block "pou-name". Otherwise, FALSE is returned. hasattribute (pou: pou-name, 'attribute'), page 550
hasattribute (variable: **variable**, attribute)	This operator causes that the expression gets the value TRUE if the attribute "attribute" is assigned to the variable "variable" using the {attribute} statement in the line in front of the variable declaration. Otherwise, FALSE is returned. hasattribute (variable: variable, 'attribute'), page 550
hastype (variable: **variable**, **type-spec**)	This operator causes that the expression gets the value TRUE if the variable "variable" is a data type. Otherwise, FALSE is returned. hastype (variable: variable, type-spec)
hasvalue (**define-ident**, **char-string**)	This operator causes that the expression gets the value TRUE if a variable is defined with an identifier "define-ident" and has the value "char-string". Otherwise, FALSE is returned. hasvalue (define-ident, char-string)
NOT **operator**	The expression receives the value TRUE if the inverse value of the operator "operator" returns TRUE. "operator" can be one of the operators described in this table. NOT operator
AND **operator**	The expression returns the value TRUE if the specified "operator" operators both return TRUE. "Operator" can be one of the operators described in this table. AND operator
OR **operator**	The expression returns TRUE if both of the specified operators "operator" return TRUE. "Operator" can be one of the operators described in this table. OR operator
(operator)	Operator parentheses

defined (identifier)

This operator causes that the expression gets the value TRUE if the identifier "identifier" was defined using a {define} statement and was not undefined afterwards with an {undefine} statement. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". The variable "hallo" is defined by a {define} statement in "App1", but not in "App2".

Example

```
{IF defined (pdef1)}
      // this code is processed in App1
{info 'pdef1 defined'}
  hugo := hugo + SINT#1;
{ELSE}
      // the following code is only processed in App2
{info 'pdef1 not defined'}
  hugo := hugo - SINT#1;
```

There is also an example of a message pragma, page 527, included:

Only the information "pdef1 defined" is displayed in the message window if the application is compiled, since "pdef1" is really defined. The message "pdef1 not defined" is output if "pdef1" is not defined.

defined (variable: variable)

This operator causes that the expression gets the value TRUE if the variable **variable** is declared in the current validity range. Otherwise, FALSE is returned.

Prerequisite: There are two applications, App1 and App2. Variable "g_bTest" is declared in "App1", but not in "App2.Precondition"

Example

```
{IF defined (variable:g_bTest)}
  // the following code is processed in App2 only
g_bTest := x > 300;
{END_IF}
```

defined (type: identifier)

This operator causes that the expression gets the value TRUE if a data type with an identifier **identifier** is declared. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". The data type "DUT" is declared in "App1", but not in "App2".

Example

```
{IF defined (type:DUT)}
   // the following code is processed in App1 only
  bDutDefined := TRUE;
{END_IF}
```

defined (pou: pou-name)

This operator causes that the expression gets the value TRUE if a function block or an action with the name "pou-name" is present. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". The block "CheckBounds" is present in "App1", but not in "App2".

Example

```
{IF defined (pou:CheckBounds)}
    // the following code is processed in App1 only
  arrTest[CheckBounds(0,i,10)] := arrTest[CheckBounds(0,i,10)]+1;
{ELSE}
    // the following code is processed in App2 only
 arrTest[i] := arrTest[i]+1;
{END_IF}
```

not yet implemented: defined (task: identifier)

This operator causes that the expression gets the value TRUE if a task with the name "identifier" is defined. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". The task "PLC_PRG_Task" is defined in "App1", but not in "App2".

Example

```
{IF defined (task:PLC_PRG_Task)}
  // the following code is processed in App1 only
 erg := plc_prg.x;
{ELSE}
  // the following code is processed in App2 only
erg := prog.x;
{END_IF}
```

not yet implemented: defined (resource: identifier)

This operator causes that the expression gets the value TRUE if a resource object with the name "identifier" is present for the application. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". a resource object "glob_var1" (Global Variable List) is present for "App1", but not for "App2".

Example

```
{IF defined (resource:glob_var1)}
  // the following code is processed in App1 only
 gvar_x := gvar_x + ivar;
{ELSE}
  // the following code is processed in App2 only
 x := x + ivar;
{END_IF}
```

hasattribute (pou: pou-name, 'attribute')

This operator causes that the expression gets the value TRUE if the attribute "attribute" is specified in the first line of the declaration of the function block pou-name. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". A function "fun1" is defined in "App1" and "App2", but in "App1" the attribute 'vision' is also assigned to it:

Definition of fun1 in App1:

```
{attribute 'vision'}
FUNCTION fun1 : INT
VAR_INPUT
 i : INT;
END_VAR
VAR
END_VAR
```

Definition of fun1 in App2:

```
FUNCTION fun1 : INT
VAR_INPUT
 i : INT;
END_VAR
VAR
END_VAR
```

Pragma statement:

```
{IF hasattribute (pou: fun1, 'vision')}
   // the following code is processed in App1 only
  ergvar := fun1(ivar);
{END_IF}
```

hasattribute (variable: variable, 'attribute')

This operator causes that the expression gets the value TRUE if the attribute **attribute** is assigned to the variable **variable** using the {attribute} statement in the line in front of the variable declaration. Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". Variable "g_globalInt" is used in "App1" and "App2", but in "App1" the attribute 'docount' is also assigned to it.

Declaration of g_globalInt in App1:

```
VAR_GLOBAL
{attribute 'docount'}
 g_globalInt : INT;
 g_multiType : STRING;
END_VAR
```

Declaration of g_globalInt in App2:

```
VAR_GLOBAL
 g_globalInt : INT;
 g_multiType : STRING;
END_VAR
```

Pragma statement:

```
{IF hasattribute (variable: g_globalInt, 'docount')}
(* the following code line is executed in App1 only,
   because there the variable g_globalInt is defined
   with the attribute 'docount'*)
 g_globalInt := g_globalInt + 1;
{END_IF}
```

hastype (variable: variable, type-spec)

This operator causes that the expression gets the value TRUE if the variable **variable** is of the data type **type-spec.** Otherwise, FALSE is returned.

Available data types of "type-spec"

ANY	LINT	WSTRING
ANY_DERIVED	DINT	STRING
ANY_ELEMENTARY	INT	
ANY_MAGNITUDE	SINT	TIME
ANY_BIT	ULINT	DATE_AND_TIME
ANY_STRING	UDINT	DATE
ANY_DATE	UINT	TIME_OF_DAY
ANY_NUM	USINT	
ANY_REAL	LWORD	
ANY_INT	DWORD	
	WORD	
LREAL	BYTE	
REAL	BOOL	

Prerequisite: There are two applications, "App1" and "App2". The variable "g_multitype" is declared in "App1" with data type LREAL, but in "App2" with the data type STRING:

Example

```
{IF (hastype (variable: g_multitype, LREAL))}
   // the following code line is executed in App1 only
  g_multitype := (0.9 + g_multitype) * 1.1;
{ELSIF (hastype (variable: g_multitype, STRING))}
   // the following code line is executed in App2 only
  g_multitype := 'this is a multitalent';
{END_IF}
```

hasvalue (define-ident, char-string)

This operator causes that the expression gets the value TRUE if a variable is defined with an identifier **define-ident** and has the value **char-string.** Otherwise, FALSE is returned.

Prerequisite: There are two applications, "App1" and "App2". The variable "test" is used in the applications "App1" and "App2"; in "App1" it has the value "1", in "App2" the value "2":

Example

```
{IF hasvalue(test,'1')}
  (* the following code is executed in App1 because
     there the value of the test variable is 1 *)
  x := x + 1;
{ELSIF hasvalue(test,'2')}
  (* the following code is executed in App1 because
     there the variable test is 2 *)
  x := x + 2;
{END_IF}
```

NOT operator

The expression gets the value TRUE if the reciprocal of the **operator** operator returns TRUE. The **operator** can be one of the operators described in this table.

Prerequisite: There are two applications, "App1" and "App2". The POU "PLC_PRG1" is present in "App1" and "App2", but the POU "CheckBounds" is only in "App1":

Example

```
{IF defined(pou: PLC_PRG1) AND NOT (defined(pou: CheckBounds))}
   // the following code line is executed in App2 only
  bANDNotTest := TRUE;
{END_IF}
```

AND operator

The expression returns the value TRUE if the specified **operator** operators both return TRUE. **Operator** can be one of the operators described in this table.

Prerequisite: There are two applications, "App1" and "App2". The POU "PLC_PRG1" is present in "App1" and "App2", but the POU "CheckBounds" is only in "App1":

Example

```
{IF defined(pou: PLC_PRG1) AND (defined(pou: CheckBounds))}
  (* the following code line is executed in App1 only,
     because there only PLC_PRG1 and CheckBounds are defined.*)
  bORTest := TRUE;
{END_IF}
```

OR operator

The expression returns TRUE if both of the specified operators **operator** return TRUE. **Operator** can be one of the operators described in this table.

Prerequisite: There are two applications, "App1" and "App2". The POU "PLC_PRG1" is present in "App1" and "App2", but the POU "CheckBounds" is only in "App1":

Example

```
{IF defined(pou: PLC_PRG1) OR (defined(pou: CheckBounds))}
  (* the following code line is execute in App1 and App 2,
     because at least one of the POEs contains PLC_PRG1
     and CheckBounds *)
 bORTest := TRUE;
{END_IF}
```

(operator)

Operator parentheses

5.2 Data Types

5.2.1 Data Types, General Information

A data type is assigned to each identifier. A data type specifies the memory reserved and the values corresponding to the memory contents.

When programming in IndraLogic, memory can be reserved for instances of

- standard data types, page 553,
- user-defined data types, page 560, and
- of function blocks

The following basic data types are supported:

	1 bit	8 bits	16 bits	32 bits	64 bits
Boolean variable	BIT*	BOOL			
Bit sequence		BYTE	WORD	DWORD	LWORD
Signed integers		SINT	INT	DINT	LINT
Unsigned integers		USINT	UINT	UDINT	ULINT
Floating point number				REAL	LREAL

	1 bit	8 bits	16 bits	32 bits	64 bits
Time				TIME TOD DATE DT	LTIME*
Character string		STRING**	WSTRING**		
Reference				REFERENCE TO*	
Pointer				POINTER TO*	

* Extension of the EN 61131-3 standard

** Memory requirement of a character of the character string

Fig.5-18: Basic Data Types

5.2.2 Basic Data Types

Basic Data Types, General Information

IndraLogic supports all data types described in the IEC 61131-3 standard.

- BIT, page 553,
- BOOL, page 553
- Bit sequence, page 554
- Integer data types, page 554
- Floating point numbers, page 554,
- Text variables, page 555,
- Time data types, page 555
- References, page 556,
- Pointer, page 557,

☞ Data types can also be defined themselves; see user-defined data types.

BIT

Variables of the data type BIT can assume the truth values TRUE (1) and FALSE (0).

☞ One bit is reserved as memory space in the structures.

If the data type is used outside the structures, one byte is reserved as memory space.

The data type BIT is not part of the standard IEC 61131-3.

Also refer to

- BOOL constants, page 616 (operands).

BOOL

Variables of the data type BOOL can accept the truth values TRUE (1) and FALSE (0).

☞ An eight bit memory space is reserved.

Also refer to

- BOOL constants, page 616 (operands).

Bit Sequence

Each data type has a different data width (number of bits).

List of available bit sequence data types (with range limits):

Data type	Lower limit	Upper limit	Memory space
BYTE	16#00	16#FF	8 bits
WORD	16#0000	16#FFFF	16 bits
DWORD	16#0000 0000	16#FFFF FFFF	32 bits
LWORD	16#0000 0000 0000 0000	16#FFFF FFFF FFFF FFFF	64 bits

Also refer to

- Number constants, page 618 (operands)
- Typed constants, page 620 (operands)

Integer Data Types

Each integer data type covers a specific number range.

List of available integer data types (with range limits):

Data type	Lower limit	Upper limit	Memory space
SINT	-128	127	8 bits
USINT	0	255	8 bits
INT	-32768	32767	16 bits
UINT	0	65535	16 bits
DINT	-2147483648	2147483647	32 bits
UDINT	0	4294967295	32 bits
LINT	-2^{63}	$2^{63}-1$	64 bits
ULINT	0	$2^{64}-1$	64 bits

☞ When data types are converted from large to small, information can be lost.

Also refer to

- Number constants, page 618 (operands)
- Typed constants, page 620 (operands)

Floating Point Numbers

The data types REAL and LREAL are so-called floating point data types. The data types REAL and LREAL are used with rational numbers. The reserved memory space is 32 bits for REAL and 64 bits for LREAL.

Values allowed for REAL:

±(1.175494351e-38 to 3.402823466e+38)

Values allowed for LREAL:

±(2.2250738585072014e-308 to 1.7976931348623158e+308)

☞ If a REAL or LREAL is converted into SINT, USINT, INT, UINT, DINT, UDINT, LINT or ULINT and the value of the REAL/LREAL number is outside the value range of the integer, the result is undefined and depends on the target system. An exception is then possible!

To get a target system-independent code, value range exceedances have to be intercepted via the application.

If the REAL/LREAL number is within the range, the conversion runs equally on all systems.

Also refer to

- REAL-/LREAL constants, page 619 (operands)

Text Variables

A variable of the data type STRING can accept any character string. The size specification for the memory reservation for the declaration refers to characters (1 byte) and can be made in parentheses or square brackets. If no size is specified, 80 characters are defined as default.

In principle, the string length is not limited, but the string functions can only process lengths between 1 - 255!

When a variable is initialized with a string that is too long for the variable data type, the string is cut at the end accordingly.

String declaration with 35 characters:

Example:

```
str:STRING(35):= 'This is a String';
```

A variable of the data type WSTRING can accept any character string. The size specification for the memory reservation for the declaration refers to characters (2 byte) and can be made in parentheses or square brackets.

Compared to the STRING (ASCII), the WSTRING is interpreted in Unicode format.

Example:

```
wstr: WSTRING:= "This is a WString";
```

When a variable is initialized with a WString that is too long for the variable data type, the WString is cut at the end accordingly.

Also refer to

- STRING constants, page 619 (operands)
- WSTRING constants, page 619 (operands).

Time Data Types

The data types "TIME", "TIME_OF_DAY" (abbreviated "TOD"), "DATE" and "DATE_AND_TIME" (abbreviated "DT") have a size of 32 bits.

TIME has a size of 32 bits and a resolution in milliseconds.

For TOD, the time is given in milliseconds and calculations begin for TOD at 00:00 o'clock.

For DATE and DT, the time is given in seconds and the calculations begin on January 1st 1970 at 00:00.

LTIME is used as time basis for high-resolution timers. LTIME has a size of 64 bits and a resolution in the nanosecond range.

Syntax:

```
LTIME#<time_declaration>
```

The time declaration can contain time units that apply for TIME constants and also:

"us" : microseconds

"ns" : nanoseconds

Example:

```
LTIME1: LTIME := LTIME#1000d15h23m12s34ms2us44ns;
```

The data types DATE, DATE_AND_TIME, TIME_OF_DAY are part of the standard IEC 61131-3. The data type LTIME is not part of the standard IEC 61131-3.

Also refer to

- TIME constants, page 616
- DATE constants, page 617
- DATE_AND_TIME constants, page 617
- TIME_OF_DAY constants, page 617

References (REFERENCE)

This string data type is an extension with regard to the IEC 61131-3 standard.

A "REFERENCE" is an "alias" for an object. It can be written or read using identifiers.

Compared to a pointer, page 557,, the value indicated is directly affected and the assignment of reference and value is fix.

The address of the reference has to be set using a separate assignment operation.

A reference is declared according to the following syntax:

Syntax:

```
<Name> : REFERENCE TO <data_type>
```

Example declaration:

```
ref_int : REFERENCE TO INT;
a : INT;
b : INT;
```

ref_int can now be used as an "Alias" for INT type variables.

Use case:

```
ref_int REF= a;     // ref_int now points to a
ref_int := 12;      // a now has the value 12
b := ref_int * 2;   // b now has the value 24
ref_int REF= b;     // ref_int now points to b
ref_int := a / 2;   // b now has the value 6
ref_int REF= 0;     // explicit initialization of the reference
```

☞ The following reference declarations cannot be made:

REFERENCE TO REFERENCE

or

ARRAY OF REFERENCE

or

POINTER TO REFERENCE.

☞ References are initialized (with 0) with the compiler version ≥ V3.3.0.0.

Check for valid references

The operator "__ISVALIDREF" can be used to check if a reference refers to a valid value, i.e. a value not equal to 0.

Syntax:

```
<boolean variable>:= __ISVALIDREF (with REFERENCE TO <data_type>);
```

<Boolean Variable> becomes TRUE if the reference points to a valid value. Otherwise, it becomes FALSE.

Example:

Declaration:

```
ivar : INT;
ref_int : REFERENCE TO INT;
ref_int0: REFERENCE TO INT;
testref: BOOL := FALSE;
testref0: BOOL  := FALSE;
```

Implementation:

```
ivar := ivar +1;
ref_int REF= ivar;
ref_int0 REF= 0;
testref := __ISVALIDREF(ref_int);
 // TRUE, because ref_int points to ivar, with value <> 0
testref0 := __ISVALIDREF(ref_int0);
 // FALSE, because ref_int0 is set to 0
```

Pointer (POINTER)

In an extension of the IEC 61131-3 standard, the usage of pointers is supported:

Pointers save the addresses of variables, programs, function blocks, methods and functions while an application program is running.

A pointer can point to each of the named objects and to every data type and even to user-defined data types.

💡 Implicit monitoring functions for pointers, page 559, can be used!

Syntax of a pointer declaration:

```
<Name>: POINTER TO <data_type | Functionblock | Program |
                            Method | Function>;
```

Dereferencing a pointer means obtaining the value that is currently at the address to which the pointer is pointing. A pointer can be dereferenced by adding the content operator "^" to the pointer indicator; see "pt^" in the example below.

The address operator "ADR" can be used to assign the address of a variable to a pointer.

Example:

```
VAR pt:POINTER TO INT; // Declaration of pointer pt
  var_int1:INT := 5;   // Declaration of variables var_int1 and var_int2
  var_int2:INT; END_VAR

pt := ADR(var_int1);   // Address of varint1 is assigned pointer pt
var_int2:= pt^;
   (* Value 5 of var_int1 is assigned to variable
       var_int2 by dereferencing pointer pt *)
```

Function pointer

In contrast to IndraLogic 1.x, function pointers that replace the INDEXOF operator are now supported as well.

These pointers can be forwarded to external libraries, but **a function pointer cannot be called in an application in the programming system.**

The function of the runtime system to register callback functions (system library function) expects the function pointer. Depending on the callback registered, the respective function is then implicitly called by the runtime system (for example at stop).

To enable a system call (runtime system), the corresponding application properties, page 240, has to be set for the function object.

The ADR operator, page 587, can be used for functions, programs, function blocks and methods.

Since functions can change after an online change, the address of a pointer that points **to the function is output instead of the address of the function**. This address is valid as long as the function exists on the target system.

Index access to pointers

An extension of the IEC 61131-3 standard allows the index access "[ ]" to variables of the type POINTER, text variables.

- `pint[i]` returns the basic data type.
- Index access to pointers is carried out arithmetically:

  If the index access is used with a variable of type pointer, the offset is calculated using

  `pint[i] = (pint + i * SIZEOF(base type))^`

  The index access also causes an implicit dereferencing of the pointer. The resulting data type is the basic data type of the pointer.

  Note: `pint[7] ≠ (pint + 7)^`!
- If the index access is used with a STRING type variable, the sign at offset index-expr. is obtained.

  The result is of type BYTE.

  str[i] returns the ith character of the string as SINT (ASCII).
- If the index access is used with a WSTRING type variable, the sign at offset index-expr. is obtained.

  The result is of type WORD.

  wstr[i] returns the ith character of the string as INT (Unicode).

☞ references, which, in contrast to pointers directly affect a value, can also be used.

Monitoring function for pointers

The implicit monitoring function "CheckPointer" can be used to monitor the memory access by pointers at runtime.

It has to be integrated into the application as an object POUs for implicit checks, page 63, using the command Add, page 234,.

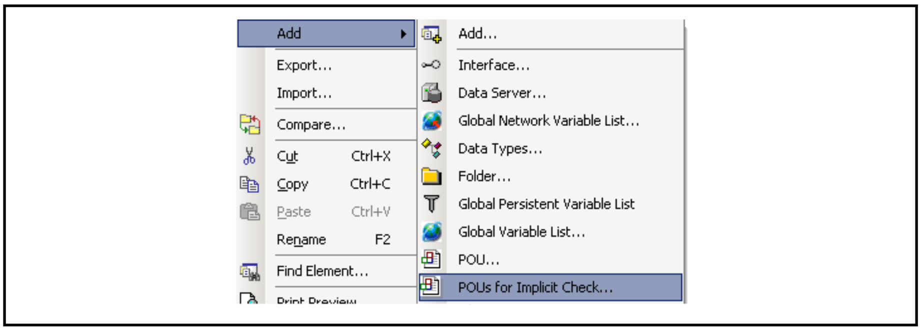
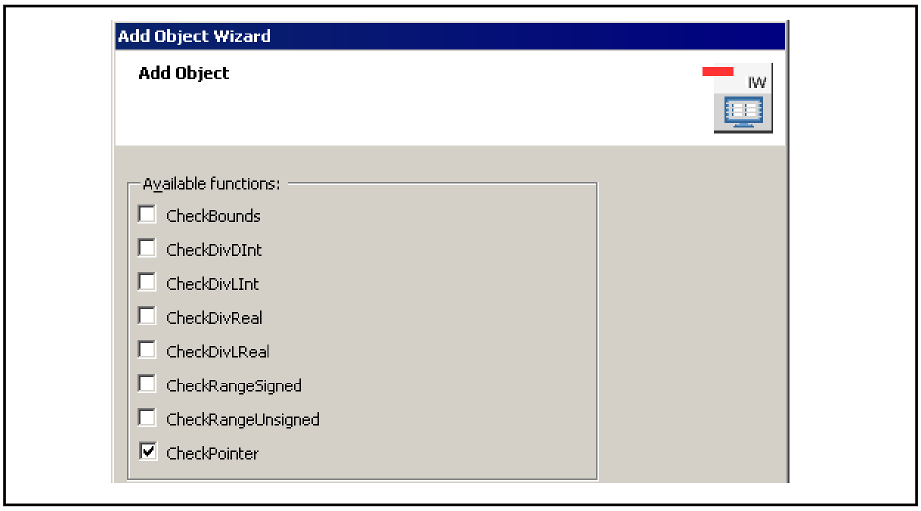


Fig.5-19: Add -> POUs for Implicit Check

After selecting the respective checkbox, select a program language and confirm it with "Open". The CheckPointer function opens in the respective editor.


Add Object Wizard

Add Object

Available functions:

- □ CheckBounds
- □ CheckDivDInt
- □ CheckDivLInt
- □ CheckDivReal
- □ CheckDivLReal
- □ CheckRangeSigned
- □ CheckRangeUnsigned
- ☑ CheckPointer

Fig.5-20: Selecting CheckPointer

☞ The declaration of the functions is specified for all languages and may not be modified! Only local variables may be added. In contrast to other monitoring functions, there is no suggestion for the implementation of the CheckPointer. Users have to carry out the implementation!

The CheckPointer function checks whether the transferred pointer points to a valid memory address and if the position of referenced memory area corresponds to the type of variables to which the pointer points. If both conditions are met, the pointer itself is returned. Otherwise, CheckPointer should perform an appropriate error handling.

☞ No implicit call of the monitoring function is carried out for the THIS pointer.

Template:

```
(* Automatically generated code: DO NOT EDIT *)
FUNCTION CheckPointer : POINTER TO BYTE
VAR_INPUT
    ptToTest : POINTER TO BYTE;
    iSize : DINT;
    iGran : DINT;
    bWrite: BOOL;
END_VAR

(* No standard implementation. Please enter your code here. *)
CheckPointer := ptToTest;
```

At the call, the functions of the following input parameters are transferred.

- ptToTest: Target address of the pointer
- iSize: Size of the referenced variable; the data type of iSize gas to be compatible with INT and cover the range of the variables.
- iGran: Granularity of the referenced size, i.e. the largest unstructured data type in the referenced variables. The data type of iGran has to be compatible with INT.
- bWrite: Type of access (TRUE=write access FALSE=read access. The data type of bWrite has to be BOOL

If the check is positive, the unmodified input pointer is returned (ptToTest).

5.2.3 User-defined Data Types

User-Defined Data Types, General Information

Except for the standard data types, users can define their own data types in a project.

These definitions can be made by creating DUT (Data Unit Type) objects, page 43, in the Project Explorer or in the declaration of a function block.

Note to be as consistent as possible at the recommendations for names, page 505, for objects.

The following user-defined data types can be created:

- ARRAY, page 560
- STRUCT, page 563,
- UNION, page 564,
- Enumeration, page 564
- Subrange type, page 566

Arrays (ARRAY)

One-, two- and three-dimensional arrays of elementary data types are supported. Arrays can be defined in the declaration of a function block and in the global variable lists. Note that there are functions to check array limits, page 560,.

Syntax:

```
<Array_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>,<ll3>..<ul3>] OF <elem.Type>
```

ll1, ll2, ll3 identify the lower limit of an array dimension,

ul1, ul2 and ul3 identify the upper limit. These limit values have to be integers.

Example:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Initialization of arrays

☞ In contrast to IndraLogic 1.x, "square brackets" have to be placed around the initialization value(s)!

Complete initialization of an array:

Example:

```
arr1 : ARRAY [1..5] OF INT := [1,2,3,4,5];
(* short for 1,7,7,7*)
arr2 : ARRAY [1..2,3..4] OF INT := [1,3(7)];
(* short for 0,0,4,4,4,4,2,3*)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := [2(0),4(4),2,3];
```

Initialization of an array of a structure:

Example:

```
TYPE STRUCT1:      // Structure definition
    STRUCT
        p1:int;
        p2:int;
        p3:dword;
    END_STRUCT
END_TYPE
   // Array initialization in the application declaration
ARRAY[1..3] OF STRUCT1:= [(p1:=1, p2:=10,p3:=4723),
                                            (p1:=2, p2:=0, p3:=299),
                          (p1:=14,p2:=5, p3:=112)];
```

Partial initialization of an array:

Example:

```
arr4 : ARRAY [1..10] OF INT := [1,2];
```

Elements without any assigned value as initialization value are initialized with the default value of the basic data type. In the example above, the elements arr4[3] to arr4[10] are initialized with 0.

Access to array components

In a two-dimensional array, the components are accessed as follows:

`<Array-Name>[Index1,Index2]>`

Example:

```
Card_game [9,2]
```

Functions to check the array limits:

To guarantee a correct access to the array elements, the "CheckBounds" function has to be available to the application.

It is integrated into the application as an object POUs for implicit checks, page 234, using the command Add, page ,63,.

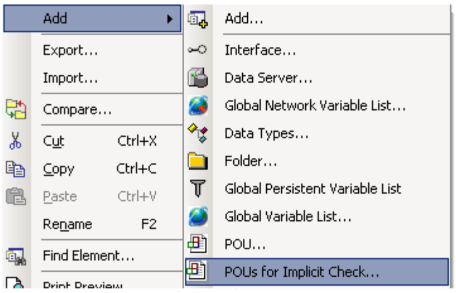
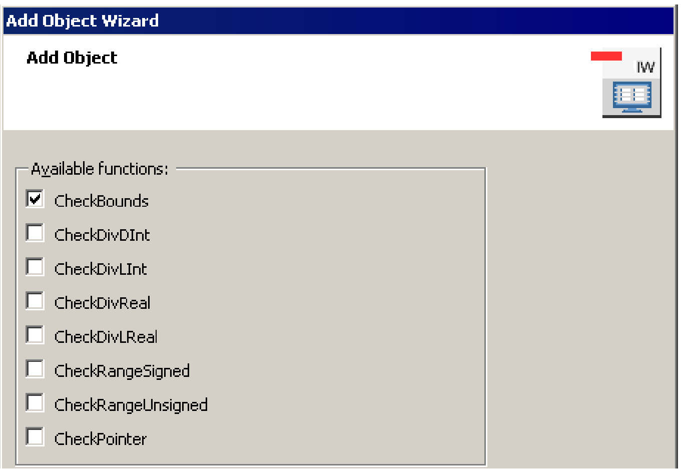


Fig.5-21: Add -> POUs for Implicit Check

After selecting "CheckBounds", select a program language and confirm it with "Open". The "CheckBounds" function opens in the respective editor.



Available functions:

☑ CheckBounds
□ CheckDivDInt
□ CheckDivLInt
□ CheckDivReal
□ CheckDivLReal
□ CheckRangeSigned
□ CheckRangeUnsigned
□ CheckPointer

Fig.5-22: Selecting CheckBounds

The declaration part of the functions is specified for all languages and may not be modified! Only local variables may be added. A suggestion to implement the function is available in the programming language ST and may be modified as desired.

The task of the CheckBounds function is to monitor array limits and their exceedances. For example, if array limits are exceeded, an error flag can be set or the array indices can be changed. The function is called implicitly as soon as values are assigned to a variable of the type array.

☞ To maintain the monitoring functionality, the declaration part of the function may not be modified!

Using the CheckBounds function. The function is programmed in ST as follows by default:

Example to use the CheckBounds function

```
// Declaration:
FUNCTION CheckBounds : INT
VAR_INPUT
    index, lower, upper: INT;
END_VAR

// Program:
(* Automatically generated code:
This is a proposal for implementation. *)
IF  index < lower THEN
    CheckBounds := lower;
ELSIF  index > upper THEN
    CheckBounds := upper;
ELSE
    CheckBounds := index;
END_IF
```

When calling, the function obtains the following input parameters:

- index: Index of the array element
- lower: Lower limit of the array dimension
- upper: Upper limit of the array dimension

The return value is the index of the array element as long as it is located in the valid range. Otherwise, depending on how the limit range has been violated, the upper or lower limit is returned.

In the program below, the defined upper limit of the array "a" has been exceeded:

Example:

```
PROGRAM PLC_PRG
VAR
    a: ARRAY[0..7] OF BOOL;
    b: INT:=10;
END_VAR

a[b]:=TRUE;
```

In this case, the implicit call of the function CheckBounds carried out during the assignment causes the index 10 to be changed to the upper limit of the array range "a".

Thus, the value TRUE is assigned to the element [7]. This corrects attempted array accesses outside the valid array range.

Structures (STRUCT)

Structures are created in the project as "DUT" (DUT (Data Unit Type) objects, page 43,) objects using the command "Add".

Alternatively, add a DUT from the application library.

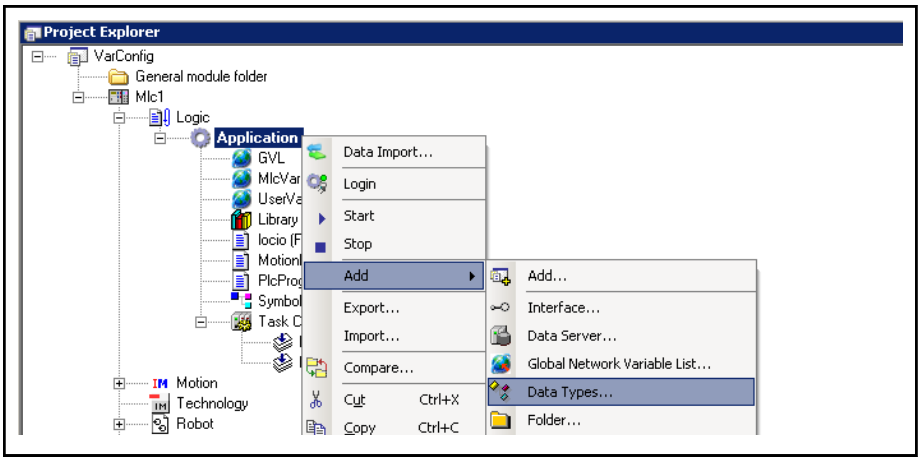


Fig.5-23: Adding a DUT

Structures start with the keywords **TYPE** and **STRUCT** and end with **END_STRUCT** and **END_TYPE**.

☞ In contrast to IndraLogic 1.x, a colon ":" has to be placed after **TYPE** in the structure declaration.

The syntax for structure declaration is as follows:

SYNTAX:

```
TYPE <structure_name>:
 STRUCT
   <variable_declaration 1>
            ...
   <variable_declaration n>
 END_STRUCT
END_TYPE
```

`<structure_name>` is a type that is identified in the entire project and that can be used as a standard data type.

Nested structures are permitted.

The only limitation is that addresses may not be assigned to variables (AT declaration is not allowed!).

A structure definition called polygon line:

Example: Structure with arrays as elements

```
TYPE Polygonline:
    STRUCT
        Start:ARRAY [1..2] OF INT;
        Point1:ARRAY [1..2] OF INT;
        Point2:ARRAY [1..2] OF INT;
        Point3:ARRAY [1..2] OF INT;
        Point4:ARRAY [1..2] OF INT;
        End:ARRAY [1..2] OF INT;
    END_STRUCT
END_TYPE
```

Initializing structures:

Example:

```
Poly_1:polygonline:=(Start:=[3,3], Point1 =[5,2], Point2:=[7,3],
                     Point3:=[8,5], Point4:=[5,7], End := [3,5]);
```

Initializations with variables are not permitted.

An example for initializing an array of a structure is located under Arrays, page 560.

Access to structure components

Structure components are accessed according to this syntax:

`<StructureName>.<ComponentName>`

For the example of the "polygon line" structure above, also use

`Poly_1.Start`

to access the "Start" component.

Union (UNION)

In an extension of the IEC 61131-3 standard it is possible to declare "unions" in user-defined data types.

In a union, all components have the same offset, i.e. they occupy the same memory location.

Thus, in the following example declaration of a union an assignment to name1.a would also apply to name1.b.

Example of a declaration in the data type editor (DUT):

```
TYPE name1: UNION
    a : LREAL;
    b : LINT;
    END_UNION
END_TYPE
```

Example of the initialization and use of UNION 'name1':

```
PROGRAM Plc_Main
VAR
  UnionInit:name1;
END_VAR

UnionInit.a:=10; // LREAL, 10.0
```

Enumerations

An enumeration is a user-defined data type consisting of a sequence of string constants.

Enumeration values are recognized globally in the project even if they are declared within a function block.

An enumeration is created in the project as "DUT" (DUT (Data Unit Type) objects, page 43,) objects using the command "Add".

☞ In contrast to IndraLogic 1.x, a local enumeration declaration can no longer be made - except in TYPE.

Syntax:

```
TYPE
 <name>:(<Enum_0>,<Enum_1>, ...,<Enum_n>)| <base_data_type>;
END_TYPE
```

A variable of type <identifier> can accept one of the enumeration values <Enum_..> and is initialized with the first of these values.

The values are compatible with integers, i.e. operations can be carried out with them as with INTEGER variables.

A number x can be assigned to each enumeration value.

If this assignment is not made explicitly in the declaration, the first component gets "0" and the next components "1, 2", etc.

If the assignment of number values is made in the declaration, ensure that the sequence of numbers increases in the enumeration.

Example:

```
TYPE TRAFFIC_SIGNAL: (red, yellow, green:=10);
 // The values for the colors are red=0, yellow=1, green=10
END_TYPE
    TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
    TRAFFIC_SIGNAL1:=0; // The value of the traffic signal is red
    FOR i:= red TO green DO
        i := i + 1;
    END_FOR;
```

Extensions of the EN 61131-3 standard:

1. The type name of an enumeration can be used (as namespace operator, page 608) to provide unique access to an enumeration constant. This allows to use the same constant in different enumerations.

Definition of two enumerations:

```
TYPE COLORS_1: (red, blue);
END_TYPE
TYPE COLORS_2: (green, blue, yellow);
END_TYPE
```

Enumeration value "blue" in a function block:

```
// Declaration:
colorvar1 : COLORS_1;
colorvar2 : COLORS_2;

// Implementation:

(* possible: *)
colorvar1 := colors_1.blue;
colorvar2 := colors_2.blue;

(* not possible: *)
colorvar1 := blue;
colorvar2 := blue;
```

2. The basic data type of the enumeration - preset as INT - can be defined by another data type.

Basic data type for the enumeration BigEnum is supposed to be DINT:

```
TYPE BigEnum : (yellow, blue, green:=16#8000) DINT;
END_TYPE
```

Subrange Types

A subrange type is a user-defined data type with a value range that only includes a subset of a basic type. Note that it can contain a range monitoring, page 566. The declaration can be made in a DUT (Data Unit Type) object, page 43,, but a variable can also be declared directly with a subrange type.

Syntax for the declaration as Data Unit Type (DUT):

```
TYPE <Name>:<Inttype> (<ll>..<ul>) END_TYPE
```

<Name>	Has to be a valid IEC identifier
<Inttype>	One of these data types is possible: SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, (BYTE, WORD, DWORD, LWORD).
<ll>	Constant that has to be compatible with the basic data type and that determines the lower limit of the range type. The lower limit itself belongs to this range.
<ul>	Constant that has to compatible with the basic data type and that determines the upper limit of the range type. The upper limit itself belongs to this range.

Examples:

Subranges in type definition:

```
TYPE
    SubInt : INT (-4095..4095);
END_TYPE
```

Direct declaration of a variable with a subrange type:

```
VAR
    i : INT (-4095..4095);
    ui : UINT (0..10000);
END_VAR
```

If a constant is assigned to a subrange type (in the declaration or in the statement) and this constant is not within the range (e.g. 1=5000), an error message is output.

Functions for range monitoring

To monitor the range limits of a subrange type at runtime, the function "CheckRangeSigned" or "CheckRangeUnsigned" has be integrated into the application.

They are integrated into the application as an object POUs for implicit checks, page 63, using the command Add, page 234,.

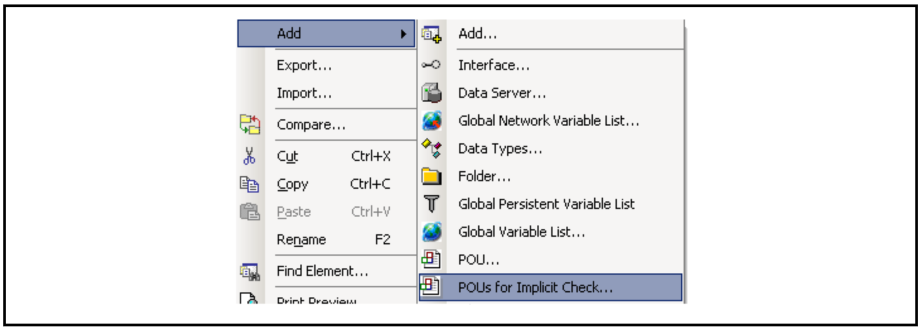
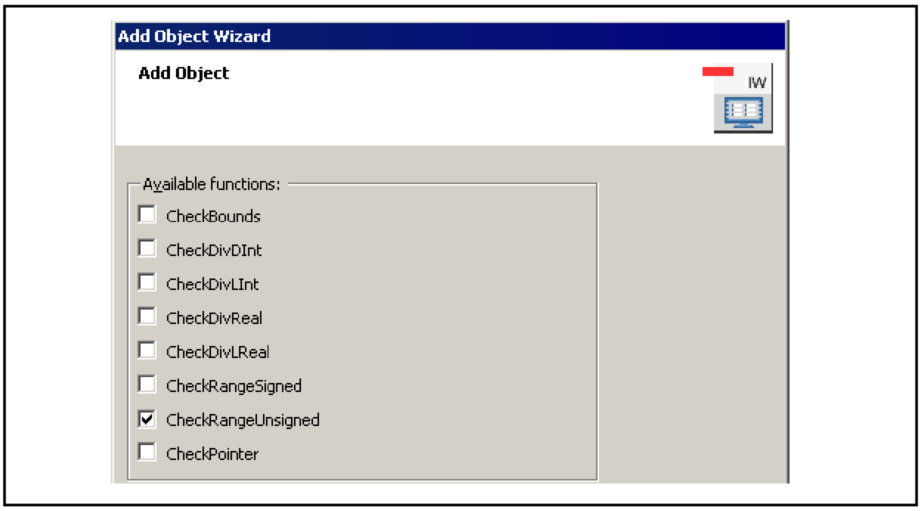


Fig.5-24: Add -> POUs for Implicit Check

After selecting the respective checkbox, select a program language and confirm it with "Open". The "CheckRangeSigned" or "CheckRangeUnsigned" function opens in the respective editor.


Add Object Wizard

Add Object

Available functions:

- □ CheckBounds
- □ CheckDivDInt
- □ CheckDivLInt
- □ CheckDivReal
- □ CheckDivLReal
- □ CheckRangeSigned
- ☑ CheckRangeUnsigned
- □ CheckPointer

Fig.5-25: Selecting "CheckRangeSigned" or "CheckRangeUnsigned"

The declaration of the functions is specified for all languages and may not be modified! Only local variables may be added. A suggestion for programming the functions is available in ST. It may be freely modified.

The task of the functions "CheckRangeSigned" or "CheckRangeUnsigned" is to monitor range limits and instances and their exceedances.

If range limits are exceeded, an error flag can be set or a value can be changed for example.

The function is called implicitly when a value is assigned to a variable of the type subrange.

☞ To maintain the monitoring functionality, the declaration part of the function may not be modified!

If a value is assigned to a variable of a signed subrange type, this causes an automatic call of the "CheckRangeSigned" function. The function, which limits the assignment value to the subrange specified at the variable declaration, is implemented in ST by default as follows:

Example:

```
  // Automatically generated code : DO NOT EDIT
FUNCTION CheckRangeSigned : DINT
  VAR_INPUT
```

```
    value, lower, upper: DINT; END_VAR

  // Automatically generated code :
          // It deals with an implementation suggestion.
IF   (value < lower) THEN
      CheckRangeSigned := lower;
ELSIF(value > upper) THEN
      CheckRangeSigned := upper;
ELSE
      CheckRangeSigned := value;
END_IF
```

At the call, the functions of the following input parameters are transferred.

- Value: Value of the variable to be assigned by the subrange type
- lower: Lower range limit
- upper: Upper range limit

The return value is the assigned value itself as long as it is within the valid range.

Otherwise, the upper or lower limit is returned depending on the violation of the subrange.

The

assignment without limit monitoring

```
i:=10*y
```

is now replaced explicitly by

the assignment with limit monitoring

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

For example, if "y" has the value "1000", the value "10*1000=10000" is not assigned to the variable "i" as intended in the code, but the value of the upper range limit instead which is "4095".

The same applies for the "CheckRangeUnsigned" function.

☞ If neither the "CheckRangeSigned" nor the "CheckRangeUnsigned" function is available, the subrange is not checked for the respective variables at runtime! In this case, any value between –32768 and 32767 can be assigned to a variable of a subrange type of the data type INT!

☞ When the functions "CheckRangeSigned" and "CheckRangeUnsigned" are integrated, endless loops can result.

This is the case for example if the counter variable of a FOR loop is a subrange type and the counting range of the loop leaves the defined subrange!

Example:

```
VAR
 ui : UINT (0..10000);
END_VAR

 FOR ui:=0 TO 10000 DO
 ...
 END_FOR
```

The FOR loop is never exited, since the monitoring function "CheckRangeSigned" prevents "ui" from being set to a value greater than 1000.

5.3 IEC Operators and Standard-Extending Functions

5.3.1 IEC Operators and Standard-Extending Functions, General Information

IndraLogic 2G supports all IEC operators. In contrast to the default functions, these operators are implicitly known across the entire project.

In addition to the IEC operators, the following operators, which are not described by the IEC standard, are also supported: ANDN, ORN, XORN, INDEXOF and SIZEOF (see arithmetic operators), ADR and BITADR and content operators (see address operators), some "namespace operators".

Operators are used in a function block like functions.

☞ For operations with floating point data types, the result of calculation depends on the target system hardware used!

- For REAL variables, the "double precision" calculation is used for the controls L45/L65.
- Due to the processor, this approach is not possible for L25 controls. Therefore, it is calculated with the "single precision" method.

Categories of operators:

- Assignment operators, MOVE, page 573
- Arithmetic operators, page 569
- Bit string operators, page 575
- Bit shift operators, page 577
- Selection operators, page 581
- Relational operators, page 584
- Address operators, page 587
- Call operators, page 589
- Type conversions, page 590
- Numeric functions, page 602
- IEC-extending "namespace operators", page 608

5.3.2 Arithmetic Operators

Arithmetic Operators, Overview

The following operators described by the IEC 61131-3 standard are supported:

- ADD, page 570
- MUL, page 570
- SUB, page 570
- DIV, page 571
- MOD, page 573
- MOVE, page 573

The following operators extend the standard:

- INDEXOF, page 574
- SIZEOF, page 574

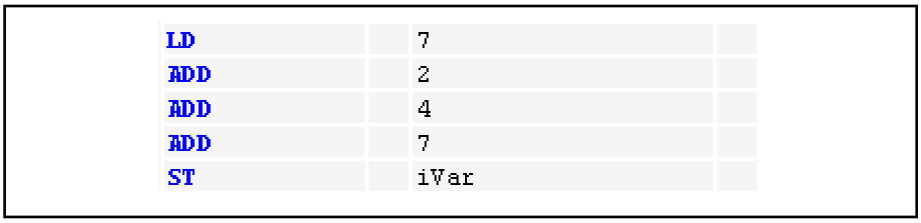
ADD

IEC operator: Addition of variables.

Types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

Two TIME variables can be added; the result can also be from a different time data type,

(e.g. t#45s + t#50s = t#1m35s).



```
LD      7
ADD     2
ADD     4
ADD     7
ST      iVar
```

Fig.5-26: Operator ADD in IL

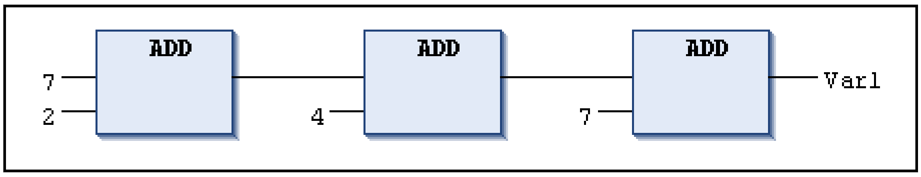


Fig.5-27: Operator ADD in FBD

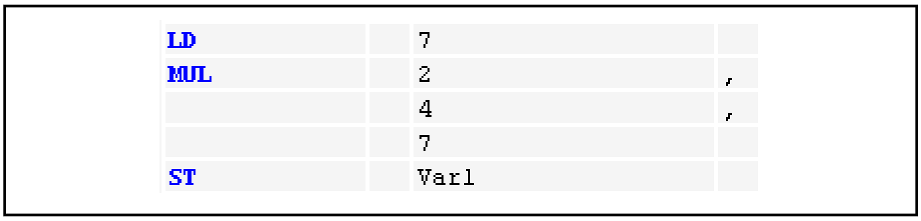
Operator ADD in ST

```
var1 := 7+2+4+7;
```

MUL

IEC operator: Multiplication of variables.

Types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.



```
LD      7
MUL     2    ,
        4    ,
        7
ST      Var1
```

Fig.5-28: Operator MUL in IL

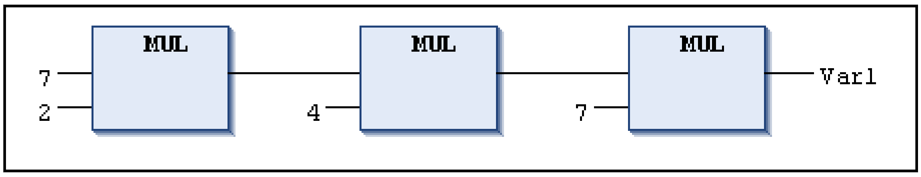


Fig.5-29: Operator MUL in FBD

Operator MUL in ST

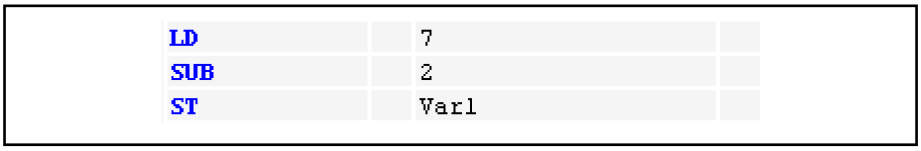
```
var1 := 7*2*4*7;
```

SUB

IEC operator: Subtraction of one variable from another.

Types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.

A TIME variable can also be subtracted from a variable of a different TIME type. The result is located in a variable of a third TIME type. However, note that negative TIME values are undefined.



```
LD   7
SUB  2
ST   Var1
```

Fig.5-30: Operator SUB in IL

Result: Content of Var1 is 5

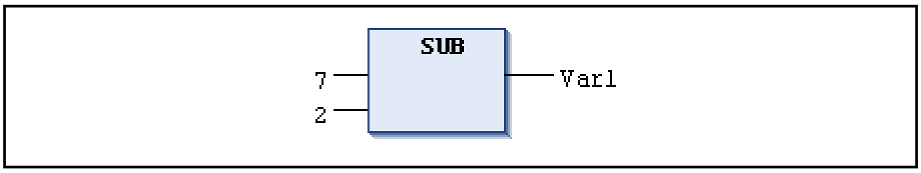


Fig.5-31: Operator SUB in FBD

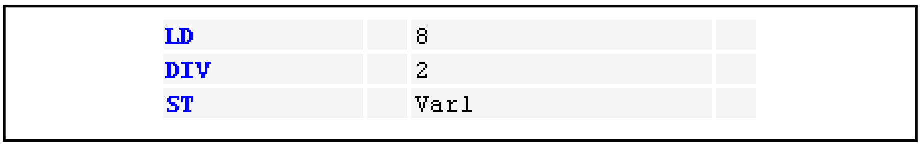
Operator SUB in ST

```
var1 := 7-2;
```

DIV

IEC operator: Division of one variable by another.

Types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT, REAL and LREAL.



```
LD   8
DIV  2
ST   Var1
```

Fig.5-32: Operator DIV in IL

Result: Content of Var1 is 4

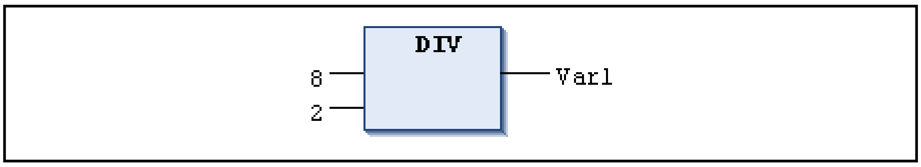


Fig.5-33: Operator DIV in FBD

Operator DIV in ST

```
var1 := 8/2;
```

The behavior when dividing by zero can depend on the target system!

The behavior at a division can be preset by the user. Configurable floating point exceptions, page 335.

Functions for check

The functions **CheckDivInt**, **CheckDivLint**, **CheckDivReal** and **CheckDivLReal** can be used to monitor the value of a divisor in order to prevent division by 0. After they are integrated into the application, its call automatically precedes each division that occurs in the corresponding code.

They have to be integrated into the application as an object POUs for Implicit Check, page 63, using the command Add, page 234,.

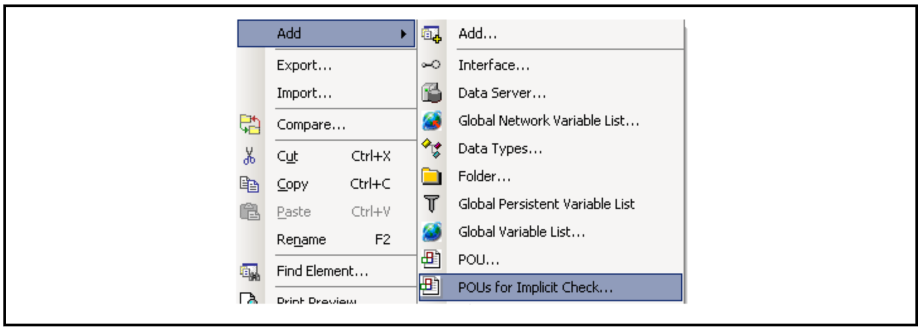
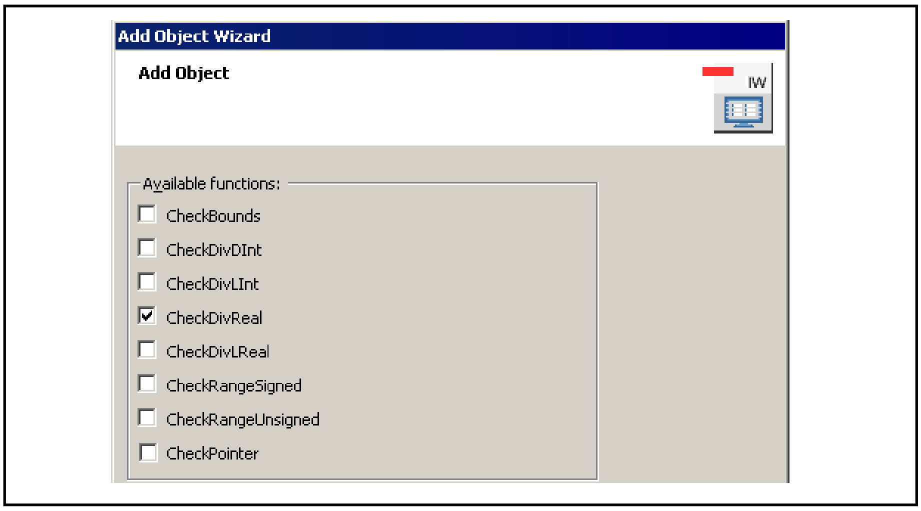


Fig.5-34: Add -> POUs for Implicit Check


Add Object Wizard

Add Object

Available functions:

- ☐ CheckBounds
- ☐ CheckDivDInt
- ☐ CheckDivLInt
- ☑ CheckDivReal
- ☐ CheckDivLReal
- ☐ CheckRangeSigned
- ☐ CheckRangeUnsigned
- ☐ CheckPointer

Fig.5-35: Selection "CheckDivReal"

The declaration part of the functions is fixedly determined and may not changed. Only local variables may be added. A suggestion to implement the functions is available in ST.

See the following example to implement the function CheckDivReal:

Default implementation of the function CheckDivReal in ST:

```
// Implicitly generated code : DO NOT EDI
FUNCTION CheckDivReal : REAL
VAR_INPUT
 divisor:REAL;
END_VAR
// Implicitly generated code :
// only an suggestion for implementation
IF divisor = 0 THEN
 CheckDivReal:=1;
ELSE
 CheckDivReal:=divisor;
END_IF;
```

The operator DIV uses the output of the function CheckDivReal as a divisor.

In the example program below, division by 0 is prevented, since the function "CheckDiv Real" changes the value of the divisor "d" - implicitly initiated with "0" - to "1" before the division is executed. The result of the division is thus 799.

Example in ST:

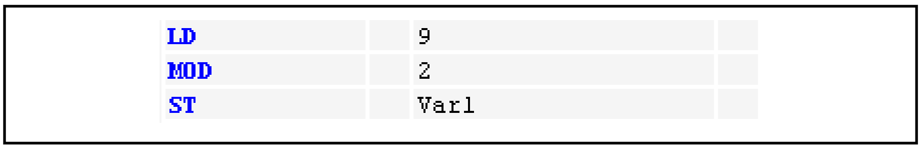
```
PROGRAM PLC_PRG
VAR
 erg:REAL;
 v1:REAL:=799;
 d:REAL;
END_VAR

erg:= v1 / d;
```

MOD

IEC operator: Modulo division of one variable by another.

Types allowed: BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT. This function returns the integer remainder of the division as a result.



```
LD     9
MOD    2
ST     Var1
```

Fig.5-36: Operator MOD in IL

Result: Content of Var1 is 1

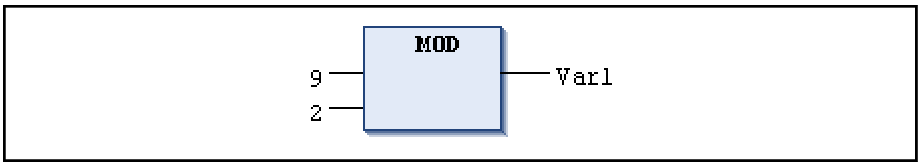


Fig.5-37: Operator MOD in FBD

Operator MOD in ST:

```
var1 := 9 MOD 2;
```

MOVE

IEC operator: Assignment of a variable to another variable of a corresponding type.

Since MOVE is available as function block in the CFC, FBD and LD editors, the EN/EN0 functionality can also be used for a variable assignment.

Example in CFC in connection with the EN/EN0 function:

Only if en_i is TRUE, the value of variable var2 is assigned to variable var1.

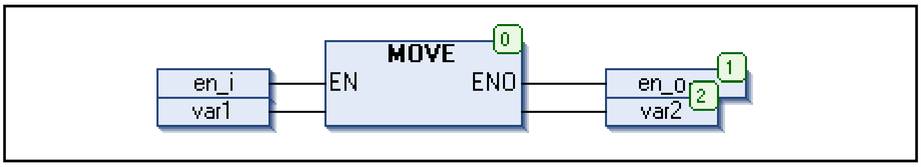
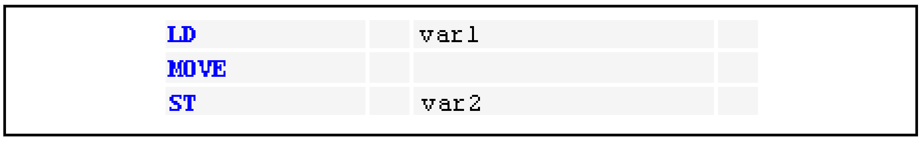


Fig.5-38: Operator MOVE in CFC

Without this EN/ENO functionality, using the MOVE operator is not reasonable, since the code piece becomes a simple assignment.

Example:

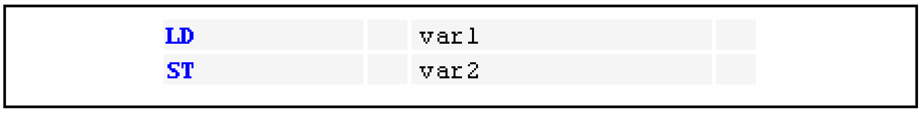
Example: Operator MOVE in IL (content of var2 is the content of var1):



```
LD      var1
MOVE
ST      var2
```

Fig.5-39: Operator MOVE in IL

This corresponds to:



```
LD      var1
ST      var2
```

Fig.5-40: Operators LD and ST in IL

Operator MOVE in ST:

```
ivar2 := MOVE(ivar1);
                      // you get the same result with:
ivar2 := ivar1;
```

SIZEOF

This arithmetic operator is not specified by the IEC 61131-3 standard.

It can be used to specify the number of bytes needed by the specified variable x.

The SIZEOF operator always returns an unsigned value. The type of the return variable adjusts to the size of the variable x.

Return value of SIZEOF(x)	Data type of the constants implicitly used for the size found
0 ≤ size of x < 256	USINT
256 ≤ size of x < 65536	UINT
65536 ≤ size of x < 4294967296	UDINT
4294967296 ≤ size of x	ULINT

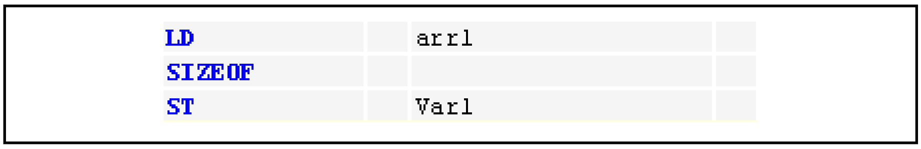
Fig.5-41:

Declaration:

```
 arr1:ARRAY[0..4] OF INT;
 Var1:INT;
```

Operator SIZEOF in ST:

```
var1 := SIZEOF(arr1); (* i.e.: var1:=USINT#10; *)
```



```
LD      arr1
SIZEOF
ST      Var1
```

Fig.5-42: Operator Sizeof in IL

Result is 10

INDEXOF

This arithmetic operator is not specified by the IEC 61131-3 standard.

The internal index of a function block can be determined with this function.

Example: Operator INDEXOF in ST:

```
var1 := INDEXOF(POU2);
```

5.3.3 Bit String Operators

Bit String Operators, Overview

The following bit string operators described by the IEC 61131-3 standard are supported:

- AND, page 575
- OR, page 576
- XOR, page 576
- NOT, page 577

Not yet available: The following bit string operators are supported in an extension of the standard:

- ANDN, page 575
- ORN, page 576
- XORN, page 577

Bit string operators compare the bits from two or more operands.

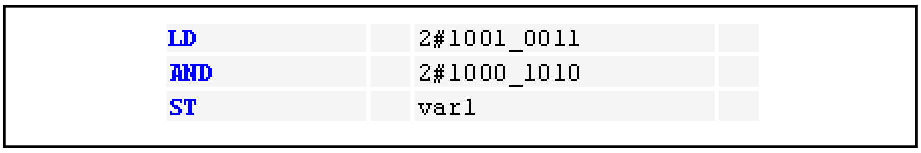
AND

IEC bit string operator: AND bit by bit from bit operands. If the input bits are 1, the output bit is 1. Otherwise, it is 0.

Types allowed: BOOL, BYTE, WORD, DWORD, LWORD.

Declaration:

```
VAR
 Var1:BYTE;
END_VAR
```



```
LD      2#1001_0011
AND     2#1000_1010
ST      var1
```

Fig.5-43: Operator AND in IL

Result: Content of Var1 is 2#1000_0010

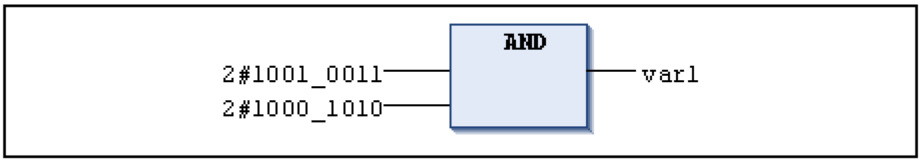


Fig.5-44: Operator AND in FBD

Operator AND in ST:

```
var1 := 2#1001_0011 AND 2#1000_1010
```

ANDN

### In preparation ###

This IEC bit string operator is not described in the IEC 61131-3 standard.

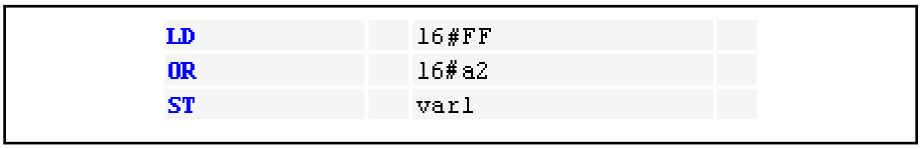
OR

IEC bit string operator: OR bit by bit from bit operands. If at least one of the input bits is 1, the output bit is 1. Otherwise, it is 0.

Types allowed: BOOL, BYTE, WORD, DWORD, LWORD.

Declaration:

```
VAR
 Var1:BYTE;
END_VAR
```



```
LD    16#FF
OR    16#a2
ST    var1
```

Fig.5-45: Operator OR in IL

Result: Content of var1 is 2#1001_1011

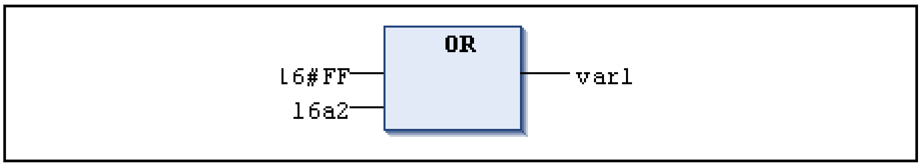


Fig.5-46: Operator OR in FBD

Operator OR in ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

ORN

### In preparation ###

This IEC bit string operator is not described in the IEC 61131-3 standard.

XOR

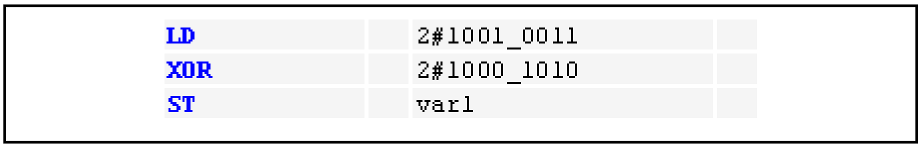
IEC bit string operator: XOR bit by bit from bit operands. If only one of the two bit inputs is 1, the result is 1. If both inputs are 0 or 1, the result is 0.

Types allowed: BOOL, BYTE, WORD, DWORD, LWORD.

☞ Note the behavior of the XOR function block in extended form, that is with more than two inputs: The inputs are checked in pairs and, in turn, the respective results are then compared with each other (meets the standard, but not necessarily expectations).

Declaration:

```
VAR
 Var1:BYTE;
END_VAR
```



```
LD    2#1001_0011
XOR   2#1000_1010
ST    var1
```

Fig.5-47: Operator XOR in IL

Result: var1 has the content 2#0001_1001

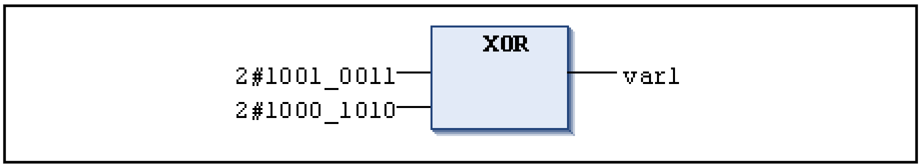


Fig.5-48: Operator XOR in FBD

Operator XOR in ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

XORN

### In preparation ###

This IEC bit string operator is not described in the IEC 61131-3 standard.

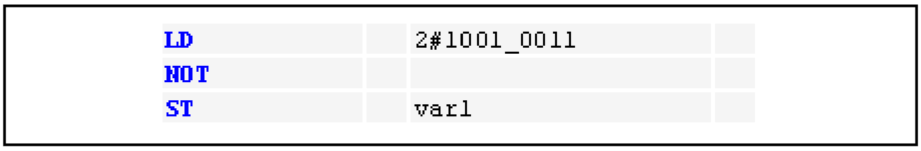
NOT

IEC bit string operator: NOT bit by bit of a bit operand. The output bit is 1 if the corresponding input bit is 0 and vice versa.

Types allowed: BOOL, BYTE, WORD, DWORD, LWORD.

Declaration:

```
VAR
 Var1:BYTE;
END_VAR
```



```
LD      2#1001_0011
NOT
ST      var1
```

Fig.5-49: Operator NOT in IL

Result: Content of var1 is 2#0110_1100

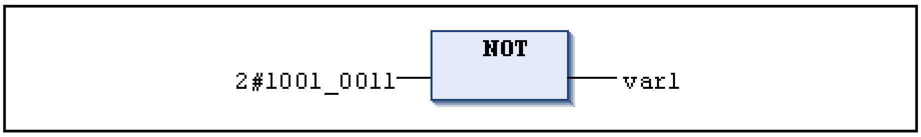


Fig.5-50: Operator NOT in FBD

Operator NOT in ST:

```
Var1 := NOT 2#1001_0011
```

5.3.4 Bit Shift Operators

Bit Shift Operators, Overview

The following bit shift operators described in the IEC 61131-3 standard are supported:

- SHL, page 577
- SHR, page 578
- ROL, page 579
- ROR, page 580

SHL

IEC operator: Shifting an operand to the left bit by bit.

SYNTAX:

```
erg:= SHL (in, n)
```

in: Operand to be shifted to the left.

n: Number of bits by which **in** is shifted to the left.

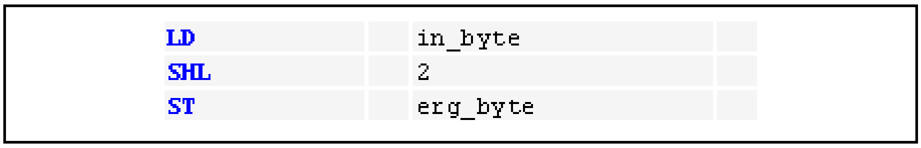
If **n** exceeds the data type width, BYTE, WORD, DWORD and LWORD operands are filled with zeroes, while fixed sign type operands, e.g. INT, are shifted arithmetically, i.e. they are filled with the value of the top bit.

> ☞ Note that the number of bits considered for the arithmetic operations is specified by the data type of the input variable "in". If this is a constant, the smallest possible data type is considered. The data type of the output variable does not affect the arithmetic operation.

In the following example in a hexadecimal display, note that the different results for erg_byte and erg_word depend on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are equal.

Operator SHL in ST:

```
PROGRAM shl_st
 VAR
  in_byte : BYTE:=16#45;
  in_word : WORD:=16#45;
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
 END_VAR
  erg_byte:=SHL(in_byte,n); (* 16#14*)
  erg_word:=SHL(in_word,n); (* 16#01141*)
```



```
LD    in_byte
SHL   2
ST    erg_byte
```

Fig.5-51: Operator SHL in IL

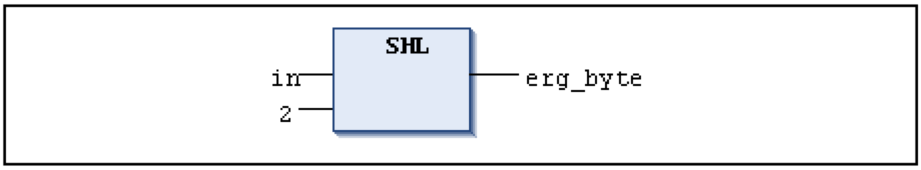


Fig.5-52: Operator SHL in FBD

SHR

IEC operator: Shifting an operand to the right bit by bit.

SYNTAX:

```
erg:= SHR (in, n)
```

in: Operand to be shifted to the right.

n: Number of bits by which **in** is shifted to the right.

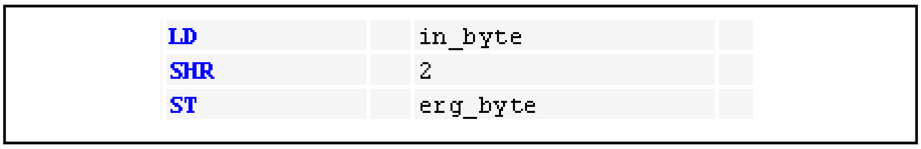
If **n** exceeds the data type width, BYTE, WORD, DWORD and LWORD operands are filled with zeroes, while fixed sign type operands, e.g. INT, are shifted arithmetically, i.e. they are filled with the value of the top bit.

Note that the number of bits considered for the arithmetic operations is specified by the data type of the input variable "in". If this is a constant, the smallest possible data type is considered. The data type of the output variable does not affect the arithmetic operation.

In the following example in a hexadecimal display, note that the different results for erg_byte and erg_word depend on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are equal.

Operator SHR in ST:

```
PROGRAM shr_st
 VAR
  in_byte : BYTE:=16#45;
  in_word : WORD:=16#45;
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
 END_VAR
 erg_byte:=SHR(in_byte,n); (* 16#11 *)
 erg_word:=SHR(in_word,n); (* 16#0011 *)
```



```
LD      in_byte
SHR     2
ST      erg_byte
```

Fig.5-53: Operator SHR in IL

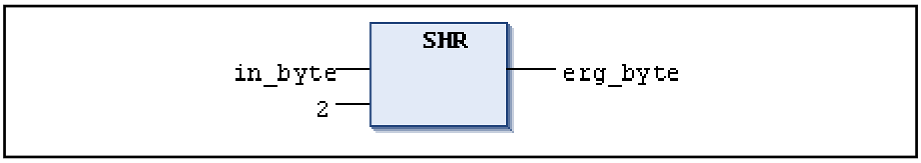


Fig.5-54: Operator SHR in FBD

ROL

IEC operator: Rotation of an operand bit by bit to the left.

SYNTAX:

```
erg:= ROL (in, n)
```

Data types allowed: BYTE, WORD, DWORD and LWORD.

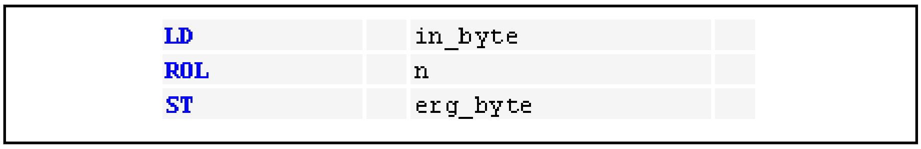
in is shifted **n** times 1 bit to the left and at the same time, the bit with the outermost position at the left is inserted again from the right.

Note that the number of bits considered for the arithmetic operations is specified by the data type of the input variable "in". If this is a constant, the smallest possible data type is considered. The data type of the output variable does not affect the arithmetic operation.

In the following example in a hexadecimal display, note that the different results for erg_byte and erg_word depend on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are equal.

Operator ROL in ST:

```
PROGRAM rol_st
 VAR
  in_byte : BYTE:=16#45;
  in_word : WORD:=16#45;
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
 END_VAR
  erg_byte:=ROL(in_byte,n); (* 16#15 *)
  erg_word:=ROL(in_word,n); (* 16#0114 *)
```



```
LD      in_byte
ROL     n
ST      erg_byte
```

Fig.5-55: Operator ROL in IL

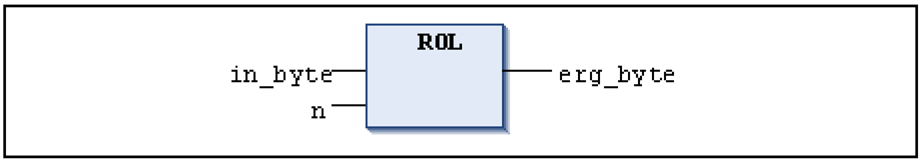


Fig.5-56: Operator ROL in FBD

ROR

IEC operator: Rotation of an operand bit by bit to the right.

SYNTAX:

```
erg = ROR (in, n)
```

Data types allowed: BYTE, WORD, DWORD and LWORD.

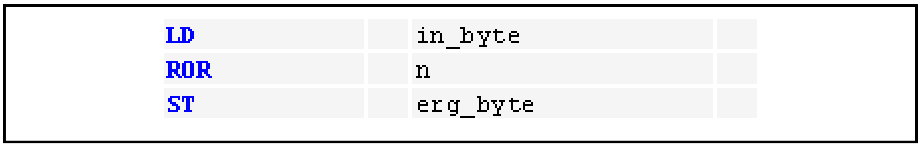
in is shifted **n** times 1 bit to the right and at the same time, the bit with the outermost position at the right is inserted again from the left.

☞ Note that the number of bits considered for the arithmetic operations is specified by the data type of the input variable "in". If this is a constant, the smallest possible data type is considered. The data type of the output variable does not affect the arithmetic operation.

In the following example in a hexadecimal display, note that the different results for erg_byte and erg_word depend on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are equal.

Operator ROR in ST:

```
PROGRAM ror_st
 VAR
  in_byte : BYTE:=16#45;
  in_word : WORD:=16#45;
  erg_byte : BYTE;
  erg_word : WORD;
  n: BYTE :=2;
 END_VAR
  erg_byte:=ROR(in_byte,n); (* 16#51*)
  erg_word:=ROR(in_word;n); (* 16#4011*)
```



```
LD      in_byte
ROR     n
ST      erg_byte
```

Fig.5-57: Operator ROR in IL

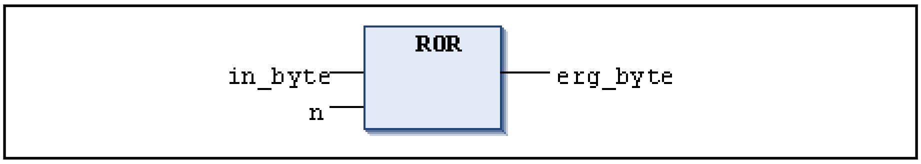


Fig.5-58: Operator ROR in FBD

5.3.5 Selection Operators

Selection Operators, Overview

All selection operators can also be used for constants and variables.

To ensure clarity in the examples, the following includes only constants:

- SEL, page 581
- MAX, page 582
- MIN, page 582
- LIMIT, page 583
- MUX, page 583

SEL

IEC selection operator: Binary selection. **G** specifies whether **IN0** or **IN1** is assigned to the variable **OUT**.

SYNTAX:

```
OUT := SEL(G, IN0, IN1) // bedeutet:
                        // OUT := IN0; if G=FALSE
                        // OUT := IN1; if G=TRUE.
```

Data types allowed:

IN0, IN1, OUT: Can be any type.

G: BOOL.

☞ An expression preceding IN0 is not calculated if G is TRUE!

Operator SEL in IL:

```
 LD   TRUE
 SEL 3,4    // IN0= 3, IN1= 4
 ST   Var1  // 4

 LD   FALSE
 SEL 3,4    // IN0= 3, IN1= 4
 ST   Var1  // 3
```

Operator SEL in ST:

```
Var1:=SEL(TRUE,3,4); // 4
```

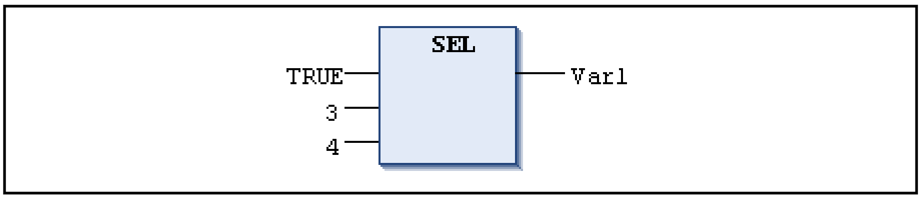


Fig.5-59: Operator SEL in FBD

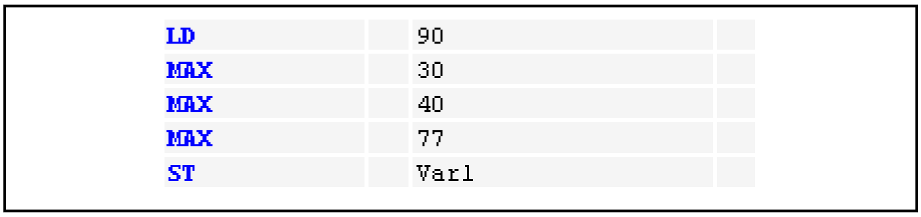
MAX

IEC selection operator: Maximum function. Returns the greater of the two values.

SYNTAX:

```
OUT := MAX(IN0, IN1)
```

IN0, IN1 and OUT can be any type.



```
LD      90
MAX     30
MAX     40
MAX     77
ST      Var1
```

Fig.5-60: Operator MAX in IL

Result is 90

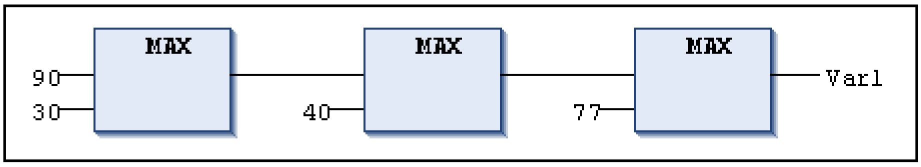


Fig.5-61: Operator MAX in FBD

Operator MAX in ST

```
Var1:=MAX(30,40);          // 40
Var1:=MAX(40,MAX(90,30)); // 90
```

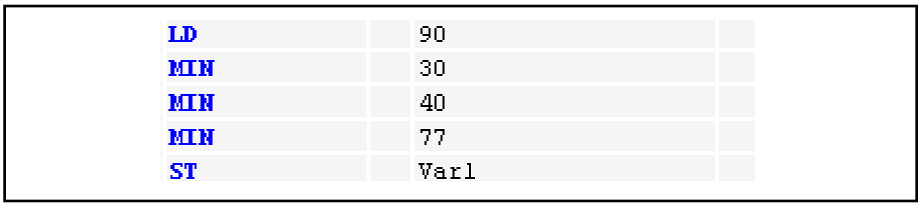
MIN

IEC selection operator: Minimum function. Returns the smaller of the two values.

SYNTAX:

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 and OUT can be any type.



```
LD      90
MIN     30
MIN     40
MIN     77
ST      Var1
```

Fig.5-62: Operator MIN in IL

Result is 30

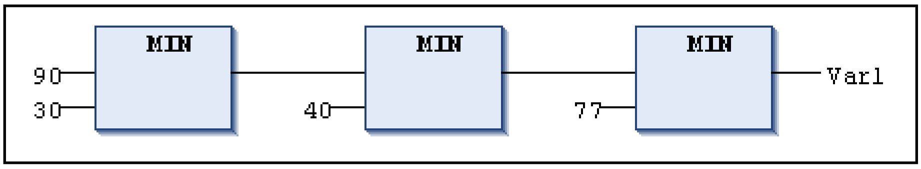


Fig.5-63: Operator MIN in FBD

Operator MIN in ST

```
Var1:=MIN(90,30);          // 30;
Var1:=MIN(MIN(90,30),40); // 30;
```

LIMIT

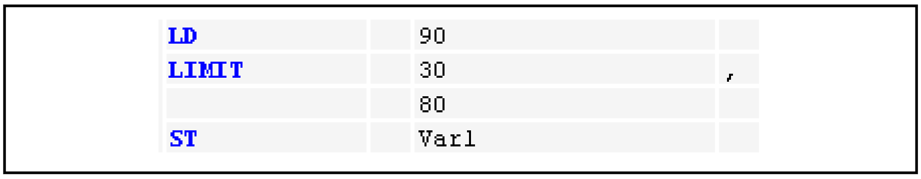
IEC selection operator: Limitation

SYNTAX:

```
OUT := LIMIT(Min, IN, Max)      // means:
OUT := MIN (MAX (IN, Min), Max)
```

"Max" is the upper limit value, "Min" the lower limit value for the result. If the value IN exceeds the upper limit Max, then LIMIT returns Max. If IN is lower than Min, then the result is Min.

IN and OUT can be any type.



```
LD      90
LIMIT   30      ,
        80
ST      Var1
```

Fig.5-64: Operator LIMIT in IL

Result is 80

Operator LIMIT in ST

```
Var1:=LIMIT(30,90,80); // 80
```

MUX

IEC selection operator: Operator: Multiplexer

SYNTAX:

```
OUT := MUX(K, IN0,...,INn) // means:
OUT := INk.
```

IN0, ...,INn and OUT can be any type.

K has to be of types BYTE, WORD, DWORD, LWORD, SINT, USINT, INT, UINT, DINT, UDINT, LINT or ULINT.

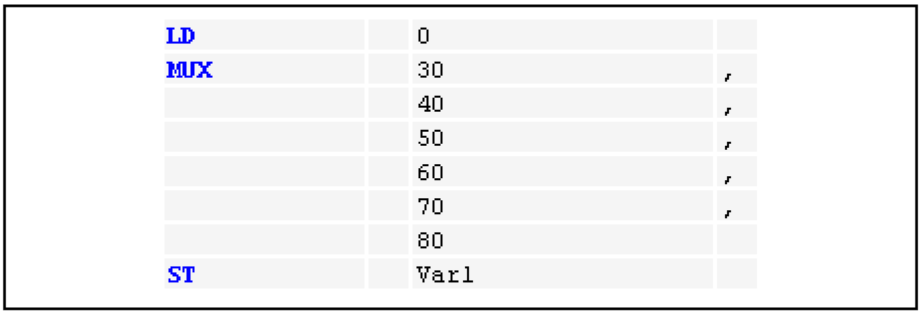
MUX selects the kth value from a set of values.

The first value is K=0.

If K is greater than the number of the other inputs (n), the final value is forwarded (INn).

To improve performance at runtime, only the expression preceding IN_k is calculated!

In contrast, all branches are calculated in the simulation.



```
LD      0
MUX     30      ,
        40      ,
        50      ,
        60      ,
        70      ,
        80
ST      Var1
```

Fig.5-65: Operator MUX in IL

Result is 30

Operator MUX in ST

```
Var1:=MUX(0,30,40,50,60,70,80); // 30;
```

5.3.6 Relational Operators

Relational Operators, Overview

The following relational operators described in the IEC 61131-3 standard are supported:

- GT, page 584
- LT, page 584
- LE, page 585
- GE, page 586
- EQ, page 586
- NE, page 587

These are Boolean operators each comparing two inputs (first and second operand).

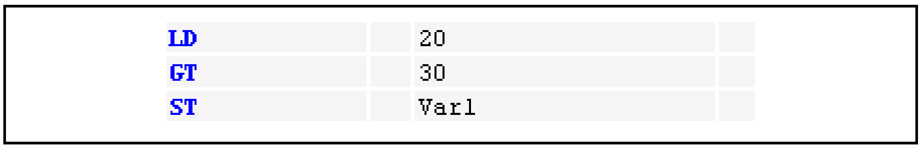
GT

IEC relational operator: greater than.

A Boolean operator with a result of TRUE if the first operand is greater than the second.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD   20
GT   30
ST   Var1
```

Fig.5-66: Operator GT in IL

Result is FALSE

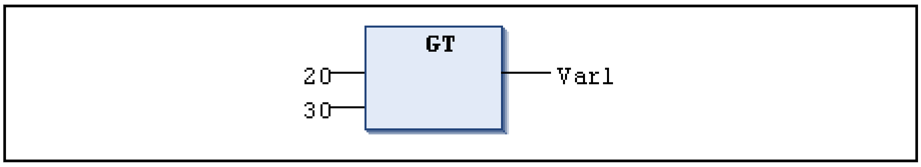


Fig.5-67: Operator GT in FBD

Operator GT in ST

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70; // FALSE
```

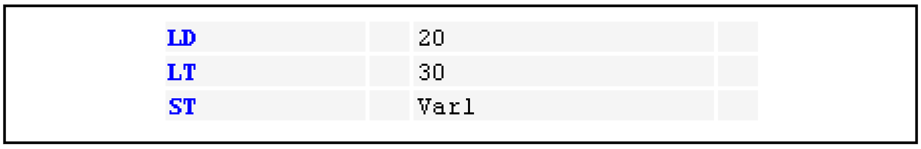
LT

IEC relational operator: Less than.

A Boolean operator with the result TRUE if the first operand is less than the second.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD    20
LT    30
ST    Var1
```

Fig.5-68: Operator LT in IL

Result is TRUE

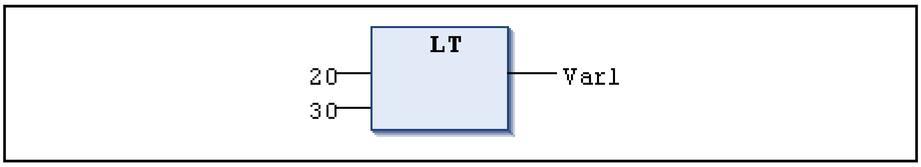


Fig.5-69: Operator LT in FBD

Operator LT in ST

```
VAR1 := 20 < 30; // TRUE
```

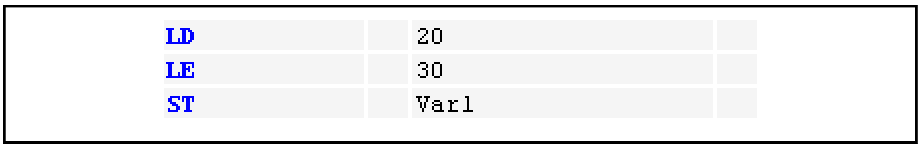
LE

IEC relational operator: Less than or equal to.

A Boolean operator with the result TRUE if the first operand is less than or equal to the second operand.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD    20
LE    30
ST    Var1
```

Fig.5-70: Operator LE in IL

Result is TRUE

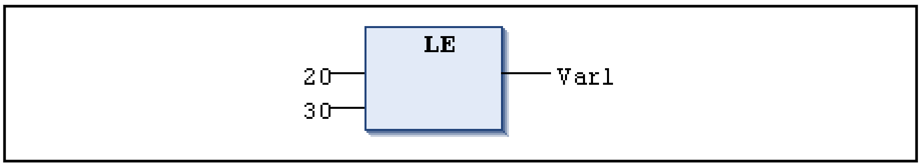


Fig.5-71: Operator LE in FBD

Operator LE in ST

```
VAR1 := 20 <= 30; // TRUE
```

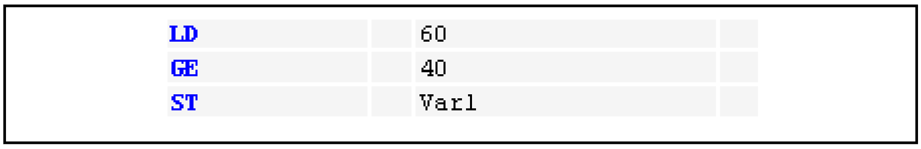
GE

IEC relational operator: Greater than or equal to

A Boolean operator with the result TRUE if the first operand is greater than or equal to the second operand.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD    60
GE    40
ST    Var1
```

Fig.5-72: Operator GE in IL

Result is TRUE

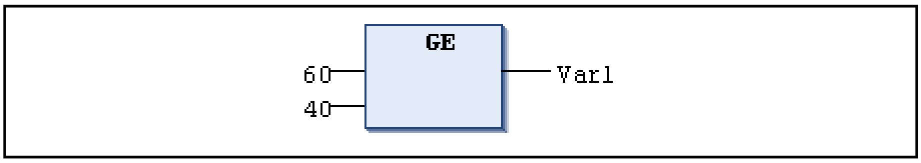


Fig.5-73: Operator GE in FBD

Operator GE in ST

```
VAR1 := 60 >= 40;  // TRUE
```

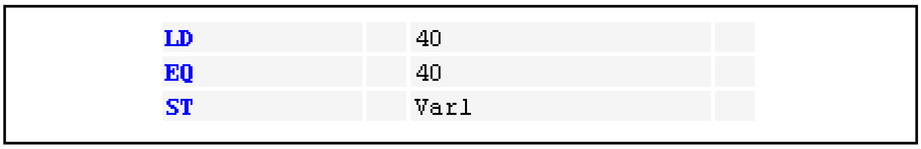
EQ

IEC relational operator: Equality

A Boolean operator with the result TRUE if the operands are equal.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD    40
EQ    40
ST    Var1
```

Fig.5-74: Operator EQ in IL

Result is TRUE

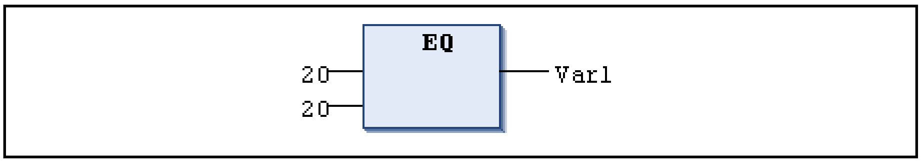


Fig.5-75: Operator EQ in FBD

Operator EQ in ST

```
VAR1 := 40 = 40; // TRUE
```

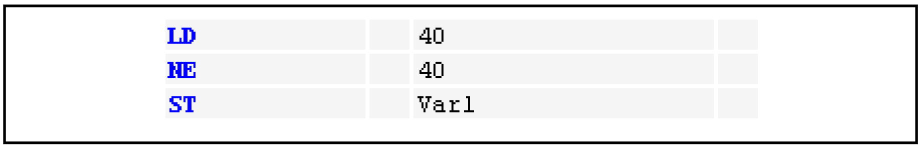
NE

IEC relational operator: Inequality

A Boolean operator with the result TRUE if the operands are not equal.

The operands can be any of the following types:

- BOOL, BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL,
- TIME, LTIME, DATE, TIME_OF_DAY, DATE_AND_TIME and
- STRING.



```
LD    40
NE    40
ST    Var1
```

Fig.5-76: Operator NE in IL

Result is FALSE

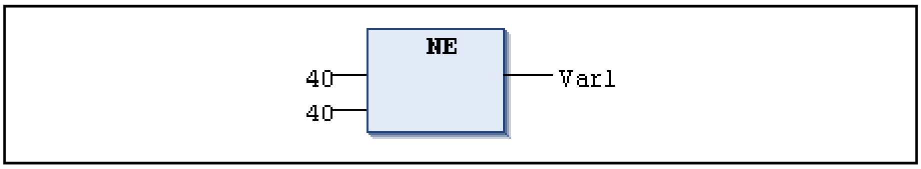


Fig.5-77: Operator NE in FBD

Operator NE in ST

```
VAR1 := 40 <> 40; // FALSE
```

5.3.7 Address Operators

Address Operators, Overview

In IndraLogic 2G, the following as address operators are available in the extension of the IEC 61131-3 standard:

- ADR, page 587
- BITADR, page 588,
- Content operator "^" , page 588

ADR

This address operator is not described in the IEC 61131-3 standard.

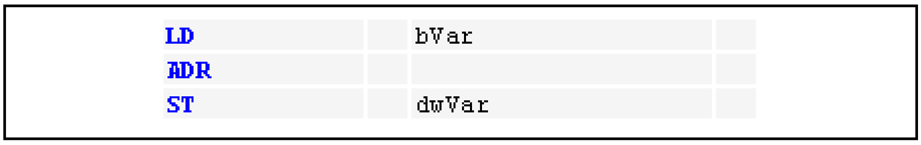
ADR returns the address, page 623, of its argument in a DWORD. This address can be sent to the vendor functions and treated there like a pointer or it can be assigned to a pointer, page 557, in the project.

☞ in contrast to IndraLogic1.x, the ADR operator can now be used with function, program, function block and method names and thus replaces the INDEXOF operator.

In this context, note the following as well: 'Function pointer, page 557'.

Always remember that function pointers can be forwarded to external libraries, but that **a function pointer cannot be called in IndraLogic**!

To enable a system call (runtime system), the corresponding object property ("Compile" tab) has to be set for the function tab.



```
LD      bVar
ADR
ST      dwVar
```

Fig.5-78: Operator ADR in IL

Operator ADR in ST

```
dwVar:=ADR(bVAR);
```

☞ If an online change, page 133, is applied, the contents of addresses can be shifted.

Thus, POINTER variables can point to an invalid memory space. To avoid problems, ensure that the pointer is not memorized but updated in each cycle.

☞ "Pointer TO Variables" of functions and methods should not be returned to the user or assigned to the global variables.

Content Operators

This address operator is not described in the IEC61131-3 standard.

It allows a pointer, page 557 to be dereferenced.

It is attached to the pointer identifier as "^".

Content operator in ST:

```
VAR
 pt:POINTER TO INT;
 var_int1:INT;
 var_int2:INT;
END_VAR
 pt := ADR(var_int1);
 var_int2:=pt^;
```

☞ If an online change, page 133, is applied, the contents of addresses can be shifted. Note when using pointers to point to addresses.

BITADR

This address operator is not described in the IEC 61131-3 standard.

BITADR returns the bit offset within a segment in a DWORD.

Note that the offset depends on whether the byte addressing option is enabled in the target system settings.

The current version of the implementation includes the following special features:

1. The most significant byte of the double word contains the additional information whether it is dealt with a flag variable (16#40), input variable (16#80) or output variable (16#C0).

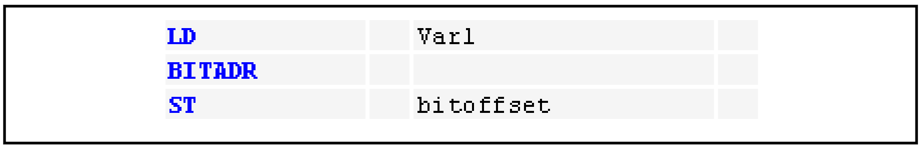
   If necessary, this part has to be masked out.
2. If the address was assigned with the complete path specification

   `<ApplicationName> <BlockName>.<VariableName>`

   , the compiler returns an error message.
3. If the address of a global variable was directly assigned, the compiler functions without an error.

Declaration:

```
VAR
 var1 AT %IX2.3:BOOL;
 bitoffset: DWORD;
END_VAR
```



```
LD       Var1
BITADR
ST       bitoffset
```

Fig.5-79: Operator BITADR in IL

Operator BITADR in ST

```
bitoffset:=BITADR(var1);
    (* Result if byte addressing=TRUE: 16x80000013,
    if byte addressing=FALSE: 16x80000023 *)
```

☞ If an online change, page 133, is applied, the contents of addresses can be shifted. Note when using pointers to point to addresses.

5.3.8 Call Operators

CAL, CALC, CALCN

IEC operator for calling a function block.

In IL, the instance of a function block, page 253, or an action, page 51, is called with CAL.

In addition to the "CAL" operator, the modifications

- CALC; call is made only if the preceding expression has the value 'TRUE'
- CALCN; call is made only if the preceding expression has the value 'FALSE'

are permitted.

Following the name of the instance of a function block in parentheses is the assignment of the input variables of the function block and the value output.

Calling the instance "Inst" of a function block with the assignment of the input variables Par1, Par2 to 0 or TRUE.

Operator CAL in IL:

```
CAL            ZAB(         // Invocation of instance ZAB of FB TON
          IN:= FALSE,       // Input parameters
          PT:= WAITINGTIME,
          ET=> ETVar)       // Result
```

5.3.9 Type Conversion Operators

Type Conversion Operators, Overview

IndraLogic 2G allows operands of different data types to be connected, although an automatic **implicit** type conversion is attempted first.

It is not permitted to implicitly convert from a "higher" type to a "smaller" type (for example from INT to BYTE or from DINT to WORD). If this should be done, use special type conversions. In principle, every elementary type can be converted into any other elementary type.

Implicit type conversion, BYTE ⇒ USINT

```
VAR
 byte1:BYTE;
 usint1:USINT;
END_VAR
 usint1:=usint1+byte1;
```

The compiler converts **BYTE** into **USINT** without a warning or error message.

Implicit type conversion, BYTE ⇒ SINT

```
VAR
 byte1:BYTE;
 sint1:SINT;
END_VAR
 sint1:=sint1+byte1;
```

The compiler converts **BYTE** into **SINT** with the **warning**:

Implicit conversion of the unsigned data type "BYTE" to the signed data type "SINT": Possible loss of sign;

If the difference between the respective data types increases, the compiler returns an **error message**.

The **explicit** type conversion can also be used.

In principle, every elementary type can be converted into any other elementary type.

Syntax:

```
<elem.Typ1>_TO_<elem.Typ2>
```

The following type conversions are supported:

- BOOL_TO conversions, page 591
- TO_BOOL conversions, page 593
- Conversions of integer types, page 594
- REAL_TO-/ LREAL_TO conversions, page 595
- TIME_TO/TIME_OF_DAY conversions, page 596
- DATE_TO/DT_TO conversions, page 597
- STRING_TO conversions, page 599

New compared to IndraLogic 1.x:

- TRUNC (conversion to DINT), page 601
- TRUNC_INT, page 601
- ANY_NUM_TO_<numeric data type>, page 601
- ANY_TO_<any data type>, page 601

For ...TO_STRING conversions, the string (character string) is generated left-justified. If it has been defined too short, it is cut off from the right.

BOOL_TO Conversions

Conversion from data type BOOL to another type.

Syntax:

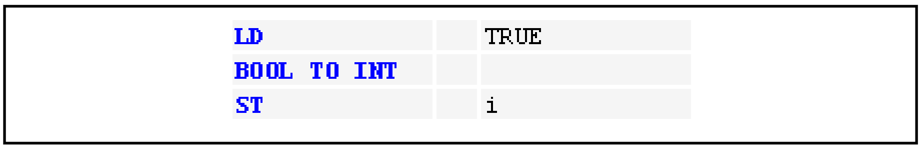
```
BOOL_TO_<data type>
```

For number types, the result is 1 if the operand is TRUE and 0 if the operand is FALSE.

For STRING types, the result is TRUE or FALSE.

Example:

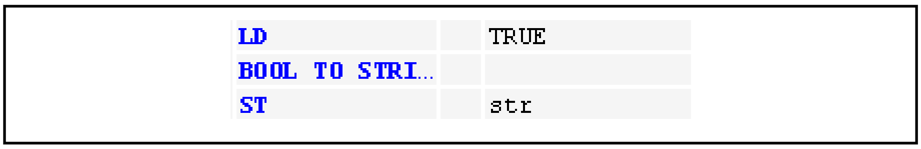
Operator BOOL_TO in IL



```
LD            TRUE
BOOL TO INT
ST            i
```

Fig.5-80: *BOOL TO INTEGER*

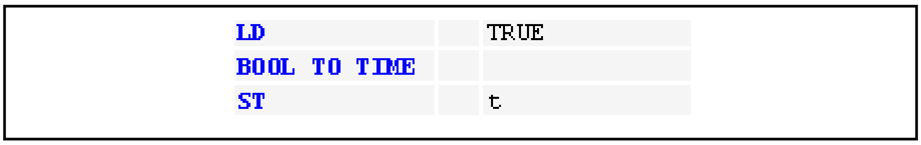
Result is 1



```
LD            TRUE
BOOL TO STRI...
ST            str
```

Fig.5-81: *BOOL TO STRING*

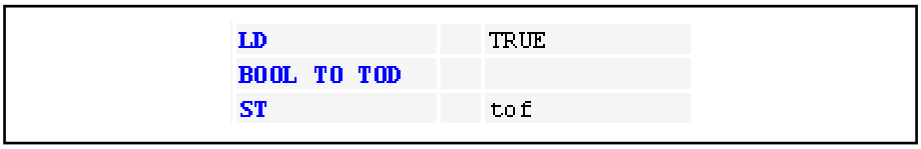
RESULT is TRUE



```
LD            TRUE
BOOL TO TIME
ST            t
```

Fig.5-82: *BOOL TO TIME*

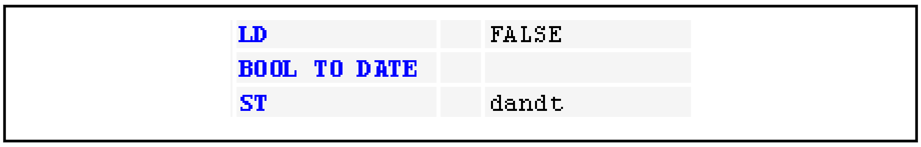
Result is T#1ms



```
LD            TRUE
BOOL TO TOD
ST            tof
```

Fig.5-83: *BOOL TO TOD*

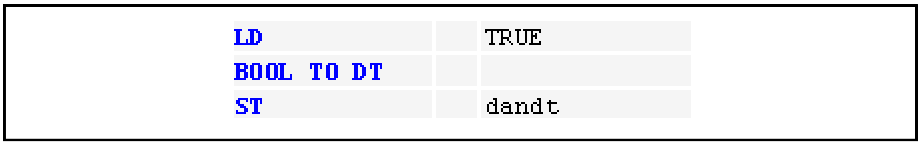
Result is TOD#00:00:00.001



```
LD            FALSE
BOOL TO DATE
ST            dandt
```

Fig.5-84: *BOOL TO DATE*

Result is D#1970-01-01



```
LD            TRUE
BOOL TO DT
ST            dandt
```

Fig.5-85: BOOL TO DT

Result is DT#1970-01-01-00:00:01

Operator BOOL_TO_ in ST

```
(*BOOL TO INTEGER:*)
i:=BOOL_TO_INT(TRUE); (*result is 1*)

(*BOOL TO STRING:*)
str:=BOOL_TO_STRING(TRUE); (*result is "TRUE"*)

(* BOOL TO TIME: *)
t:=BOOL_TO_TIME(TRUE); (*result is T#1ms*)

(*BOOL TO TIME OF DAY:*)
tof:=BOOL_TO_TOD(TRUE); (*result is TOD#00:00:00.001*)

(*BOOL TO DATE:*)
dat:=BOOL_TO_DATE(FALSE); (*result is D#1970*)

(*BOOL TO DATE AND TIME:*)
dandt:=BOOL_TO_DT(TRUE); (*result is DT#1970-01-01-00:00:01*)
```

Example:

Operator BOOL TO in FBD:

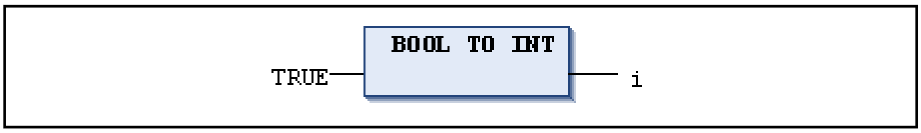


Fig.5-86: Operator BOOL TO Integer

Result is 1

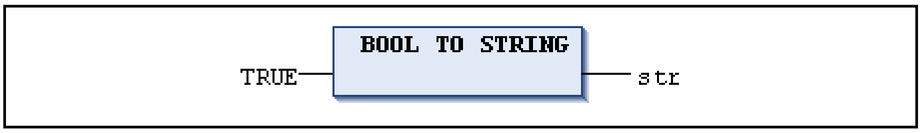


Fig.5-87: Operator BOOL TO STRING

Result is TRUE

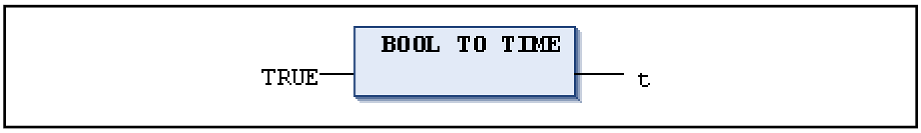


Fig.5-88: Operator BOOL TO TIME

Result is T#1ms

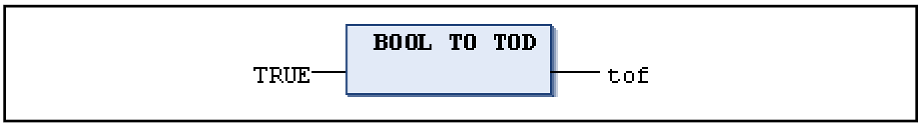


Fig.5-89: Operator BOOL TO TOD

Result is TOD#00:00:00.001

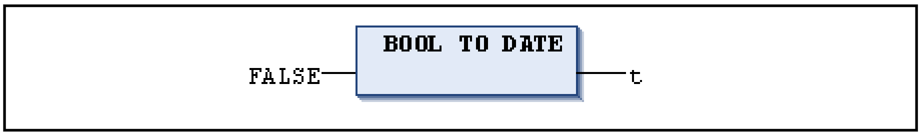


Fig.5-90: Operator BOOL TO DATE

Result is D#1970-01-01

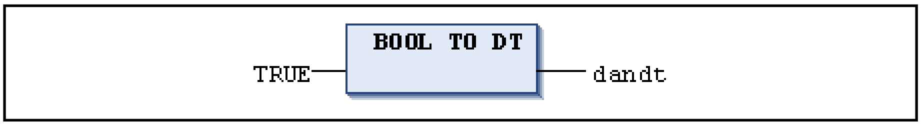


Fig.5-91: Operator BOOL TO DATE AND TIME

Result is DT#1970-01-01-00:00:01

TO_BOOL Conversions

Conversion from any data type to BOOL.

Syntax:

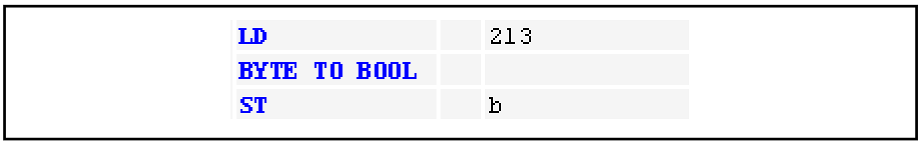
```
<data type>_TO_BOOL
```

The result is TRUE if the operand is not equal to 0. The result is FALSE if the operand is equal to 0.

For STRING types, the result is TRUE if the operand is 'TRUE'. Otherwise, the result is FALSE.

Example:

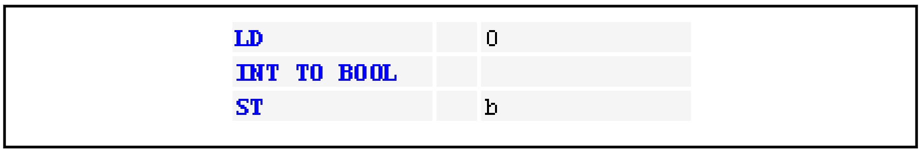
Operator TO BOOL in IL:



```
LD            213
BYTE TO BOOL
ST            b
```

Fig.5-92: Operator BYTE TO BOOL

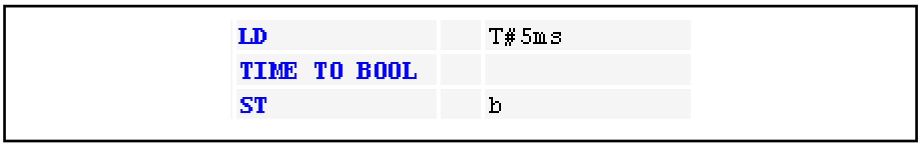
Result is TRUE



```
LD            0
INT TO BOOL
ST            b
```

Fig.5-93: Operator INTEGER TO BOOL

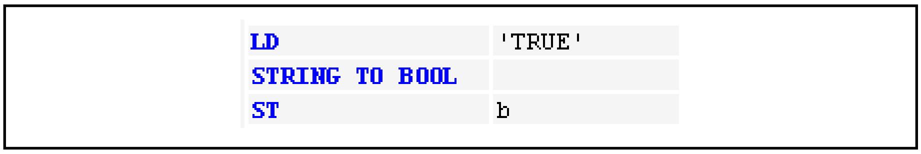
Result is FALSE



```
LD            T#5ms
TIME TO BOOL
ST            b
```

Fig.5-94: Operator TIME TO BOOL

Result is TRUE



```
LD              'TRUE'
STRING TO BOOL
ST              b
```

Fig.5-95: Operator STRING TO BOOL

Result is TRUE

Operator TO_BOOL in ST

```
(*BYTE TO BOOL:*)
b := BYTE_TO_BOOL(2#11010101);(*result is TRUE*)

(*INTEGER TO BOOL:*)
b := INT_TO_BOOL(0); (*result is FALSE*)

(*TIME TO BOOL:*)
b := TIME_TO_BOOL(T#5ms); (*result is TRUE *)

(*STRING TO BOOL:*)
b := STRING_TO_BOOL('TRUE'); (*result is TRUE*)
```

Example:

Operator TO BOOL in FBD:

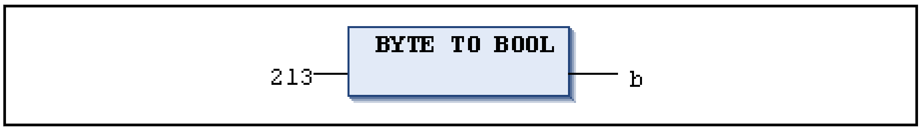


Fig.5-96: Operator BYTE TO BOOL

Result is TRUE

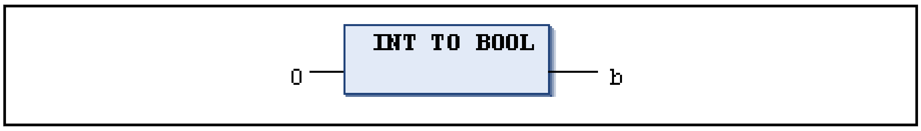


Fig.5-97: Operator INTEGER TO BOOL

Result is FALSE

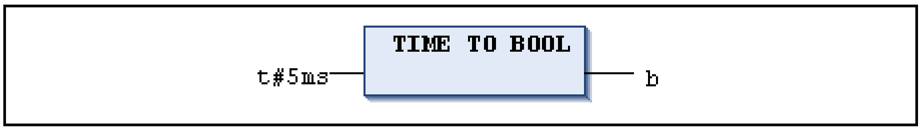


Fig.5-98: Operator TIME TO BOOL

Result is TRUE

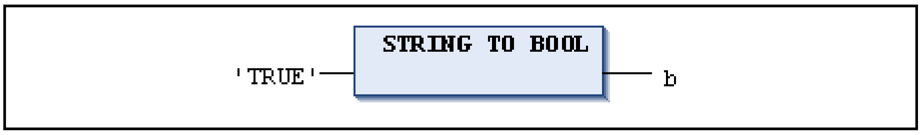


Fig.5-99: Operator STRING TO BOOL

Result is TRUE

Conversion of Integer Types

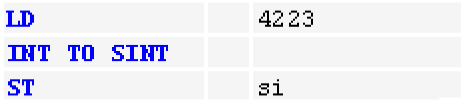
Conversion of an integer type to another number type:

Syntax:

```
<INT data type>_TO_<INT data type>
```

☞ When converting larger types to smaller types, information can be lost.

If the number to be converted exceeds the range limit, the initial bytes of the number are not considered.



```
LD              4223
INT TO SINT
ST              si
```

Fig.5-100: Operator INT TO SINT in IL

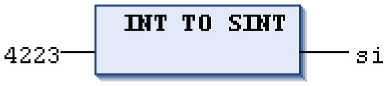


Fig.5-101: Operator INT TO SINT in FBD

☞ Save the integer 4223 (16#107f in hexadecimal) to a SINT variable. The, it gets the number 127 (16#7f in hexadecimal).

Operator INTEGER TO SIGNED INTEGER in ST

```
si := INT_TO_SINT(4223); (*result is 127*)
```

REAL_TO/LREAL_TO Conversions

Conversion from REAL or LREAL type to another type.

It is rounded up or down to an integer value and converted into the respective type. Exceptions are the types STRING, BOOL, REAL and LREAL.

☞ If REAL or LREAL is converted into SINT, USINT, INT, UINT, DINT, UDINT, LINT or ULINT and the value of the REAL/LREAL number is outside the value range of the integer, the result is undefined and depends on the target system.

An exception is then possible!

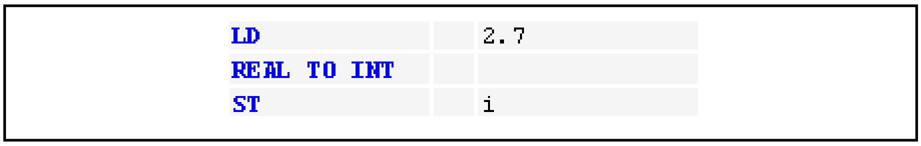
To get a target system-independent code, value range exceedances have to be intercepted via the application.

If the REAL/LREAL number is within the range, the conversion runs equally on all systems.

Note that for conversions into STRING type, the number of decimal places is limited to 16. IF the (L)REAL number contains more places, the 16th place is rounded and displayed in the STRING.

If the STRING defined is too short for the number, the number is shortened from the right.

☞ When converting larger types to smaller types, information can be lost.



```
LD           2.7
REAL TO INT
ST           i
```

Fig.5-102: Operator REAL TO INT in IL

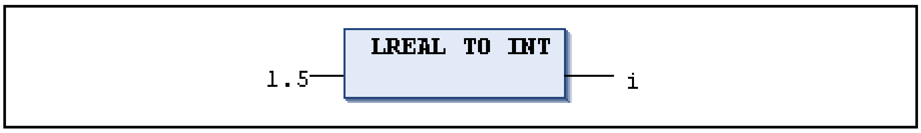


Fig.5-103: Operator LREAL TO INT

Operator REAL TO INTEGER in ST

```
i := REAL_TO_INT(1.5); (*result is 2*)
j := REAL_TO_INT(1.4); (*result is 1*)
k := REAL_TO_INT(-1.5); (*result is -2*)
l := REAL_TO_INT(-1.4); (*result is -1*)
```

TIME_TO/TIME_OF_DAY Conversions

Conversion from TIME or TIME_OF_DAY type to another type.

Syntax:

```
<TIME-data type>_TO_<data type>
```

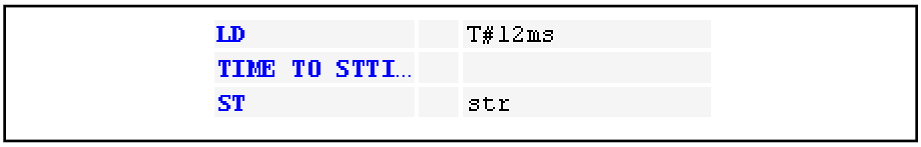
The time is internally saved into a DWORD in milliseconds (for TIME_OF_DAY starting at 00:00 o'clock). This value is converted.

For STRING types, the result is the time constant.

☞ When converting larger types to smaller types, information can be lost.

Example:

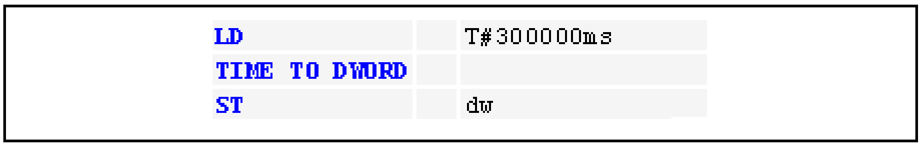
Operator TIME TO in IL



```
LD            T#12ms
TIME TO STRI…
ST            str
```

Fig.5-104: Operator TIME TO STRING

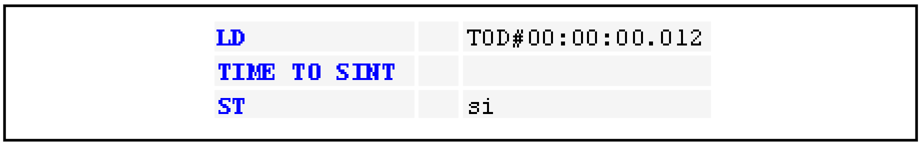
Result is T#12ms



```
LD             T#300000ms
TIME TO DWORD
ST             dw
```

Fig.5-105: Operator TIME TO DWORD

Result is 300000



```
LD              TOD#00:00:00.012
TIME TO SINT
ST              si
```

Fig.5-106: Operator TIME TO SINT

Result is 12

Operator TIME TO in ST

```
(*TIME TO STRING:*)
str:=TIME_TO_STRING(T#12ms); (*result is T#12ms*)

(*TIME TO DWORD:*)
dw:=TIME_TO_DWORD(T#5m); (*result is 300000*)

(*TIME TO SINT:*)
si:=TOD_TO_SINT(TOD#00:00:00.012); (*result is 12*)
```

Example:

Operator TIME TO in FBD

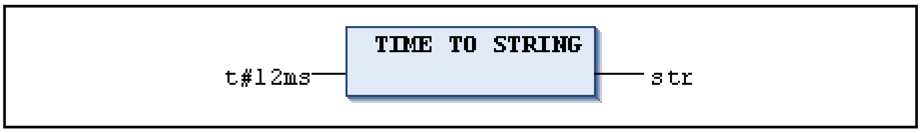


Fig.5-107: Operator TIME TO STRING

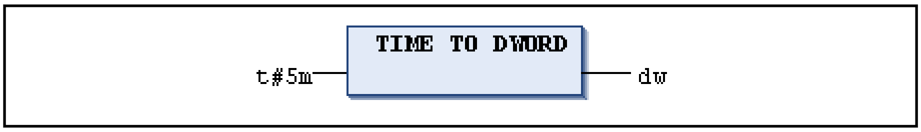


Fig.5-108: Operator TIME TO DWORD

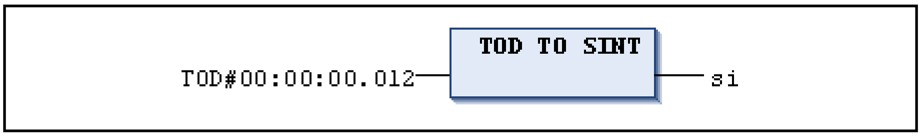


Fig.5-109: Operator TOD TO SINT

DATE_TO/DT_TO Conversions

IEC operator: Conversion from DATE or DATE_AND_TIME type to another type.

Syntax:

```
<DATE data type>_TO_<data type>
```

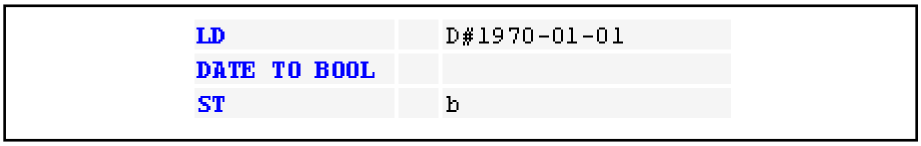
The date is internally saved in seconds starting on January 1st 1970. This value is converted.

For STRING types, the result is the date constant.

☞ When converting larger types to smaller types, information can be lost.

Example:

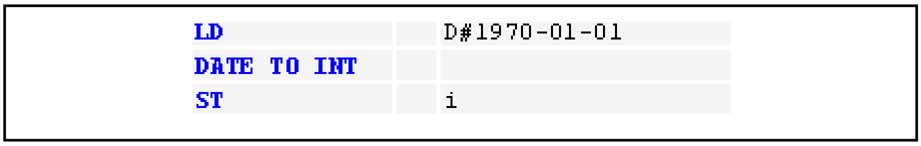
Operator DATE TO in IL



```
LD              D#1970-01-01
DATE TO BOOL
ST              b
```

Fig.5-110: Operator DATE TO BOOL

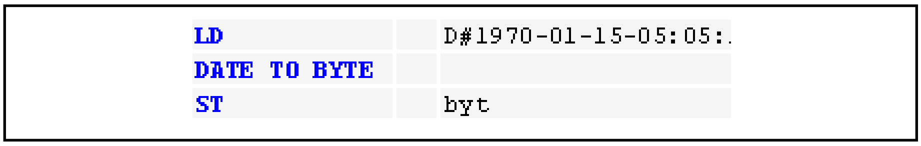
Result is FALSE



```
LD              D#1970-01-01
DATE TO INT
ST              i
```

Fig.5-111: Operator DATE TO INT

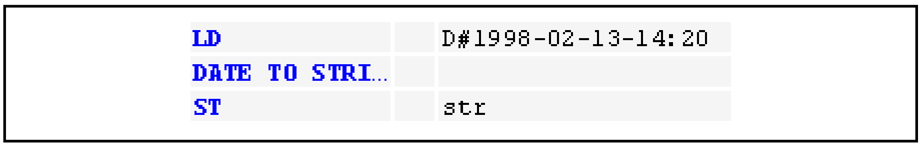
Result is 29952



```
LD              D#1970-01-15-05:05:.
DATE TO BYTE
ST              byt
```

Fig.5-112: Operator DATE TO BYTE

Result is 129



```
LD              D#1998-02-13-14:20
DATE TO STRI...
ST              str
```

Fig.5-113: Operator DATE TO STRING

Result is DT#1998-02-13-14:20

Operator DATE TO in ST:

```
(*Operator DATE_TO_BOOL:*)
b:=DATE_TO_BOOL(D#1970-01-01); (*result is FALSE*)

(*Operator DATE_TO_INT:*)
i :=DATE_TO_INT(D#1970-01-15); (*result is 29952*)

(*Operator DATE_TO_BYTE:*)
byt:=DT_TO_BYTE(DT#1970-01-15-05:05:05); (*result is 129*)

(*Operator DATE_TO_STRING:*)
str:=DT_TO_STRING(DT#1998-02-13-14:20); (*result is )
                                          'DT#1998-02-13-14:20'*)
```

Example:

Operator DATE TO in FBD

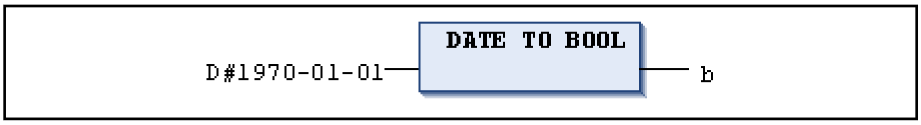


Fig.5-114: Operator DATE TO BOOL

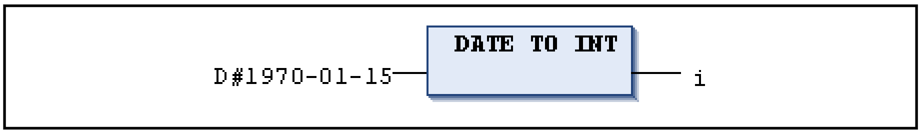


Fig.5-115: Operator DATE TO INT

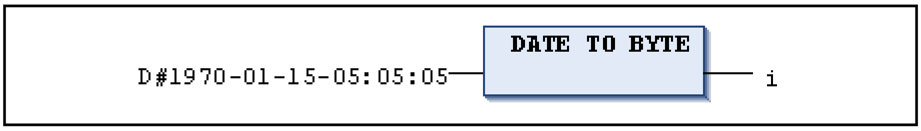


Fig.5-116: Operator DATE TO BYTE

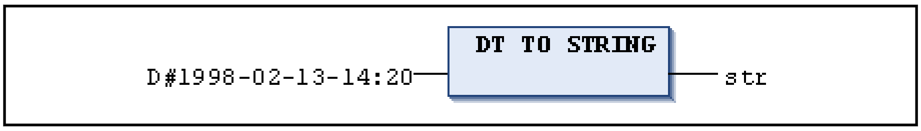


Fig.5-117: Operator DT TO STRING:

STRING_TO Conversions

IEC operator: Conversion from STRING type to another type.

The conversion runs according to standard C: STRING to INT and then INT to BYTE. The higher byte is cut, i.e. only values from 0 - 255 result.

Syntax:

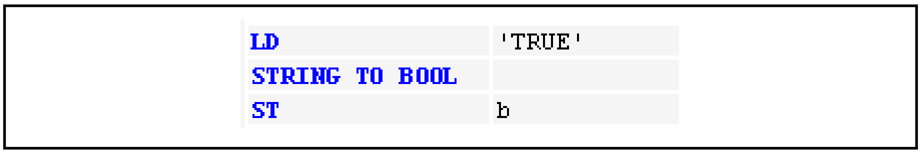
```
STRING_TO_<data type>
```

The operand of type STRING has to have a valid value of the target type. Otherwise, the result is 0.

☞ When converting larger types to smaller types, information can be lost.

Example:

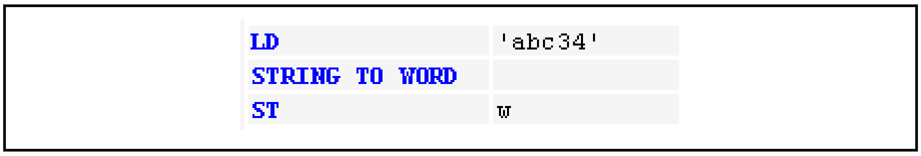
Operator STRING_TO in IL



```
LD              'TRUE'
STRING TO BOOL
ST              b
```

Fig.5-118: Operator STRING TO BOOL

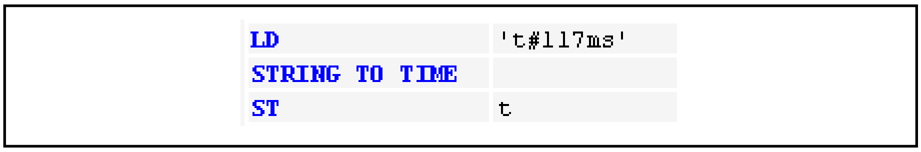
Result is TRUE



```
LD              'abc34'
STRING TO WORD
ST              w
```

Fig.5-119: Operator STRING TO WORD

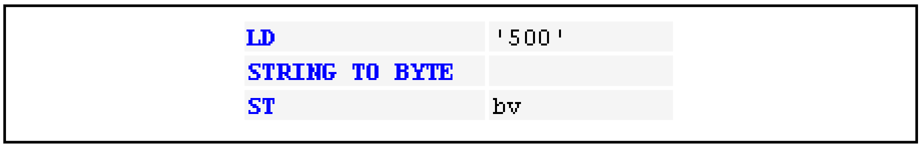
Result is 0



```
LD               't#117ms'
STRING TO TIME
ST               t
```

Fig.5-120: Operator STRING TO TIME

Result is T#117ms



```
LD               '500'
STRING TO BYTE
ST               bv
```

Fig.5-121: Operator STRING TO BYTE

Result is 244

Operator STRING_TO in ST

```
(*Operator STRING_TO_BOOL:*)
b:=STRING_TO_BOOL('TRUE'); (*result is TRUE*)

(*Operator STRING_TO_WORD:*)
w:=STRING_TO_WORD('abc34'); (*result is 0*)

(*Operator STRING_TO_TIME:*)
t:=STRING_TO_TIME('T#127ms'); (*result is T#127ms*)

(*Operator STRING_TO_BYTE:*)
bv:=STRING:TO_BYTE('500'); (*result is 244*)
```

Example:

Operator STRING TO in FBD

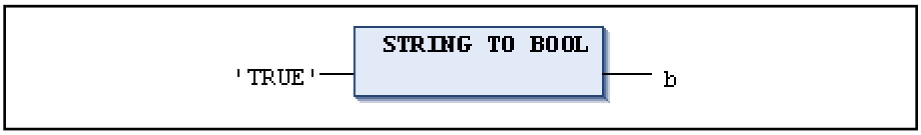


Fig.5-122: Operator STRING TO BOOL

Result is TRUE

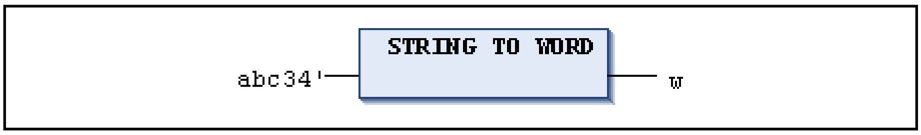


Fig.5-123: Operator STRING TO WORD

Result is 0

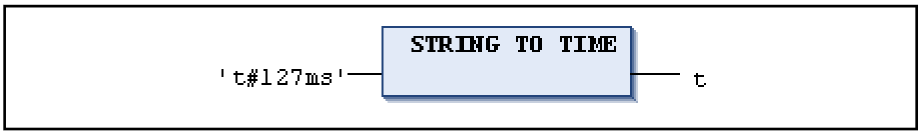


Fig.5-124: Operator STRING TO TIME

Result is T#127ms

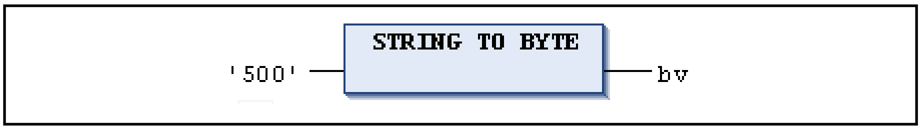


Fig.5-125: Operator STRING TO BYTE

Result is 244

TRUNC

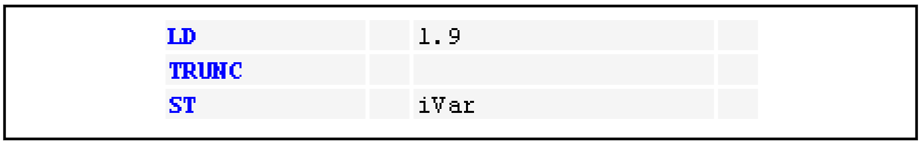
Conversion from REAL type to **DINT** type. Only the **value of the integer part of the number** is used.

In IndraLogic 1.x, the TRUNC operator converts from REAL to INT.

If a 1.x project is imported, TRUNC is automatically replaced by TRUNC_INT.

Operator TRUNC in ST

```
i:=TRUNC(1.9); (*result is 1*)
j:=TRUNC(-1.4); (*result is -1*)
```



```
LD      1.9
TRUNC
ST      iVar
```

Fig.5-126: Operator TRUNC in IL

TRUNC_INT

Conversion from REAL type to **INT** type. Only the value of the integer part of the number is used.

Note: TRUNC_INT corresponds to the TRUNC operator in IndraLogic 1.x and when 1.x projects are imported, it is automatically used at its place. Note the different function of TRUNC in IndraLogic 2G.

Operator TRUNC_INT in IL

```
LD                  1.9
TRUNC_INT
ST                  iVar
```

Operator TRUNC_INT in ST

```
i:=TRUNC_INT(1.9);  (*result is 1*)
i:=TRUNC_INT(-1.4); (*result is -1*)
```

ANY_TO Conversions

Conversion of any data type ANY or a specific numeric data type ANY_NUM into another type. Reasonable selection of data types is required.

Syntax:

```
ANY_NUM_TO_<numeric data type>
ANY_TO_<any data type>
```

Operator ANY_..._TO in ST

```
VAR
  re: REAL := 1.234;
  i:  INT;
END_VAR
  i:= ANY_TO_INT(re)// REAL to INT
```

5.3.10 Numeric Operators

Numeric Operators, Overview

The following numeric operators described in the IEC 61131-3 standard are supported:

- ABS, page 602
- SQRT, page 603
- LN, page 603
- LOG, page 604
- EXP, page 604
- SIN, page 605
- COS, page 605
- TAN, page 606
- ASIN, page 606
- ACOS, page 607
- ATAN, page 607
- EXPT, page 608

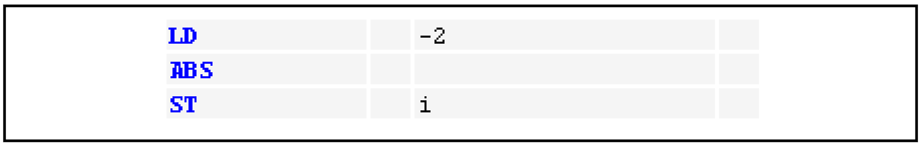
ABS

NumericIEC operator: Returns the absolute value of a number.

The following type combinations for IN and OUT are possible:

IN	OUT
SINT	SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LWORD
INT	INT, DINT, LINT, UINT, UDINT, ULINT, REAL, LREAL, WORD, DWORD, LWORD
DINT	DINT, LINT, UDINT, ULINT, REAL, LREAL, DWORD, LWORD
LINT	LINT, ULINT, REAL, LREAL, LWORD
USINT	USINT, UINT, UDINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LWORD
UINT	UINT, UDINT, ULINT, REAL, LREAL, WORD, DWORD, LWORD
UDINT	UDINT, ULINT, REAL, LREAL, DWORD, LWORD
ULINT	ULINT, REAL, LREAL, LWORD
REAL	REAL, LREAL
LREAL	LREAL
BYTE	USINT, UINT, UDINT, ULINT, REAL, LREAL, BYTE, WORD, DWORD, LWORD
WORD	UINT, UDINT, ULINT, REAL, LREAL, WORD, DWORD, LWORD

DWORD	UDINT, ULINT, LREAL, DWORD, LWORD
LWORD	ULINT, LREAL, LWORD



```
LD      -2
ABS
ST      i
```

Fig.5-127: Operator ABS in IL

Result in i is 2

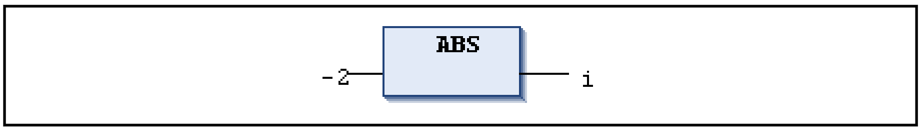


Fig.5-128: Operator ABS in FBD

Operator ABS in ST

```
i:=ABS(-2);
```

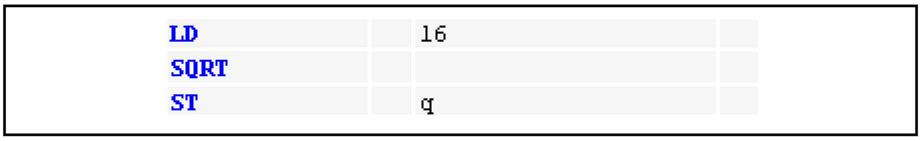
SQRT

NumericIEC operator: Returns the square root of a number.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      16
SQRT
ST      q
```

Fig.5-129: Operator SQRT in IL

Result in q is 4

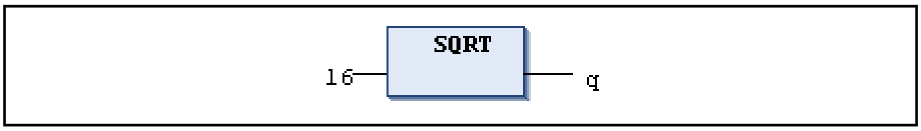


Fig.5-130: Operator SQRT in FBD

Operator SQRT in ST

```
q:=SQRT(16);
```

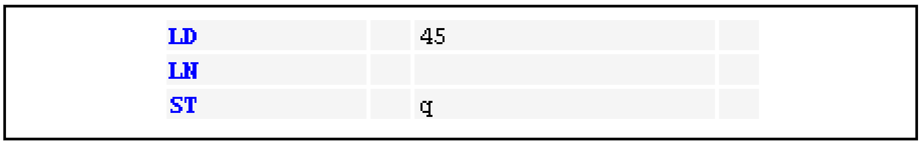
LN

NumericIEC operator: Returns the natural logarithm of a number.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD    45
LN
ST    q
```

Fig.5-131: Operator LN in IL

Result in q is 3.80666

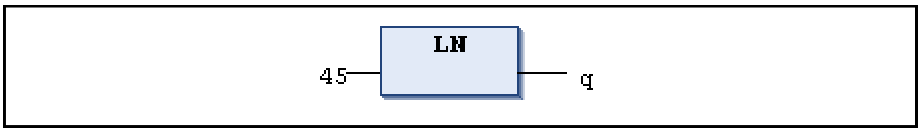


Fig.5-132: Operator LN in FBD

Operator LN in ST:

```
q:=LN(45);
```

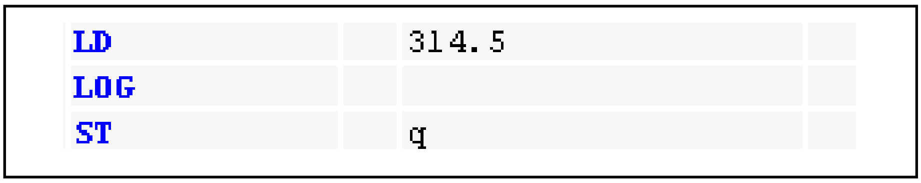
LOG

NumericIEC operator: Returns the logarithm to the basis of 10 to a number.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD    314.5
LOG
ST    q
```

Fig.5-133: Operator LOG in IL

Result in q is 2.49762

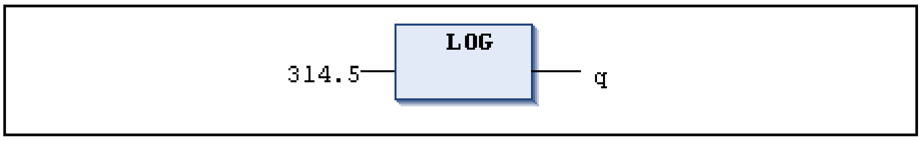


Fig.5-134: Operator LOG in FBD

Operator LOG in ST

```
q:=LOG(314.5);
```

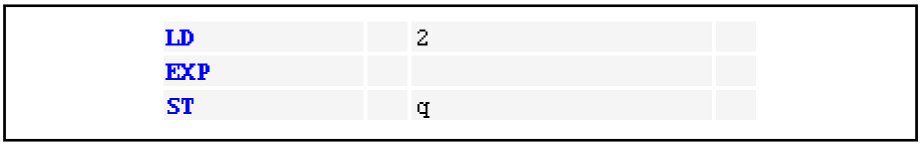
EXP

NumericIEC operator: Returns the exponential function.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD    2
EXP
ST    q
```

Fig.5-135: Operator EXP in IL

Result in q is 7.389056099

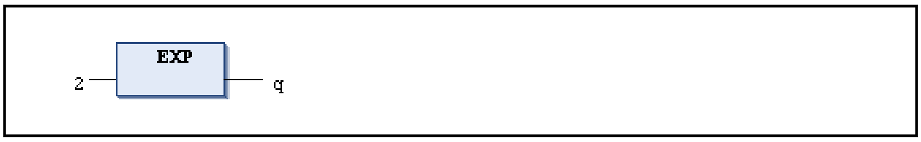


Fig.5-136: Operator EXP in FBD

Operator EXP in ST

```
q:=EXP(2);
```

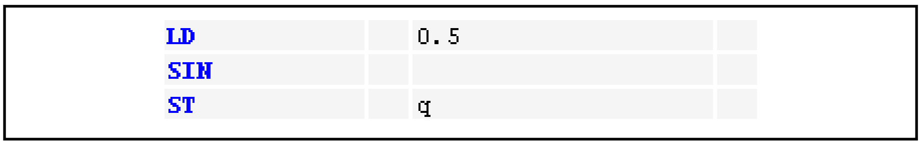
SIN

NumericIEC operator: Returns the sine value of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD    0.5
SIN
ST    q
```

Fig.5-137: Operator SIN in IL

Result in q is 0.479426

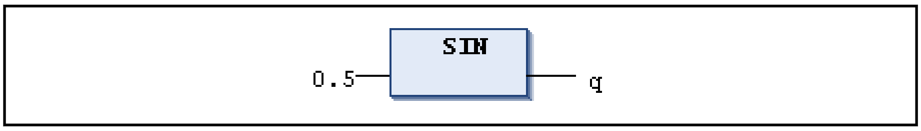


Fig.5-138: Operator SIN in FBD

Operator SIN in ST

```
q:=SIN(0.5);
```

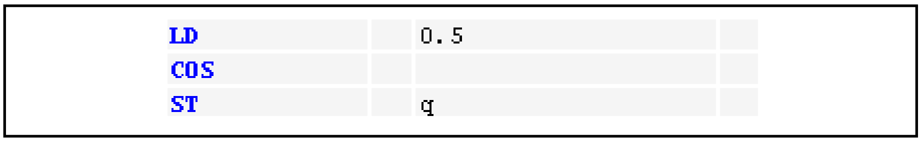
COS

NumericIEC operator: Returns the cosine value of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      0.5
COS
ST      q
```

Fig.5-139: Operator COS in IL

Result in q is 0.877583

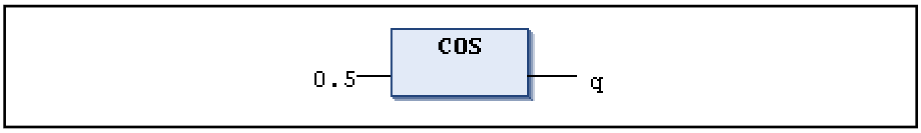


Fig.5-140: Operator COS in FBD

Operator COS in ST

```
q:=COS(0.5);
```

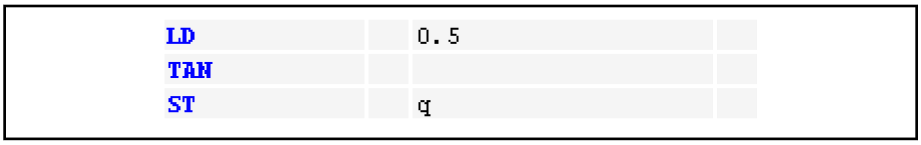
TAN

NumericIEC operator: Returns the tangent value of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      0.5
TAN
ST      q
```

Fig.5-141: Operator TAN in IL

Result in q is 0.546302

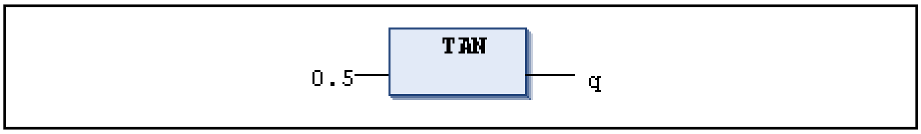


Fig.5-142: Operator TAN in FBD

Operator TAN in ST

```
q:=TAN(0.5);
```

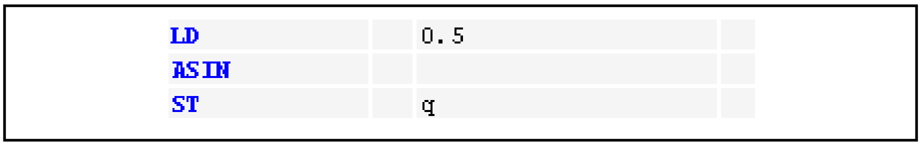
ASIN

NumericIEC operator: Returns the arc sine value (inverse function of sine) of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      0.5
ASIN
ST      q
```

Fig.5-143: Operator ASIN in IL

Result in q is 0.523599

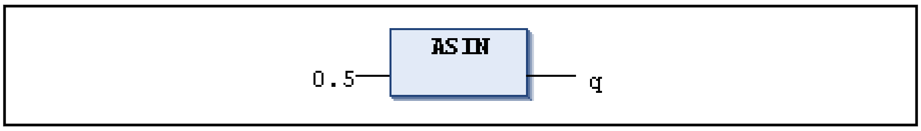


Fig.5-144: Operator ASIN in FBD

Operator ASIN in ST

```
q:=ASIN(0.5);
```

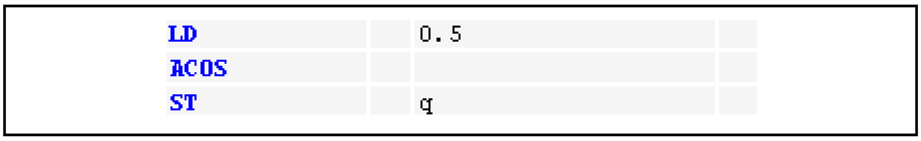
ACOS

NumericIEC operator: Returns the arc cosine value (inverse function of cosine) of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      0.5
ACOS
ST      q
```

Fig.5-145: Operator ACOS in IL

Result in q is 1.0472

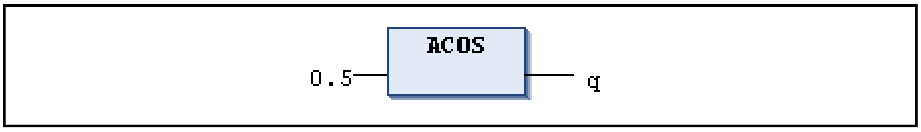


Fig.5-146: Operator ACOS in FBD

Operator ACOS in ST

```
q:=ACOS(0.5);
```

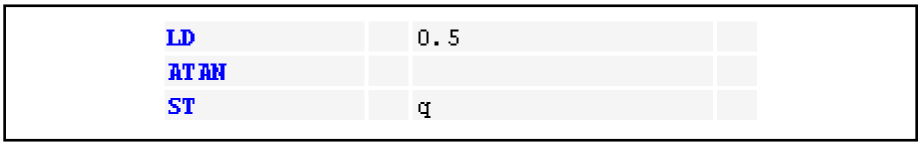
ATAN

NumericIEC operator: Returns the arc tangent value (inverse function of tangent) of a number. The value is calculated in radians.

IN can be one of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      0.5
ATAN
ST      q
```

Fig.5-147: Operator ATAN in IL

Result in q is 0.463648

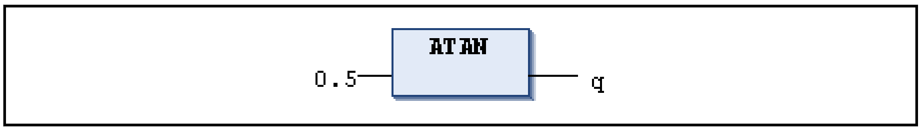


Fig.5-148: Operator ATAN in FBD

Operator ATAN in ST

```
q:=ATAN(0.5);
```

EXPT

NumericIEC operator: Exponentiation of one variable with another.

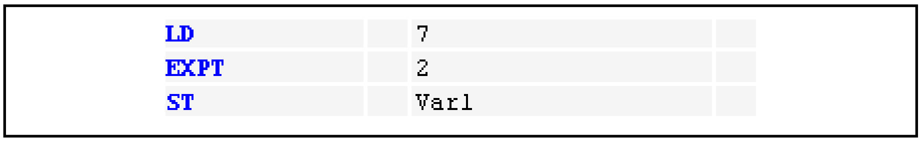
$$OUT := IN1^{IN2}$$

Fig.5-149: Exponentiation of the variable IN1 with IN2

IN1 and IN2 can be any of the following types:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL, LREAL.

OUT has to be of type REAL or LREAL.



```
LD      7
EXPT    2
ST      Var1
```

Fig.5-150: Operator EXPT in IL

Result is 49

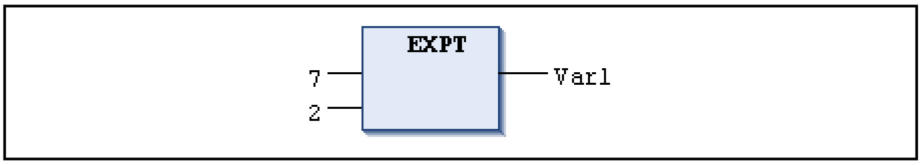


Fig.5-151: Operator EXPT in FBD

Operator EXPT in ST

```
var1 := EXPT(7,2);
```

5.3.11 Namespace Operators

Namespace Operators, Overview

An extension of the IEC 61131-3 standard operators, there are a few ways to design unique access to variables or modules even if the same variable or module name is used several times in the project.

The following namespace operators can be used to define the respectively valid namespace:

- "." (Global namespace operator), page 609.
- "<Name of a global variable list>.", page 609.
- "<Name of a library>.", page 609.
- "<Name of an enumeration>.", page 610.

Global Namespace Operators

Namespace operator: Extension of the EN 61131-3 standard.

An instance path that begins with a dot "." always opens a global namespace.

So if there is a local variable with the same name as a global variable, "<varname>", ".<varname>" is used to address the global variable.

Namespace for Global Variable Lists

Namespace operator: Extension of the EN 61131-3 standard.

The name of a global variable list (GVL) can be used as a namespace identifier for the variables defined in the list. Thus, variables with the same name in different global variable lists can be used which still have unique access to a specific variable by attaching the name of the list to the front of the variable name separated by a dot "."

Syntax:

```
<global variable list name>.<variable>
```

Example:

The global variable lists **globlist1** and **globlist2** contain a variable **varx** each.

In the following line of code, **varx** is copied from the list **globlist2** to **varx** in the list **globlist1**:

Example:

```
globlist1.varx := globlist2.varx;
```

☞ If a variable that is declared in several global variable lists is referenced without using the list name prefix, an error message is output.

Library Namespace

Namespace operator: Extension of the EN 61131-3 standard.

The library name can be used in order to uniquely address a library function block.

Syntax:

```
<namespace>.<module name>
```

Example:

If a library integrated into a project contains a function block "fun1" and there is also a local function block "fun1" in the project, the library name can be attached to the front of the function block name separated by a dot "." in order to create unique access.

e.g. "lib1.fun1".

By default, the "namespace" of a library has the same name as the library. However, a different identifier can be used for it.

This can already be entered in the project information when the library project, page 191, is created or later on in the Properties dialog, page 368, of an integrated library.

Example:

There is a function **fun1** in the library **lib** and a function **fun1** declared locally in the project. By default, the namespace of the library is called "lib":

Example:

```
res1 := fun(in := 12); // call of the project function fun
res2 := lib.fun(in := 12); // call of the library function fun
```

Enumeration Namespace

Namespace operator: Extension of the EN 61131-3 standard.

The TYPE name of an enumeration, page 564, can be used for a unique access to an enumeration constant.

This way, constants with the same name can be used in several enumerations.

The enumeration name is attached to the front of the constant name separated by a dot ".".

Syntax:

```
<enumeration name>.<constant name>
```

Example:

The constant **Blue** is a component of the enumeration **Colors** and the enumeration **Feelings.**

Example:

```
    // Access to enum value Blue
color := Colors.Blue;    // in type Colors
feeling := Feelings.Blue;// in type Feelings
```

5.3.12 IEC-Extending Operands

IEC-Extending Operands, Overview

In addition to the operators described in the IEC standard, IndraLogic supports the following operators:

- __DELETE, page 610, releases the memory of instances generated with _NEW dynamic.
- __ISVALIDREF, page 611, checks whether a reference refers to a valid value.
- __NEW, page 612 allocates memory for instances of function blocks or arrays of standard data types.
- __QUERYINTERFACE, page 614, enables the type conversion of one interface reference to another at runtime.
- __QUERYPOINTER, page 615 enables the type conversion of an interface reference of a function block to a pointer at runtime.

__DELETE

This operator is not described in the IEC 61131-3 standard.

Due to compatibility reasons, the compiler version has to be >= 3.3.2.0.

The operator __DELETE releases the memory of instances generated with _NEW dynamic.

__DELETE has no return value and after this operation, the operand is reset to 0.

Memory is reserved with __NEW.

Syntax:

```
__DELETE (<Pointer>)
```

If <Pointer> points to a function block, the associated method FB_EXIT is called before the pointer is set to 0.

Example with function block:

```
FUNCTION_BLOCK FBDynamic
VAR_INPUT
   in1, in2 : INT;
END_VAR
VAR_OUTPUT
   out : INT;
END_VAR
VAR
   test1 : INT := 1234;
   _inc : INT := 0;
   _dut : POINTER TO DUT;
   neu : BOOL;
END_VAR
out := in1 + in2;

METHOD FB_Exit : BOOL
VAR_INPUT
   bInCopyCode : BOOL;
END_VAR
__Delete(_dut);

METHOD FB_Init : BOOL
VAR_INPUT
   bInitRetains : BOOL;
   bInCopyCode : BOOL;
END_VAR
_dut := __NEW(DUT);

METHOD INC : INT
VAR_INPUT
END_VAR
_inc := _inc + 1;
INC := _inc;
// ------------------------------------
PLC_PRG(PRG)
VAR
   pFB : POINTER TO FBDynamic;
   bInit: BOOL := TRUE;
   bDelete: BOOL;
   loc : INT;
END_VAR
IF (bInit) THEN
  pFB := __NEW(FBDynamic);
  bInit := FALSE;
END_IF
IF (pFB <> 0) THEN
  pFB^(in1 := 1, in2 := loc, out => loc);
  pFB^.INC();
END_IF
IF (bDelete) THEN
  __DELETE(pFB);
END_IF
```

__ISVALIDREF

This operator is not described in the IEC 61131-3 standard.

The operator "__ISVALIDREF" can be used to check if a reference, page 556, refers to a valid value, i.e. a value not equal to 0.

Syntax:

```
<boolean variable>:= __ISVALIDREF
 (with REFERENCE TO <data_type>);
```

<Boolean Variable> becomes TRUE if the reference points to a valid value. Otherwise, it becomes FALSE.

Declaration:

```
 ivar : INT;
 ref_int : REFERENCE TO INT;
 ref_int0: REFERENCE TO INT;
 testref: BOOL := FALSE;
 testref0: BOOL := FALSE;
```

Implementation:

```
 ivar := ivar +1;
 ref_int REF= ivar;
 ref_int0 REF= 0;
 testref := __ISVALIDREF(ref_int);
  // TRUE, because ref_int points to ivar, with value <> 0
 testref0 := __ISVALIDREF(ref_int0);
  // FALSE, because ref_int0 is set to 0
```

__NEW

This operator is not described in the IEC 61131-3 standard.

☞ Due to compatibility reasons, the compiler version has to be >= 3.3.2.0.

__NEW, allocates memory for instances of function blocks or arrays of standard data types. The operator returns a pointer to the correctly typed object which is not equal to 0. If the operator is not used in any assignment, an error message is output.

If the attempt to reserve memory fails, __NEW returns 0.

__DELETE releases the memory again.

Syntax:

```
__NEW (<Type>, [<Size>])
```

The operator generates an instance on the specified type <Type> and returns a pointer to this instance. Then, the initialization of the instance is called.

If <Type> is a scalar type, the optional operand <Size> is set. Then, the operator generates an array of type <Type> and of size <Size>.

Example:

```
pScalarType := __New(ScalarType, length);
```

☞ For dynamically generated instances, copy code methods are not possible in online mode!

Due to this reason, function blocks from libraries that cannot be changed and function blocks with the attribute "enable_dynamic_creation" can be applied to the __NEW operator. If a function block with this identifier is to be changed so that copy code is required, an error message is output.

☞ The code for the memory allocation has to be able to be introduced again.

A semaphore (SysSemEnter) can be used to prevent two tasks from attempting to allocate memory simultaneously. As a result, comprehensive use of __NEW causes greater jitter.

Example with a scalar type:

```
TYPE DUT :
STRUCT
a,b,c,d,e,f : INT;
END_STRUCT
END_TYPE
// ------------------------------------
PLC_PRG (PRG)
VAR
  pDut : POINTER TO DUT;
  bInit: BOOL := TRUE;
  bDelete: BOOL;
END_VAR
IF (bInit) THEN
   pDut := __NEW(DUT);
   bInit := FALSE;
END_IF
IF (bDelete) THEN
   __DELETE(pDut);
END_IF
```

Example with a function block:

```
FBDynamic(FP)
{attribute 'enable_dynamic_creation'}
FUNCTION_BLOCK FBDynamic
VAR_INPUT
   in1, in2 : INT;
END_VAR
VAR_OUTPUT
   out : INT;
END_VAR
VAR
   test1 : INT := 1234;
   _inc : INT := 0;
   _dut : POINTER TO DUT;
   neu : BOOL;
END_VAR
out := in1 + in2;
// ------------------------------------
PLC_PRG(PRG)
VAR
   pFB : POINTER TO FBDynamic;
   loc : INT;
   bInit: BOOL := TRUE;
   bDelete: BOOL;
END_VAR
IF (pFB <> 0) THEN
   pFB^(in1 := 1, in2 := loc, out => loc);
   pFB^.INC();
END_IF
IF (bDelete) THEN
   __DELETE(pFB);
END_IF
```

Example with an array:

```
PLC_PRG(PRG)
VAR
  bInit: BOOL := TRUE;
  bDelete: BOOL;
  pArrayBytes : POINTER TO BYTE;
  pArrayDuts : POINTER TO BYTE;
  test: INT;
  parr : POINTER TO BYTE;
END_VAR
IF (bInit) THEN
```

```
   pArrayBytes := __NEW(BYTE, 25);
   bInit := FALSE;
END_IF
IF (pArrayBytes <> 0) THEN
   pArrayBytes[24] := 125;
   test := pArrayBytes[24];
END_IF
IF (bDelete) THEN
   __DELETE(pArrayBytes);
END_IF
```

__QUERYINTERFACE

This operator is not described in the IEC 61131-3 standard.

__QUERYINTERFACE, enables type conversion of one interface reference to another at runtime. The operator returns a result of type BOOL. TRUE means that the conversion was performed successfully.

☞ Due to compatibility reasons, the definition of the reference to be converted has to be an extension of the basic interface __SYSTEM.IQueryInterface and the compiler version has to be >= 3.3.0.10.

Syntax:

```
 __QUERYINTERFACE(<ITF_Source>, < ITF_Dest>)
```

As first operand, this operator receives an interface reference or a function blocks instance. As second operand, it receives an interface reference with the desired target types. After __QUERYINTERFACE is processed, ITF_Dest contains a reference on the desired interface if the object referenced by ITF_Source implements the interface. Then, the conversion was successful and the result of the operator returns TRUE. In all other cases, FALSE is returned.

The prerequisite for the explicit conversion is that ITF_Source and ITF_Dest are derived from Interface __System.IQueryInterface. This interface is implicitly available and does not require a library.

Example:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived1 : BOOL
END_METHOD

INTERFACE ItfDerived2 EXTENDS ItfBase
METHOD mderived2 : BOOL
END_METHOD
// ------------------------------------
 POU (PRG)
VAR
   itfderived1 : ItfDerived1;
   itfderived2 : ItfDerived2;
   bTest1, bTest2, xResult1, xResult2: BOOL;
END_VAR
xResult1 := __QUERYINTERFACE(itfbase, itfderived1);
      // variables of type ItfBase resp. ItfDerived1
xResult2 := __QUERYINTERFACE(itfbase, itfderived2);
      // variables of type ItfBase resp. ItfDerived2
IF (xResult1 = TRUE) THEN
   bTest1 := itfderived1.mderived1();
ELSIF xResult2 THEN
   bTest2 := itfderived2.mderived2();
END_IF
```

__QUERYPOINTER

This operator is not described in the IEC 61131-3 standard.

__QUERYPOINTER enables the type conversion of an interface reference of a function block to a pointer at runtime. The operator returns a result of type BOOL. TRUE means that the conversion was performed successfully.

☞ Due to compatibility reasons, the definition of the pointer to be converted has to be an extension of the basic interface __SYSTEM.IQueryInterface and the compiler version has to be >= 3.3.0.10.

Syntax:

```
__QUERYPOINTER (<ITF_Source>, < Pointer_Dest>)
```

As first operand, this operator receives an interface reference or a function block instance with the desired target types. As second operand, it receives a pointer. After __QUERYPOINTER has been processed, Pointer_Dest contains the pointer to the reference or instance of a function block to which the interface ITF_Source currently references. Pointer_Dest is not typed and can be cast to any desired type. The type has to be ensured by the programmer. For example, the interface could offer a method that returns a type code.

The prerequisite for the explicit conversion is that the ITF_Source is derived from Interface __System.IQueryInterface. This interface is implicitly available and does not require a library.

Example:

```
INTERFACE ItfBase EXTENDS __System.IQueryInterface
METHOD mbase : BOOL
END_METHOD

INTERFACE ItfDerived1 EXTENDS ItfBase
METHOD mderived : BOOL
END_METHOD
// ------------------------------------

FUNCTION_BLOCK FBVariante IMPLEMENTS ITFDerived
// ------------------------------------

PROGRAMM POU
VAR
  itfderived : ItfDerived;
  insV : FBVariante;
  xResult, xTest : BOOL;
  pVar: POINTER TO DWORD;
END_VAR
itfderived := insV;
xResult := __QUERYPOINTER(itfderived, pVar);
IF xResult THEN
     xTest := pVar.mderived();
END_IF
```

5.4 Operands

5.4.1 Operands, General Information

The following can be used in IndraLogic 2G as operands:

- *Constants*
  – BOOL, page 616
  – TIME, page 616
  – DATE, page 617

  - TIME_OF_DAY, page 617
  - DATE_AND_TIME, page 618
  - Number, page 618
  - REAL/LREAL, page 619
  - STRING, page 619
  - WSTRING, page 619
  - Typed constants, page 620
- Variables, page 621
- Addresses, page 623
- Functions, page 624

5.4.2 Constants

BOOL Constants

BOOL constants are the truth values TRUE and FALSE.

Also refer to

- Data type BOOL, page 553.

TIME Constants

TIME constants are especially used to operate the default timer module.

In addition to the time constant TIME with a size of 32 bits and which meets the IEC 61131-3 standard, LTIME is also supported as time basis for high resolution timers in the extension of the standard.

LTIME has a size of 64 bits and a resolution within nanoseconds range.

Syntax for a time constant:

```
t#<time declaration>
T#<time declaration>
time#<time declaration>
TIME#<time declaration>

LTIME#<time declaration>
```

Instead of "t#", the following spellings are also valid: "T#", "time#", "TIME#".

The time declaration can include the following time units. They have to be arranged in the following sequence, but it is not required to enter all units.

"d": Days

"h": Hours

"m": Minutes

"s": Seconds

"ms": Milliseconds

"us" : Microseconds (only for LTIME#<time declaration>)

"ns" : Nanoseconds (only for LTIME#<time declaration>)

Examples of correct time constants in an ST assignment:

TIME1 := T#14ms;	
TIME1 := T#100S12ms;	// Overflow is permitted in the highest component
TIME1 := t#12h34m15s;	

Examples of incorrect usage:

TIME1 := t#5m68s;	// Overflow at a low position
TIME1 := 15ms;	// T# missing
TIME1 := t#4ms13d;	// Incorrect time specification sequence

Also refer to

- Time data types, page 555.

DATE Constants

These constants can be used in order to enter date information.

Syntax:

```
d#<date declaration>
D#<date declaration>
date#<date declaration>
DATE#<date declaration>
```

Instead of "d#", the following spelling is also valid: "D#", "date", "DATE".

The date declaration has to be specified in the "<year-month-day>" format.

DATE (abbreviated D) values are internally treated as DWORD. The time is given in seconds and the calculations begin on January 1st 1970 at 00:00.

Examples

```
DATE#2011-05-06
d#2010-03-29
```

Also refer to

- Time data types, page 555.

TIME_OF_DAY Constants

Times can be specified with these kinds of constants.

Syntax:

```
tod#<time_of_day declaration>
TOD#<time_of_day declaration>
time_of_day#<time_of_day declaration>
TIME_OF_DAY#<time_of_day declaration>
```

Instead of "tod#", the following spelling is also valid: "TOD#", "time_of_day#", "TIME_OF_DAY#".

The time declaration has to be entered in the format "<hour:minute:second>".

Seconds can be specified as real numbers and fractions of seconds can also be specified.

TIME_OF_DAY (abbreviated TOD) values are internally treated as DWORD. The time is given in milliseconds and the calculations start at 00:00 o'clock.

Examples:

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

Also refer to

- Time data types, page 555.

DATE_AND_TIME Constants

DATE constants and TIME_OF_DAY constants can be combined to so-called DATE_AND_TIME constants.

Syntax:

```
dt#<Date_and_time declaration>
DT#<Date_and_time declaration>
date_and_time#<Date_and_time declaration>
DATE_AND_TIME#<Date_and_time declaration>
```

Instead of "dt#", the following spelling is also valid: "DT#", "date_and_time#", "DATE_AND_TIME#".

The date and time declaration have to be entered in the format "<year-month-day-hour:minute:second>".

Seconds can be specified as real numbers and fractions of seconds can also be specified.

DATE_AND_TIME (abbreviated DT) values are treated as DWORD internally. The time is given in seconds and the calculations begin on January 1st 1970 at 00:00.

Examples:

```
DATE_AND_TIME#2011-05-06-15:36:30
dt#2010-03-29-00:00:00
```

Also refer to

- Time data types, page 555.

Number Constants

Number values can appear as binary numbers, octal numbers, decimal numbers and hexadecimal numbers.

If an integer value is not a decimal number, its base has to be written in front of the integer constant followed by a hash (#).

In hexadecimal numbers, the numeric values for the numbers 10 to 15 are usually specified by the letters A-F.

Underscores are allowed within a number value.

14	(Decimal number)
2#1001_0011	(Binary number)
8#67	(Octal number)
16#A	(Hexadecimal number)

The type of these number values can be one of the following:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL or LREAL.

☞ Implicit conversions of larger types to smaller types are not allowed, i.e. a DINT variable cannot be used as an INT variable. Use type conversions.

💡 For bit character strings, displaying hexadecimal numbers has to begin with leading zeroes.

BYTE: 16#00 ... 16#FF

WORD: 16#0000 ... 16#FFFF etc.

REAL/LREAL Constants

REAL and LREAL constants can be specified as decimal fractions and with exponents.

To do this, use the American format with a dot.

7.4	instead of 7,4
1.64e+009	instead 1.64e^{+009}

STRING Constants

A STRING, page 555, is any character string in ASCII code (8 bit).

STRING constants are enclosed by simple apostrophes before and after.

Spaces and umlauts can also be entered. They are treated as any other characters.

In character strings, the combination of the dollar sign ($) followed by two hexadecimal numbers is interpreted as the hexadecimal display of the eight bit character code. In addition, combinations of characters that start with a dollar sign are interpreted as follows:

Specified combination	Interpretation
'$<two hex. numbers>'	Hexadecimal display of the eight bit character code
' '	Space
'$$'	Dollar sign
'$''	Simple apostrophe
'$L' or '$l'	Line feed
'$N' or '$n'	New line
'$P' or '$p'	Page feed
'$R' or '$r'	Line break
'$T' or '$t'	Tab

Example:

Valid character strings:

'w1Wüß?'

'Abby and Craig'

':-)'

'costs ($$)'

WSTRING Constants

A WSTRING, page 555, is any character string in Unicode.

WSTRING constants are enclosed by quotation marks before and after.

Spaces and umlauts can also be entered. They are treated as any other characters.

In character strings, the combination of the dollar sign ($) followed by four hexadecimal numbers is interpreted as the hexadecimal display of the 16 bit character code. In addition, combinations of characters that start with a dollar sign are interpreted as follows:

Specified combination	Interpretation
'$<four hex. numbers>"	Hexadecimal display of the 16 bit character code
" "	Space
"$$"	Dollar sign
"$""	Simple quotation marks
"$L" or "$l"	Line feed
"$N" or "$n"	New line
"$P" or "$p"	Page feed
"$R" or "$r"	Line break
"$T" or "$t"	Tab

Example:

Valid character strings:

"w1Wüß?"

"Abby and Craig"

":-)"

"costs ($$)"

Typed Constants (Typed Literals)

Except REAL/LREAL constants (LREAL is always used here), the smallest possible data type is used when calculating with IEC constants. If a different data type is to be used, typed literals (typed constants) can be used without having to declare the constant explicitly. The constants are provided with a prefix that specifies the type:

Syntax:

```
<Type>#<Literal>
```

<Type> indicates the desired data type. Possible specifications are:

- BYTE, WORD, DWORD, LWORD,
- SINT, USINT, INT, UINT, DINT, UDINT, LINT, ULINT,
- REAL or LREAL.

The type has to be written in upper case letters.

<Literal> indicates the constant. The input has to match the data types specified under <Type>.

Example:

```
var1:=DINT#34;
```

If the constants cannot be transferred to the target type without loss of data, an error message is output.

Typed constants can be used anywhere normal constants can be used.

5.4.3 Variables

Variables, General Information

Variables are either declared locally in the declaration part of a function block or in the global variable lists.

The available variables can be called using the input assistance, page 98,.

Further information

- on the declaration of a variable, page 503, and
- on the multiple use of variable identifiers, page 608

Access to Variables of Arrays, Structures and Function Blocks

The following access methods are possible:

Two-dimensional array components:

```
<ArrayName>[Index1, Index2]
```

Structure variables:

```
<StructureName>.<VariableName>
```

Function block and program variables:

```
<FunctionBlockInstanceName>.<VariableName>
```

Further information

- on arrays, page 560,
- on structures, page 563,
- on function blocks, page 33 and programs, page 29.

Bit Access to Variables

In integer variables, individual bits can be addressed. To do this, the index of the bit to be addressed is attached to the variable separated by a dot. The bit index can be indicated by any desired constant. The indexing is based on 0.

☞ Bit accesses used in visualizations that are executed using a data server connection, page 318, only work if they contain literal bit index information (i.e. no defined constants).

Syntax:

```
<Variablename>.<Bitindex>
```

Example:

```
VAR
 a : INT;
 b : BOOL;
END_VAR

a.2 := b;
```

The **third** bit of the variable a is set to the value of variable b.

If the index is greater than the bit width of the variable, the following error is output:

Index <n> outside of the valid range for variable <var>!

Bit addressing is possible for the following variable types: SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT, BYTE, WORD, DWORD, LWORD.

If the type of variable is not permitted, the following error message is output:

Illegal data type <Type> for direct indexing.

A bit access may not be assigned to a VAR_IN_OUT variable!

Bit access with a global constant:

If a global constant is declared that defines the bit number, this constant can be used for the bit access.

The variable **enable** defines which bit is to be accessed:

Declaration in a global variable list for both examples:

```
VAR_GLOBAL CONSTANT
  enable:INT:=2;
END_VAR
```

Example 1, bit access to an integer variable:

```
// Function block declaration
VAR
    xxx:INT;
END_VAR

// Bit access
xxx.enable:=true;
```

This sets the third bit in the variable xxx to TRUE.

Example 2, bit access to integer structure components:

```
// DUT declaration of stru1
TYPE stru1 :
    STRUCT
        bvar:BOOL;
        rvar:REAL;
        wvar:WORD;
        {bitaccess enable 42 'Start drive'}
    END_STRUCT
END_TYPE
// FB declaration
VAR
    x:stru1;
END_VAR

// Bit access
 x.enable:=true;
```

This sets the 42nd bit in the variable x to TRUE.

Since bvar contains 8 bits and rvar 32 bits, this access is on the second bit in the variable wvar which receives the a value of 4.

5.4.4 Addresses

Notes on addresses

☞ If the online change is used, the contents of addresses can be shifted. Remember this when using pointers, page 557, to addresses!

Address

Individual memory cells are displayed directly using special character strings.

Syntax:

```
%<memory area prefix><size prefix><number|.number|.number...>
```

The following range prefixes are supported:

I	Input (input, physical inputs via input drivers, "sensors")
Q	Output (output, physical outputs via output driver, "actuators")
M	Flag memory space

The following prefixes are supported for the sizes:

X	Single bit
None	Single bit
B	Byte (8 bits)
W	Word (16 bits) [word (32 bits)]
D	Double word (32 bits)
L	Long word (64 bits)

Examples

%QX7.5 and %Q7.5	Output bit 7.5
%IW215	Input word 215
%QB7	Output byte 7
%MD48	Double word at memory location 48 in flag area
ivar AT %IW0 : WORD;	Example of a variable declaration with address specification.

Also refer to

- AT declarations, page 512.

Ensure that the address is valid:

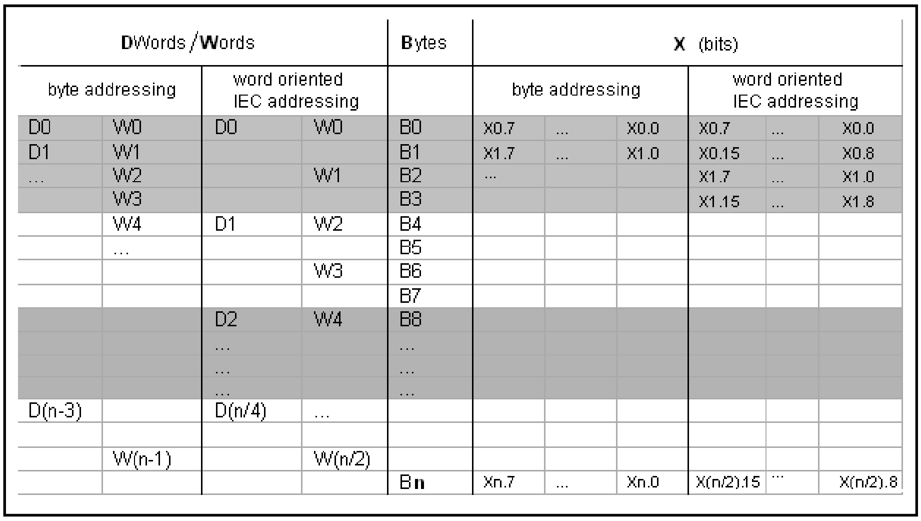
To assign a valid address, page 622, in an application, the requested position in the process image has to be known, that is the responsible **memory space**: Input (**I**), output (Q) or flag memory spaces (M), see above. The required **size** has to be specified as well: bit, byte, Word, DWord (see above: X, B, W, D)

Decisive is the **currently used device configuration** and its settings (hardware structure, device description, I/O settings). Note that there are differences in the address interpretation of devices with "byte addressing mode" and devices with word-oriented "IEC addressing mode".

Depending on the size and the addressing mode, different memory cells can be addressed with the same address specification.

The following table shows a comparison of **byte addressing** and **word-oriented IEC addressing** for bit, BYTE, WORD and DWORD. It also shows overlapping memory spaces that are also present for byte addressing (also refer to the example below the table).

With regard to the spelling, note that the IEC addressing mode is always word-oriented. That means that the word number is in front of the point and then the bit number.



DWords / Words				Bytes	X (bits)					
byte addressing		word oriented IEC addressing			byte addressing			word oriented IEC addressing		
D0	W0	D0	W0	B0	X0.7	...	X0.0	X0.7	...	X0.0
D1	W1			B1	X1.7	...	X1.0	X0.15	...	X0.8
...	W2		W1	B2	...			X1.7	...	X1.0
	W3			B3				X1.15	...	X1.8
	W4	D1	W2	B4						
	...			B5						
			W3	B6						
				B7						
		D2	W4	B8						
		...		...						
		...		...						
		...		...						
D(n-3)		D(n/4)	...							
	W(n-1)		W(n/2)							
				Bn	Xn.7	...	Xn.0	X(n/2).15	...	X(n/2).8

Fig.5-152: *Comparison of byte-oriented or word-oriented addressing of the address formats D, W, B and X*

n = byte number; word-oriented, addressing

Memory space overlapping in byte addressing mode, example:

D0 contains B0 - B3, W0 contains B0 and B1, W1 contains B1 and B2, W2 contains B2 and B3 ->

To avoid overlapping, W1 or D1, D2, D3 may not be used as addressing!

☞ Boolean values are reserved byte by byte if a single bit address is not explicitly specified.

Example: A value change of varbool1 AT %QW0 affects the range from QX0.0 to QX0.7.

☞ If the online change is used, the contents of addresses can be shifted. Remember this when using pointers, page 557, to addresses!

5.4.5 Functions

Functions

In ST, a function call, page 31, can also occur as operand.

Example:

```
Result := Fct(7) + 3;
```

TIME() function

This function provides the time in milliseconds passed since the system start.

The data type is TIME.

Example in IL:

```
TIME
ST    systime
```

Example in ST:

```
systime:=TIME();
```

5.5 Visualization

5.5.1 Visualization, General Information

Depending on the application, IndraWorks provides two visualization tools implemented at different levels of performance:

- WinStudio; This tool should only be used for the visualization of applications that run on visualization devices (BTV / Vxx).

  See also "Rexroth WinStudio; Brief Description, DOK-CONTRL-WIS*PC**V06-KB..-EN-P".

  The Symbol configuration, page 306 object is used to transfer the data to be visualized.
- Visualization onboard; This tool described below should be used to support the commissioning of own applications.

5.5.2 Visualization in IndraLogic 2G

All visualizations in IndraLogic 2G have the following characteristics:

- Within a visualization, any desired expressions, even function calls, are permitted.
- A visualization can be created as object below an application (or in the "General module" folder).
- In contrast to IndraLogic 2.x, all of the visualization elements as well as the visualization objects themselves are implemented as IEC61131-3 function blocks. This way, visualizations can be instanced in other visualizations, i.e. they can be referenced and used to extend other function blocks.
- It includes a placeholder concept that treats a placeholder as an input variable for a visualization, see Frames, references, interfaces, placeholders, page 628.
- Interfaces can be edited in an interface editor. The use of frame elements makes it easy to switch from a display of one visualization to another one within a visualization object.
- Visualizations in the Project Explorer are managed by a Visualization Manager, page 496, for each application. The manager defines common settings for all application-specific visualizations.
- Some general settings for all visualizations in a project (file paths, size adjustment, etc.) can be made in the visualization options, page 199,.
- Each individual visualization has properties such as its planned usage (as "visualization", "Numpad/Keypad" or "dialog") or the display size.

  In this context, ensure that a visualization that can be created explicitly to be used as user input dialog in other visualizations. As an implicit option, there is also a prefabricated numpad and keypad mask for this purpose. The use of such numpad/keypad templates and dialogs can be defined in the configuration of a visualization element.
- The visualization editor also includes a toolbox, page 446, which provides the available visualization elements and a properties editor to configure the added elements.

  It is easy to arrange and group the visualization elements. The standard elements are provided in the corresponding visualization libraries in the project, see Visualization libraries, page 632. IndraLogic 1.x visualizations can be imported.

- Visualizations can be animated, page 451 using IndraLogic project variables that are entered directly or using expressions, i.e. combinations of the variables with operators and constants. This allows variable values to be scaled for example so that the variable values can be used in the visualization.
- Language selection (ANSI or UNICODE) in a visualization is possible by using text lists, see 55,, see Text and language in visualizations, page 628.
- Variable values of an application can be modified and displayed using a Text field element, page 629,. The formatting for these inputs and output is based on the standard function sprintf (C library).
- External data sources, page 632, can be used.
- A tooltip text, page 451, can be assigned to each visualization element.
- In addition to the zoom function, automatic scaling, page 496, is also possible (visualization adjusts to the screen size).
- Visualizations can be saved in libraries, page 632, and thus provided for other projects.
- Visualization element repositories (### in preparation ###) are used to manage visualization elements on the local system.

☞ The visualization works with an integrated runtime system. Thus, start and download messages appear when work is being done in the editor.

5.5.3 Prerequisites

A project with visualization requires the same basic structure as a project without visualization. A device with the application and task configuration as well as an application program have to be present in the project tree (Project Explorer).

The IndraLogic 2G visualization is realized according to the IEC 61131-3 standard. Thus, certain libraries, page 632, have to be integrated into the project.

To select the visualization libraries and to exactly define the selection of elements to be provided in the visualization editor, visualization profiles, page 633, are used. Thus, each visualization project has to be based on such as profile.

5.5.4 Options

In the "Options" dialog, standard entries for the project settings can be made in the 'Visualization' category such as visualization directories for language and image files that have to be available to configure visualizations.

Therefore, click on **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Visualizations** in the main menu to specify the visualization options.

For more information, refer to: Options, Visualizations, page 199.

5.5.5 Creating a Visualization Object

A visualization can be added to the "General module" folder (global object pool) or below an "Application" depending on whether the visualization is to be used globally or only for a specific application.

To create a visualization object in the project, use **Add** ▸ **Visualization...** in the context menu.

Alternatively, the visualization object is available in the "PLC objects" library in the "VI logic objects" folder and can be added to the project via drag&drop.

As soon as the first visualization object is added to the project, the libraries defined in the currently active visualization profile are automatically integrated into the Library Manager.

As soon as the first visualization is added below an application, the Visualization manager object, page 627, is also automatically added.

The newly created visualization object is automatically opened in the visualization editor

5.5.6 Editing a Visualization

The **visualization editor** to create visualizations works together with the toolbox which provides the visualization elements from the integrated element libraries and with the **properties editor** to configure visualization elements.

If necessary, call both editors manually in the main menu under **View ▸ Other windows.**

To open a visualization object, double-click on the visualization object in the Project Explorer.

☞ See also the following chapters on editing visualizations:

- Visualization editor, page 445
- Visualization tools, page 446 (available visualization elements)
- Visualization element properties, page 451 (configuring the elements)

5.5.7 Visualization Manager

When a visualization is added below an application in the Project Explorer, a visualization manager object is also automatically added. The "Visualization Manager" defines common settings for all visualizations in the application, e.g. start visualization.

For more information, refer to: Visualization Manager, page 496.

5.5.8 Start Visualization

The "start visualization", i.e. the visualization object to be displayed first after logging into the control with the application, has to be added below the respective application object in the Project Explorer.

If only one visualization is assigned to the application, it is automatically used as start visualization.

If several visualizations are assigned to the application, the start visualization has to be explicitly selected in the Visualization Manager, page 627,. Afterwards, this option is automatically disabled in all other visualization objects.

☞ The start visualizations below an application switches to online mode with the application.

Since visualizations in the POUs do not have a direct reference to an application, these do not switch to the online mode.

If a change in visualization to a visualization from the POUs is configured in a visualization below the application, it can thus be reached in online mode.

5.5.9 Frames, References, Interfaces, Placeholders

The concept of visualization references and placeholders in IndraLogic 1.x is replaced by a similar concept in IndraLogic 2G.

- In principle, a visualization can be added to another visualization and thus referenced. The "Frame" element which can include one or more "visualization references" is used for this purpose.

  The visualizations available depend on their positions relative to the visualization just edited in the Project Explorer.
- Each visualization is treated as a function block and has an "interface" in which the input variables for a function block can be defined (Interface editor for visualizations, page 491).

  These input parameters function as "placeholders".

  In an instance of the visualization, they have to be replaced by values or expressions for the specific usage in the local object. The replacement has to be be carried out in the "Property" dialog of the "Frame" element that integrates the visualization instances. Note that the input variables of a visualization instance allocated to a specific application have to be assigned to valid variables for this application.
- Switching visualizations:

  If a "frame" element includes several visualization references (Frame selection, page 302), user inputs for another visualization element can be configured so that it causes the display of these references to switch in the frame.

  To do this, the input configuration provides the "Switch visualization" option.

Example:

A visualization contains three buttons and a frame to which the visualizations Visu1, Visu2 and Visu3 are assigned.

The buttons are configured (input, OnMouse actions, visualization switch) so that a user input to a button calls the respective visualization in the frame.

Thus - in contrast to IndraLogic 1.x - different visualizations can be displayed in alternation in one visualization. In other words, a switchboard can be visualized.

- In addition, note the following: A visualization is treated as a function block and can be used to extend another function block, see Extending a function block, page 36.

5.5.10 Text and Language in Visualizations

A text can be assigned to a visualization element in the respective fields of the "Properties" editor.

"Text lists" can be used to manage texts. A text list is a POU that contains text strings that can be referenced uniquely within a project by combining a text list name with a list-internal text ID. Different language versions of a text (local country language) can be defined in a text list. Thus, each text is also identified by a language code. At least the default language has to be defined for every text.

Detailed information on the usage of text lists can be found in text lists, page 55.

☞ Directories that provide text lists for usage in visualizations can be specified in the visualization options. Click on **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Visualizations** in the main menu to enter visualization options, see Visualization options, page 199.

Enter the text list name and text ID (string format) in the text properties of a visualization element in order to define which text is displayed in the element in online mode. In addition, use formatting sequences to affect how the text is displayed, see Formatting text, page 629. To enable **dynamic switching among texts** a variable to specify the text ID has to be used as well.

The language set in IndraLogic determines the local country language version of the text used. A **language selection** in the visualization can also be forced during online mode if a corresponding input configuration is present for a visualization element.

For example, users can then switch to another language via mouse click (configuration in the "Input" category in the "Properties" editor), see Visualization elements - Properties, page 451.

This allows dynamic display and language selection for visualization texts.

There are two types of text in a visualization:

1. **Static texts:**

   Static text cannot be changed in online mode. It is configured in the "Texts" category in the visualization element properties and is managed in an automatically generated text list "GlobalTextList".

2. **Dynamic texts:**

   Dynamic text can be modified in online mode by the user input or by a variable of the application. It is configured in the "Dynamic texts" category in the visualization element properties. The name of the dynamic text lists and the text ID for the desired text have to be specified.

Formatting text

In addition to entering the pure text in the element configuration, formatting specifications can also be edited to format the text display in online mode. A formatting specification always begins with a "%" followed by a character that determines the type of formatting. Use formatting by itself or in combination with the actual text.

If a **%s** is included in the text specification, in online mode, it is replaced by the value of the variable specified in the "Text variable" properties field in the "Text variables" category.

To display the instance name of a variable that was transferred to the visualization function block as input parameter, use the pragma attribute "parameterstringof", page 543.

Note that any formatting specification in accordance with the standard C library function "sprintf" can be used if it matches the data type of the variable used.

The following table contains some examples of formatting specifications.

Characters following "%"	Argument/output as
d,i	Decimal number
o	Unsigned octal number (without a preceding zero)
x	Unsigned hexadecimal number (without a preceding "0x")
u	Unsigned decimal number

Characters following "%"	Argument/output as
c	Single character
s	Character string: In online mode, this is replaced by the value of the variables specified in the "Text variable" properties field in the "Text variables" category.
f	REAL values **Syntax:** `%|<Alignment><LowestWidth><Accuracy>|f` Alignment is defined by a minus sign (left-justified) or plus sign (right-justified, default). Accuracy is defined by the number of places after the decimal (preset value: 6). See the following example.

Fig.5-153: Formatting specifications

Example:

Entry in the "Text" properties field: Fill level %2.5f mm

Entry in the "Text variable" field (REAL variable to specify the fill level), e.g.: Plc_Main.fvar1

⇒ Output in online mode, e.g.

```
Fill level 32.8999 mm
```

If a "percent sign %" is to be displayed together with one of the previously described formatting specifications, "%%" has to be entered.

Example:

Enter "Rate in %%: %s" to display the following in the online mode:

```
Rate in %: 12
```

(if the value of the text variable is currently "12").

To process all character sequences in the visualization in Unicode format, select the option "Use Unicode strings", page 497, in the Visualization Manager of your application.

System time output

A combination of "**%t**" and the following special characters in square brackets is replaced by the actual system time in online mode. The placeholders define the display format, refer to the table below.

☞ Import of IndraLogic 1.x projects:

The time format %t used in IndraLogic 1.x is automatically converted into the new %t[] format when an old project is imported. However, the following placeholders are no longer supported: %U, %W, %z, %Z.

Valid placeholders:

ddd	Name of weekday, abbreviated, e.g. "Wed"
dddd	Name of weekday, e.g. "Wednesday"
ddddd	Weekday as number (0 - 6; Sunday is 0)
MMM	Name of month, abbreviated, e.g. "Feb"
MMMM	Name of month, e.g. "February"
d	Day in month as number (1 – 31), e.g. "8"
dd	Day in month as number (01 – 31), e.g. "08"
M	Month as number (1 – 12), e.g. "4"
MM	Month as number (01 – 12), e.g. "04"
yyy	Day in year as number (01 – -366), e.g. "067"
y	Year without century indication (0-99), e.g. "9"
yy	Year without century indication (00-99), e.g. "09"
yyy	Year with century indication, e.g. "2009"
HH	Hour, 24 hour format (01-24), e.g. "16"
hh	Hour, 12 hour format (01-12), e.g. "8" for 4 PM
m	Minutes (0-59), without preceding zero, e.g. "6"
mm	Minutes (00-59), without preceding zero, e.g. "06"
s	Seconds (0-59), without preceding zero, e.g. "6"
ss	Seconds (00-59), without preceding zero, e.g. "06"
ms	Milliseconds (0-999), without preceding zero, e.g. "322"
t	Identifier for the display in 12 hour format: A (hours <12) or P (hours >12), e.g. "A" if it is 9 a.m.
tt	Identifier for the display in 12 hour format: AM (hours <12) or PM (hours >12), e.g. "AM" if it is 9 o'clock in the morning
' '	Character strings that contain one of the placeholders listed above have to be enclosed in single apostrophes. All other texts in the format string can remain without apostrophes: e.g. "update", since it contains a "d" and a "t"

Fig.5-154: Valid placeholders in the new format

Example:

%t['Last update:' ddd MMM dd.MM.yy 'at' HH:mm:ss] ⇒

Display in online mode:

Last update: Wed Aug 28.08.02 at 16:32:45

Font The font used in online operation can also be defined in the visualization element properties. See the categories 'Text properties' and 'Font variables'.

Alignment The horizontal and vertical alignment of the element text can be defined in the element properties in the "Text properties" category.

☞ When configuring visualization elements (e.g. text fields, tooltips, alignment, font), also refer to the description in Visualization elements - Properties, page 451.

5.5.11 Images in a Visualization

The "Image" element can be used to add an image from an external image file to a visualization.

☞ Directories that provide image files for usage in visualizations can be specified in the visualization options. Click on **Tools** ▸ **Options** ▸ **IndraLogic 2G** ▸ **Visualizations** in the main menu to enter visualization options; see Visualization Options, page 199.

The image files are managed in the project in image pools, page 61,.

Individual images can be uniquely referenced by a combination of image pool name and ID specified for each image file in the pool.

This ID and (for unique access) the image pool name can be specified in the element properties of an image element in order to determine the image displayed in online mode. The image reference can also be specified using a variable, which enables **dynamic switching among images**.

In addition, a background image for a visualization can be defined in the main menu with **VI logic visualization** ▸ **Background**.

Alternatively, **Background** can also be selected via the respective context menu.

☞ When configuring visualization elements (e.g. text fields, tooltips, alignment, font), also refer to the description in the visualization elements - properties, page 451.

5.5.12 External Data Sources

External (remote) data sources can also be used in visualizations in order to set up a point to point connection between an application (on the "local" device) and a remote data source.

For this purpose, create a data server for the local application which manages the related data sources; see Using data sources in visualizations, page 325.

5.5.13 Visualization Libraries

The libraries necessary for creating and executing a visualization are automatically added to the library manager in the global "General module" folder as soon as a visualization is created in the project.

Since the visualization elements in IndraLogic 2G are created as function blocks, the basic elements are provided by libraries.

If a visualization object is added to the project, certain visualization libraries are integrated into the Library Manager. The names and versions of these libraries are defined in the visualization profile currently used. The profile also specifies exactly which elements from these libraries are available in the visualization editor toolbox.

A visualization library is always created as a special type of a "placeholder library". The result is that the exact version of the library to be used is not specified as long as it is not integrated into a project. The current visualization profile specifies which version is actually needed. Note that this library type differs from the device-specific placeholder libraries at which the placeholders are resolved from the device description.

See the following basic libraries that are integrated by default as soon as a visualization element is added to a standard project. They reference other libraries that are not listed here:

- **VisuElems.library**, (basic set of visualization elements)
- **VisuElemMeter.library**, (meter and bar display elements)
- **VisuElemWinControls.library**, (table, text field, scrollbar elements)
- **VisuElemTrace.library**, (trace element)
- **VisuInputs.library**, (handling inputs on a visualization)

5.5.14 Visualization Profiles

Each visualization project, that is a project that contains at least one visualization object, has to be based on a visualization profile. This profile defines the following:

- the names and versions of the visualization libraries, page 632 that are automatically integrated into the project as soon as a visualization object is created, page 626.
- a selection of the visualization elements from the integrated libraries provided in the toolbox of the visualization editor, page 446,

The **profile configuration** is completed outside the project in the visualization element repository (### in preparation ###).

Several profiles can be defined and stored on the local system.

The **profile that is used by default in the project** is defined in the project settings (visualization profile).

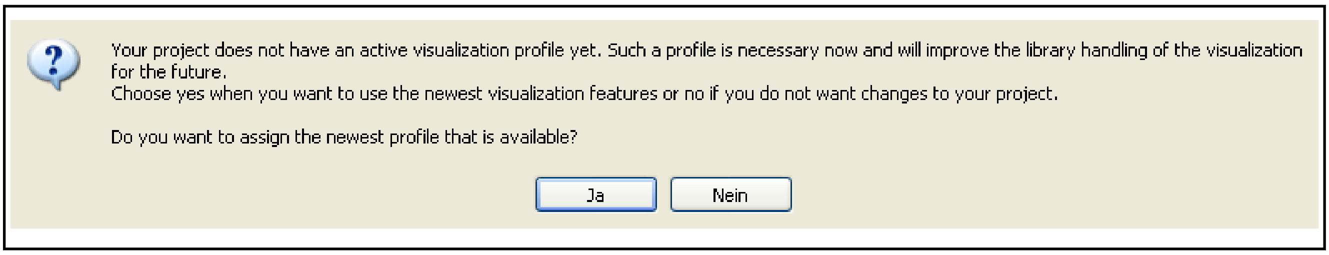
Switching to another profile is possible at any time. Note that the selected profile applies to all devices and applications.

In case of a profile switch, a message appears that indicates that the switch could prevent logging in without an online change or download.

Switching profiles causes an automatic update in the library manager with respect to the required libraries. These do not have to be adjusted manually.

A visualization profile has to define at least one **"main" visualization library**, i.e. a library that defines the replacements for the placeholders used in other visualization libraries. In a standard installation, this library is **VisuElems.library**.

When opening an **old project** that was not created with a visualization profile, you are prompted whether you want to assign the newest profile available.



Your project does not have an active visualization profile yet. Such a profile is necessary now and will improve the library handling of the visualization for the future.
Choose yes when you want to use the newest visualization features or no if you do not want changes to your project.

Do you want to assign the newest profile that is available?

Fig.5-155: Message box if there is no profile

If "Yes", the "Newest profile" is used. If "No", the oldest available profile is entered in the project settings (it is called "**Compatibility profile**"), but no further changes are made in the project.

5.5.15 Visualization Running in the Programming System

For diagnostic purposes, it can be desirable to let one of the visualization(s) belonging to an application only run in the programming system without having to load visualization code to the control.

This"**Diagnostic visualization**" available **from V3.3.2.0** is **automatically** used if no client object "TargetVisualization" is attached below the Visualization Manager in the Project Explorer. Then, no **visualization code** is generated and loaded to the control. However, a few limitations result which are listed in the following.

If you do not want to remove the TargetVisualization objects below the Visualization Manager to obtain the diagnostic mode of the visualization, exclude the client objects from compilation by selecting the setting "Exclude from compilation" in the respective object properties.

☞ If a fixed compiler version was set in a project that was originally created with a version < V3.3.2, the project acts according to the possibilities available in that version!

For versions lower than V3.3.2.0, the setting "Do not use the PLC for visualization" has to be selected so that a visualization only runs in the programming system.

Numerical values displayed in an element in a diagnostic visualization ("%s" in the "Text" property) are displayed in accordance with the currently set display format (binary, decimal, hexadecimal).

☞ In the diagnostic visualization, the VAR_INPUT variables act like VAR_IN_OUT variables.

Restrictions

Expressions, monitoring

The diagnostic visualization supports only those expressions that can be handled by the monitoring mechanism of the programming system. These are:

- **Normal variable accesses such as PlcProg.myPou.nCounter**
- *Complex accesses as listed in the following. Note that a runtime system of V3.3 SP2 or later is required:*
  - Access to an array of scalar data types in which one variable is used as index (a[i])
  - Access to an array of complex data types (structures, function blocks, arrays) in which one variable is used as index (a[i].x)
  - Access to a multidimensional array by all types of data types with one or more variable indices (a[i, 1, j].x)
  - Access to an array with a constant index (a[3])
  - Accesses as described above in which simple operators are used for the calculations within the index brackets (a[i+3])
  - Nested combinations of the complex expressions listed above (a[i + 4 * j].aInner[j * 3].x)
- Operators supported in index calculations: +, -, *, /, MOD
- Pointer monitoring such as p^.x
- Methods or function calls are not supported except for the following: All standard text functions, all type conversion functions such as INT_TO_DWORD, all operators such as SEL, MIN, ...

Inputs:

Within the input action "Execute ST code" only a list of assignments is supported.

Example:

```
PlcProg.n := 20 * PlcProg.m;
```

Not permitted:

```
  IF PlcProg.n < MAX_COUNT THEN
     PlcProg.n := PlcProg.n + 1;
  END_IF
```

Use the following instead:

```
PlcProg.n := MIN(MAX_COUNT, PlcProg.n + 1);
```

☞ If a list of assignments is used, the value on the left side is assigned in the next cycle. Immediate processing in the next line is not possible.

Visualization interface:

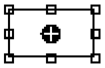
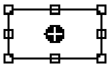
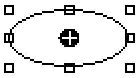
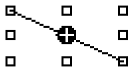
Within the interface definition of a visualization, the type "interface" ("INTERFACE") may not be used.

DataServer:

The diagnostic visualization (visualization runs in the programming system) is only for observation purposes of the current application. It does not display the current values of variables transmitted from another data source via the data server. Only the initial values or the most recently transferred values are displayed.

5.5.16 Visualization Elements of Type "Visualization"

The category "Visualization" of the "ToolBox" tab in the visualization editor contains the following elements:

Rectangle		Rectangle, rounded rectangle and ellipse are the same type of element. Each shape can be converted into the other by changing the type property.
Rounded Rectangle		
Ellipse		
Line		Line: Adjust the slope of a "line" element between "upper left - lower right" and "lower left - upper right".

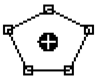
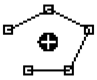
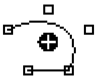
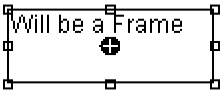
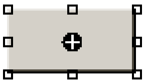
Polygon		Polygon, polyline and curve are the same type of element. Each shape can be converted into the other by changing the type property. An additional position point that specifies the segments and the shape of an element can be inserted by clicking on an existing point (small rectangle on the frame line) while holding down the <Ctrl> key. A point can be deleted by clicking on the point while pressing and holding down the <Shift> + <Ctrl> keys. In a curve element, the points next to the line are used as "grips" to modify the curve shape.
Polyline		
Curve		
Frame		Frame: A frame defines a section of the current visualization to which one or several other visualizations are assigned of which one is displayed in online mode. In principle a frame can be configured as a rectangle element. With the correct configuration, a user input can be used to switch among the visualizations displayed. With the correct configuration, a user input can be used to switch among the visualizations displayed. Assign visualizations to a frame using the "Configuration of frame visualizations" which opens via the command Frame selection, page 302). All available visualizations are provided for selection. For information about configuring a visualization switch by user inputs, refer to the possible mouse actions, page 451.
Button		An image and a "height" (for the relief view) can be assigned to a button element.
Image		An image element is filled by an image file defined by an ID and name of the image pool, page 61 in which it is managed. The file specification can also be dynamically configured, i.e. via a project variable.

Fig.5-156: Standard elements in the VisuElems.library

When configuring visualization elements (e.g. text fields, tooltips, alignment, font), also refer to the description in Visualization elements - Properties, page 451.

5.5.17 Visualization Elements of Type "Complex Controls Visualization"

Elements overview

The category "Complex Controls" in the toolbox, page 446, in the visualization editor contains the following elements.

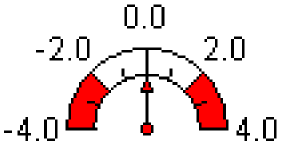
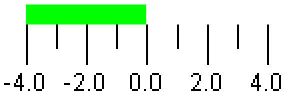
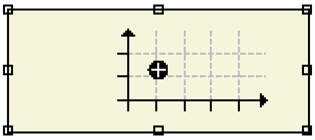
Element name	Element	Description
meter		The "meter" element inserts a tachometer into the visualization for which a minimum and maximum display value can be entered. The needle position indicates the current value of the associated input variable. Specific background colors can be defined for certain value ranges.
Bar display		The "bar display" element inserts a bar graph into the visualization for which a minimum and maximum display value can be entered. The length of a the bar indicates the current value of the associated input variable. Specific colors can be defined for certain value ranges. A horizontal bar display is preset. After the element has been inserted, the orientation can be changed to vertical in the element properties.
Trace		The trace, page 431, element allows a trace to be integrated into a visualization. The name of the trace to be displayed is input in the element properties. However, the recorded variables are configured in the Trace configuration, page 436, dialog.

Fig.5-157: Elements in the "Complex Controls" category

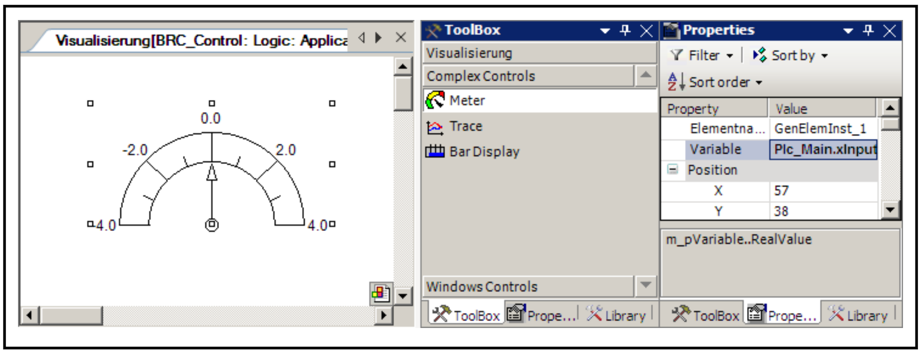
An explanation using examples can be found under:

- Configuring the meter element, page 637
- Configuring the bar display element, page 640
- Configuring the trace element. page 643

Configuring the Meter Element

To configure the "meter" element, carry out the following steps:

1. The "meter" visualization element looks like a tachometer and allows the current value of the associated input variable to be displayed. Insert the "meter" visualization element into a visualization by dragging it from the toolbox, page 446, and dropping it in the visualization.



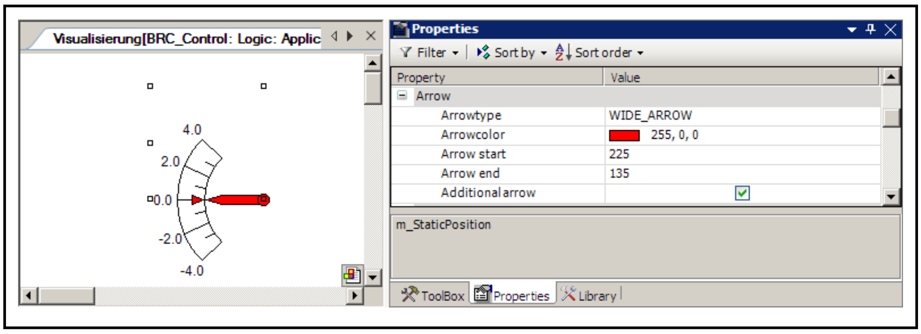
Property	Value
Elementna...	GenElemInst_1
Variable	Plc_Main.xInput
Position	
X	57
Y	38

m_pVariable..RealValue

Fig.5-158: Inserting the meter element and link to the input variables

In the element properties for the "meter" element, enter the associated input variable (e. g. "xInput") into the "Value" column. After clicking into the input field, the ... button is available. It can be used to search through the project for the input variable. Ensure to specify the input variable by entering its complete path in the Project Explorer.

2. The orientation and size or the scale display as well as the color and appearance of the arrow can be specified in the "Arrow" section in the element properties.



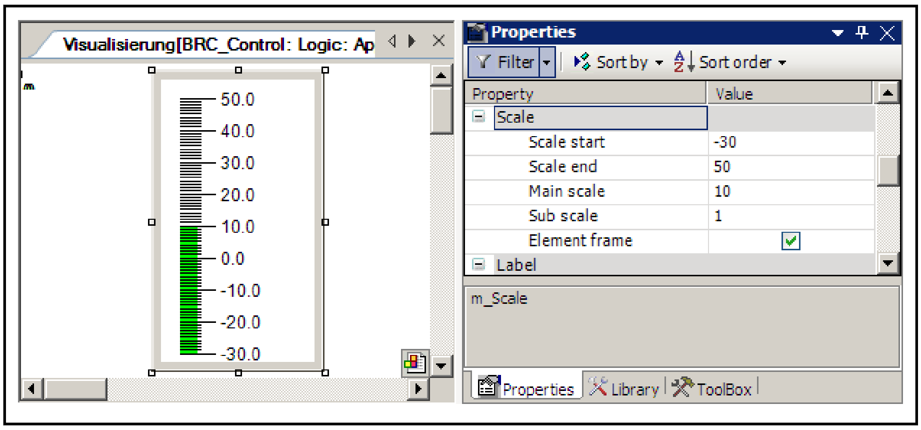
Property	Value
⊟ Arrow	
Arrowtype	WIDE_ARROW
Arrowcolor	🟥 255, 0, 0
Arrow start	225
Arrow end	135
Additional arrow	☑

m_StaticPosition

Fig.5-159: Defining the appearance of the scale display and the arrow

The number in the "Arrow start" and "Arrow end" input field specifies the angle (in degrees) spanned by the left/right margins of the scale. This angle is mathematically oriented, i.e. counterclockwise. Thus, the value in the "Arrow start" field always has to be greater than the value in "Arrow end"! The pair of starting and end value is periodic with 360 degrees.

3. In the "Scale" section, specify the numerical range for the scale as well as rough and precision scaling.



Property	Value
⊟ Scale	
Scale start	-30
Scale end	50
Main scale	10
Sub scale	1
Element frame	☑
⊟ Label	

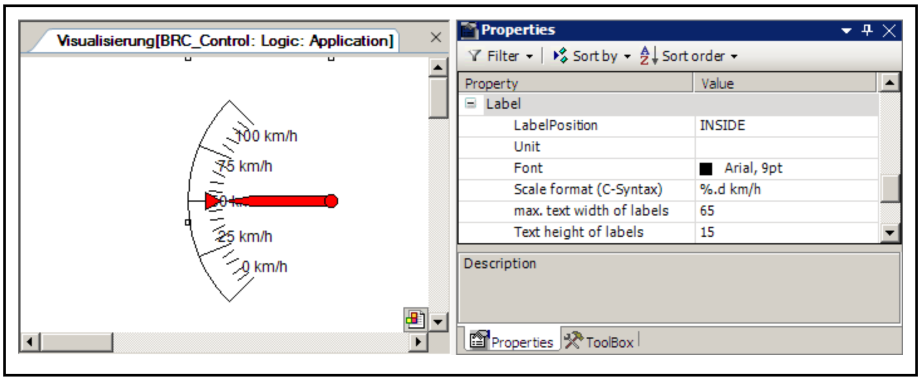
m_Scale

Fig.5-160: Configuration within the "Scale" section in the element properties

Enter the starting point for the scale in "Scale start". The "Scale start" value has to be lower than the "Scale end" value inserted at the end of the scale.

In contrast to rough scaling, precision scaling ("Sub scale") can be omitted by setting the value for the spacing to 0. In this case, precision scaling lines are not displayed. In the example, the "Frame inside" checkbox is deselected so that the inner arc of the scale is hidden.

4. Format the scale label in the "Label" section.



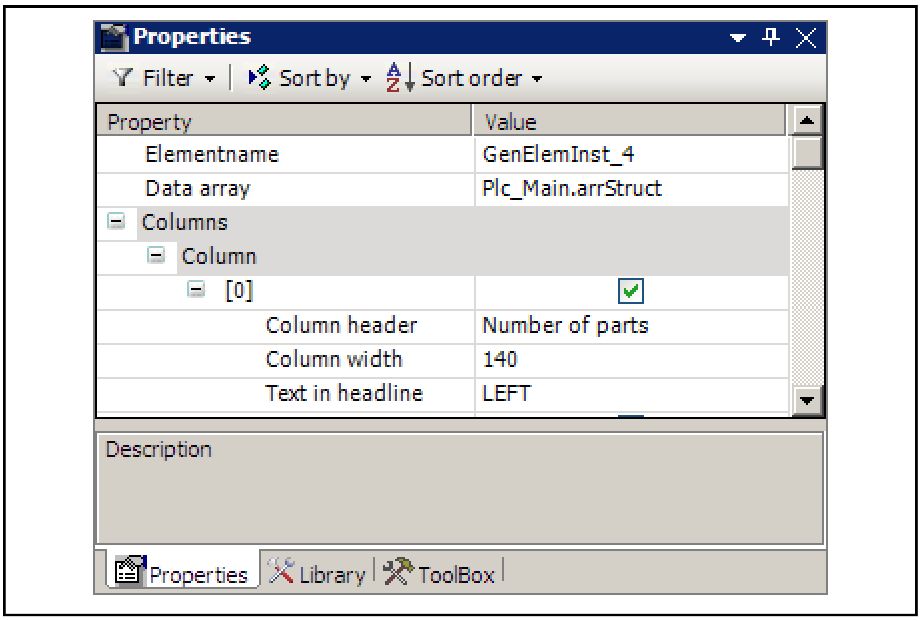
Property	Value
Label	
LabelPosition	INSIDE
Unit	
Font	■ Arial, 9pt
Scale format (C-Syntax)	%.d km/h
max. text width of labels	65
Text height of labels	15

Fig.5-161: Configuration within the "Label" section in the element properties

By changing "LabelPosition" from OUTSIDE to INSIDE, move the scale labels to the inside of the arc. The entry in the "Unit" field appears below the lowest point of the arrow. In the "Font" line, use the ... button to define the font of the display.

In the "Scale format (C syntax)" line, adjust the formatting of the scale labels. The numerical value of the scale has to be formatted using the syntax of the programming language C (use %d for integers and %.Xf for floating point numbers where X can be replaced by the desired number of digits following the decimal point); see Text and language in visualizations, page 628. The values in the lines "Max. text width of labels" and "Text height of labels" are automatically entered based on your previous settings in this section. These values have to be changed only if the automatic adjustment does not lead to the desired result.

5. In the "Colors" section, select colors for specific scale ranges by creating "Color areas" with the Create new button. The color areas are numbered in ascending order. Each color area has its own input fields within the element properties.



Property	Value
Elementname	GenElemInst_4
Data array	Plc_Main.arrStruct
Columns	
Column	
[0]	☑
Column header	Number of parts
Column width	140
Text in headline	LEFT

Fig.5-162: Configuration within the "Colors - Color areas" in the element properties

In the "Begin of area" and "End of area" fields, specify the area that is to be filled with color. Select the color from the pop-up menu in the "Colors" line.

Use the ☒ Delete button to delete existing color areas.

The effect of the "Durable color areas" checkbox is only visible in online mode.

The right "meter" element only displays the color area where the arrow is currently located. The left "meter" element displays all of the color areas because there is a check in the "Durable color areas" checkbox.

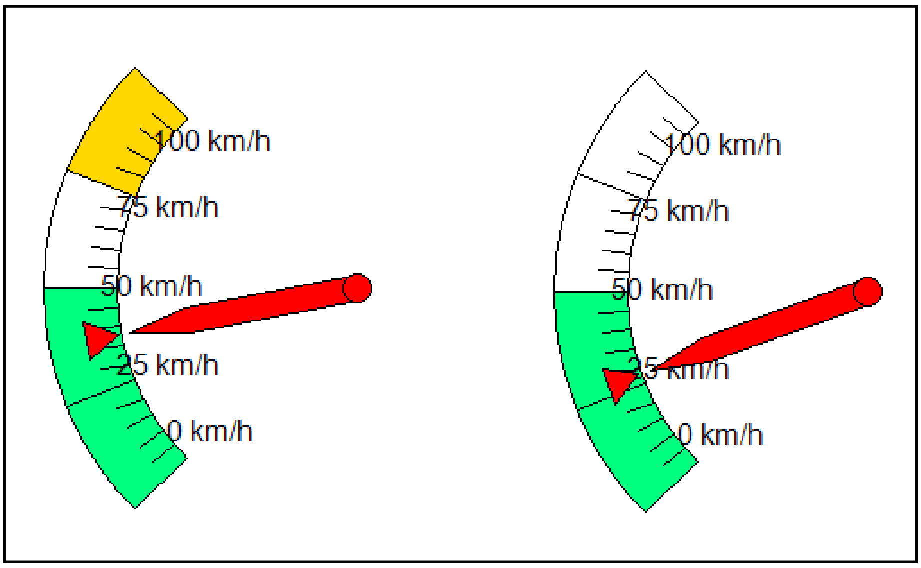
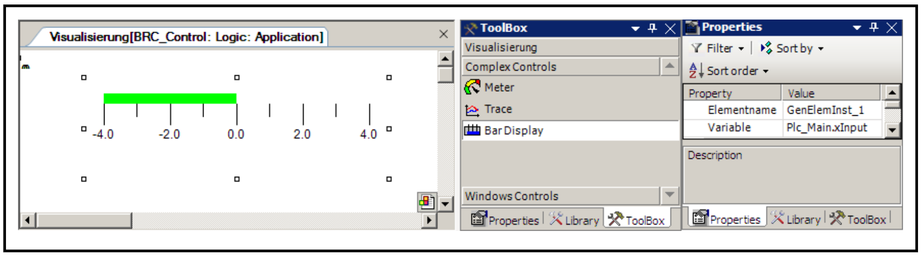


Fig.5-163: Durable color areas option

Configuring the bar display element

To configure the "bar display" element, carry out the following steps:

1. Insert the "bar display" visualization element into a visualization by dragging it from the Toolbox, page 446, and dropping it in the visualization. It looks like a thermometer and allows the current value of the associated input variable to be displayed.

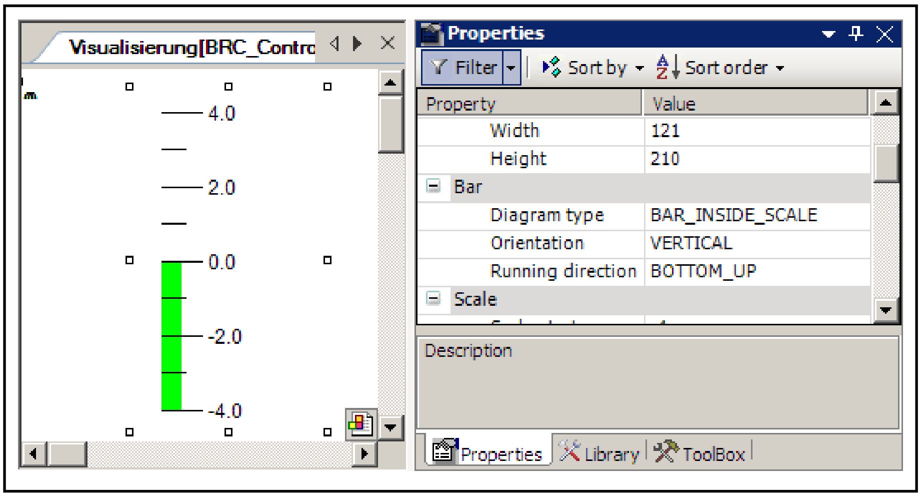


Property	Value
Elementname	GenElemInst_1
Variable	Plc_Main.xInput

Fig.5-164: Inserting the bar display element and linking to the input variables

In the element properties for the "bar display" element enter the associated input variable (e. g. "xInput") in the "Variable" line. After clicking into the input field, the ... button is available. It can be used to search through the project for the input variable. Ensure to specify the input variable by entering its complete path in the Project Explorer.

2. The orientation and placement of the bar with respect to the scale can be set in the "Bar" section of the element properties. The position of bar with respect to the scale can be specified in the "Diagram type" line. The selection field offers SCALE_BESIDE_BAR, SCALE_INSIDE_BAR and BAR_INSIDE_SCALE.



Property	Value
Width	121
Height	210
⊟ Bar	
Diagram type	BAR_INSIDE_SCALE
Orientation	VERTICAL
Running direction	BOTTOM_UP
⊟ Scale	

Description

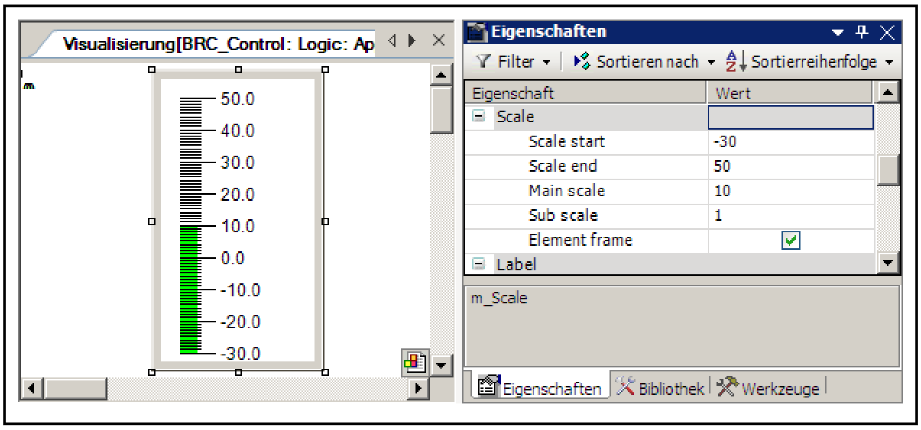
Fig.5-165: Configuration within the "Bar" section in the element properties

In the example, the orientation of the bar has been changed from HORIZONTAL to VERTICAL in the "Orientation" line. This setting also changes the possible entries for the "Running direction" line.

Orientation	Running direction	Description
HORIZONTAL		
	LEFT_RIGHT	The bar runs from left to right
	RIGHT_LEFT	The bar runs from right to left
VERTICAL		
	BOTTOM_UP	The bar runs from top to bottom
	TOP_DOWN	The bar runs from bottom to top

Fig.5-166: Settings for the "bar" property

3. In the "Scale" section, specify the value range of the scale and the subdivisions for rough and precision scaling.



Eigenschaft	Wert
⊟ Scale	
Scale start	-30
Scale end	50
Main scale	10
Sub scale	1
Element frame	☑
⊟ Label	

m_Scale

Fig.5-167: Configuration within the "Scale" section in the element properties

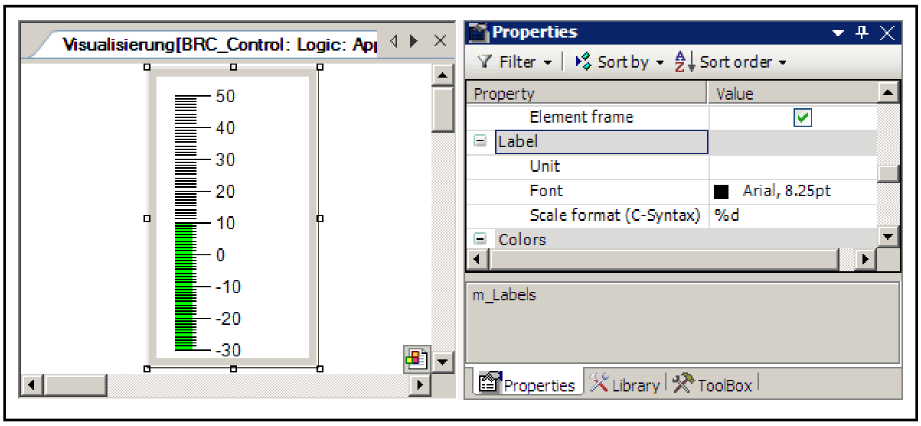
Limit the value range of the scale starting at the bottom using the value in "Scale start" and from the top using the value in "Scale end". The value for "Scale start" has to be lower than that for "Scale end".

The entry for spacing in "Main scale" specifies the marking for rough scaling. With the entry for spacing in "Sub scale" YOU specify the marking for precision scaling. The entries for spacing in "Main scale" and

"Sub scale" can be set to 0 to switch off the display of scaling marks. If the value for rough scaling (Main scale) is set to 0, no scaling marks are displayed, regardless of the value set for precision scaling. If precision scaling (Sub scale) is set to 0, only the scaling marks for rough scaling are displayed.

Place a check in the "Element frame" checkbox to place a frame around the "bar display" element.

4. Format the bar label in the "Label" section.



Property	Value
Element frame	☑
⊟ Label	
Unit	
Font	■ Arial, 8.25pt
Scale format (C-Syntax)	%d
⊟ Colors	

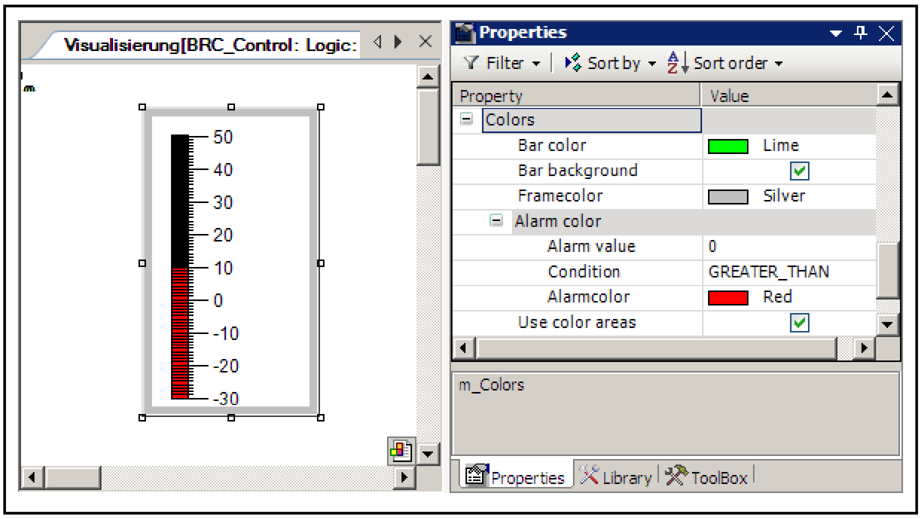
m_Labels

Fig.5-168: Configuration within the "Label" section in the element properties

In the "Unit" input field, specify the unit for the display and it is centered below the bar. In the "Font" line, use the ... button to define the font of the display.

In the "Scale format (C syntax)" line, adjust the formatting of the scale labels. The numerical value of the scale has to be formatted using the syntax of the programming language C (use %d for integers and %.Xf for floating point numbers where X can be replaced by the desired number of digits following the decimal point); see Text and language in visualizations, page 628.

5. In the "Colors" section, specify the coloring of the element.



Property	Value
⊟ Colors	
Bar color	🟩 Lime
Bar background	☑
Framecolor	▭ Silver
⊟ Alarm color	
Alarm value	0
Condition	GREATER_THAN
Alarmcolor	🟥 Red
Use color areas	☑

m_Colors

Fig.5-169: Configuration within the "Color" section in the element properties

In the "Bar color" line, specify the color of the bar.

The default background for the bar that is not currently filled in is white.

Use the "Bar background" checkbox to change the background color to black.

In the "Frame color" line, specify the color of the frame.

In the "Alarm color" subsection, enter an alarm color in the "Alarm color" line. The bar is displayed in this color as soon as the current value of the input variable meets the requirements for the alarm state. The alarm color is determined by the definition of the alarm value in the "Alarm value" line.

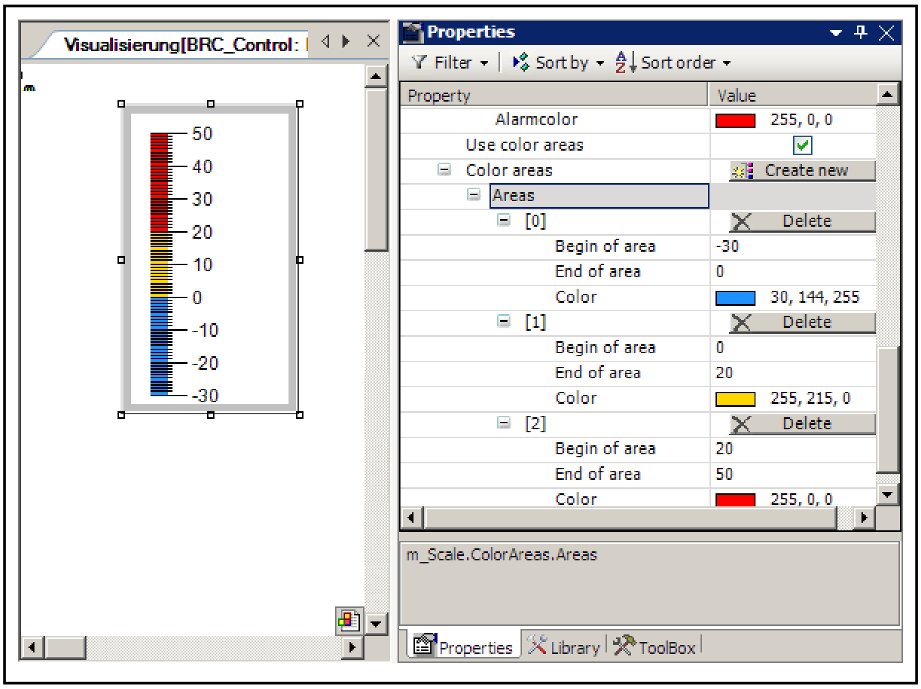
Values may either not exceed or not be less than the "Alarm value" entered. This setting can be made in the "Condition" line with the options GREATER_THAN or LESS_THAN.

By selecting the checkbox "Use color areas", define the color areas for the "bar display" element.

Subsections of the scale can be assigned a certain color by creating a color area with the Create new button. The color areas are numbered in ascending order. Each color area has its own input fields within the element properties.

In the "Begin of area" and "End of area" fields, specify the area that is to be filled with color. Select the color from the pop-up menu in the "Colors" line.

Use the Delete button to delete existing color areas.



Property	Value
Alarmcolor	🟥 255, 0, 0
Use color areas	☑
⊟ Color areas	Create new
⊟ Areas	
⊟ [0]	Delete
Begin of area	-30
End of area	0
Color	🟦 30, 144, 255
⊟ [1]	Delete
Begin of area	0
End of area	20
Color	🟨 255, 215, 0
⊟ [2]	Delete
Begin of area	20
End of area	50
Color	🟥 255, 0, 0

m_Scale.ColorAreas.Areas

Fig.5-170: Configuration within the "Colors - Color areas" section in the element properties

Configuring the Trace Element

To display traced variables within a visualization, insert a trace object into the project first. In the trace object, configure the variables to be recorded, a trigger variable (as needed) and the trace (axes, update interval, etc.).

Activating the option "Create trace block for visualization" within the trace configuration, page 436, implicitly generates the "<TraceName>_<RecordName>_VISU" function block. That is the prerequisite for using the trace object as an input for the trace visualization element.

Creating the trace object

The presence of the trace object is the prerequisite for its used in the visualization.

In the context menu of the application node click on **Add ▸ Trace** to add the "trace" object.

Open the trace object by double clicking on it.

In the trace editor, click on "Configuration" and activate the option "Create trace block for visualization".

Activating the option "Create trace block for visualization" within the trace configuration implicitly generates the "<TraceName>_<RecordName>_VISU" function block in the trace editor, see Record settings, page 436,.

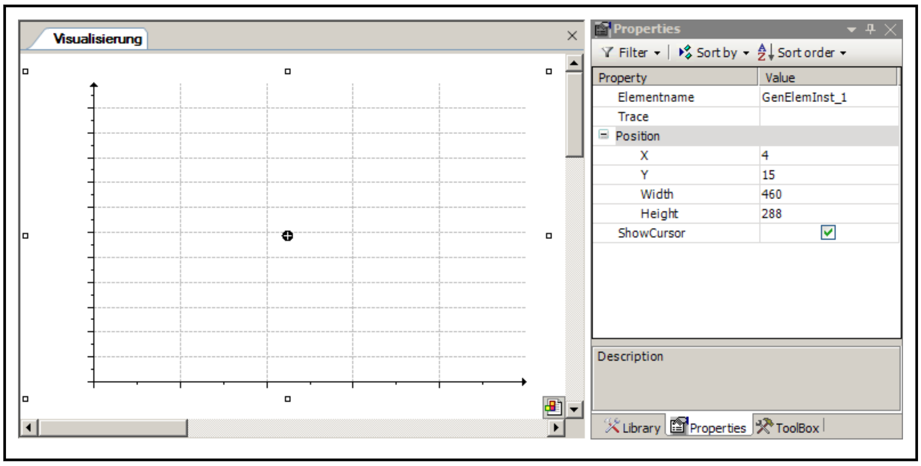
That is the prerequisite for using the trace object as an input for the trace visualization element.

Further information on the "trace" object can be found in trace editor, page 431, and trace commands, page 294.

Using the trace object in the visualization

To configure the "trace" element, carry out the following steps:

1. Insert the "trace" visualization element into a visualization by dragging it from the Toolbox, page 446, and dropping it in the visualization.



Property	Value
Elementname	GenElemInst_1
Trace	
⊟ Position	
X	4
Y	15
Width	460
Height	288
ShowCursor	☑

Fig.5-171: Inserting and configuring the trace visualization element

The related trace object is defined within the element properties. In the "Trace" line enter the complete name of the trace, including its path in the project tree (for sequence tracking)! Click the … button for input assistance.

2. To affect how the "trace" is displayed in online mode, add further visualization elements. The following template includes four buttons for managing trace behavior in the visualization.

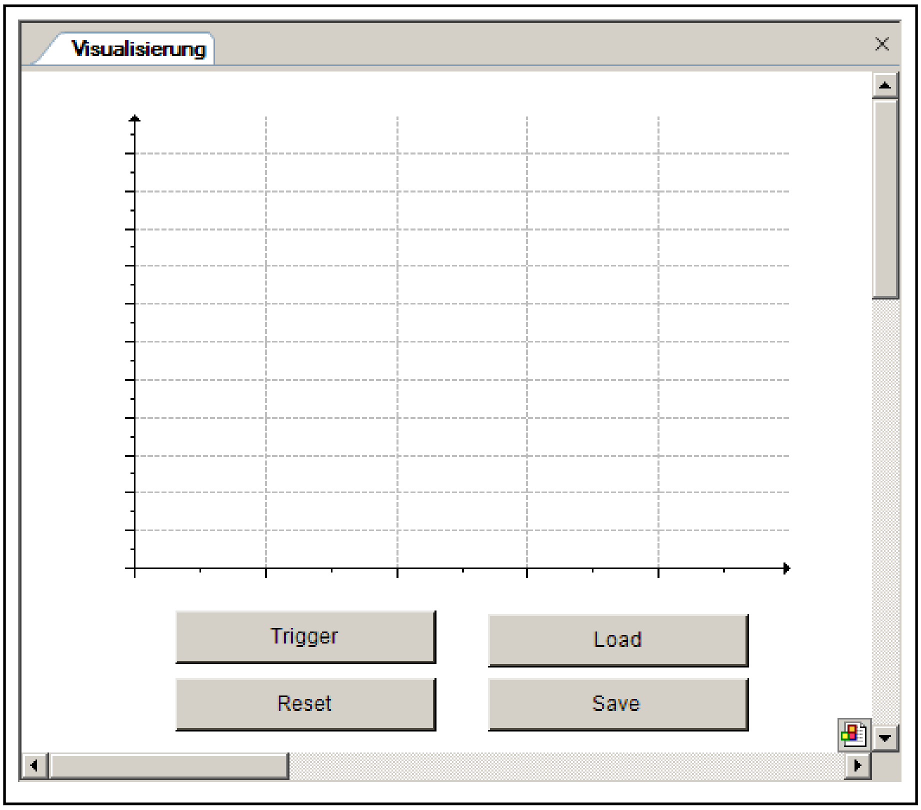
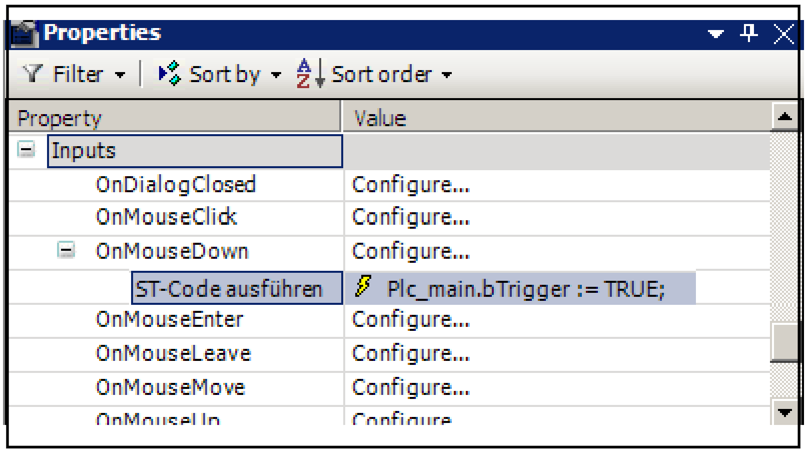


Fig.5-172: Additional elements for managing the trace

- Trigger

  The button labeled "Trigger" triggers the trace, i. e. the value "TRUE" is assigned to the associated trigger variable. This can be implemented in the element properties of the button. The "bTrigger" trigger variable has to be announced in the declarations of the main program. To function, the trigger button requires the corresponding instructions in the statements of the main program.

  The special "OnMouseDown" action is connected with the execution of ST code that sets the associated trigger variable to TRUE ("Plc_Main.bTrigger:=TRUE;").



Property	Value
Inputs	
OnDialogClosed	Configure...
OnMouseClick	Configure...
OnMouseDown	Configure...
ST-Code ausführen	Plc_main.bTrigger := TRUE;
OnMouseEnter	Configure...
OnMouseLeave	Configure...
OnMouseMove	Configure...
OnMouseUp	Configure

Fig.5-173: Element properties dialog

- Reset

  The "Reset" button should reset the trace, for example, by setting the trigger variable back to FALSE ("Plc_Main.bTrigger:=FALSE;")

and the associated reset variable is set to TRUE (e. g. "Plc_Main.bReset:=TRUE;"). The "bReset" reset variable has to be made known in the declarations of the main program.

To function, the reset button requires the corresponding instructions in the statements of the main program.

- Load

  The button labeled "Load" allows the displayed trace to be modified without requiring a logout from the related application.

  To do this, ST code that sets the trigger variable to FALSE is attached and has to be executed. ("Plc_Main.bTrigger:=FALSE;").

  Furthermore, in the code, the "bLoad" variable is set to TRUE, for example with "Trace_PlcTask_VISU.bLoad:=TRUE;".

  This Boolean variable "bLoad" is part of the implicitly generated "<TraceName>_<RecordName>_<_VISU>" block and it monitors the loading of a trace configuration file with the suffix .tcg. The trace visualization (and of course, the trace itself) are adapted for the configuration specified in this file.

  A detailed description of the configuration file can be found in Example of a trace configuration file, page 648.

- Save

  The "Save" button allows the trace data currently displayed in the visualization to be saved. To do this, the attached ST code sets the trigger variable to FALSE and the bSave variable to TRUE (for example with

  ```
  Plc_Main.bTrigger:=FALSE;
  ```

  ```
  Trace_PlcTask_VISU.bSave:=TRUE;
  ```

  This Boolean variable "Trace_PlcTask_VISU.bSave" is part of the implicitly generated block

  ```
  <TraceName>_<RecordName>_VISU
  ```

  block and it requires that a

  ```
  <TraceName>_<RecordName>.dta
  ```

  data file be created in the runtime system. This file contains the currently displayed values of the recorded variables and if necessary it can be read into the trace again at a later time.

The following figure shows the visualization of a trace in online mode after the "Trigger" button has been clicked:

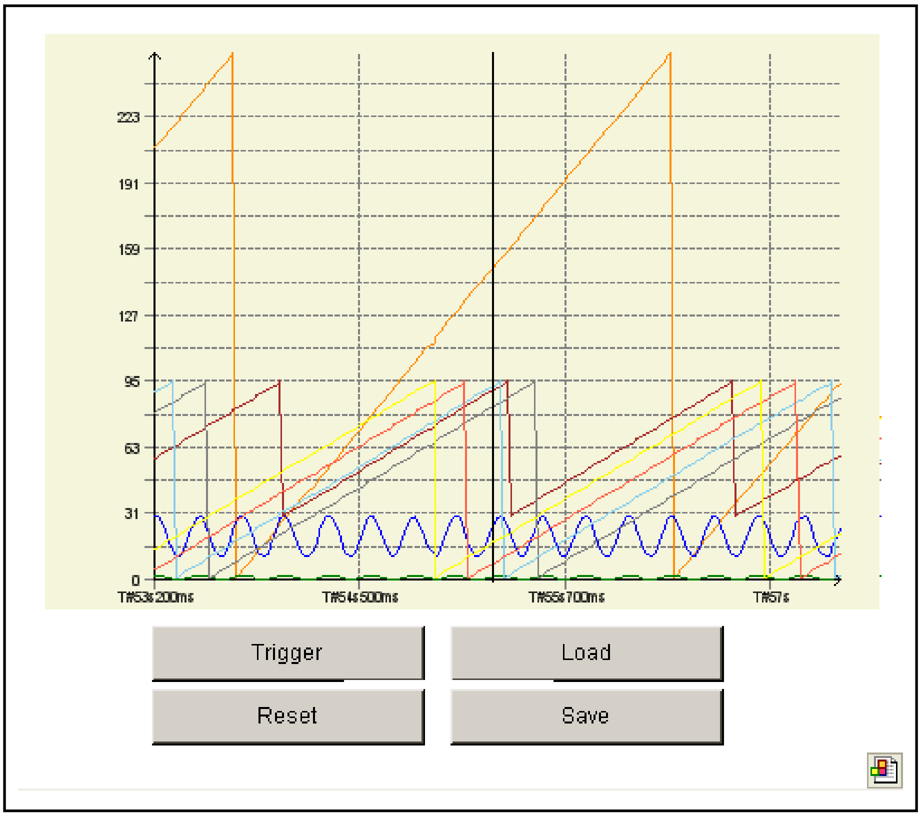
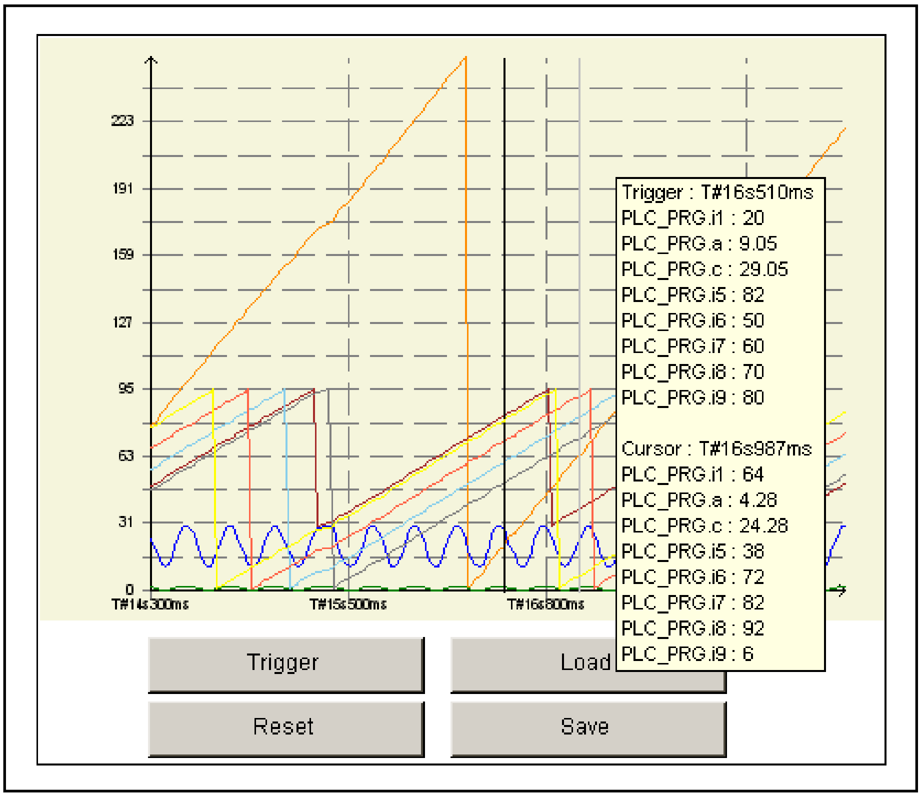


Fig.5-174: Visualization of a trace in online mode

A black vertical line marks the edge of the trigger variables. The recording of the variables continues up to a specified percentage value (TriggerPosition%) of the entire trace interval; then the visualization is stopped.

Activating the "ShowCursor" option in the element properties of the trace visualization element causes a tooltip to display the traced and trigger variables related to the position highlighted by the cursor in the trace visualization element.



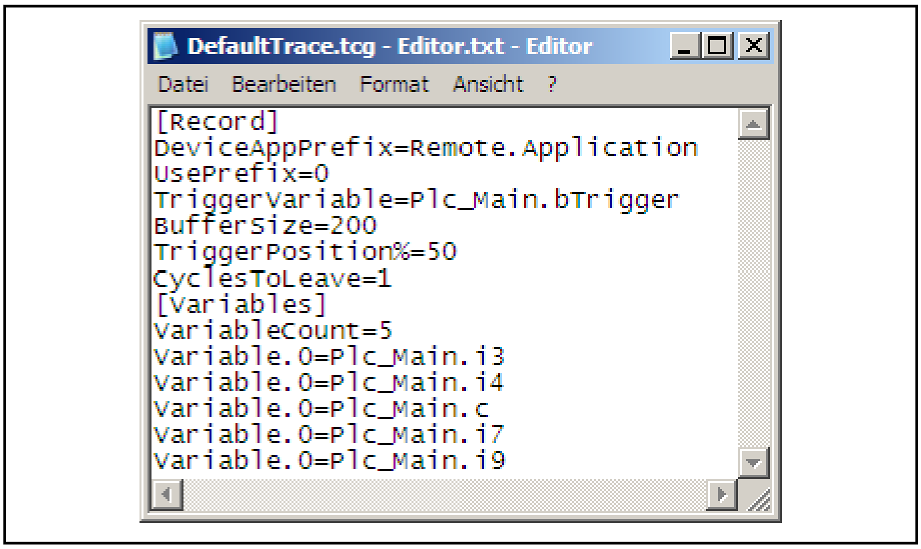
```
Trigger : T#16s510ms
PLC_PRG.i1 : 20
PLC_PRG.a : 9.05
PLC_PRG.c : 29.05
PLC_PRG.i5 : 82
PLC_PRG.i6 : 50
PLC_PRG.i7 : 60
PLC_PRG.i8 : 70
PLC_PRG.i9 : 80

Cursor : T#16s987ms
PLC_PRG.i1 : 64
PLC_PRG.a : 4.28
PLC_PRG.c : 24.28
PLC_PRG.i5 : 38
PLC_PRG.i6 : 72
PLC_PRG.i7 : 82
PLC_PRG.i8 : 92
PLC_PRG.i9 : 6
```

Fig.5-175: Visualization of a trace in online mode with the ShowCursor option

Trace configuration file

The trace configuration file "<TraceName>_<RecordName>.tcg" has to be stored in the directory of the runtime system, for example in "LW:\Programs \Rexroth\IndraWorks\GatewayPLC".



```
[Record]
DeviceAppPrefix=Remote.Application
UsePrefix=0
TriggerVariable=Plc_Main.bTrigger
BufferSize=200
TriggerPosition%=50
CyclesToLeave=1
[Variables]
VariableCount=5
Variable.0=Plc_Main.i3
Variable.0=Plc_Main.i4
Variable.0=Plc_Main.c
Variable.0=Plc_Main.i7
Variable.0=Plc_Main.i9
```

Fig.5-176: Trace configuration file

- [Record] section:

  To avoid having to enter each respective complete name for individual trace variables, set the "DeviceAppPrefix" to the same path percentage as the variables to be traced. This prefix is also displayed in the tooltip if the "UsePrefix" parameter is set to 1. If "UsePrefix" is set to 0, the prefix is hidden in the tooltip.

A detailed description of the TriggerVariable, BufferSize and TriggerPosition% entries can be found in trace editor, configuration, page 436.

The "CyclesToLeave" parameter corresponds with the "Record at every <x> – th cycle" parameter in the trace configuration.

- **[Variables] section:**

  The number of recorded variables can be specified in "VariableCount". Here, the number in the original trace may be maintained or decreased, but it may not be exceeded. All variables to be recorded have be made known with a symbol configuration. Alternatively, declare all of the variables to be recorded by using a data source located in the device that contains the trace.

5.5.18 Visualization Elements of Type "Windows Controls"

Elements overview

The "Windows Controls" category on the "ToolBox" tab contains the following elements:

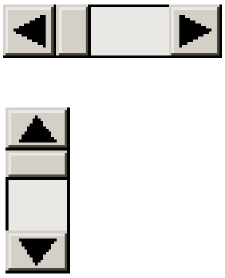
Element name	Element	Description
Table		The "table" element is used to display the values of an array, a structure or the local values of a function block. A column or line is used to display the elements of a one-dimensional array or a structure or a POU. Two-dimensional arrays and arrays that contain structures or POUs as elements are displayed in a matrix structure (columns and lines). A basic type variable can be understood as a one-dimensional array with an element and so it can also be displayed with the table elements (as a table with a single entry).
Text field		The "text field" element is used to display text that is either entered directly into the element properties or that comes from a text input variable. In contrast to the normal rectangle, the frame can be displayed as shadowed.
Scrollbar		The "Scrollbar" element generates a scrollbar for which minimum and maximum values can be defined. The position of the controller is then linked to the value of the input variables. If an output variable is defined, its value can be written by manually positioning the controller. By default, the scrollbar is horizontal, but if the shape of the element changes (by dragging with the mouse) after the scrollbar has been inserted such that the height of the element is greater than its width, the scrollbar switches to vertical.

Fig.5-177: Elements in the "Windows Control" category

Explanations on the table element can be found in examples under the following link: Table, page 650.

A detailed description can be found under Properties of these elements, page 451.

Configuring the table element

A table element can be inserted into a visualization to visualize one- or two-dimensional arrays, structures or the local variables of a POU.

The table is configured in the element properties window.

Insert the "table" visualization element into a visualization by dragging it from the Toolbox, page 446, and dropping it in the visualization.

First, enter the variable to be visualized with the table element in the "Data array" line.

Its structure determines the number of lines and columns in the table.

For an ARRAY:

- The lines represent the first dimension of the array entered.
- If the array is two-dimensional, the columns represent the second index.

For a structure: the individual components represent the structure.

For a POU: they represent the local variables.

A one-dimensional array of elements of the basic type is represented by a single column, whereas

an array of structure type elements is displayed in a table in which the number of lines corresponds with the number of array elements and the number of columns corresponds with the number of structure components.

If the dimensions of the linked variables are changed later, e. g. if the array boundaries are changed or if structure components are added or removed, the table can also be updated by clicking the <Return> key in a table field.

The configuration of the valid components remains.

A **table column can be duplicated** by holding down the <Ctrl> key while left-clicking in the title cell of the column to be duplicated.

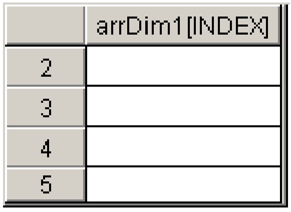
A **table column can be deleted** by holding down the <Ctrl> + <Shift> keys while left-clicking in the title cell of the column to be deleted.

☞ If the entry in the "Data array" field in the element properties is changed, all of the other settings for the element properties are reset to the default settings!

Display of table elements

- **Declaration of a one-dimensional field:**

  `arrDim1: ARRAY [2..5] OF INT;`



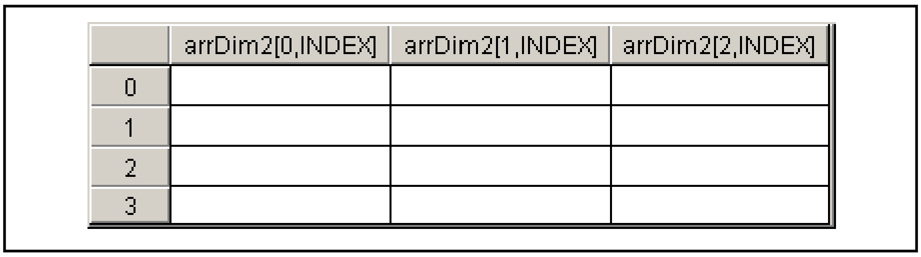
	arrDim1[INDEX]
2	
3	
4	
5	

Fig.5-178: "Table" visualization element as a one-dimensional array

Note that the indexing in the line labeling matches the indexes of the first dimension of the array; for this reason, the line numbering in this example starts with "2"!

- **Declaration of a two-dimensional field:**

```
arrDim2: ARRAY [0..2, 0..3] OF INT;
```

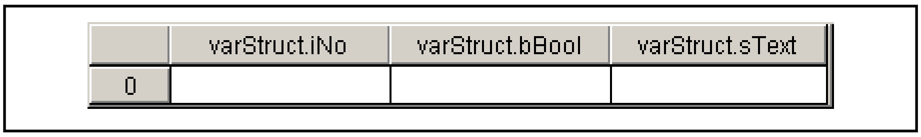


	arrDim2[0,INDEX]	arrDim2[1,INDEX]	arrDim2[2,INDEX]
0			
1			
2			
3			

Fig.5-179: "Table" visualization element as a two-dimensional array

- Type declaration for the structure "MyStruct" and its use:

```
TYPE MyStruct :
STRUCT
  iNo: INT;
  bBool: BOOL;
  sText: STRING;
END_STRUCT
END_TYPE
VAR
  varStruct: MyStruct;
END_VAR.
```

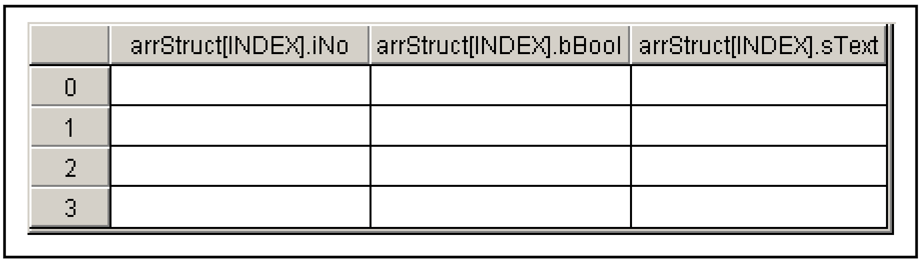


	varStruct.iNo	varStruct.bBool	varStruct.sText
0			

Fig.5-180: "Table" visualization element as a structure with a line

- Declaration of a field based on "MyStruct":

```
VAR
  arrStruct: ARRAY[0..3] OF MyStruct;
END_VAR.
```

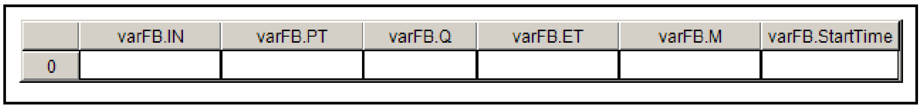


	arrStruct[INDEX].iNo	arrStruct[INDEX].bBool	arrStruct[INDEX].sText
0			
1			
2			
3			

Fig.5-181: "Table" visualization element as an ARRAY OF STRUCT with several lines

- Declaration of an FB instance:

```
VAR
  varFB: TON;
END_VAR.
```



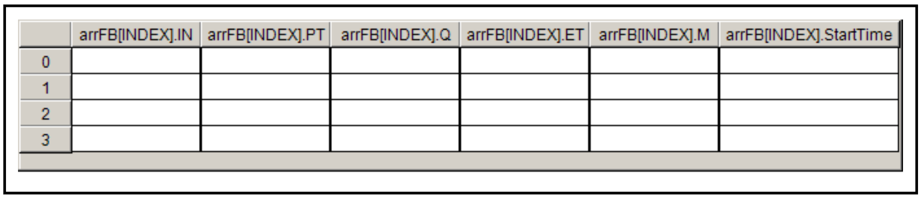
	varFB.IN	varFB.PT	varFB.Q	varFB.ET	varFB.M	varFB.StartTime
0						

Fig.5-182: "Table" visualization element as a one-dimensional function block

When calling a function block, all of the inputs and outputs have their own columns.

- **Declaration of a field of FB instances:**

```
VAR
  arrFB: ARRAY[0..3] OF TON;
END_VAR.
```



	arrFB[INDEX].IN	arrFB[INDEX].PT	arrFB[INDEX].Q	arrFB[INDEX].ET	arrFB[INDEX].M	arrFB[INDEX].StartTime
0						
1						
2						
3						

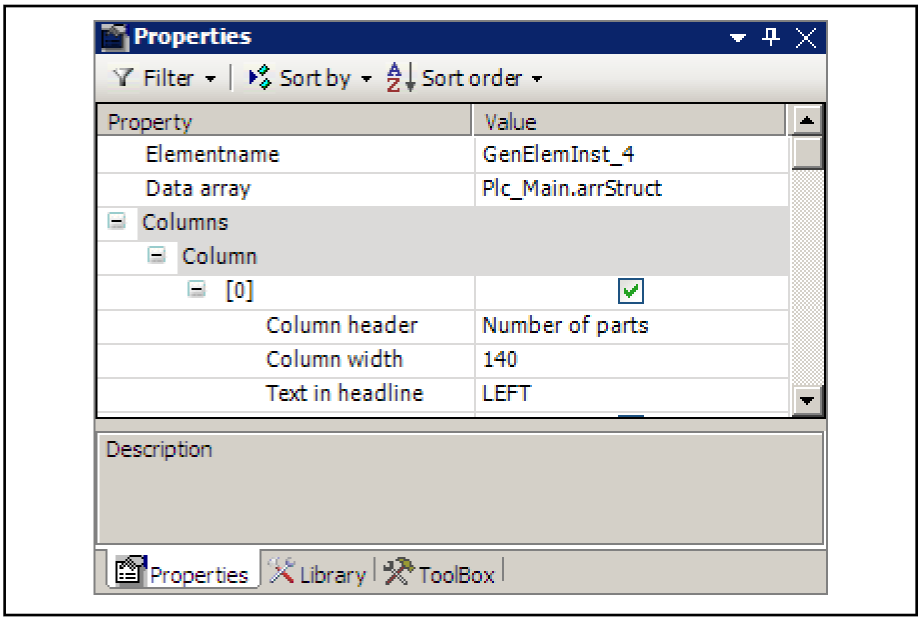
Fig.5-183: "Table" visualization element as a two-dimensional function block

In addition to a few basic properties, e. g. size, position or line width in the table, a wide variety of specific properties can be set for the table. A selection of these properties are explained further here using a step-by-step continuation of the "arrStruct" example.

Properties of a table elements

A selection of these properties are explained further here using a step-by-step continuation of the "arrStruct" example.

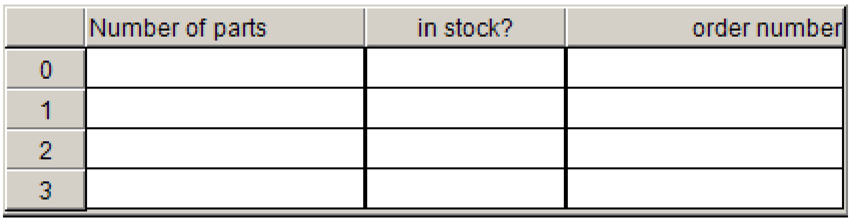
1. In the element properties, change the default entries in the "Column header" column to more meaningful titles, adjust the width of the columns and specify the alignment of the heading.



Property	Value
Elementname	GenElemInst_4
Data array	Plc_Main.arrStruct
⊟ Columns	
⊟ Column	
⊟ [0]	☑
Column header	Number of parts
Column width	140
Text in headline	LEFT

Fig.5-184: Specifying column width, heading and alignment

The effects of these changes:

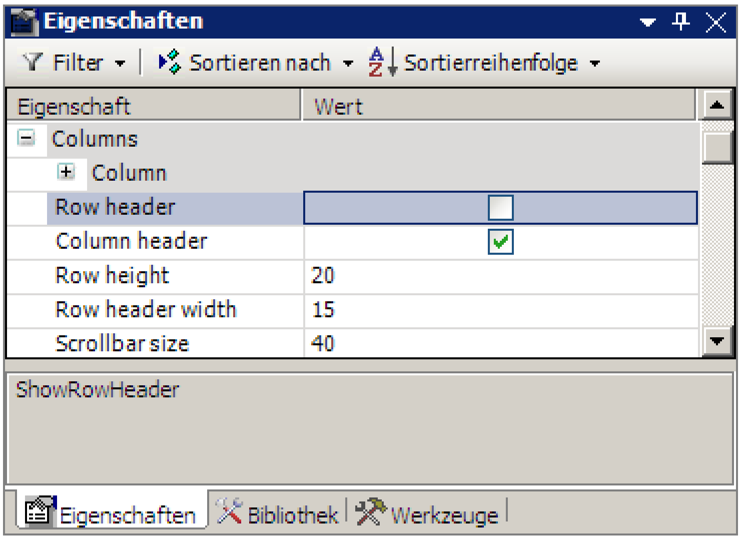


	Number of parts	in stock?	order number
0			
1			
2			
3			

Fig.5-185: Column heading as defined

2. Specify the height of the lines as 20 pixels in the "Row height" line. Switch off the line labeling by removing the checkmark from the "Row header" line, so that now the first column that contains the array indexes is missing.

   The following figure also shows the wider scrollbars (line: "Scrollbar size"), which are inserted in the table as soon as its size decreases so that it is less than the comprehensive line heights or column widths.

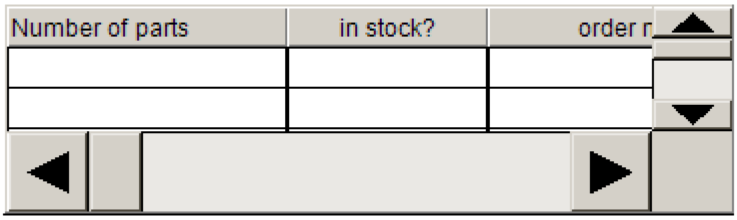


Eigenschaft	Wert
⊟ Columns	
⊞ Column	
Row header	□
Column header	☑
Row height	20
Row header width	15
Scrollbar size	40

ShowRowHeader

Fig.5-186: Specifying column height and scrollbars; switching off line labeling

The effects of these changes:



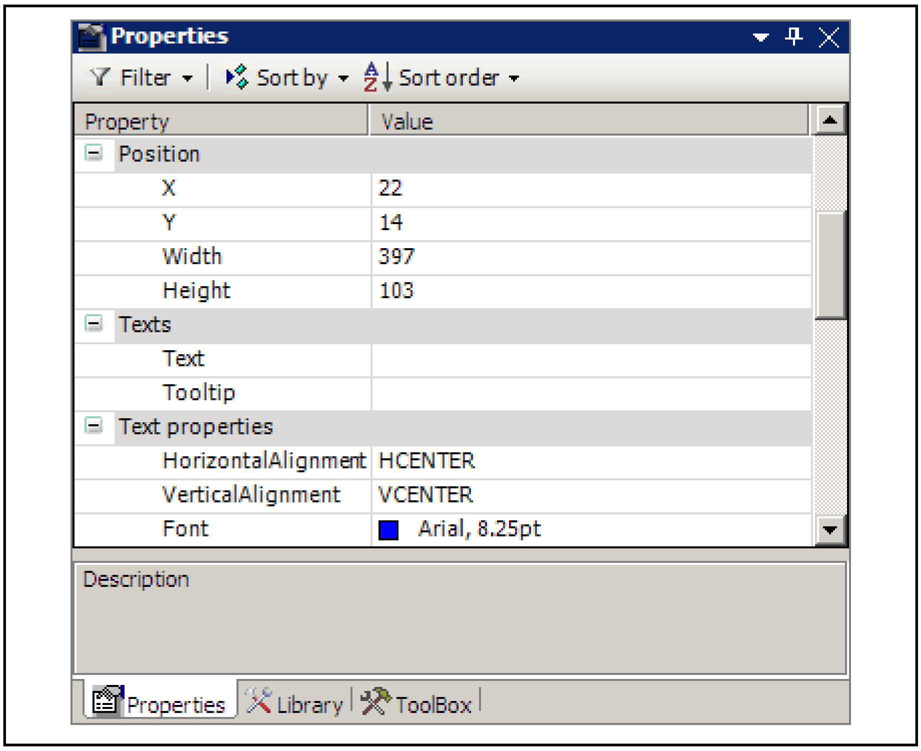
Number of parts	in stock?	order r

Fig.5-187: Expanded column options

3. The position of the "table" element is specified in the "Position" element property in the X and Y lines or by dragging with the mouse to the desired position.

   The scrollbars can be hidden with the "WITH" and "High" lines by increasing the default values. Their height is determined by the line height multiplied by the number of lines. The width of the table is the sum of the individual column widths and the width of the line labeling (if it is active).

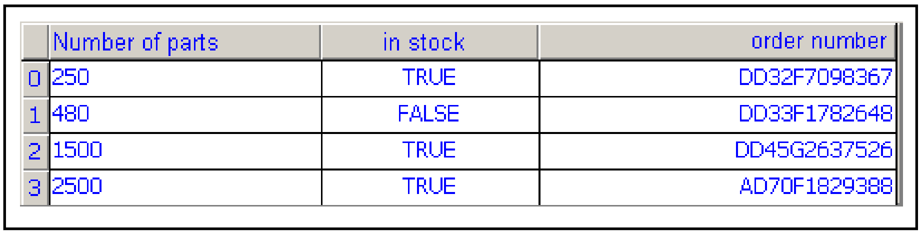
The text display is specified in the "Text properties" element property. In the "Font" line, use the ... button to define the font of the display. The horizontal text alignment in the "HorizontalAlignment" line only applies for line labeling. The horizontal text alignment in the "VerticalAlignment" line only applies for column labeling. The alignment of the texts in individual table cells is based on the settings made in the column properties.



Property	Value
⊟ Position	
X	22
Y	14
Width	397
Height	103
⊟ Texts	
Text	
Tooltip	
⊟ Text properties	
HorizontalAlignment	HCENTER
VerticalAlignment	VCENTER
Font	🟦 Arial, 8.25pt

Fig.5-188: Hiding scrollbars; text options

The appearance of the (online) display of the table has now changed.



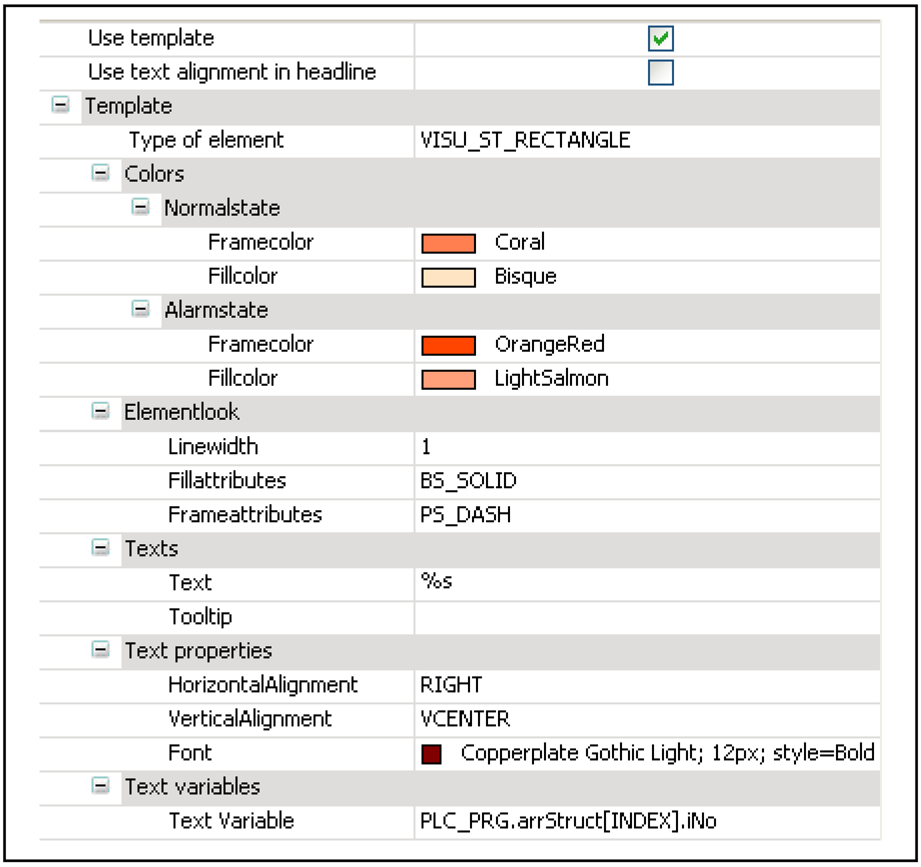
	Number of parts	in stock	order number
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Fig.5-189: Online display

Define the values in the columns and lines in your program.

Note that the alignment of the entries in the table cells corresponds with the setting for the individual columns.

4. Table fields can be edited line by line. For the column selected, select the "Use template" checkbox: Then, another "Template" point is inserted and expands in the column properties.



Use template	☑
Use text alignment in headline	□
⊟ Template	
Type of element	VISU_ST_RECTANGLE
⊟ Colors	
⊟ Normalstate	
Framecolor	■ Coral
Fillcolor	■ Bisque
⊟ Alarmstate	
Framecolor	■ OrangeRed
Fillcolor	■ LightSalmon
⊟ Elementlook	
Linewidth	1
Fillattributes	BS_SOLID
Frameattributes	PS_DASH
⊟ Texts	
Text	%s
Tooltip	
⊟ Text properties	
HorizontalAlignment	RIGHT
VerticalAlignment	VCENTER
Font	■ Copperplate Gothic Light; 12px; style=Bold
⊟ Text variables	
Text Variable	PLC_PRG.arrStruct[INDEX].iNo

Fig.5-190: Expanded options for editing table fields line by line

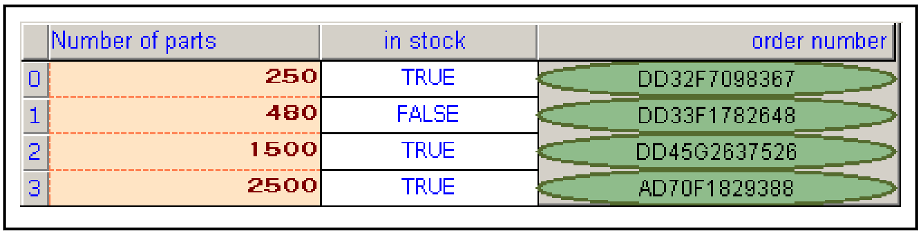
In the "Type of element" element property, three types are available in the menu for displaying the cells:

- VISU_ST_RECTANGLE (rectangle)
- VISU_ST_ROUNDRECT (rectangle with rounded corners)
- VISU_ST_CIRCLE (circular)

The related configuration possibilities show the element properties for the selected template element. Select the fill and frame colors for the normal and alarm states. They can also be toggled with a variable. The alignment of the entries in the column cells is now specified by the template setting (with the exception of the heading).

Instead of the array components entered by default, any other variable in the project can be entered in the "Text variables" field. In this way it is possible to display the array elements in the table in a different order.

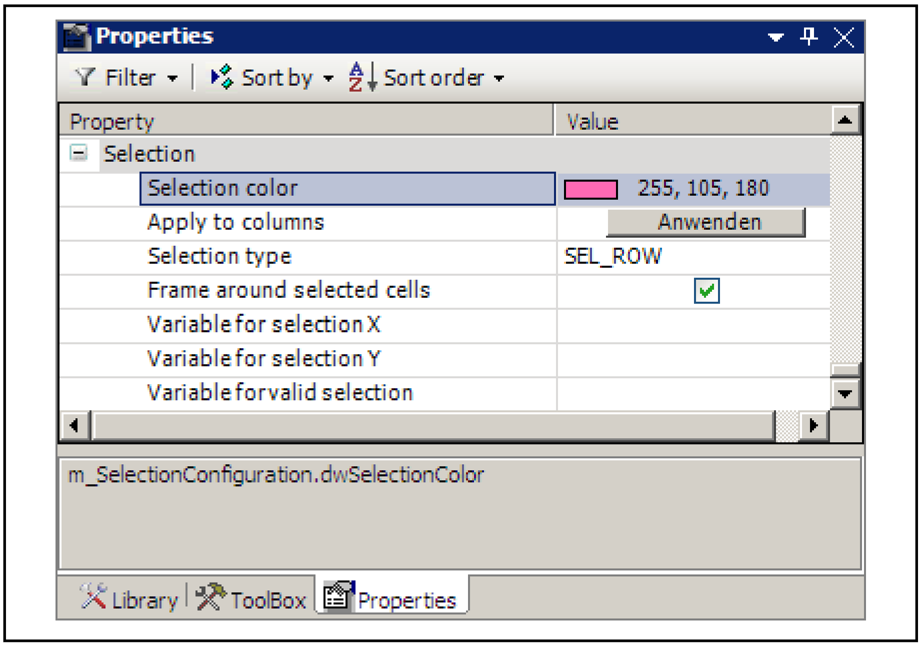
In the example, a template was used for the first and third columns.



	Number of parts	in stock	order number
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Fig.5-191: Example configuration with templates

5. The alarm state cannot only be triggered by a variable value, but also by highlighting a table cell by clicking on it in online mode. The behavior of the selected region can be specified in the "Selection" section.



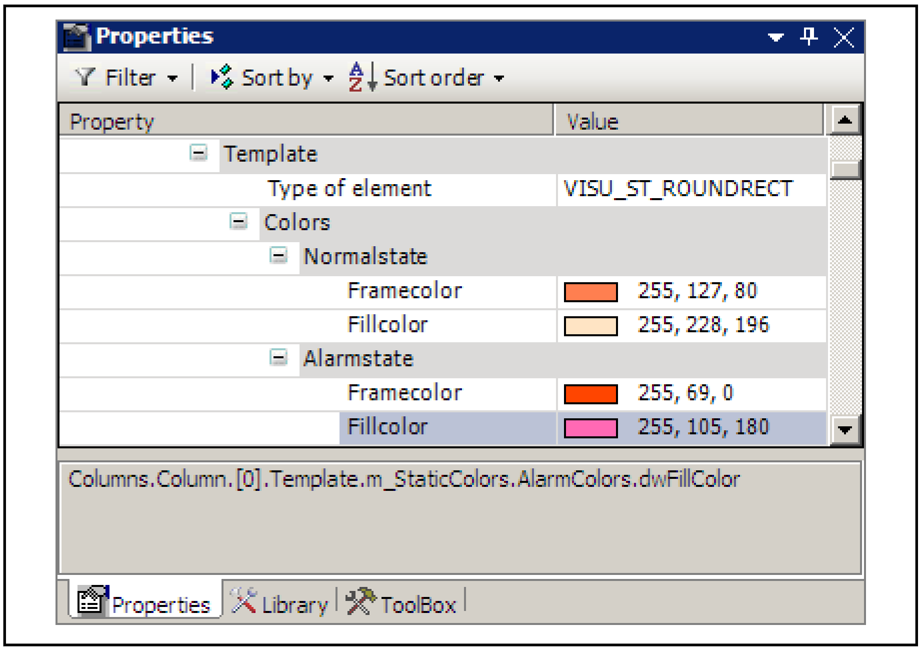
Property	Value
⊟ Selection	
Selection color	🟪 255, 105, 180
Apply to columns	Anwenden
Selection type	SEL_ROW
Frame around selected cells	☑
Variable for selection X	
Variable for selection Y	
Variable for valid selection	

m_SelectionConfiguration.dwSelectionColor

Fig.5-192: Defining alarm color in online mode

The specified background color in the "Selection color" line for highlighted elements applies for all highlighted elements for which no alarm color was specified (within a template). So in our example, a highlighted cell in the second column is filled with hot pink, while a highlighted cell in the first column is filled with light salmon, the alarm color of the associated template.

After selecting the color, click on <Apply> to replace the alarm colors in the specific template settings with the selection made in the "Selection color" field. As a result, the following entries are made in the corresponding fields in the template for the first column of our example:



Property	Value
⊟ Template	
Type of element	VISU_ST_ROUNDRECT
⊟ Colors	
⊟ Normalstate	
Framecolor	🟧 255, 127, 80
Fillcolor	🟧 255, 228, 196
⊟ Alarmstate	
Framecolor	🟥 255, 69, 0
Fillcolor	🟪 255, 105, 180

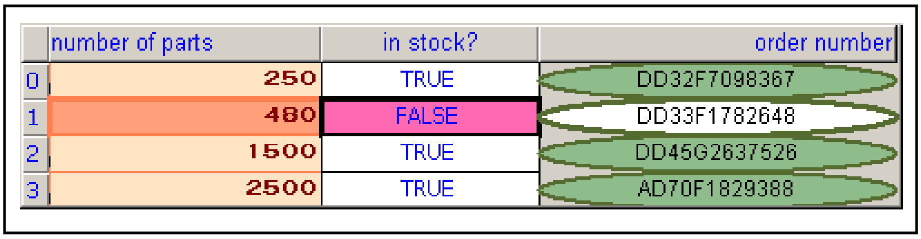
Columns.Column.[0].Template.m_StaticColors.AlarmColors.dwFillColor

Fig.5-193: Alarm colors accepted into the template

The "Selection type" field specifies which elements are to be highlighted by clicking on a single table cell. In this example, the SEL_ROW type was selected, and so all of the cells that are in the same line as the cell that was clicked are highlighted and framed.

If no columns of the display in the table were excluded, the position of the selected cells within the table represents the indexes of the visualized array associated with the entry. These indexes can be saved in variables that are entered in the last three lines of the "Selection" section for this purpose. The third variable is Boolean and is set to TRUE as long as the selection is valid.

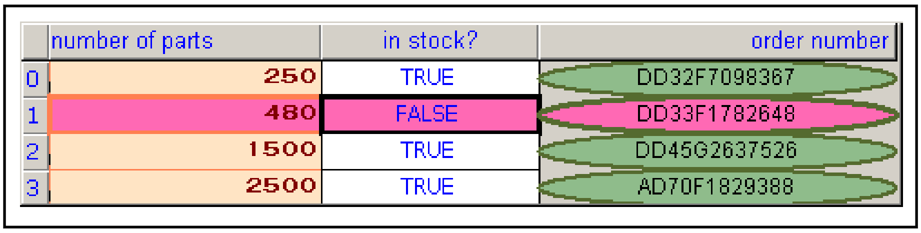
After the cell with the entry "FALSE" is clicked, all of the cells in the second line are highlighted. Only the highlighted cell in the second column has the selected color because the color for the alarm state in the other columns is set by the templates.



	number of parts	in stock?	order number
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Fig.5-194: Selected color for alarm state

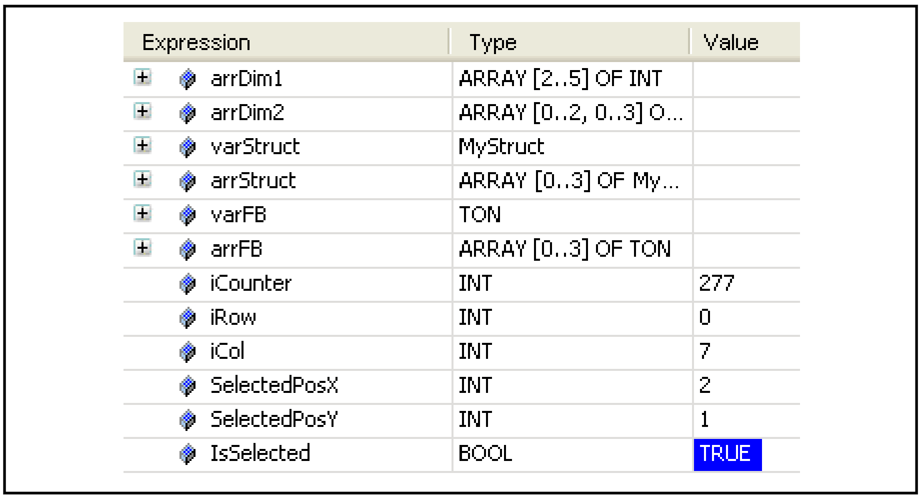
After applying the selected color for the templates as well, the highlighting looks different.



	number of parts	in stock?	order number
0	250	TRUE	DD32F7098367
1	480	FALSE	DD33F1782648
2	1500	TRUE	DD45G2637526
3	2500	TRUE	AD70F1829388

Fig.5-195: Identical colors for the alarm state in all of the columns

The array index related to the selected cell is assigned to the associated variables as a value.



Expression	Type	Value
arrDim1	ARRAY [2..5] OF INT	
arrDim2	ARRAY [0..2, 0..3] O...	
varStruct	MyStruct	
arrStruct	ARRAY [0..3] OF My...	
varFB	TON	
arrFB	ARRAY [0..3] OF TON	
iCounter	INT	277
iRow	INT	0
iCol	INT	7
SelectedPosX	INT	2
SelectedPosY	INT	1
IsSelected	BOOL	TRUE

Fig.5-196: Variables in the online view

5.5.19 Keyboard Operation in Online Mode

Some standard shortcuts are supported by every device for keyboard operation in a visualization in online mode. In addition, depending on the device, other key (combinations) can be used.

Default shortcuts:

Key(s)	Action
<Tab>	The next element according to the chronological sequence of insertion is selected; in this case, within a table, each individual cell is selected; if a frame element is selected, the selection is forwarded to the individual elements contained
<Shift>+ <Tab>	Previous element is selected; reverse order than with <Tab>
<Enter>	Input action executed for the selected element
<Arrow keys>	Next element in the direction indicated by the arrow key is selected

Fig.5-197: Shortcuts

For the integrated visualization, page 634, keyboard operation can be activated and deactivated explicitly using the command Activate Keyboard Operation, page 306,. This can be desirable because as long as keyboard operation is activated for the visualization, other commands that are given using shortcuts are not executed.

Device-specific keyboard operation:

In addition to the standard keys, it depends on the device which other key (combinations) can be used for the visualizations that run on the device. See Standard shortcuts in the Visualization Manager, page 498 and Keyboard configuration in the visualization editor, page 494.

There is an option for handling keyboard events in the application code.

5.5.20 Note Events and Input Actions

Overview

Using methods provided by the visualization libraries, the application can take notice of specific events triggered by user inputs in a visualization:

- Note the closing of an Edit Controls (writing a variable), page 658,
- Note keyboard events, page 661.

Note the Closing of an Edit Controls (Writing a Variable)

The application can determine when an "Edit Control" is closed in an associated visualization, i.e. when a variable value is written due to user input in an input field. This can be useful if another action is to be executed after an Edit Control is closed.

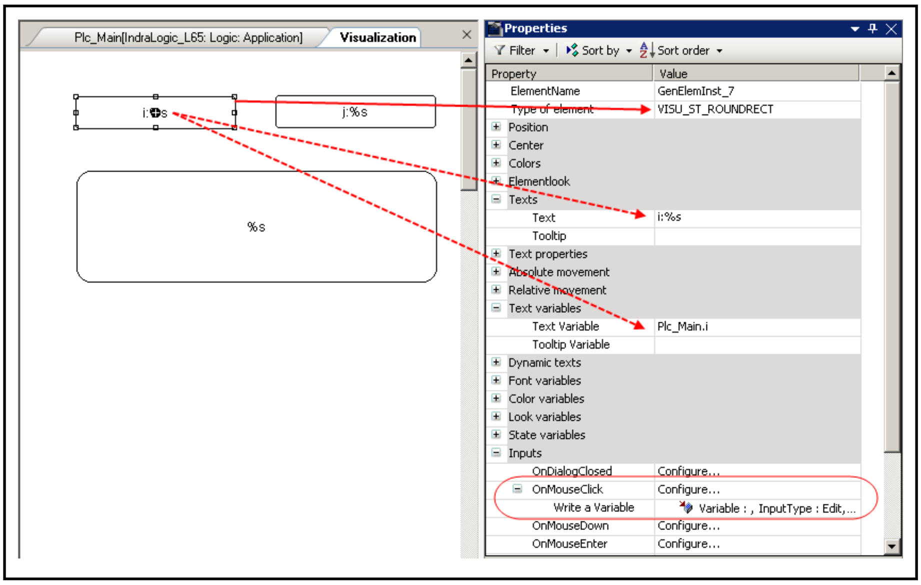
To receive information about the closing, a function block instance (that the interface VisuElems.IEditBoxInputHandler from the VisuElemBase.library implements) can be assigned to the global instance VisuElems.Visu_Globals.g_VisuEventManager; in this case, the method SetEditBoxEventHandler of this instance is used as shown in the example below.

VisuElems.IEditBoxInputHandler requires the method VariableWritten, which is called after a variable value is written due to user input in a visualization.

Example:

There is a visualization with the following elements:

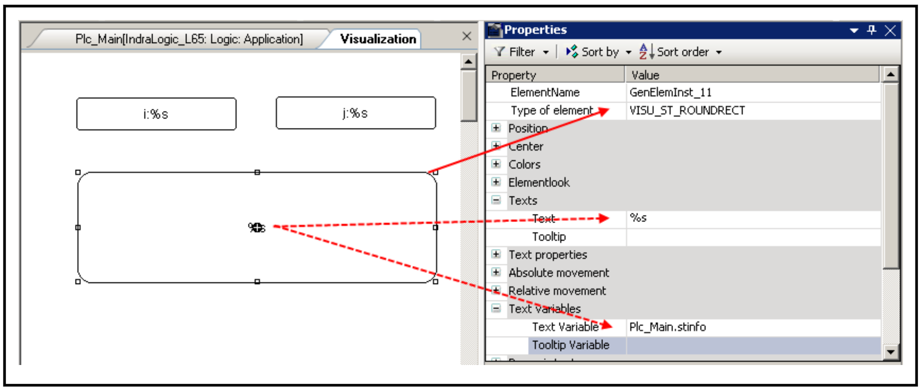
1. two rectangles, each with an Edit Control field (input "Write variable") in which the user can input a value for variable i or j.



Property	Value
ElementName	GenElemInst_7
Type of element	VISU_ST_ROUNDRECT
Position	
Center	
Colors	
Elementlook	
Texts	
Text	i:%s
Tooltip	
Text properties	
Absolute movement	
Relative movement	
Text variables	
Text Variable	Plc_Main.i
Tooltip Variable	
Dynamic texts	
Font variables	
Color variables	
Look variables	
State variables	
Inputs	
OnDialogClosed	Configure...
OnMouseClick	Configure...
Write a Variable	Variable : , InputType : Edit,...
OnMouseDown	Configure...
OnMouseEnter	Configure...

Fig.5-198: Input elements

2. Another rectangle in which a text (text variable stInfo) with its own information is shown if one of the input fields in either of the two upper rectangles is closed, i.e. if a value is written.



Property	Value
ElementName	GenElemInst_11
Type of element	VISU_ST_ROUNDRECT
Position	
Center	
Colors	
Elementlook	
Texts	
Text	%s
Tooltip	
Text properties	
Absolute movement	
Relative movement	
Text variables	
Text Variable	Plc_Main.stinfo
Tooltip Variable	

Fig.5-199: Text output element

The application contains the following blocks:

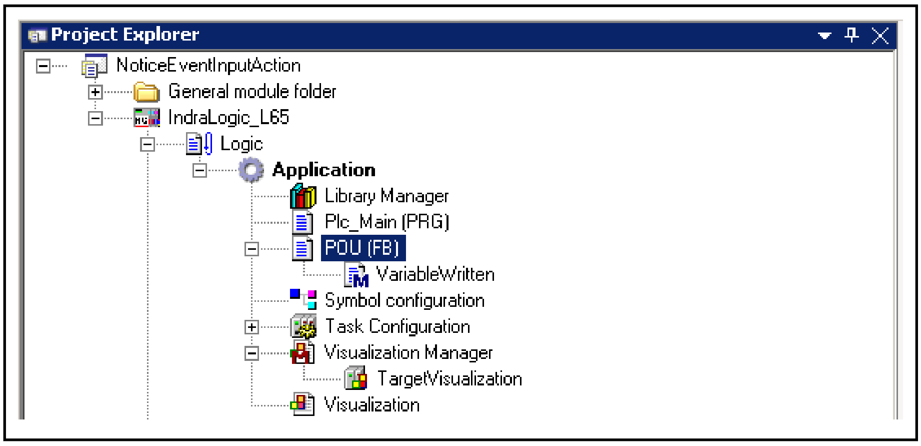


Fig.5-200: Application objects in the Project Explorer

PROGRAM Plc_Main

```
PROGRAM Plc_Main
VAR_INPUT
  i:INT;          // variable to be written by user input in visualization
  j:REAL;         // variable to be written by user input in visualization
  stInfo : STRING; (* information on the user input via the edit control field;
                     string gets composed by method VariableWritten;
                     result is displayed in the lower rectangle of the visualization *)
END_VAR
VAR
 inst : POU;
 bFirst : BOOL := TRUE;
END_VAR

 IF bFirst THEN
 bFirst := FALSE;
 VisuElems.Visu_Globals.g_VisuEventManager.SetEditBoxEventHandler(inst);
                  // Call of method VariableWritten
END_IF
```

Function block POU

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IEditBoxInputHandler
```

(no further declarations, no implementation code)

The method "VariableWritten" provides information when an Edit Control in the visualization is closed, i.e. when a variable is written based on user input in one of the upper rectangles, to the text variable "stinfo", which is to be displayed in the lower rectangle.

The text variable is combined.

Method VariableWritten, part of POU

```
METHOD VariableWritten : BOOL (* provides some information
       always when an edit control field is closed in the visualization,
       i.e. a variable gets written by user input in one of the upper rectangles *)
VAR_INPUT
 pVar : POINTER TO BYTE;
 varType : VisuElems.Visu_Types;
 iMaxSize : INT;
 pClient : POINTER TO VisuElems.VisuStructClientData;
END_VAR
// String stinfo, which will be displayed in the lower rectangle, is composed here
Plc_Main.stInfo := 'Variable written; type: ';
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, INT_TO_STRING(varType));
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, ', adr: ');
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, DWORD_TO_STRING(pVar));
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, ', by: ');
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, SEL(pClient^.globaldata.clienttype =
 VisuElems.Visu_ClientType.Targetvisualization, 'other visu', 'targetvisu'));
```

Note keyboard events

The application can specify key actions (keyboard event) in an associated visualization, i.e. it notices when the user **presses and releases a key** when the visualization is active.

☞ Note the configuration possibilities for keyboard operation in visualizations.

To receive information about such events, a function block (that the interface VisuElems.IVisuUserEventManager from the VisuElemBase.library implements) can be assigned to the global instance VisuElems.Visu_Globals.g_VisuEventManager and the method **SetKeyEventHandler** of this instance is used as shown in the following example:

Example: There is a visualization with a visualization element that displays the value Plc_Main.stInfo as text. That means that in online operation, the following information is output in this element on the currently activated key action:

"KeyEvent up: <dwKey>,

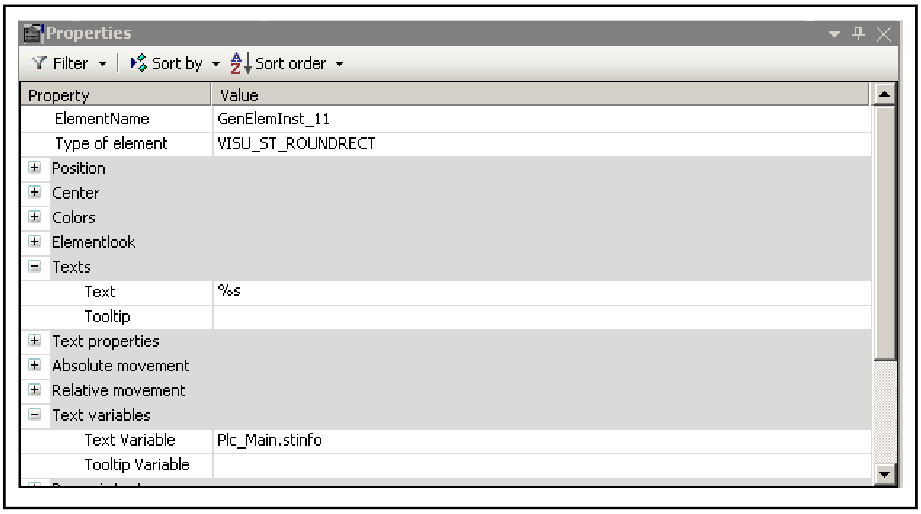
key: <key id>,

modifier: <dwModifiers>,

by: <pClient^.globaldata.clienttype.....>"

.

Example: "KeyEvent up: TRUE, key: 75, modifier: 0, by: targetvisu".



Property	Value
ElementName	GenElemInst_11
Type of element	VISU_ST_ROUNDRECT
⊞ Position	
⊞ Center	
⊞ Colors	
⊞ Elementlook	
⊟ Texts	
Text	%s
Tooltip	
⊞ Text properties	
⊞ Absolute movement	
⊞ Relative movement	
⊟ Text variables	
Text Variable	Plc_Main.stinfo
Tooltip Variable	

Fig.5-201: Text configuration of the visualization element

Programming Reference

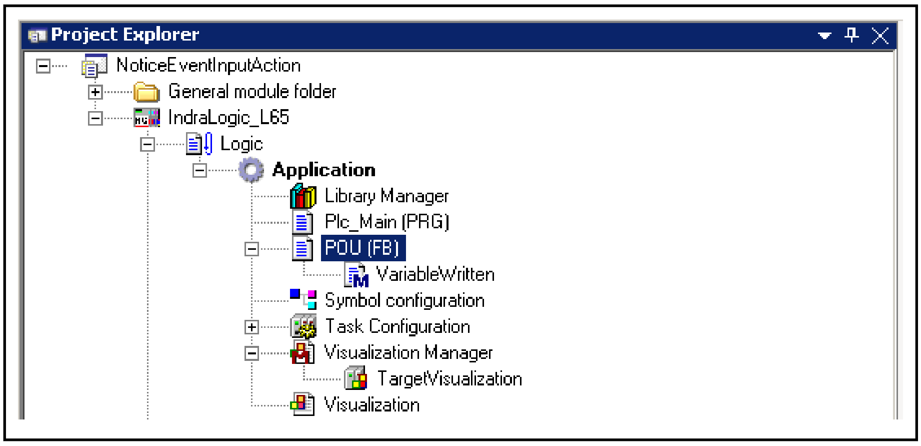


Fig.5-202: Application objects in the Project Explorer

PROGRAM Plc_Main

```
PROGRAM Plc_Main
VAR_INPUT
  stInfo : STRING;
END_VAR
VAR_OUTPUT
     g_VisuEventManager: INT;
END_VAR
VAR
 inst : POU;
 bFirst : BOOL := TRUE;
END_VAR
 IF bFirst THEN
 bFirst := FALSE;
 VisuElems.Visu_Globals.g_VisuEventManager.SetKeyEventHandler(inst);
END_IF
```

This makes it possible to determine when a keyboard event is triggered.

Function block POU

```
FUNCTION_BLOCK POU IMPLEMENTS VisuElems.IKeyEventHandler
```

(no further declarations, no implementation code)

The interface VisuElems.IVisuUserEventManager requires a method with the following signature that is called with all of the available information:

Method HandleKeyEvent, assigned to FB POU

```
   // This method will be called after a key event is released.
   // RETURN:
   // TRUE -  When the handler has handled this event and it should not be handled by someone else
   // FALSE - When the event is not handled by this handler
METHOD HandleKeyEvent : BOOL
VAR_INPUT
   // Event type. The value is true if a key-up event was released.
 bKeyUpEvent : BOOL;
   // Key code
 dwKey : DWORD;
   // Modifier. Values:
   // VISU_KEYMOD_SHIFT : DWORD := 1;
   // VISU_KEYMOD_ALT : DWORD := 2;
   // VISU_KEYMOD_CTRL : DWORD := 4;
 dwModifiers : DWORD;
   // Pointer to the client structure were the event was released
 pClient : POINTER TO VisuStructClientData;
END_VAR
VAR
END_VAR
Plc_Main.stInfo := 'KeyEvent up: ';
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, BOOL_TO_STRING(bKeyUpEvent));
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, ', key: ');
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, DWORD_TO_STRING(dwKey));
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, ', modifier: ');
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, DWORD_TO_STRING(dwModifiers));
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, ', by: ');
Plc_Main.stInfo := CONCAT(Plc_Main.stInfo, SEL(pClient^.globaldata.clienttype =
 VisuElems.Visu_ClientType.Targetvisualization, 'other visu', 'targetvisu'));
```

5.5.21 Input Dialogs

A visualization can be created as an input dialog and defined in its object properties as a "dialog" (visualization object, context menu**Properties...** ▸ **Visualization**, "Visualization used as:" dialog).

Dialog visualizations can then be used in other visualizations to provide the user with an input mask. Dialog visualizations are offered when configuring an "OpenDialog" or "OnDialogClosed" property of a visualization element, for example.

In addition to the dialog visualizations created by yourself, the standard login dialog visualization provided with the **VisuDialogs.library** library can always be used.

The parameters defined in the interface of a dialog visualization are automatically written in a

`<DialogName>_VISU_STRUCT`

("DialogName" = Name of the visualization) structure; see Interface editor for visualizations, page 491.

This structure can then be used by the application, e. g. using a function that is called if there is user input for a visualization element (configuration of the visualization element). In this way, opening a dialog in a visualization and the reaction to user input in this dialog can be programmed.

DialogManager

All visualizations that are configured as dialogs are automatically instanced and managed by the internal DialogManager.

This manager can be addressed by the VisuManager, which is also internal, by calling the **GetDialogManager()** method.

The DialogManager provides some methods for the application to handle a visualization dialog:

The following provides an example for configuring a login dialog:

Configuration of a login dialog

In this example, a button element in a visualization is configured so that if the "Login" button is clicked, a login dialog opens, in which a user name and a password can be entered. The default login dialog is provided by the "Visu-Dialog.library", but a self-created dialog could be handled in the same way:

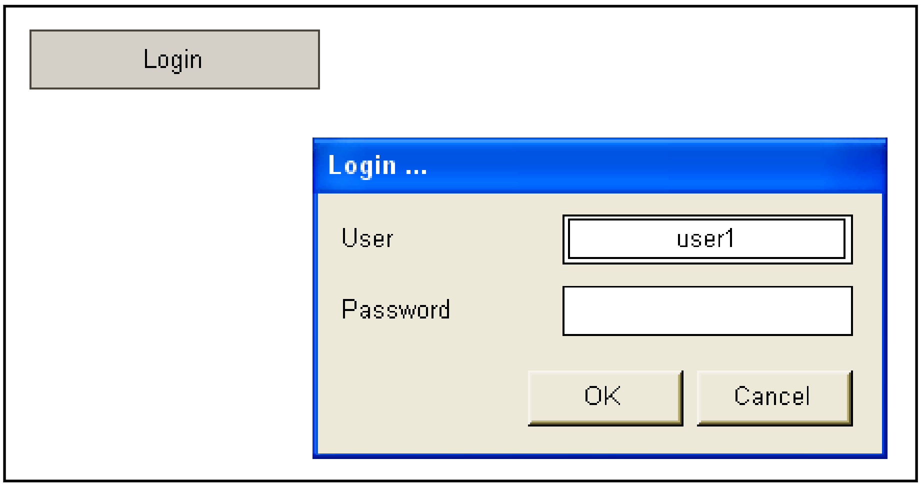


Fig.5-203: Login dialog

A button element is added to a visualization. It is configured with the text property "Login" and the following two input properties:

Configuring the "Login" button:

1. In your project, add a visualization to an application.

   Then add a function to the global "General module" folder and name the function "OpenLoginDialog".

   Add a second function "OnLoginDialogClosed".

   The following is now displayed in Project Explorer:

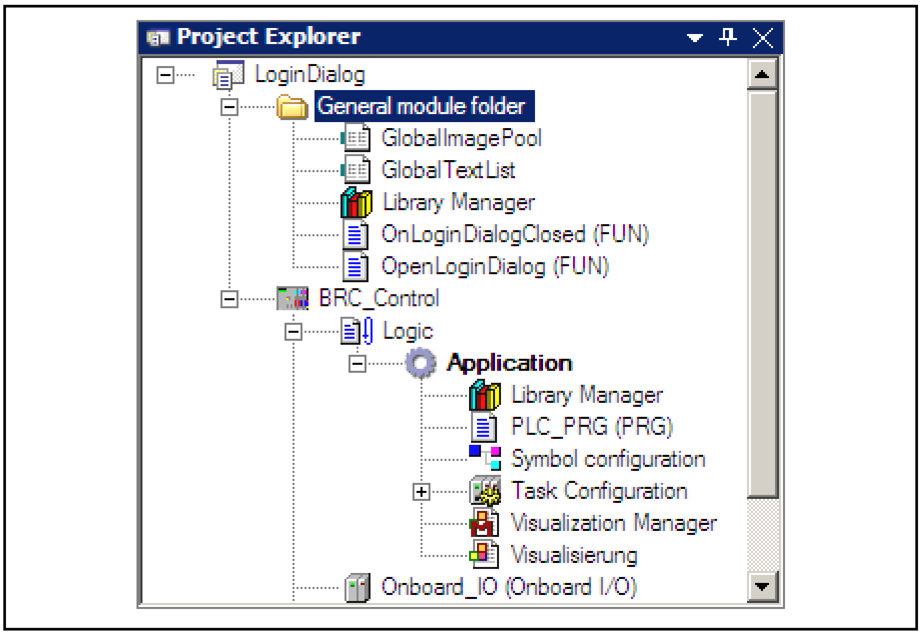


Fig.5-204: Project LoginDialog in Project Explorer

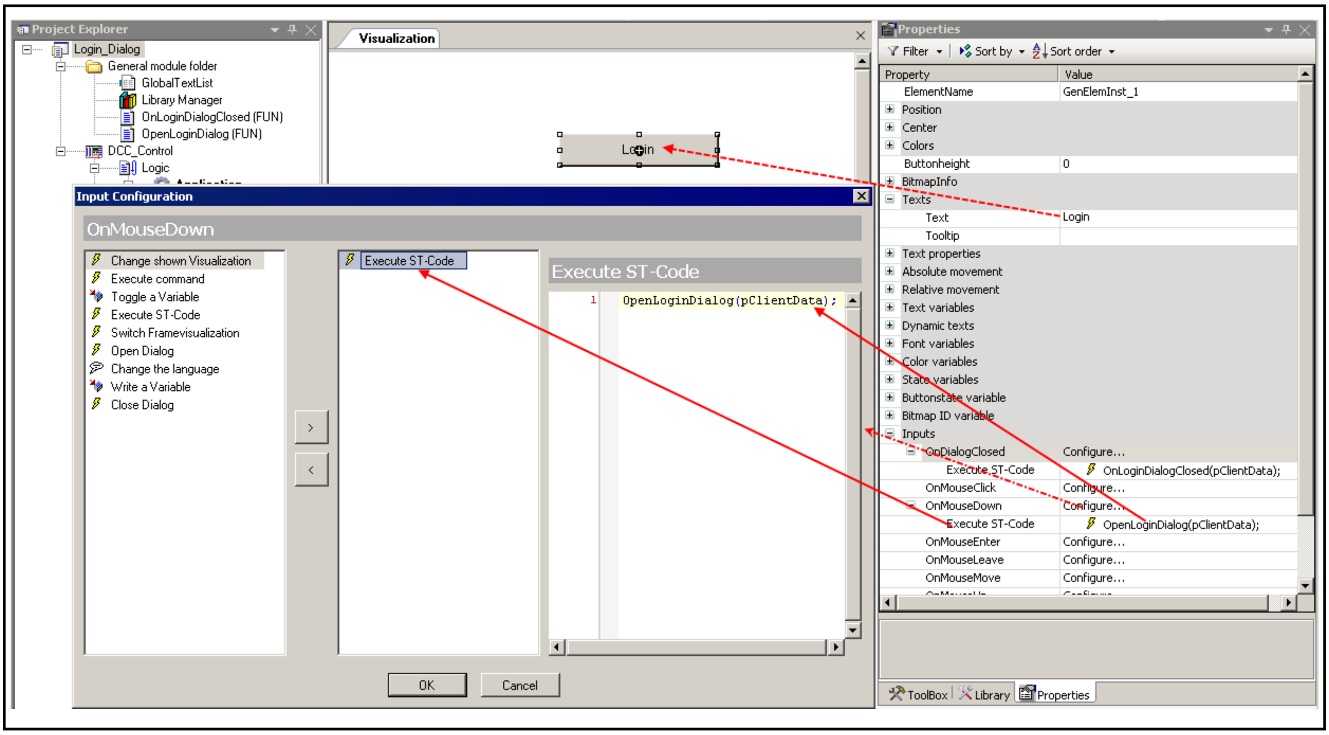
- The function "OpenLoginDialog" is called when the Login dialog opens, e. g. during an OnMouseDown action on the button element. This function reads the parameter of the open Login dialog.

  An example for a possible implementation can be seen in Function OpenLoginDialog, page 665.

- The function "OnLoginDialogClosed" defines the reaction to the closing of the Login dialog.

  An example for a possible implementation can be seen in Function OpenLoginDialogClosed, page 666.

2. Add the "Button" element to a visualization. Name the "Button" element "Login" in the "Text" property. In the "Inputs" property, configure the following inputs.



Execute ST-Code

```
OpenLoginDialog(pClientData);
```

Property	Value
ElementName	GenElemInst_1
Position	
Center	
Colors	
Buttonheight	0
BitmapInfo	
Texts	
Text	Login
Tooltip	
Text properties	
Absolute movement	
Relative movement	
Text variables	
Dynamic texts	
Font variables	
Color variables	
State variables	
Buttonstate variable	
Bitmap ID variable	
Inputs	
OnDialogClosed	Configure...
Execute ST-Code	OnLoginDialogClosed(pClientData);
OnMouseClick	Configure...
OnMouseDown	Configure...
Execute ST-Code	OpenLoginDialog(pClientData);
OnMouseEnter	Configure...
OnMouseLeave	Configure...
OnMouseMove	Configure...

Fig.5-205: Configuring the "Login" button

2.1 OnMouseDown / Execute ST code:

Function call '**OpenLoginDialog(pClientData)**'.

The variable 'pClientData' has to be transferred to a respective 'OpenLoginDialog()' function existing in the project.

2.2 OnDialogClosed / Close dialog / Execute ST code:

Function call '**OnLoginDialogClosed(pClientData)**'.

The respective functions 'OpenLoginDialog()' and 'OnLoginDialogClosed()' are present as function blocks in the project.

See a possible implementation as follows:

Implementation example:

The function 'OpenLoginDialog()' is called when the dialog opens, e.g. during an OnMouseDown action on the button element. This function reads the parameter of the open dialog.

Function OpenLoginDialog

```
FUNCTION OpenLoginDialog : BOOL
VAR_INPUT
 pClientData : POINTER TO VisuStructClientData;
END_VAR
VAR
 dialogMan : IDialogManager;
 loginDialog : IVisualisationDialog;
 pLoginInfo : POINTER TO Login_VISU_STRUCT;
 (* Login_VISU_STRUCT contains the parameters that are
   defined in the interface of the visualisation "Login" *)
 result : Visu_DialogResult;
 stTitle : STRING := 'Login ...';
 stPasswordLabelText: STRING;
 stUserLabelText: STRING;
 stUsername: STRING;
END_VAR
dialogMan := g_VisuManager.GetDialogManager();
 (* The DialogManager is provided by the VisuManager that is
   implicitly available *)
```

Programming Reference

```
IF dialogMan <> 0 AND pClientData <> 0 THEN
  loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); (* The dialog to be opened is indicated *)
  IF loginDialog <> 0 THEN
    pLoginInfo := dialogMan.GetClientInterface(loginDialog, pClientData);
    IF pLoginInfo <> 0 THEN      (* Now the parameters of the Login dialog of
   Login_VISU_STRUCT are read *)
      pLoginInfo^.stTitle := stTitle;
      pLoginInfo^.stPasswordLabelTxt := stPasswordLabelText;
      pLoginInfo^.stUserLabelTxt := stUserLabelText;
      dialogMan.OpenDialog(loginDialog, pClientData, TRUE, 0);
    END_IF
  END_IF
END_IF
```

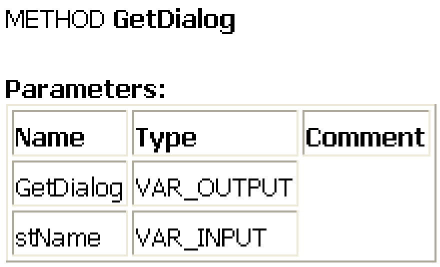
The function "OnLoginDialogClosed()" defines the reaction to the closing of the Login dialog.

Function OnLoginDialogClosed

```
FUNCTION OnLoginDialogClosed : BOOL
VAR_INPUT
 pClientData : POINTER TO VisuStructClientData;
END_VAR
VAR
 dialogMan : IDialogManager;
 loginDialog : IVisualisationDialog;
 pLoginInfo : POINTER TO Login_VISU_STRUCT;
 result : Visu_DialogResult;
 stPassword: STRING;
 stUsername: STRING;
END_VAR
dialogMan := g_VisuManager.GetDialogManager();      (* The DialogManager is provided by the VisuManager
   that is implicitly available *)
IF dialogMan <> 0 AND pVisuClient <> 0 THEN
  loginDialog := dialogMan.GetDialog('VisuDialogs.Login'); (* recognizes the Login dialog *)
  IF loginDialog <> 0 THEN
  result := loginDialog.GetResult(); (* recognizes the result (OK, Cancel) of the dialog *)
  IF result = Visu_DialogResult.OK THEN
    loginDialog.SetResult(Visu_DialogResult.None); (* Reset to Default (none) *)
    pLoginInfo := dialogMan.GetClientInterface(loginDialog, pVisuClient);
           (* Structure Login_VISU_STRUCT is read; in the following
   the structure parameters can be set *)
    IF pLoginInfo <> 0 THEN
      stPassword := pLoginInfo^.stPassword;
      pLoginInfo^.stPassword := ''; (* Password is reset *)
      stUsername := pLoginInfo^.stUsername;
    END_IF
    ELSIF result = Visu_DialogResult.Cancel THEN
      loginDialog.SetResult(Visu_DialogResult.None);
      (* Response to "Cancel" ("Abbrechen") *)
    ELSE
      (* nothing to do here *)
    END_IF
  END_IF
END_IF
```

Methods of the DialogManager

- **GetDialog(STRING stName):** Provides the dialog currently affected by an event as an IVisualizationDialog.

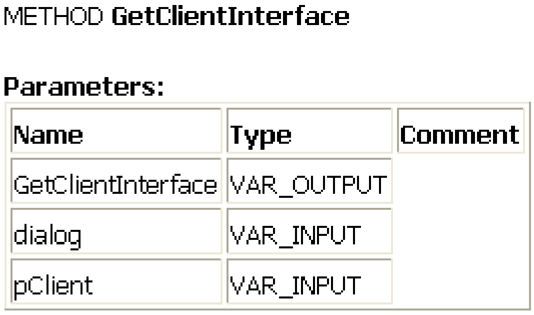


METHOD **GetDialog**

Parameters:

Name	Type	Comment
GetDialog	VAR_OUTPUT	
stName	VAR_INPUT	

- **GetClientInterface():** Provides a pointer to the dialog structure.

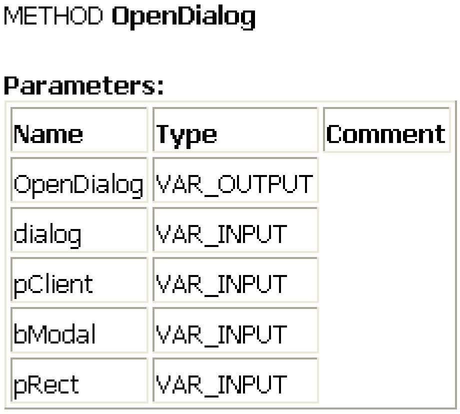


METHOD **GetClientInterface**

Parameters:

Name	Type	Comment
GetClientInterface	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

- OpenDialog(): Opens the dialog for the client.

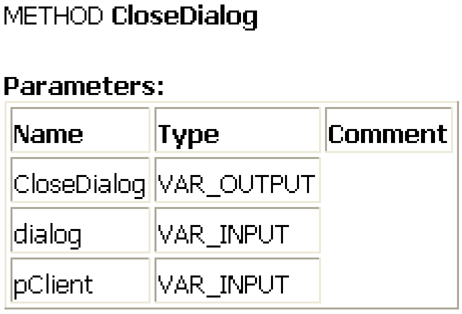


METHOD **OpenDialog**

Parameters:

Name	Type	Comment
OpenDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	
bModal	VAR_INPUT	
pRect	VAR_INPUT	

- CloseDialog(): Closes the dialog for the client.



METHOD **CloseDialog**

Parameters:

Name	Type	Comment
CloseDialog	VAR_OUTPUT	
dialog	VAR_INPUT	
pClient	VAR_INPUT	

6 Libraries

6.1 Libraries, General Information

The libraries Standard.library and Util.library are installed with the IndraLogic standard installation.

It contains functions and function blocks required for an IEC programming system according the IEC 61131-3 standard.

In addition, some "implicit" libraries are required for a variety of programming system functionalities, e.g. visualizing, profiling, etc., which are automatically integrated into a project. Users do not need to deal with these libraries explicitly.

More details:

- General Information on Library Management, page 83
- Installing Libraries, page185
- Library Manager, page 367
- Creating a "Library" or "Compiled Library" in the IndraWorks Environment, page 191

7 Service and Support

Our service helpdesk at our headquarters in Lohr, Germany and our worldwide service will assist you with all kinds of enquiries. You can reach us **around the clock - even on weekend and on holidays.**

	Helpdesk	Service Hotline Worldwide
Phone	+49 (0) 9352 40 50 60	Outwith Germany please contact our sales/service office in your area first. For hotline numbers refer to the sales office addresses on the Internet.
Fax	+49 (0) 9352 40 49 41	
E-mail	service.svc@boschrexroth.de	
Internet	http://www.boschrexroth.com You will also find additional notes regarding service, maintenance (e.g. delivery addresses) and training.	

Preparing Information

For quick and efficient help please have the following information ready:

- Detailed description of the fault and the circumstances
- Information on the type plate of the affected products, especially type codes and serial numbers
- Your phone, fax numbers and e-mail address so we can contact you in case of questions.

Index

Index

Index

Index

Index

Index

Index

Index

Index

Index

Notes

Rexroth
Bosch Group

Bosch Rexroth AG
Electric Drives and Controls
P.O. Box 13 57
97803 Lohr, Germany
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr, Germany
Tel. +49 (0)93 52-40-0
Fax +49 (0)93 52-48 85
www.boschrexroth.com/electrics

R911334390

Printed in Germany
DOK-IWORKS-IL2GPRO*V12-AP01-EN-P