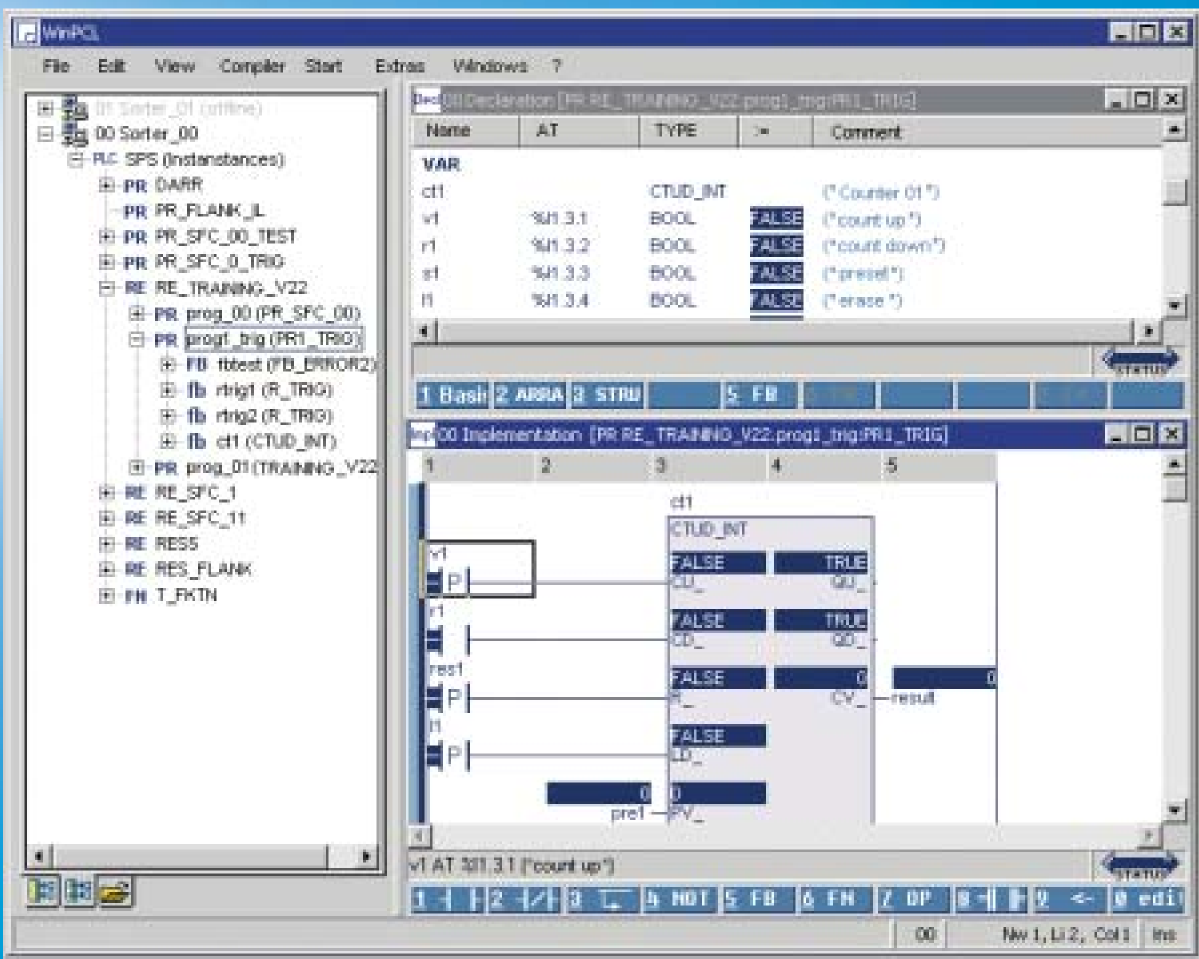
Industrial Hydraulics | **Electric Drives and Controls** | Linear Motion and Assembly Technologies | Pneumatics | Service Automation | Mobile Hydraulics

Rexroth
Bosch Group

Rexroth WinPCL 06VRS

298446
Edition 01

Operating and Programming Guide



Title Rexroth WinPCL 06VRS

Type of Documentation Operating and Programming Guide

Document Typecode DOK-CONTRL-WINPCL*06VRS-AW01-EN-P

Internal File Reference Document Number, 120-0401-B308-01/EN

Purpose of Documentation This documentation describes the operating and programming interface WinPCL.

Record of Revisions

Description	Release Date	Notes
120-0401-B308-01/EN	07/03	First Edition

Published by Bosch Rexroth AG
Bgm.-Dr.-Nebel-Str. 2 • D-97816 Lohr a. Main

Telephone +49 (0)93 52/40-0 • Tx 68 94 21 • Fax +49 (0)93 52/40-48 85

http://www.boschrexroth.com/

Dept. EPY (NH)

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

Contents

1 Preliminary Remarks

1.1 Contents of this Documentation

This documentation contains the description of the Windows programming system for the programmable logic control (PLC) offered by Bosch Rexroth.

It is also available as online help in the programming system, if you choose menu item Help ? \ Help Topics (Contents & Index).

The individual menu and submenu items are described in chapter "WinPCL".

The chapters

- Declaration Editors
  - Declaration Editor, Resource
  - Declaration Editor, Program
  - Declaration Editor, Function Block
  - Declaration Editor, Function
  - Declaration Editor, ARRAY
  - Declaration Editor, Structure
- Instruction List Editor
- Ladder Diagram Editor
- Sequential Function Chart Editor (SFC)
- Action Block Editor and
- IO Editor

describe the editors and lists belonging to the system.

The chapters

- Data Types in WinPCL
- Functions in WinPCL
- Function Blocks in WinPCL
- Programs and Resources in WinPCL

are subdivided into standard, firmware and user elements. Here you can find a description for each standard and firmware element that is available as <F1> help if you enter the name of the element as search criterion. For user elements you can find the construction possibilities.

Chapter "Programs and Resources in WinPCL" contains explanations of tasks and their execution.

Chapter "Troubleshooting in WinPCL" describes the mechanism for error identification and transfer. Errors can be identified and evaluated for standard, firmware and user elements if you work with the error variables S#ErrorFlg, S#ErrorTyp and S#ErrorNr.

1.2 Further Documentation

Number	Title	Contents	Document Typecode
/1/	MTC200 and ISP200 see "Systems and Controls in the Product Catalog"	– Application Areas – User advantages – System kit MTC200 – Control components – Visualization and control terminals – I/O units – Drive electronics and motors	Current product catalog on our webside http://info.indramat.de/products/prod_catalog/ecat/default.asp Hardcopy and English version in preparation.
/2/	Programming with IndraStep SFCs with Mode Control and Diagnosis	– SFC modes – IndraStep modes – SFC diagnosis	DOK-CONTRL-SPS*ISTEP02-AW..-EN-P
/3/	SyConPB System Configurator for PROFIBUS	– PROFIBUS Configuration – Data exchange – Menus SyConPB – Troubleshooting – FBs for bus control	DOK-CONTRL-SYCON****-DP-AW..-EN-P
/4/	SyConDN System Configurator for DeviceNet	– DeviceNet configuration – Data exchange – Menus SyConDN – Troubleshooting	DOK-CONTRL-SYCON****-DN-AW..-EN-P
/5/	CMD System Configurator for INTERBUS	– INTERBUS configuration – Data exchange – Menus CMD – Troubleshooting	DOK-CONTRL-IBS*CMD****-AW..-EN-P
/6/	INTERBUS Diagnostic Primer	Controller board, Generation 4	Phönix Contact IBS SYS DIAG DSC UM Rev. B Art.-No. 2747280
/7/	WinHMI		
	System Configurator SYSCON	– Hardware Configuration of controls, single and as compound	DOK-CONTRL-INS*BOF*V22-AW...-EN-P DOK-CONTRL-SETUP***-V22-AW..-EN-P Online Help in \..\BasicData\Help\ Syscon_en.hlp
	PC Compound	– Description of the activities when connecting a PC compound – Precondition for remote programming	DOK-CONTRL-PC**NET*V22-AW..-EN-P
/8/	Screenmanager 05VRS Project planning tool to program miniature control panels Operating and Programming Guide	– Miniature control panels	DOK-SUPPL*-SCM*PROG*V5-AW..-EN-P
/9/	Screenmanager 04VRS Application Manual	– Miniature control panels	DOK-SUPPL*-SCM*BEDIEN*-AW..-EN-P In preparation.

Fig. 1-1: Further documentation

2 Important Directions for Use

2.1 Appropriate Use

Introduction

Bosch Rexroth products represent state-of-the-art developments and manufacturing. They are tested prior to delivery to ensure operating safety and reliability.

The products may only be used in the manner that is defined as appropriate. If they are used in an inappropriate manner, then situations can develop that may lead to property damage or injury to personnel.

Note: Bosch Rexroth, as manufacturer, is not liable for any damages resulting from inappropriate use. In such cases, the guarantee and the right to payment of damages resulting from inappropriate use are forfeited. The user alone carries all responsibility of the risks.

Before using Bosch Rexroth products, make sure that all the pre-requisites for appropriate use of the products are satisfied:

- Personnel that in any way, shape or form uses our products must first read and understand the relevant safety instructions and be familiar with appropriate use.
- If the product takes the form of hardware, then they must remain in their original state, in other words, no structural changes are permitted. It is not permitted to decompile software products or alter source codes.
- Do not mount damaged or faulty products or use them in operation.
- Make sure that the products have been installed in the manner described in the relevant documentation.

Areas of Use and Application

The user and programming interface WinPCL is a development environment to create application programs for programmable logic controls (PLC) of Bosch Rexroth. WinPCL is designed for use in the following cases:

- Commissioning of programmable logic controls,
- Programming of programmable logic controls,
- Support to create diagnosis and mode controls (ProVi, IndraStep).

Note: Operation is only permitted in the specified configurations and combinations of components using the software and firmware as specified in the relevant function descriptions.

2.2 Inappropriate Use

Using the user and programming interface outside of the above-referenced areas of application or under operating conditions other than described in the document and the technical data specified is defined as "inappropriate use".

- WinPCL may not be used if it is subject to operating conditions that do not meet the above specified ambient conditions.
- Furthermore, WinPCL must not be used for applications Bosch Rexroth has not specifically released for that intended purpose. Please note the specifications outlined in the general Safety Instructions!

2.3 Delivery Stipulations for Computer Programs

The copyrights, present and future commercial proprietary rights of all kinds, as well as all the rights of exploitation to delivered computer programs -- in equipment or separate from it -- belong exclusively to the Supplier.

A computer program may only be used in one single piece of equipment. Exceptions are commissioning software, which are marked with the designation -COPY at the end. These can be copied freely within the context of regular product usage by the customer.

Every act exceeding the minimum use outlined in the proprietary rights requires the consent of the Supplier. If a computer program delivered by the Supplier is not protected by proprietary rights, then the minimum use stated in the proprietary rights laws is declared as agreed upon.

If the Orderer transfers a computer program then he must completely surrender the program carrier and all copies in their entirety to the Acquiring Party, or these must be erased. A limitation of use corresponding to these stipulations (1 through 6) must be agreed upon with the Acquiring Party.

The Supplier will eliminate any fault in the computer program either by a circumvention of the fault, which is agreeable to the Orderer, or by delivering a new program.

All documents and information needed to reconstruct a fault must accompany the notification of a fault in the computer program.

Otherwise, the general delivery stipulations outlined by Rexroth apply.

3 Safety Instructions for Electric Drives and Controls

3.1 Introduction

Read these instructions before the initial startup of the equipment in order to eliminate the risk of bodily harm or material damage. Follow these safety instructions at all times.

Do not attempt to install or start up this equipment without first reading all documentation provided with the product. Read and understand these safety instructions and all user documentation of the equipment prior to working with the equipment at any time. If you do not have the user documentation for your equipment, contact your local Bosch Rexroth representative to send this documentation immediately to the person or persons responsible for the safe operation of this equipment.

If the equipment is resold, rented or transferred or passed on to others, then these safety instructions must be delivered with the equipment.

WARNING

Improper use of this equipment, failure to follow the safety instructions in this document or tampering with the product, including disabling of safety devices, may result in material damage, bodily harm, electric shock or even death!

3.2 Explanations

The safety instructions describe the following degrees of hazard seriousness in compliance with ANSI Z535. The degree of hazard seriousness informs about the consequences resulting from non-compliance with the safety instructions.

Warning symbol with signal word	Degree of hazard seriousness according to ANSI
DANGER	Death or severe bodily harm will occur.
WARNING	Death or severe bodily harm may occur.
CAUTION	Bodily harm or material damage may occur.

Fig. 3-1: Hazard classification (according to ANSI Z535)

3.3 Hazards by Improper Use

DANGER

High voltage and high discharge current! Danger to life or severe bodily harm by electric shock!

DANGER

Dangerous movements! Danger to life, severe bodily harm or material damage by unintentional motor movements!

WARNING

High electrical voltage due to wrong connections! Danger to life or bodily harm by electric shock!

WARNING

Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electrical equipment!

CAUTION

Surface of machine housing could be extremely hot! Danger of injury! Danger of burns!

CAUTION

Risk of injury due to improper handling! Bodily harm caused by crushing, shearing, cutting and mechanical shock or incorrect handling of pressurized systems!

CAUTION

Risk of injury due to incorrect handling of batteries!

3.4 General Information

- Bosch Rexroth AG is not liable for damages resulting from failure to observe the warnings provided in this documentation.
- Read the operating, maintenance and safety instructions in your language before starting up the machine. If you find that you cannot completely understand the documentation for your product, please ask your supplier to clarify.
- Proper and correct transport, storage, assembly and installation as well as care in operation and maintenance are prerequisites for optimal and safe operation of this equipment.
- Only persons who are trained and qualified for the use and operation of the equipment may work on this equipment or within its proximity.
  - The persons are qualified if they have sufficient knowledge of the assembly, installation and operation of the equipment as well as an understanding of all warnings and precautionary measures noted in these instructions.
  - Furthermore, they must be trained, instructed and qualified to switch electrical circuits and equipment on and off in accordance with technical safety regulations, to ground them and to mark them according to the requirements of safe work practices. They must have adequate safety equipment and be trained in first aid.
- Only use spare parts and accessories approved by the manufacturer.
- Follow all safety regulations and requirements for the specific application as practiced in the country of use.
- The equipment is designed for installation in industrial machinery.
- The ambient conditions given in the product documentation must be observed.
- Use only safety features and applications that are clearly and explicitly approved in the Project Planning Manual.
  For example, the following areas of use are not permitted: construction cranes, elevators used for people or freight, devices and vehicles to transport people, medical applications, refinery plants, transport of hazardous goods, nuclear applications, applications sensitive to high frequency, mining, food processing, control of protection equipment (also in a machine).
- The information given in the documentation of the product with regard to the use of the delivered components contains only examples of applications and suggestions.
  The machine and installation manufacturer must
  - make sure that the delivered components are suited for his individual application and check the information given in this documentation with regard to the use of the components,
  - make sure that his application complies with the applicable safety regulations and standards and carry out the required measures, modifications and complements.
- Startup of the delivered components is only permitted once it is sure that the machine or installation in which they are installed complies with the national regulations, safety specifications and standards of the application.
- Technical data, connections and operational conditions are specified in the product documentation and must be followed at all times.

- Operation is only permitted if the national EMC regulations for the application are met.
  The instructions for installation in accordance with EMC requirements can be found in the documentation "EMC in Drive and Control Systems".
  The machine or installation manufacturer is responsible for compliance with the limiting values as prescribed in the national regulations.

3.5 Protection Against Contact with Electrical Parts

Note: This section refers to equipment and drive components with voltages above 50 Volts.

Touching live parts with voltages of 50 Volts and more with bare hands or conductive tools or touching ungrounded housings can be dangerous and cause electric shock. In order to operate electrical equipment, certain parts must unavoidably have dangerous voltages applied to them.

DANGER

High electrical voltage! Danger to life, severe bodily harm by electric shock!

⇒ Only those trained and qualified to work with or on electrical equipment are permitted to operate, maintain or repair this equipment.

⇒ Follow general construction and safety regulations when working on high voltage installations.

⇒ Before switching on power the ground wire must be permanently connected to all electrical units according to the connection diagram.

⇒ Do not operate electrical equipment at any time, even for brief measurements or tests, if the ground wire is not permanently connected to the points of the components provided for this purpose.

⇒ Before working with electrical parts with voltage higher than 50 V, the equipment must be disconnected from the mains voltage or power supply. Make sure the equipment cannot be switched on again unintended.

⇒ The following should be observed with electrical drive and filter components:

⇒ Wait five (5) minutes after switching off power to allow capacitors to discharge before beginning to work. Measure the voltage on the capacitors before beginning to work to make sure that the equipment is safe to touch.

⇒ Never touch the electrical connection points of a component while power is turned on.

⇒ Install the covers and guards provided with the equipment properly before switching the equipment on. Prevent contact with live parts at any time.

⇒ A residual-current-operated protective device (RCD) must not be used on electric drives! Indirect contact must be prevented by other means, for example, by an overcurrent protective device.

⇒ Electrical components with exposed live parts and uncovered high voltage terminals must be installed in a protective housing, for example, in a control cabinet.

To be observed with electrical drive and filter components:

DANGER

High electrical voltage on the housing!
High leakage current! Danger to life, danger of injury by electric shock!

⇒ Connect the electrical equipment, the housings of all electrical units and motors permanently with the safety conductor at the ground points before power is switched on. Look at the connection diagram. This is even necessary for brief tests.

⇒ Connect the safety conductor of the electrical equipment always permanently and firmly to the supply mains. Leakage current exceeds 3.5 mA in normal operation.

⇒ Use a copper conductor with at least 10 mm² cross section over its entire course for this safety conductor connection!

⇒ Prior to startups, even for brief tests, always connect the protective conductor or connect with ground wire. Otherwise, high voltages can occur on the housing that lead to electric shock.

3.6 Protection Against Electric Shock by Protective Low Voltage (PELV)

All connections and terminals with voltages between 0 and 50 Volts on Rexroth products are protective low voltages designed in accordance with international standards on electrical safety.

WARNING

High electrical voltage due to wrong connections! Danger to life, bodily harm by electric shock!

⇒ Only connect equipment, electrical components and cables of the protective low voltage type (PELV = Protective Extra Low Voltage) to all terminals and clamps with voltages of 0 to 50 Volts.

⇒ Only electrical circuits may be connected which are safely isolated against high voltage circuits. Safe isolation is achieved, for example, with an isolating transformer, an opto-electronic coupler or when battery-operated.

3.7 Protection Against Dangerous Movements

Dangerous movements can be caused by faulty control of the connected motors. Some common examples are:

- improper or wrong wiring of cable connections
- incorrect operation of the equipment components
- wrong input of parameters before operation
- malfunction of sensors, encoders and monitoring devices
- defective components
- software or firmware errors

Dangerous movements can occur immediately after equipment is switched on or even after an unspecified time of trouble-free operation.

The monitoring in the drive components will normally be sufficient to avoid faulty operation in the connected drives. Regarding personal safety, especially the danger of bodily injury and material damage, this alone cannot be relied upon to ensure complete safety. Until the integrated monitoring functions become effective, it must be assumed in any case that faulty drive movements will occur. The extent of faulty drive movements depends upon the type of control and the state of operation.

DANGER

Dangerous movements! Danger to life, risk of injury, severe bodily harm or material damage!

⇒ Ensure personal safety by means of qualified and tested higher-level monitoring devices or measures integrated in the installation. Unintended machine motion is possible if monitoring devices are disabled, bypassed or not activated.

⇒ Pay attention to unintended machine motion or other malfunction in any mode of operation.

⇒ Keep free and clear of the machine’s range of motion and moving parts. Possible measures to prevent people from accidentally entering the machine’s range of motion:

- use safety fences
- use safety guards
- use protective coverings
- install light curtains or light barriers

⇒ Fences and coverings must be strong enough to resist maximum possible momentum, especially if there is a possibility of loose parts flying off.

⇒ Mount the emergency stop switch in the immediate reach of the operator. Verify that the emergency stop works before startup. Don’t operate the machine if the emergency stop is not working.

⇒ Isolate the drive power connection by means of an emergency stop circuit or use a starting lockout to prevent unintentional start.

⇒ Make sure that the drives are brought to a safe standstill before accessing or entering the danger zone. Safe standstill can be achieved by switching off the power supply contactor or by safe mechanical locking of moving parts.

⇒ Secure vertical axes against falling or dropping after switching off the motor power by, for example:

- mechanically securing the vertical axes
- adding an external braking/ arrester/ clamping mechanism
- ensuring sufficient equilibration of the vertical axes

The standard equipment motor brake or an external brake controlled directly by the drive controller are not sufficient to guarantee personal safety!

⇒ Disconnect electrical power to the equipment using a master switch and secure the switch against reconnection for:

- maintenance and repair work
- cleaning of equipment
- long periods of discontinued equipment use

⇒ Prevent the operation of high-frequency, remote control and radio equipment near electronics circuits and supply leads. If the use of such equipment cannot be avoided, verify the system and the installation for possible malfunctions in all possible positions of normal use before initial startup. If necessary, perform a special electromagnetic compatibility (EMC) test on the installation.

3.8 Protection Against Magnetic and Electromagnetic Fields During Operation and Mounting

Magnetic and electromagnetic fields generated near current-carrying conductors and permanent magnets in motors represent a serious health hazard to persons with heart pacemakers, metal implants and hearing aids.

WARNING

Health hazard for persons with heart pacemakers, metal implants and hearing aids in proximity to electrical equipment!

⇒ Persons with heart pacemakers, hearing aids and metal implants are not permitted to enter the following areas:

- Areas in which electrical equipment and parts are mounted, being operated or started up.
- Areas in which parts of motors with permanent magnets are being stored, operated, repaired or mounted.

⇒ If it is necessary for a person with a heart pacemaker to enter such an area, then a doctor must be consulted prior to doing so. Heart pacemakers that are already implanted or will be implanted in the future, have a considerable variation in their electrical noise immunity. Therefore there are no rules with general validity.

⇒ Persons with hearing aids, metal implants or metal pieces must consult a doctor before they enter the areas described above. Otherwise, health hazards will occur.

3.9 Protection Against Contact with Hot Parts

CAUTION

Housing surfaces could be extremely hot! Danger of injury! Danger of burns!

⇒ Do not touch housing surfaces near sources of heat! Danger of burns!

⇒ After switching the equipment off, wait at least ten (10) minutes to allow it to cool down before touching it.

⇒ Do not touch hot parts of the equipment, such as housings with integrated heat sinks and resistors. Danger of burns!

3.10 Protection During Handling and Mounting

Under certain conditions, incorrect handling and mounting of parts and components may cause injuries.

CAUTION

Risk of injury by incorrect handling! Bodily harm caused by crushing, shearing, cutting and mechanical shock!

⇒ Observe general installation and safety instructions with regard to handling and mounting.

⇒ Use appropriate mounting and transport equipment.

⇒ Take precautions to avoid pinching and crushing.

⇒ Use only appropriate tools. If specified by the product documentation, special tools must be used.

⇒ Use lifting devices and tools correctly and safely.

⇒ For safe protection wear appropriate protective clothing, e.g. safety glasses, safety shoes and safety gloves.

⇒ Never stand under suspended loads.

⇒ Clean up liquids from the floor immediately to prevent slipping.

3.11 Battery Safety

Batteries contain reactive chemicals in a solid housing. Inappropriate handling may result in injuries or material damage.

CAUTION

Risk of injury by incorrect handling!

⇒ Do not attempt to reactivate discharged batteries by heating or other methods (danger of explosion and cauterization).

⇒ Never charge non-chargeable batteries (danger of leakage and explosion).

⇒ Never throw batteries into a fire.

⇒ Do not dismantle batteries.

⇒ Do not damage electrical components installed in the equipment.

Note: Be aware of environmental protection and disposal! The batteries contained in the product should be considered as hazardous material for land, air and sea transport in the sense of the legal requirements (danger of explosion). Dispose batteries separately from other waste. Observe the legal requirements in the country of installation.

3.12 Protection Against Pressurized Systems

Certain motors and drive controllers, corresponding to the information in the respective Project Planning Manual, must be provided with pressurized media, such as compressed air, hydraulic oil, cooling fluid and cooling lubricant supplied by external systems. Incorrect handling of the supply and connections of pressurized systems can lead to injuries or accidents. In these cases, improper handling of external supply systems, supply lines or connections can cause injuries or material damage.

CAUTION

Danger of injury by incorrect handling of pressurized systems !

⇒ Do not attempt to disassemble, to open or to cut a pressurized system (danger of explosion).

⇒ Observe the operation instructions of the respective manufacturer.

⇒ Before disassembling pressurized systems, release pressure and drain off the fluid or gas.

⇒ Use suitable protective clothing (for example safety glasses, safety shoes and safety gloves)

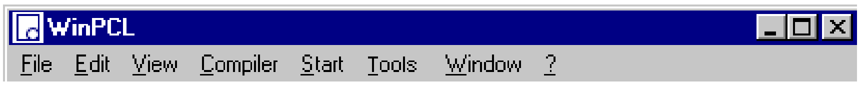
⇒ Remove any fluid that has leaked out onto the floor immediately.

Note: Environmental protection and disposal! The media used in the operation of the pressurized system equipment may not be environmentally compatible. Media that are damaging the environment must be disposed separately from normal waste. Observe the legal requirements in the country of installation.

Notes

4 WinPCL

4.1 Main Menu Line



menue_komplett.bmp

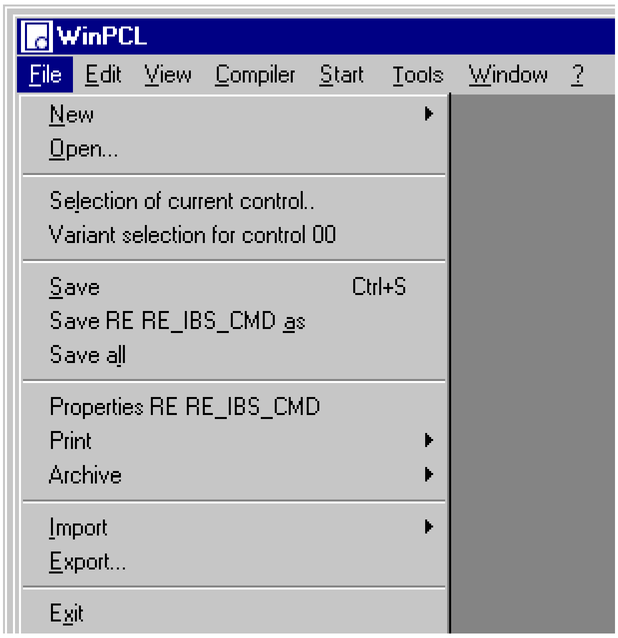
Fig. 4-1: Main menu

Like all other Windows programs, the WinPCL menu bar shows the following menu items:

- File
- Edit
- View
- Compiler
- Start
- Tools
- Window
- ? Help
- Miscellaneous, as Language Conversion, Remote Programming

The individual menu items contain several submenu items, which are indicated in gray, which means that they are inactive, when they are not useful for the moment or not relevant for the user.

4.2 File



menue_Datei.bmp

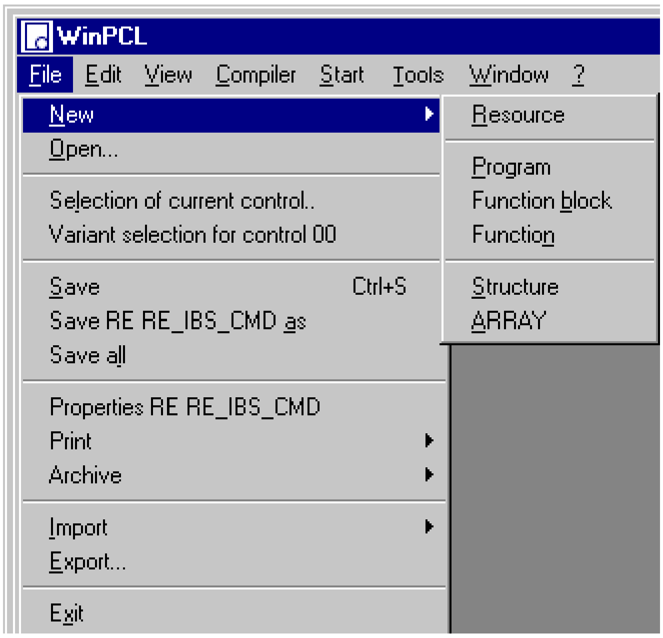
Fig. 4-2: "File" menu item

The "File" menu item combines all file-related operations.

It covers six groups with commands:

- **New** creates a new WinPCL file / **Open** opens an existing file.
- **Selecting the Current Control** / **Selecting the Variant for a Control "xx"**
- **Save** / **Save as** with specification of a new name, if necessary with different properties (password) / **Save all** saves all edited files.
- **Properties** of the focussed file, such as information on the file, passwords and statistics, can be defined and/or edited. It is not possible to permit "write on inputs" for the file. / **Print** / **Archive**
- **Import** imports text files as DOS_ASCII or WIN_ANSI text / **Export** exports the current file.
- **Exit** of WinPCL

New



menue_Datei_neu.bmp

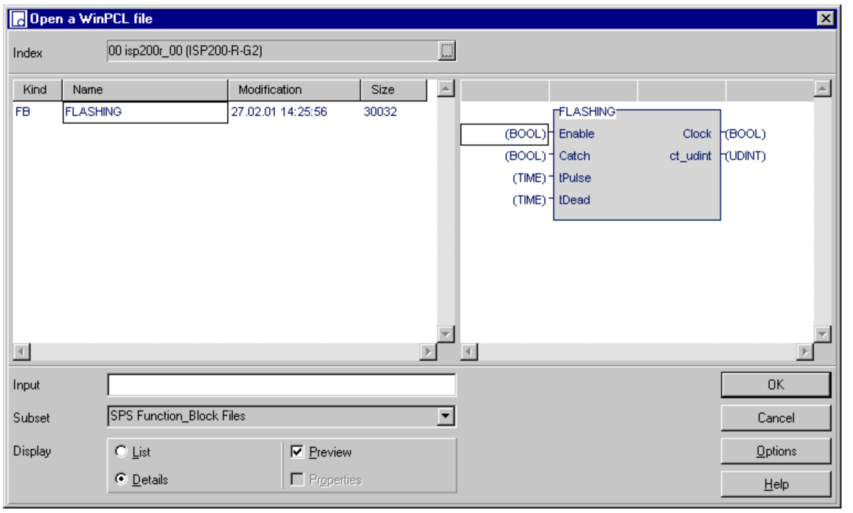
Fig. 4-3: "File / New" menu item

Using the "File / New" menu items, new resources, programs, function blocks, functions, structures, and ARRAYs can be created for the chosen control unit with the variant selected. Programs, function blocks and functions are also called program organization units (POU). After selection of the desired POU type, the declaration editor opens for definition of the interface of the POU or the data type.

Note: The name of a resource may not exceed a length of 32 characters.

If this length is exceeded, excess characters may be cut off outside of Win PCL.

Open



menue_öffnen.bmp

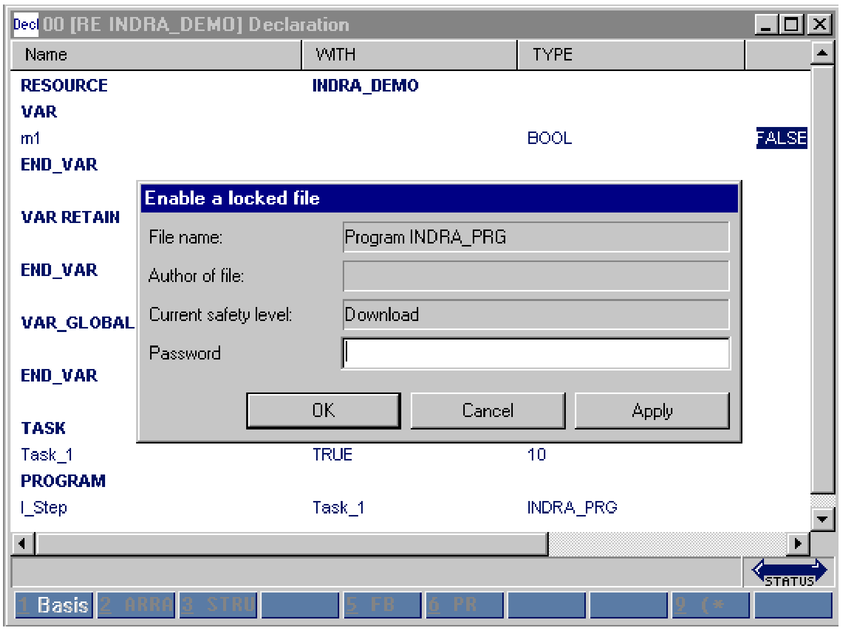
Fig. 4-4: "File / Open" menu item

The "File / Open" menu item activates the "Open" dialog for the chosen control system and with the variant selected.

The name of the desired file can be entered in the input line. Then you have to choose the respective subset. The example shows a PLC function block. If you choose the option "Preview" you can see the interface of the selected file at the right side.

If the file is coded, a window appears to enter the password, with which you unlock the file for editing or viewing.

(see also Properties of files: download / view / edit.)

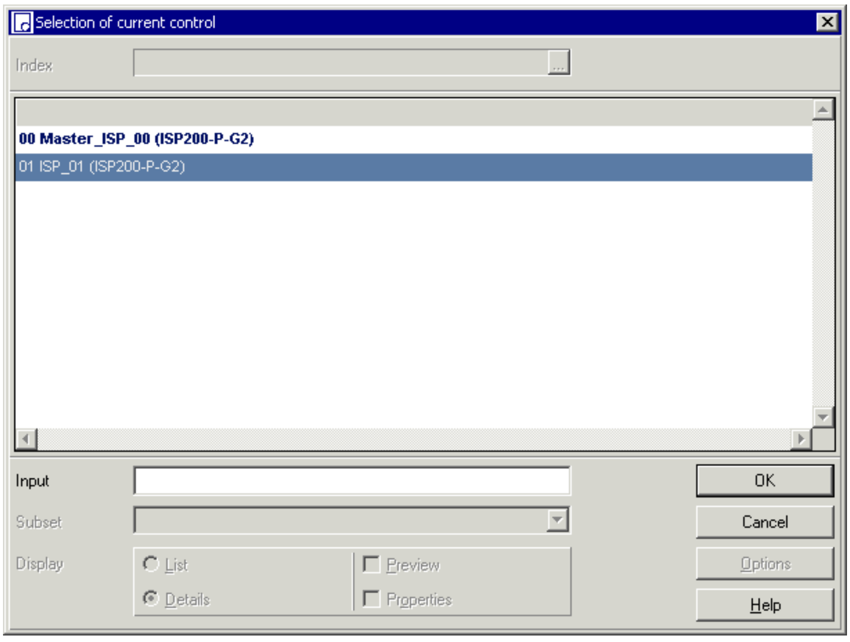


freischalten.bmp

Fig. 4-5: Password input box to open coded files

 Rexroth Bosch Group

Selecting the Current Control



auswahl akt Stg.bmp

Fig. 4-6: Selection of current control

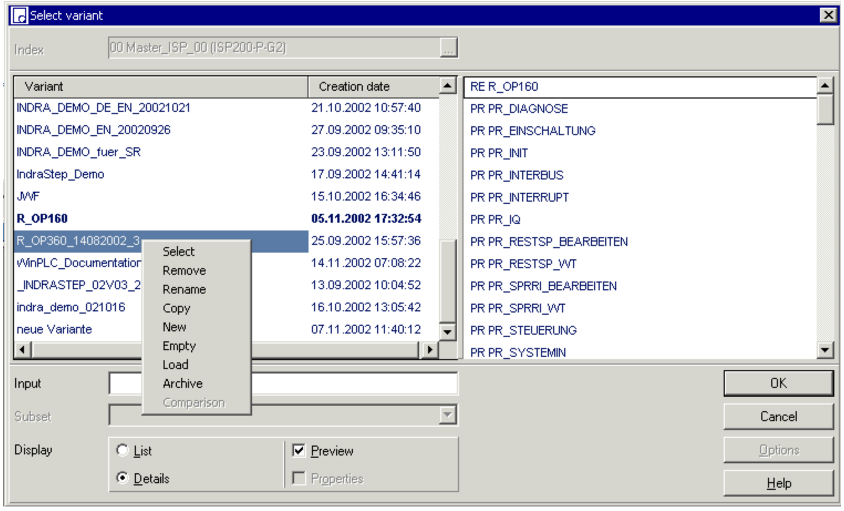
The "Selection of current control" dialog window shows the control systems which were entered with the system configuration.

The number of the control is indicated to the left, the name in the middle and (in brackets) the type of the control to the right.

The desired control can be selected with the mouse or the cursor keys.

The selected control is marked with a "*".

Selecting the Variant for a Control "xx"



auswahl variante.bmp

Fig. 4-7: Selecting the current variant

Each control has, after having been entered in the system configuration, a <<basic directory>>, to which the PLC files can be stored.

For technically real projects the number of files increases strongly so that it is be useful to combine files that belong together in variants.

Variants can contain e.g. different development stages of the same project.

A new variant can be generated by means of pop-up menus, that open when you click the right mouse button or press the <Shift>+<F10> keys.

Using the "New" menu item, a variant can be created with the standard name "new variant", which can be renamed using the "Rename" menu item.

If files of an earlier variant have to be applied, this variant first has to be marked with an "*" by using the "Select" command.

Using the "Copy" item of the pop-up menu, the chosen variant is then copied and used as source for the destination variant. After having executed the "Copy" command, the same window is opened again for selecting the destination variant. Already existing files of the destination variant are overwritten if the names are identical. A warning, however, is displayed before.

Using the "Remove" menu item, a variant including all files contained therein can be deleted. The "Empty" menu item keeps the name of the variant.

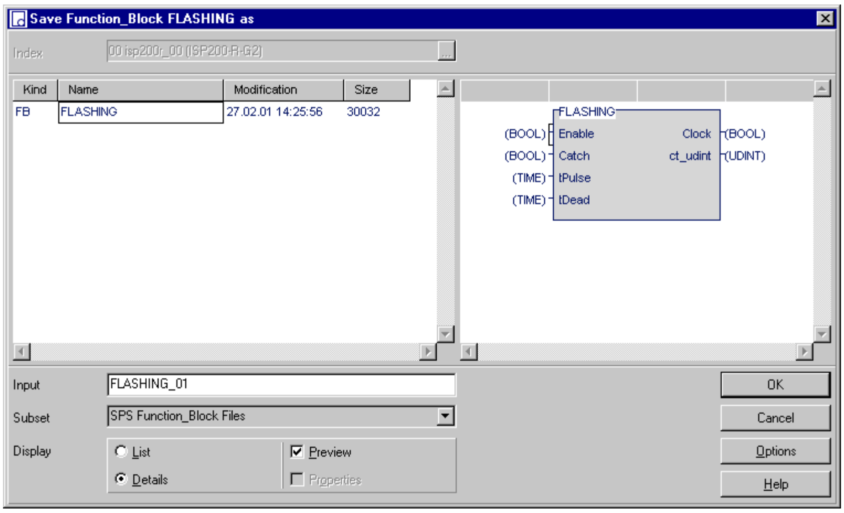
Save

The focused file is saved when you use the menu item "Save" <Ctrl>+<S>

The time of the last file modification is entered as file time.

A file which has not been saved since the last modification is marked with an "*" behind the file name.

Save as



speichern als.bmp

Fig. 4-8: "File / Save as" menu item

Using the "Save as" menu item, the focused file is saved under another name, and/or the "Options" button saves it with other file properties.

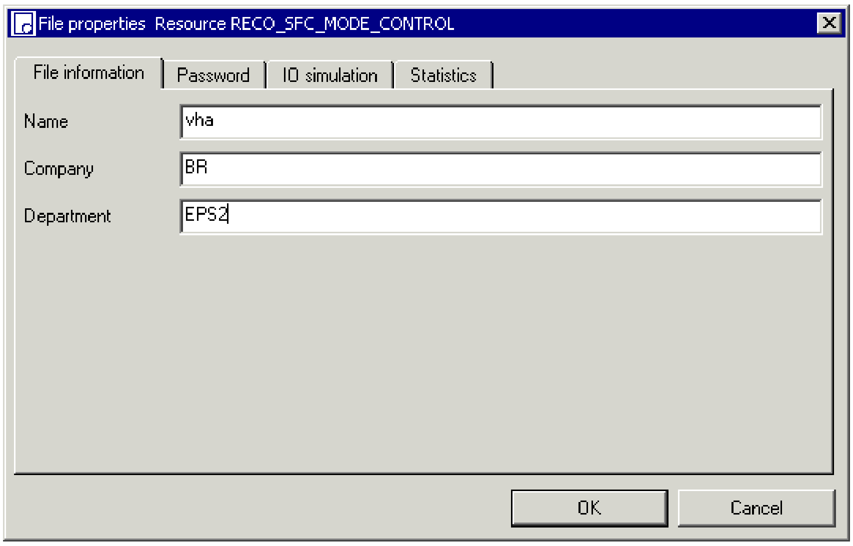
Save all

This menu item effects the storage of all currently opened files; changed files get the file time of the last change, all other files retain their file time.

Properties

The "Properties" menu item serves

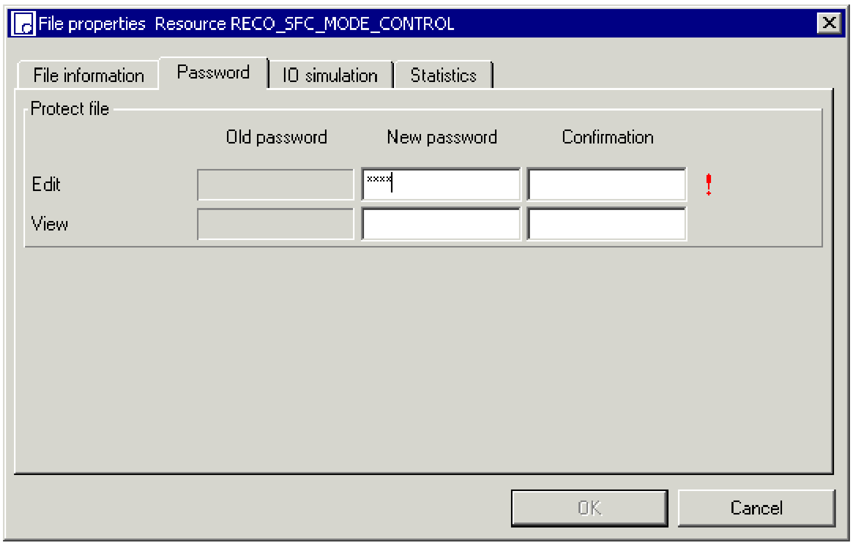
- for displaying and modifying file information like name (of the person in charge), company, department,



info01.bmp

Fig. 4-9: File properties, "File information"

- for changing file-related passwords, which permit editing and/or viewing.



info02.bmp

Fig. 4-10: File properties, changing the password(s)

For the so far unprotected function block - the field for the old password is marked in gray -, a new password can be entered permitting to edit and/or view, but not change, the function block. If this password is not confirmed or a wrong confirmation was entered, this is indicated with a red "!" at the end of the line.

The red "!" becomes a green "Y" if the input is correct.

Note: Changes have to be saved and become effective only after reload.

Edit	**View**	**Possibilities**
Fulfilled	Not of any importance	The file can be modified.
Not fulfilled	Fulfilled	The file can be viewed.
Not fulfilled	Not fulfilled	The file cannot be viewed; compilation and download are possible.

Fig. 4-11: Access with entered and fulfilled / not fulfilled password

- for allowing to write on inputs (IO simulation) in ***this*** file (resources or programs, or, if VAR EXTERNAL is used, also in programs or function blocks) (also see Search in Compound, Write on Input Addresses - Search in Compound).

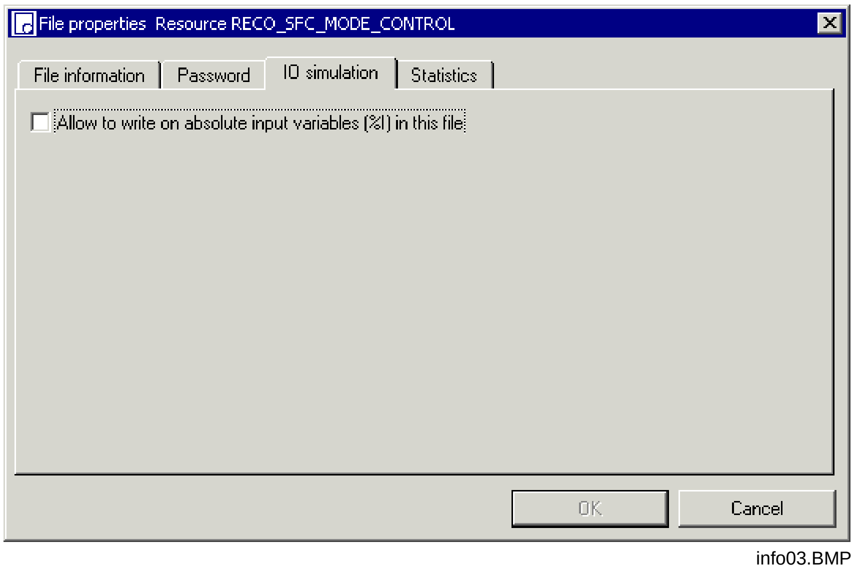


Fig. 4-12: Allow to write on absolute input variables in this file

Note: This permission is decisive only for the current file whose properties are affected.

Example: On resource level, writing on global %I variables has not been enabled; writing is enabled in the FB xyz => result: writing on the global %I variable is enabled in the FB using VAR EXTERNAL.

- for displaying statistical data, such as file name and type, last modification, modified with version, last user, and information on table usage,

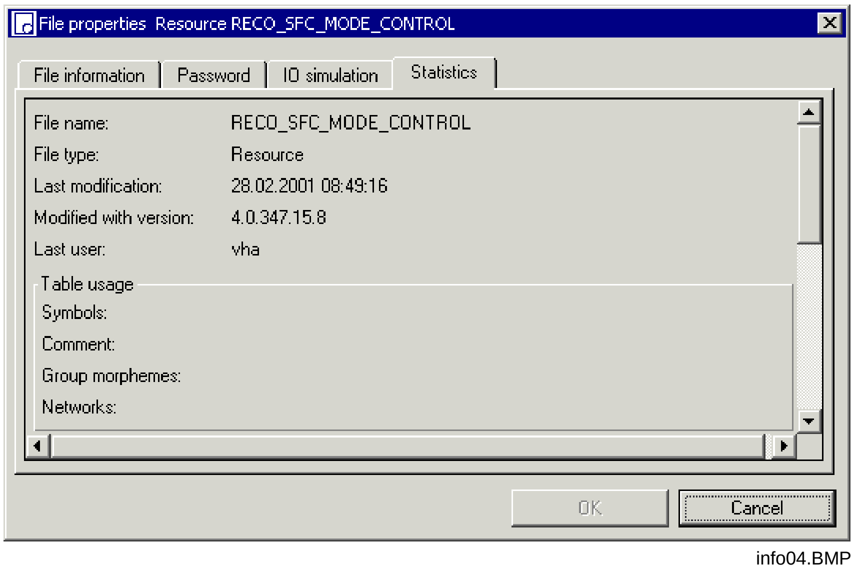
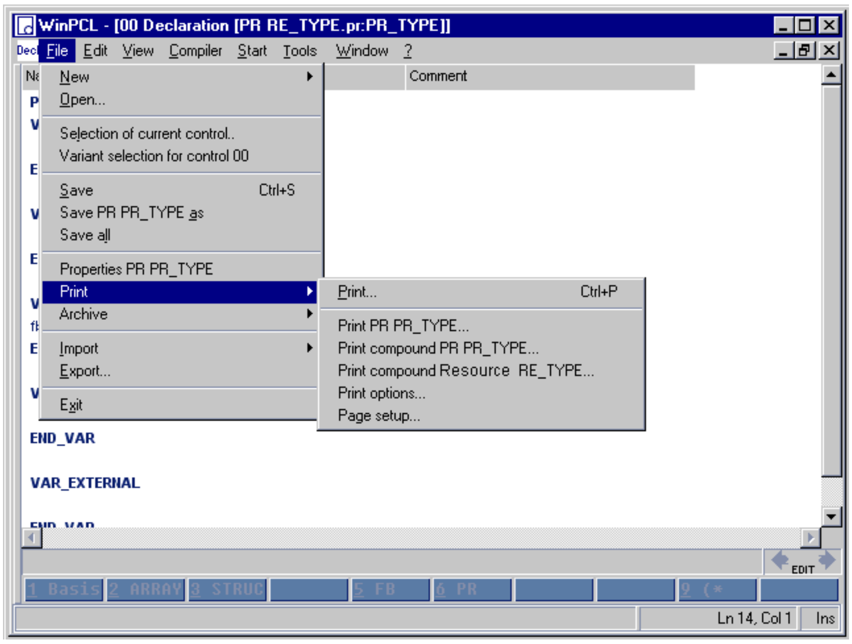


Fig. 4-13: File properties, "Statistics"

Print

The figure below shows all print methods.



Drucken.bmp

Focused file: Program PR_TYPE, declaration editor
Current resource: Resource RE_TYPE

Fig. 4-14: "File / Print" menu item

Print <Ctrl>+<P>

The menu item starts printing of the focused file; the current editor content is printed out.

In this example: Declaration of the program PR_TYPE

Note: Print <Ctrl+P> prints the currently displayed editor contents without considering the language set under Tools\Options\Print.

Print "xx"

The menu item starts printing of the focused file; the components set in the WinPCL options are printed consecutively.

For example:

Declaration, implementation, cross reference of the program PR_TYPE

Print compound "xx"

The menu item starts printing of the focused file with all its used / released files; here too the components set in the print options are printed consecutively.

For example:

Declaration, implementation, cross reference of the program PR_TYPE

Declaration, implementation, cross reference of the FBs FB_TYPE....

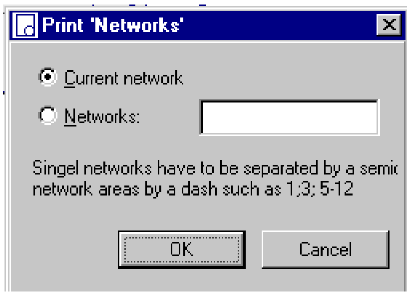
Print compound of resource "yy"

This menu item starts printing of the loaded resource (to be set under "Compiler / Selection of resource") and all of the files used / released by it; the components set in the WinPCL options are printed consecutively.

Note: The focused file is contained only if it is used by the current resource.

Print Rungs (LD / IL)

The rungs to be printed can be selected in the ladder diagram or in the instruction list by pressing the right mouse button (Print 'Rungs').



Print_Network.bmp

Fig. 4-15: Print rungs

Note: The command "Print rungs" initiates the printing of the currently represented editor content without considering the options set in menu Tools \ Options \ Print.

The context menu Print \ All corresponds to the printing with <Ctrl>+<P>.

Print Options

The settings in the print options are valid for all PLC projects, they do not have, however, any influence on the printouts with <Ctrl>+<P> (except for settings of the footer).

There are two ways to move to the menu item 'Print options':

- Tools \ Options on the page print
- File \ Print \ Print options.

The standard settings can be restored on all pages by pressing the button "Standard". In fact, this affects only the current page, the settings of the other pages are not changed.

Print Options – Content of the Printout (Contents)

On this page you can select, which parts of the project are to be printed.

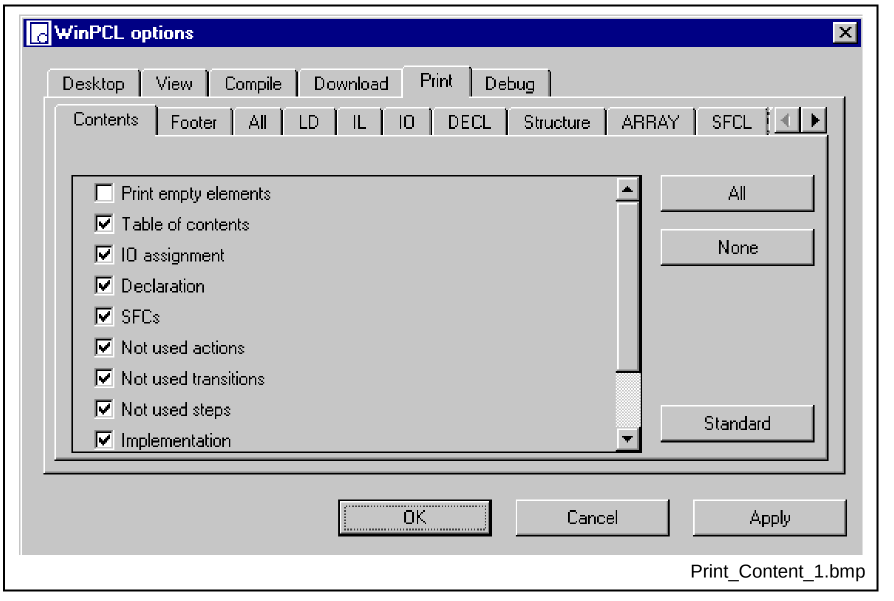


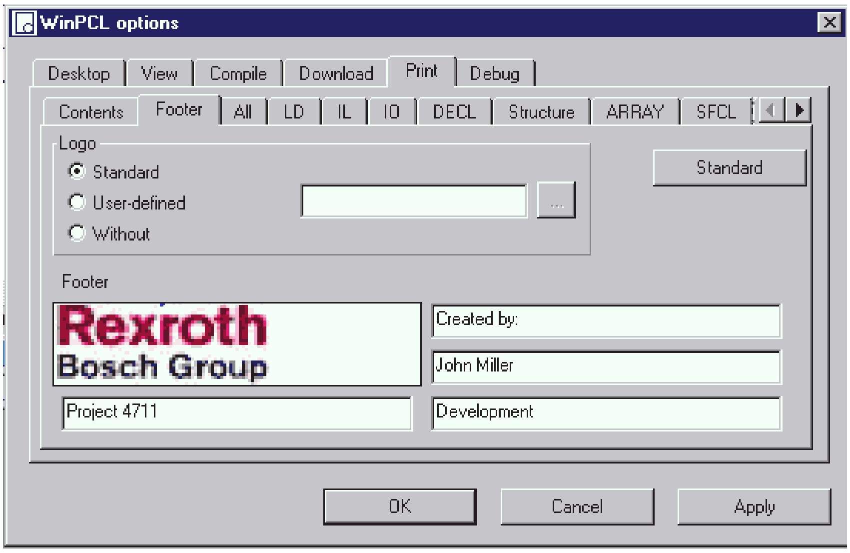
Fig. 4-16: Print options, content of the printout (Contents)

Components	Meaning
Print empty elements	Are printed also empty action transitions, steps, SFC's, imports, IO reservations
Table of contents	* Only print of the complete files and compound print * Common print for complete documentation
IO assignment	Only components of resources
Declaration	
SFCs	Only if provided, not for functions, ARRAYs and structures
Not used actions	Only if provided, not for functions, ARRAYs and structures
Not used transitions	Only if provided, not for functions, ARRAYs and structures
Not used steps	Only if provided, not for functions, ARRAYs and structures
Implementation	Only if provided, not for ARRAYs and structures
Cross reference list	Not for ARRAYs and structures
Import	Only if provided

Fig. 4-17: Supplement to the print options, content of the printout (Contents)

The "All" and the "None" buttons accelerate the selection of the content.

Print Options – Settings of the footer (Footer)

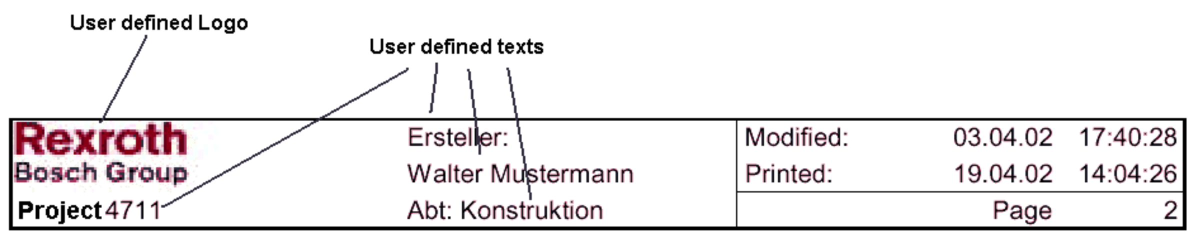


print_footer.bmp

Fig. 4-18: Print options, settings of the footer (Footer)

On this page you can configure the content for the printout of the footer. These settings are also valid when printing with <Ctrl>+<P>.

The configuration refers to the left side of the footer where it is possible to design the logo and additional texts.



print_example_footer.bmp

Fig. 4-19: Supplement to the print options, settings of the footer (Footer)

Logo:

As a logo an arbitrary file can be selected. The logo has to be available as Enhanced Meta File (EMF).

The logo is searched in the following directories:

..\Projekt_000\CustomData\Bitmap

..\Projekt_000\OemData\Bitmap

..\BasicData\Bitmap

If the same file is in several directories, the first found logo is used.

Note: The preview shows the logo that is, in fact, printed.

It is possible to use an own logo as standard by storing it under the name Userdoc_Logo.emf for example in the directory CustomData\Bitmap

Possible settings

Standard

- The standard logo is printed (Userdoc_Logo.emf)

In an user-defined way

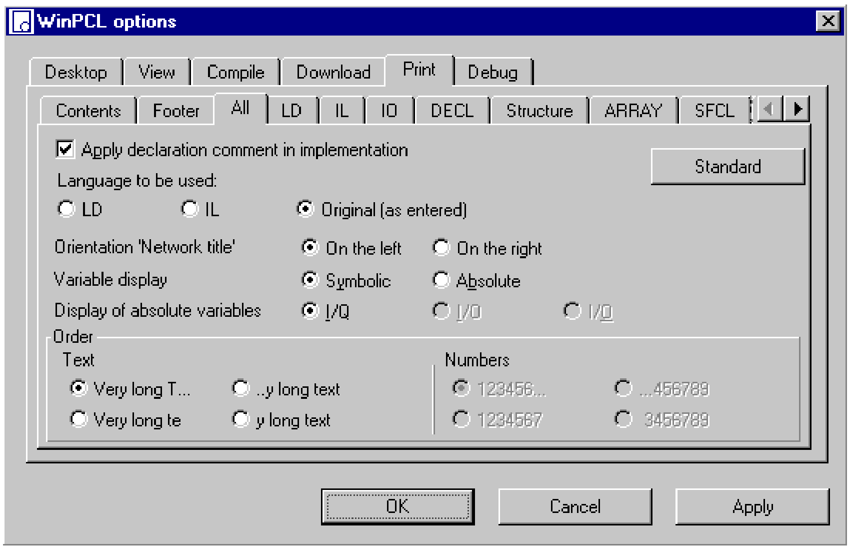
- An own logo can be selected
- Only the name of the file is indicated. The corresponding directory is searched according to the above described rule.

Note: The selection file takes over only the name of the file and not the selected path.

Without

- No print out of a logo. Instead of the logo two additional texts can be entered in the footer.

Print Options, View of All Editors (All)



Print_all.bmp

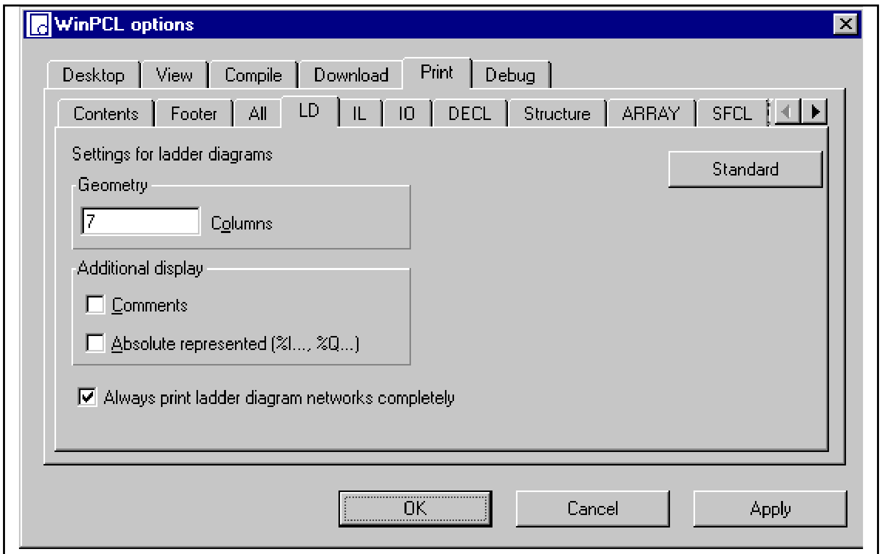
Fig. 4-20: Print options, view of all editors (All)

Print options	Comment
Apply declaration comment in implementation	The declaration comment of variables is printed in the implementation as a default-comment.
Language to be used	**LD** All rungs that can be represented in the LD are printed in the LD. **IL** All rungs are printed in the IL. **Original** All rungs are printed in the language set and stored in the editor.
Orientation 'Rung title'	Right- or left-justified orientation of the rung title in the LD or IL.
Variable display 'Symbolic / Absolute'	Instead of the variable's name its IO address can be printed.

Print options	Comment
Display of absolute variables	Representation with the letters I/Q, E/A or I/O.
Truncating very long texts	Selection where to truncate, on the right or left side, with or without "...".
Truncating very long numbers	Selection where to truncate, on the right or left side, with or without "...".

Fig. 4-21: Complement to printing options, view of all editors (All)

Print Options, Ladder Diagram Editor (LD)



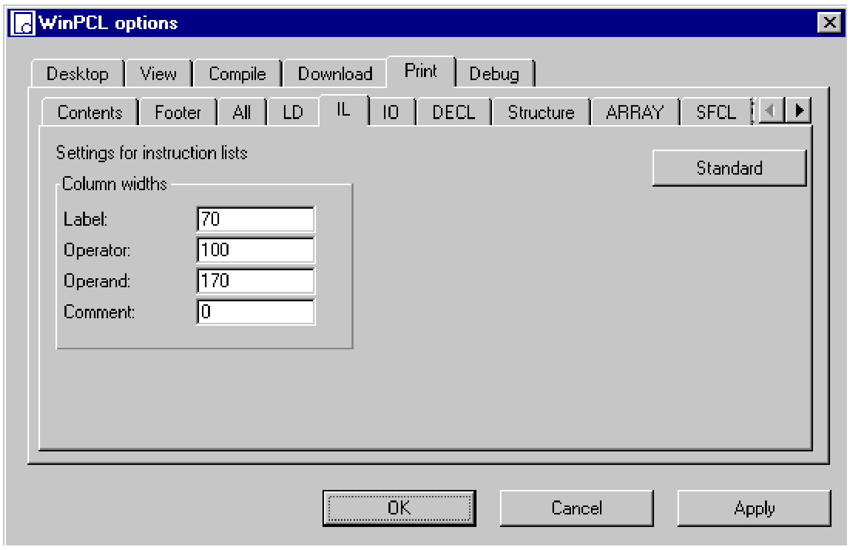
print_kop.bmp

Fig. 4-22: Print options, ladder diagram editor (LD)

Print Options	Meaning
Settings for ladder diagrams:	
Geometry	This option can be used to specify the number and width the columns.
Additional display:	
Comments	The comment on the variable is printed above the ladders.
Direct represented %I..%Q	The absolute address of the variable is printed above the ladder.
Always print ladder diagram rungs completely	LD rungs are printed on the following page, if they do not fit anymore on the current page. If the rung does not fit on one page, it is divided up.

Fig. 4-23: Supplement to print options, ladder diagram editor (LD)

Print Options, Instruction List Editor (IL)



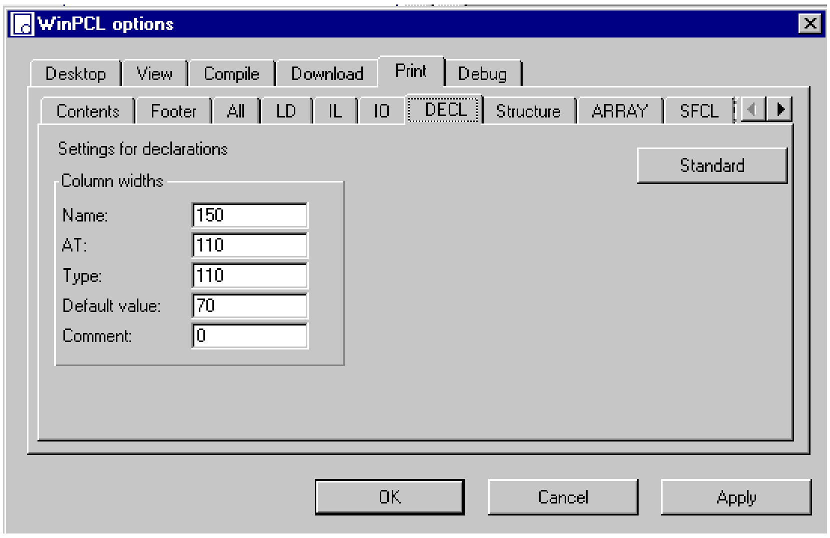
print_awl.bmp

Fig. 4-24: Print options, instruction list editor (IL)

On this page you can set the column widths for the printout of the instruction list editor.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, Declaration Editor (DECL)



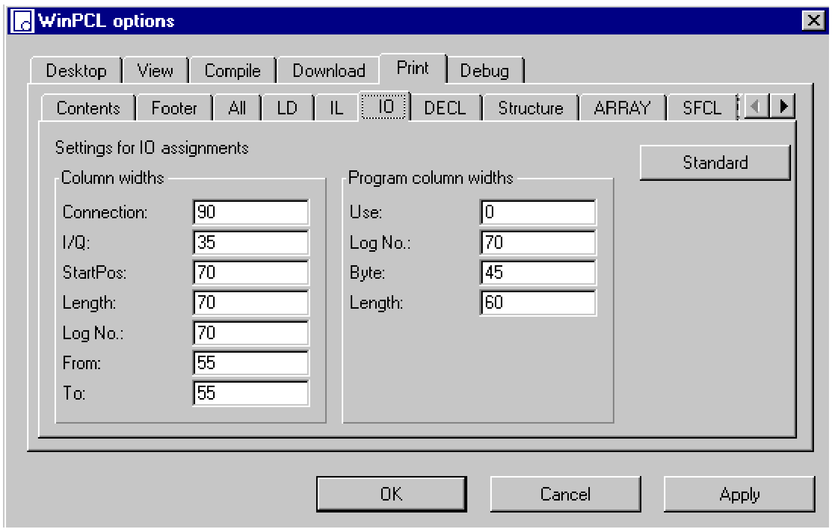
print_decl.bmp

Fig. 4-25: Print options, declaration editor (DECL)

On this page you can set the column widths for the printout of declaration editor.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, IO Editor (IO)



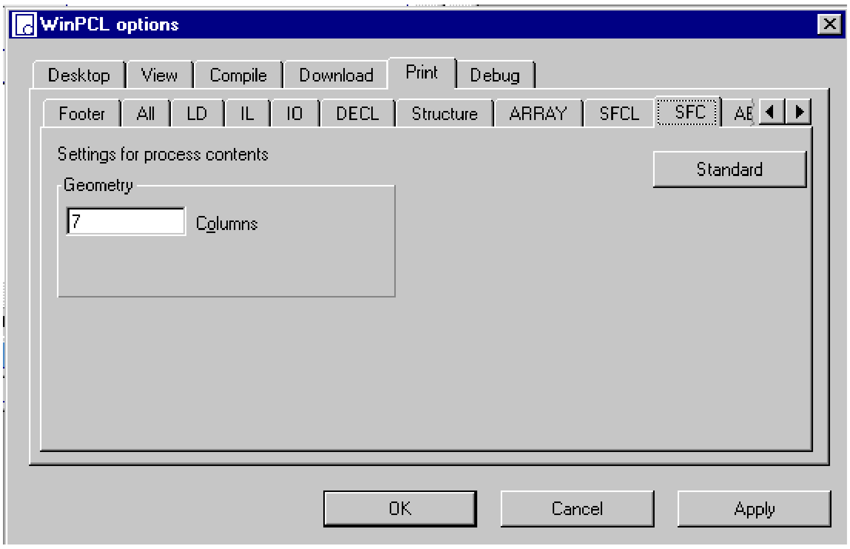
print_io.bmp

Fig. 4-26: Print options, IO editor (IO)

On this page you can set the column widths for the printout of the IO editor.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, SFC List (SFC)

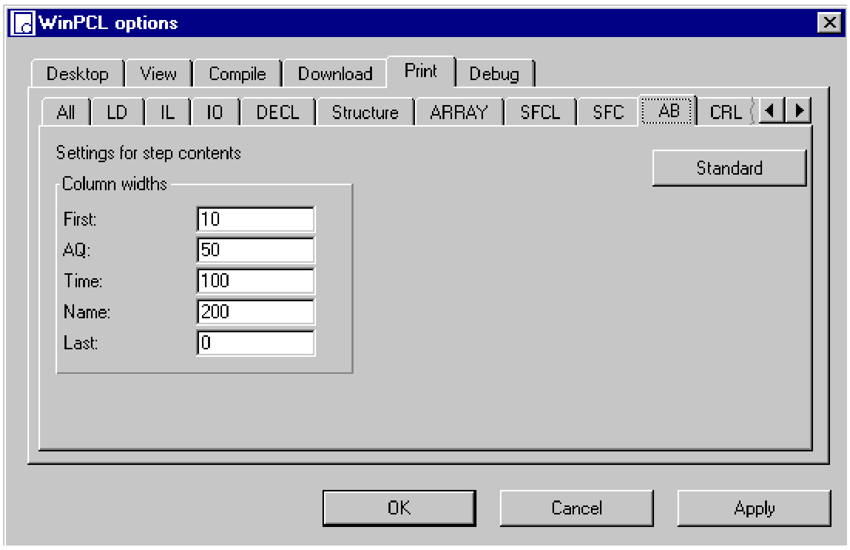


print_sfc.bmp

Fig. 4-27: Print options, SFC list (SFC)

On this page you can set the column widths for the printout of the SFC list.

Print Options, Action Block Editor (AB)



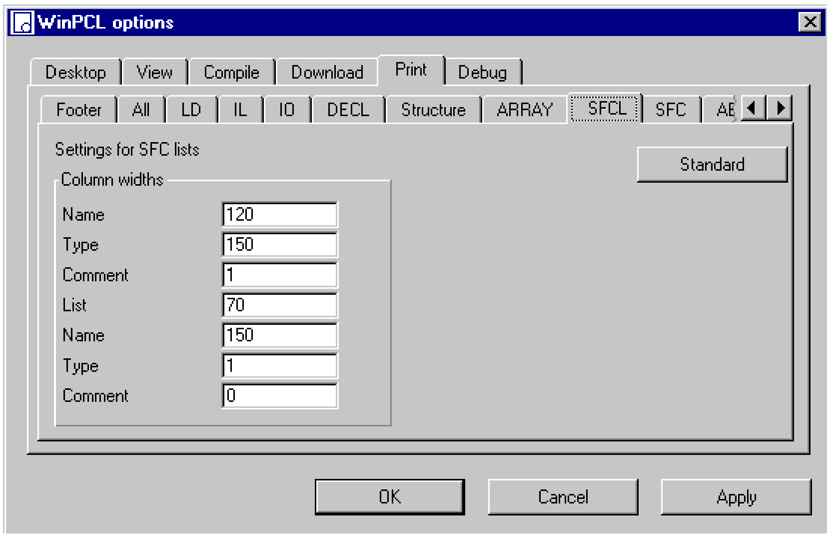
print_ab.bmp

Fig. 4-28: Print options, action block (AB)

On this page you can set the column widths for the printout of the action block editor.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, SFCL List (SFCL)



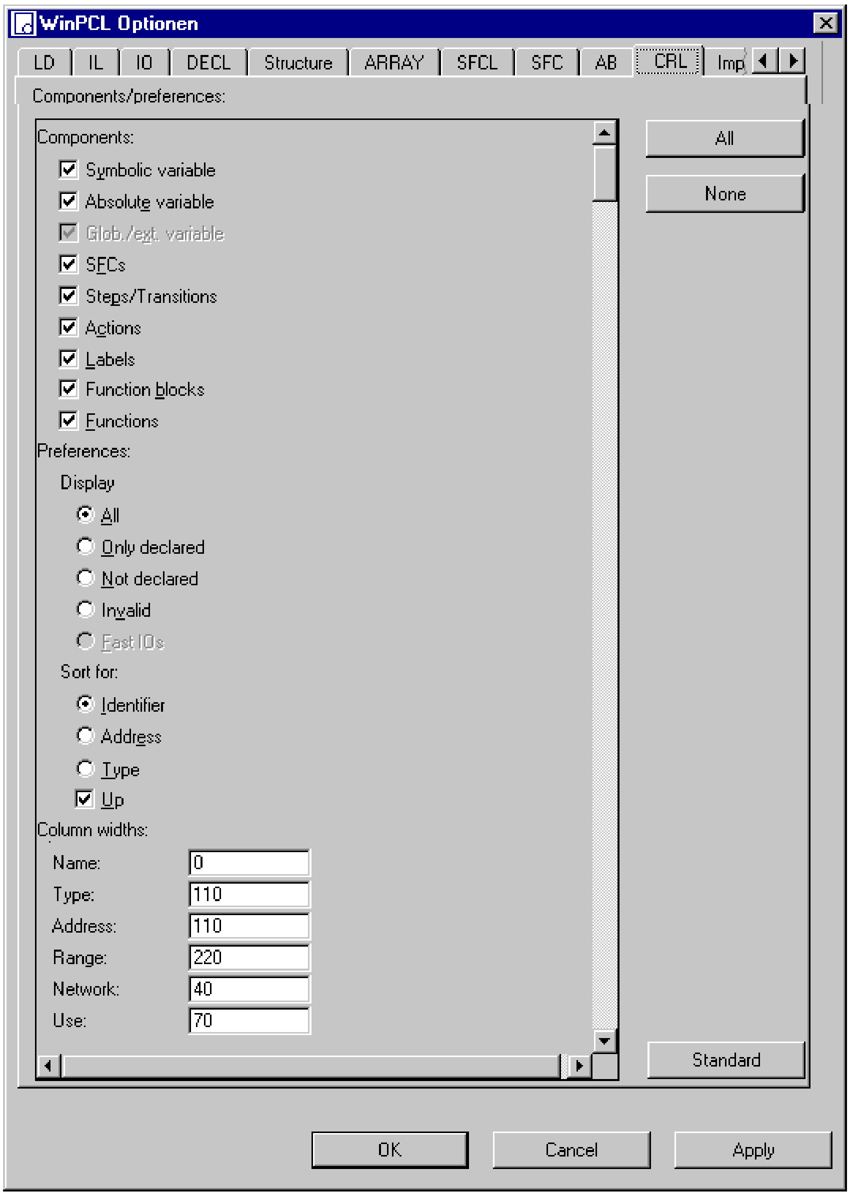
print_sfcl.bmp

Fig. 4-29: Print options, SFCL list (SFC Lists)

On this page you can set the column widths for the printout of the SFCL list.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, Cross Reference List (CRL)



print_qvl.bmp

Fig. 4-30: Print options, cross reference list (CRL)

The desired components for the cross-reference list can be selected as shown in the figure above.

All / Only declared / Not declared / Invalid cross references can be printed.

They can be sorted in ascending or descending order by identifier / address or type.

Furthermore, the column width can be preset.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

The button "All" and the button "None" accelerate the selection of the components.

Print Options, ARRAYs

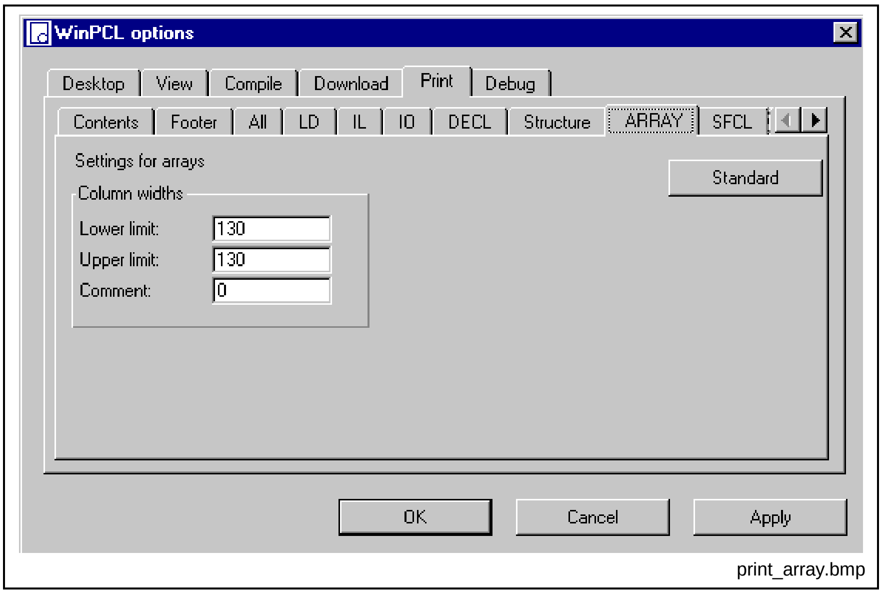


Fig. 4-31: Print options, arrays)

On this page you can set the column widths for the printout of the arrays.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, Structures

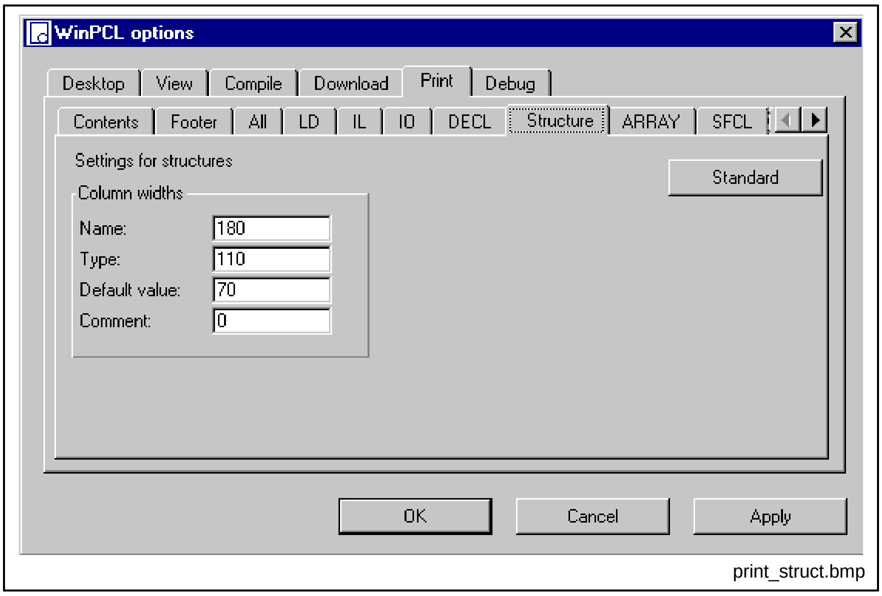


Fig. 4-32: Print options, structures)

On this page you can set the column widths for the printout of the structures.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Print Options, Imports

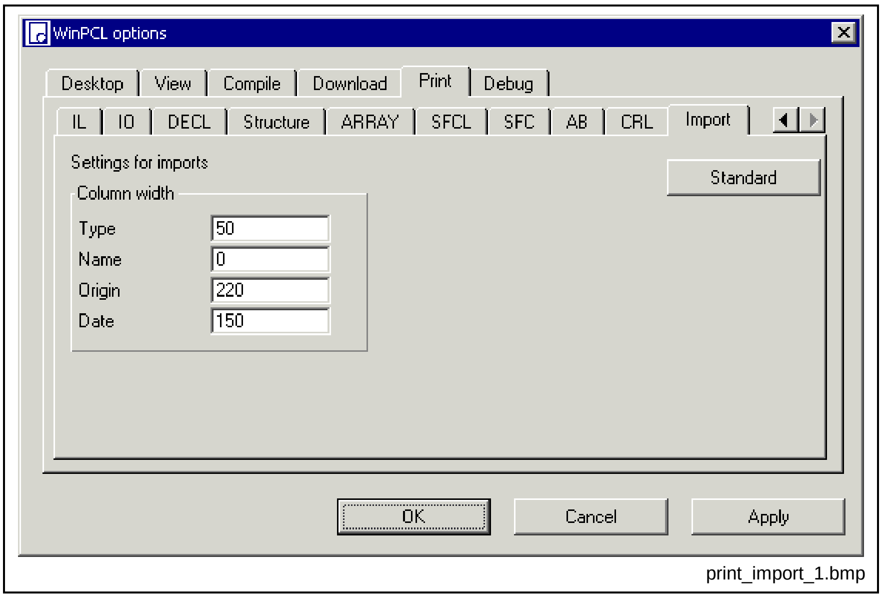


Fig. 4-33: Print options, imports

On this page you can set the column widths for the printout of the imports.

If you enter 0 as column width, the rest of the page is automatically assigned to this column.

Printer Selection

The printing process itself is initiated with the usual Windows print window.

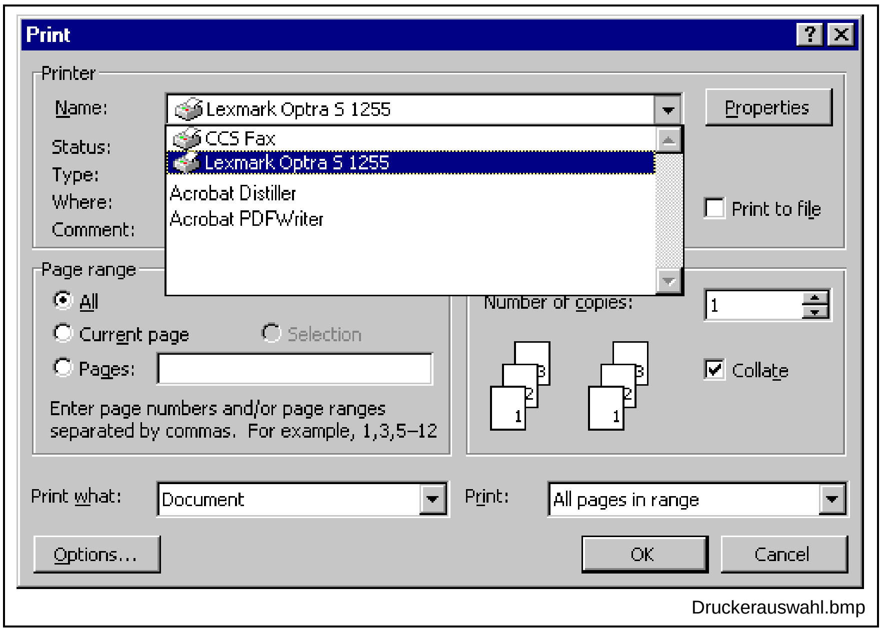


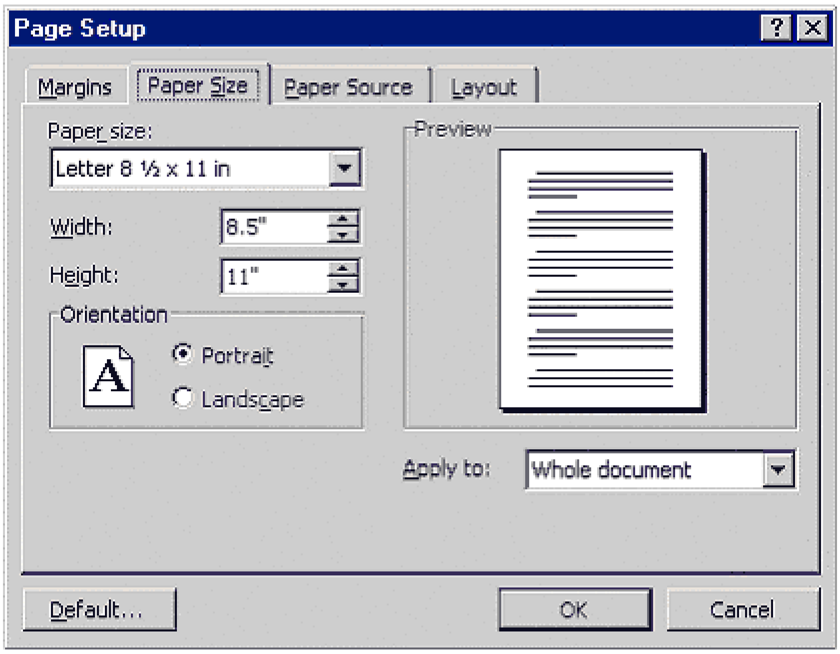
Fig. 4-34: Printer selection

As the number of pages of the desired files can vary depending on the set print options, we recommend to create a PDF file before printing is started to find out the required amount of paper (additional installation - not included in the Rexroth scope of delivery!)

Page Setup …

As for all Windows programs, the printer-dependent possibilities to set-up the printed page are shown under this menu item.

Note: Format limitations are specified by the printer or printer driver.

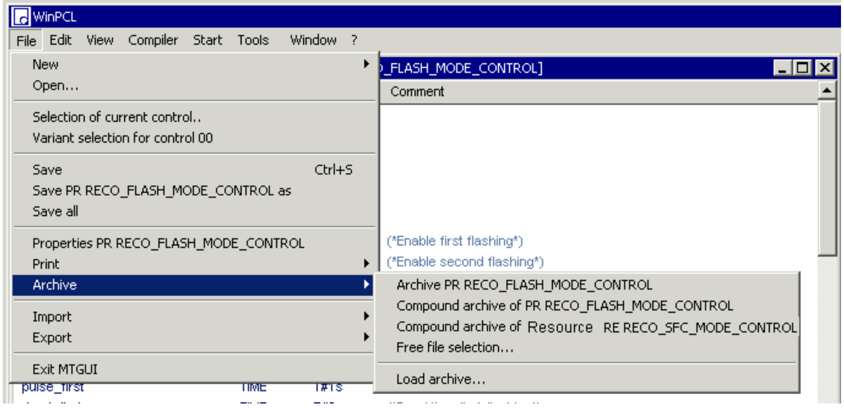


Seite_einrichten.bmp

Fig. 4-35: Set up page for printing

Archive

The programming system offers numerous methods for creating file and project archives.



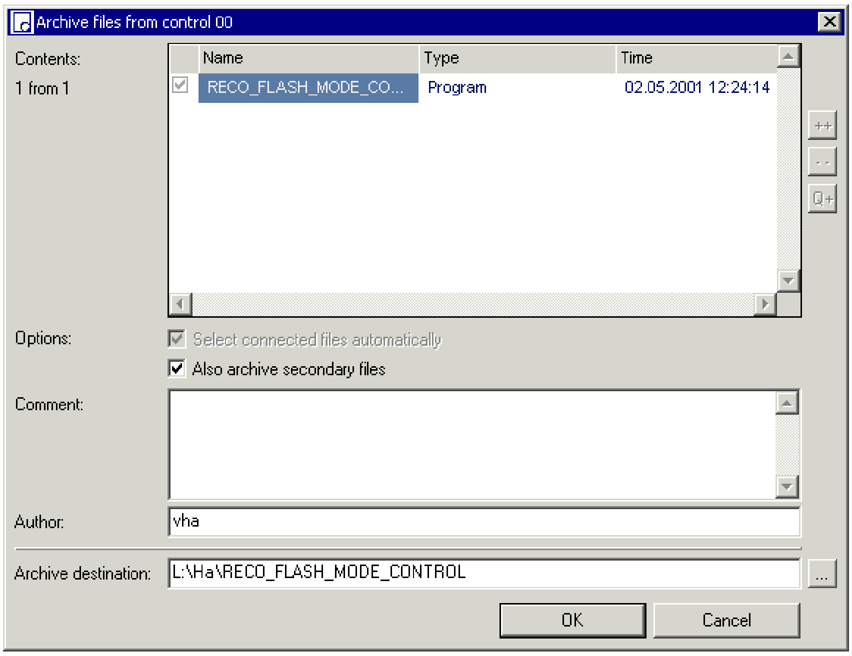
menue_ablage.bmp

Fig. 4-36: "File / Archive" menu item

The figure of the "File / Archive" menu item shows an overview of all archive methods:

Archive "xx"

The focused file can be stored with this menu item.



Ablage_datei.bmp

Fig. 4-37: File archive

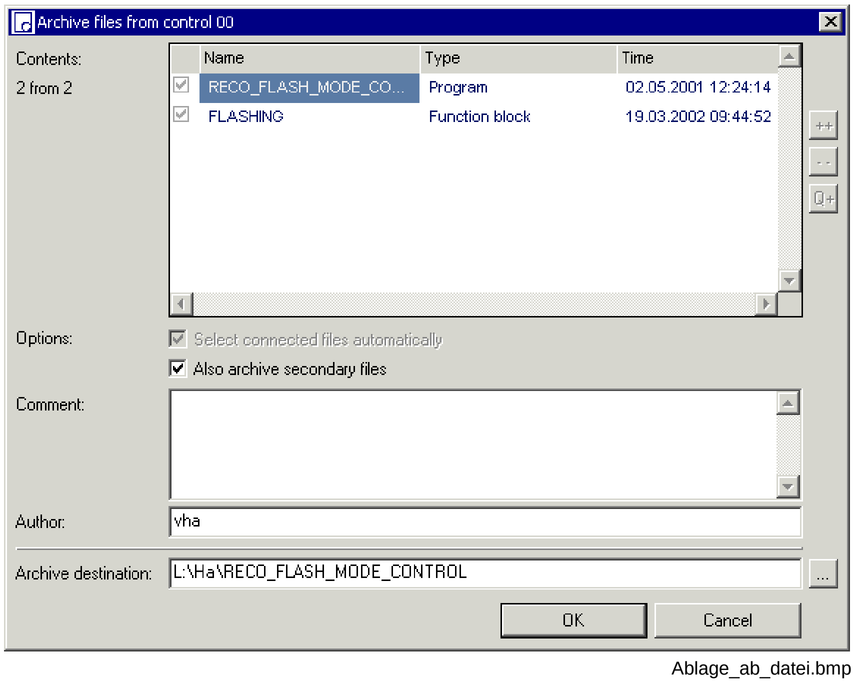
The currently focused file is displayed. It can be stored together with its secondary files and, if required, to the destination archive including comments.

The destination archive and the name of the file to be stored to the archive can be defined by the user.

Note:	If a name is entered with point such as 123.456, the file is archived under the name 123.apv. Res.tst is archived as Res.apv. If the file name is entered completely with Extension.apv that is added normally automatically, the archive is generated correctly with Res.tst.apv.

Compound archive of "xx"

This menu item starts archiving the focused file and all of the files used by it.



Ablage_ab_datei.bmp

Fig. 4-38: Compound archive beginning with the loaded file

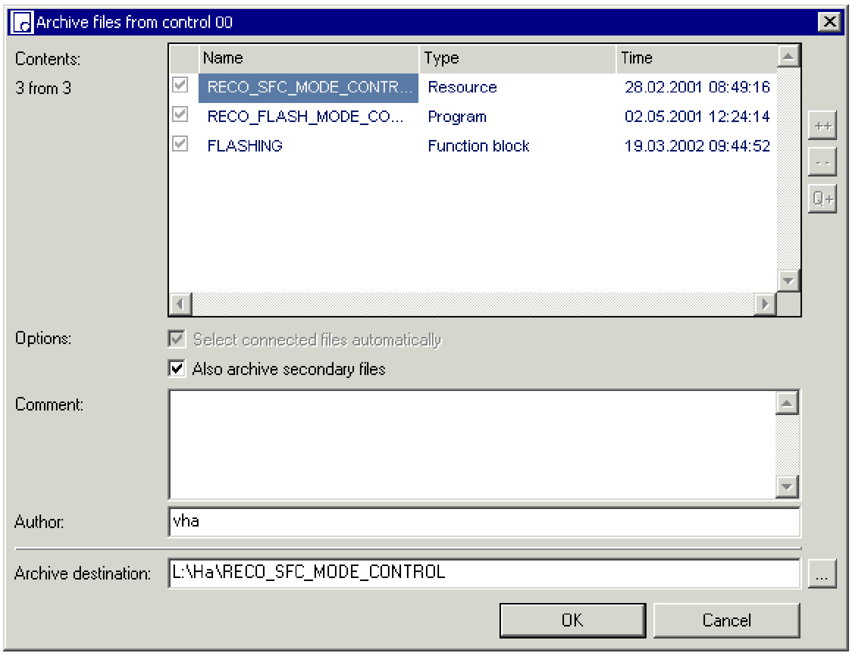
The file just being focused and its used files are displayed. They can be stored to the archive together with their secondary files and, if required, in the destination archive including comments.

The destination archive and the name of the file to be stored to the archive can be defined by the user.

Note:	If a name is entered with point such as 123.456, the file is stored to the archive under the name 123.apv. Res.tst is archived as Res.apv. If the file name is entered completely with Extension.apv that is added normally automatically, the archive is generated correctly with Res.tst.apv.

Compound archive starting form current resource

The menu item starts archiving the current resource (settings under "Compiler / Selection of current resource") and all files used by it.



Ablage_hauptdatei.bmp

Fig. 4-39: Compound archive starting from the current resource

The current resource and its used files are displayed. They can be stored to the archive together with their secondary files and, if required, in the destination archive including comments.

The destination archive and the name of the file to be stored to the archive can be defined by the user.

Note: If a name is entered with point such as 123.456, the file is stored to the archive under the name 123.apv.

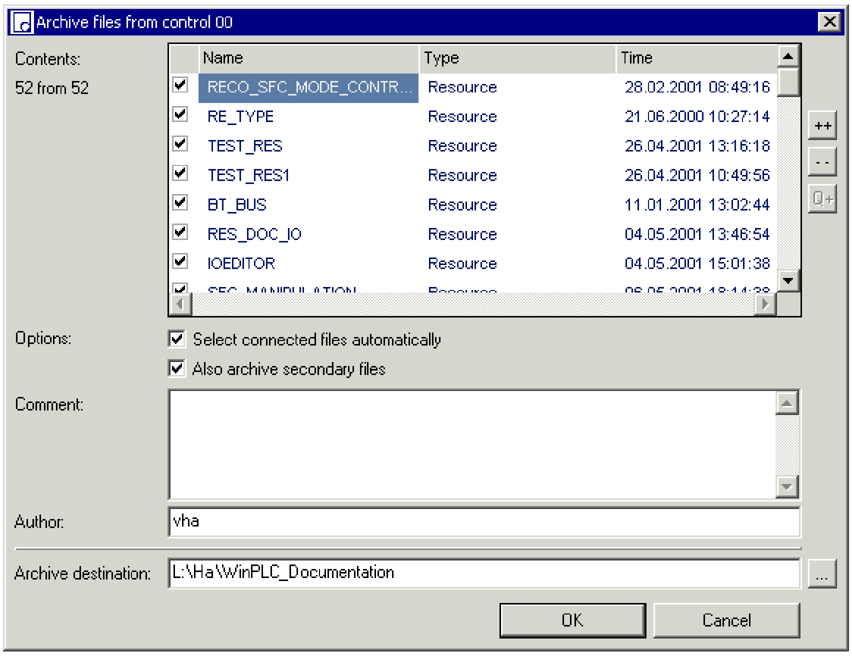
Res.tst is archived as Res.apv.

If the file name is entered completely with Extension.apv that is added normally automatically, the archive is generated correctly with Res.tst.apv.

Note: The focused file is contained only if it is used by the current resource.

Free File Selection

The "Free file selection" allows to archive any file selected by the user.



Ablage_frei.bmp

Fig. 4-40: Free file selection

All PLC files of the variant are displayed. They can be selected individually.

An options checkbox allows to select whether the connected files and, if necessary, the secondary files are to be archived as well.

A comment is possible.

The archive destination and the name of the file to be stored can be defined.

Note: If a name is entered with point such as 123.456, the file is stored to the archive under the name 123.apv.

Res.tst is archived as Res.apv.

If the file name is entered completely with Extension.apv that is added normally automatically, the archive is generated correctly with Res.tst.apv.

Load Archive

The menu item serves for loading a stored file.

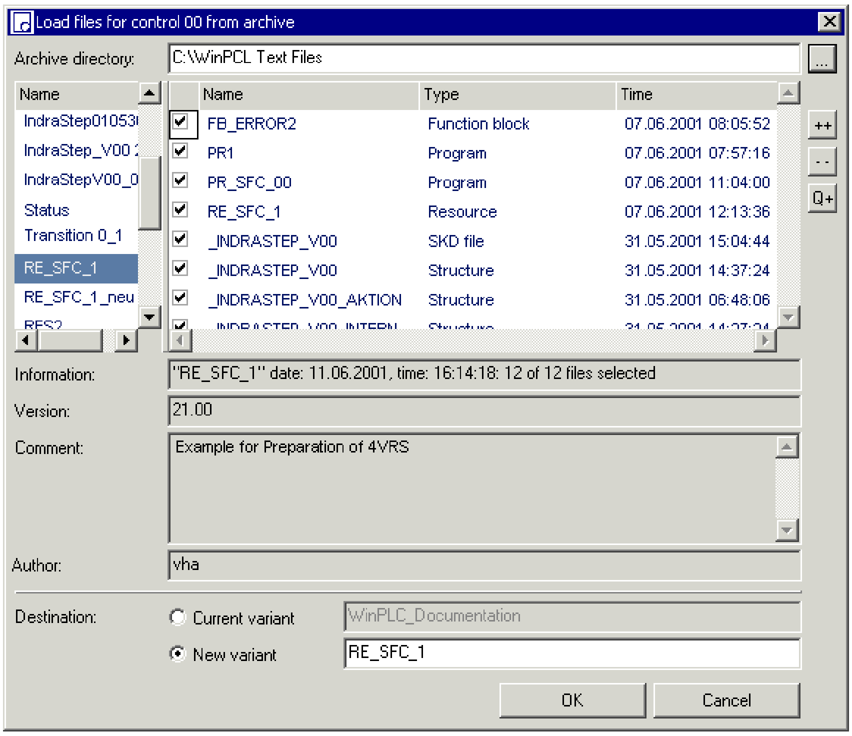


Fig. 4-41: Load archive

The storage directory, the desired file is to be loaded from, can be set with this menu item. The names of the different archive files which are residing in this directory are displayed to the left, if this option is activated.

Files which are residing in the archive file selected, can be selected separately to the right.

A new variant can be defined as destination archive or the file can be applied to the current variant.

Note: If the file is applied to the current variant, files with same names are overwritten.

Archives provided to support the firmware functionality

WinHMI and WinPCL are using the folders

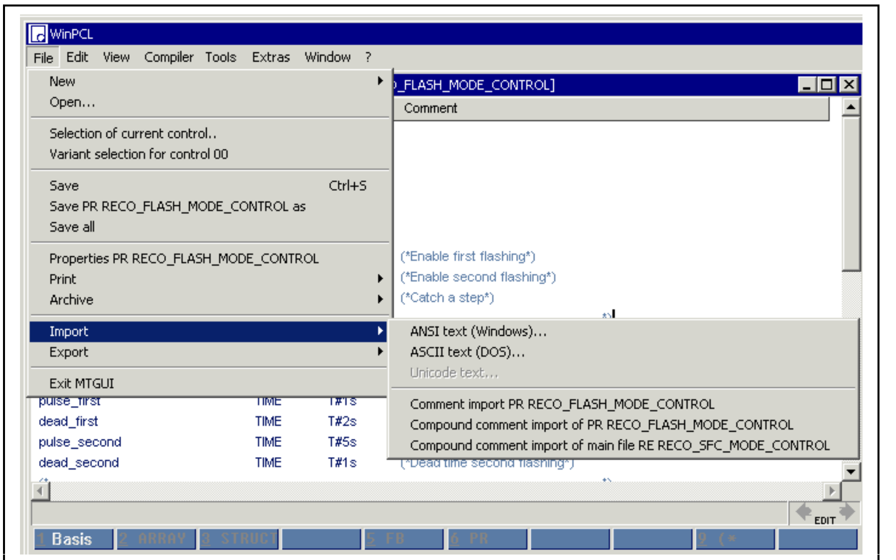
- ...Mtgui\BasicData\TEMPLEATES
- ...WinPCL\BasicData\TEMPLATES

for archiving templates which support the firmware functionality (e.g. ibs_control.apv for INTERBUS file types, functions and function blocks).

Import

Import of Files and Compound Files

This menu item allows an import of text files and compiles these files into PLC files.



import.bmp

Fig. 4-42: File import

Import is possible for the following files:

- DOS_ASCII text files from PCL
- WIN_ANSI text files, exported by means of the "File / Export" file menu
- if enabled, Unicode files, exported by means of the "File / Export" menu item

Comment Import

(Also see **Fehler! Verweisquelle konnte nicht gefunden werden.**)

- Comment import for file nn
- Compound comment import of file nn
- Compound comment import of the current resource nn

Export

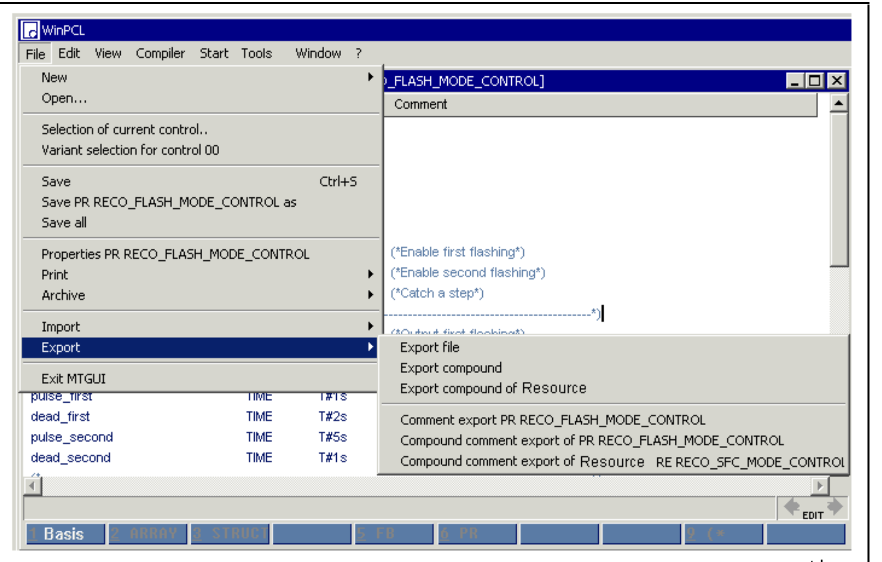
This menu item allows ANSI export.

In addition, it allows export of comments for compilation.

Export of Files and Compound Files

This menu item allows ANSI export

- of the focused file,
- of the focused file and the files used by it,
- of all files, starting with the current resource.



export.bmp

Fig. 4-43: Export

The archive destination (and the file name outside of WinPCL) can be selected for ANSI export.

Note: The text file itself contains a second name in the file, which is activated later when it is imported. This is the original name of the file in WinPCL.

Compounds are saved in a file.

Comment Export

In addition, this menu item allows export of comments for compilation.

(Also see Comment Import)

- Comment export for file nn
- Compound comment export of file nn

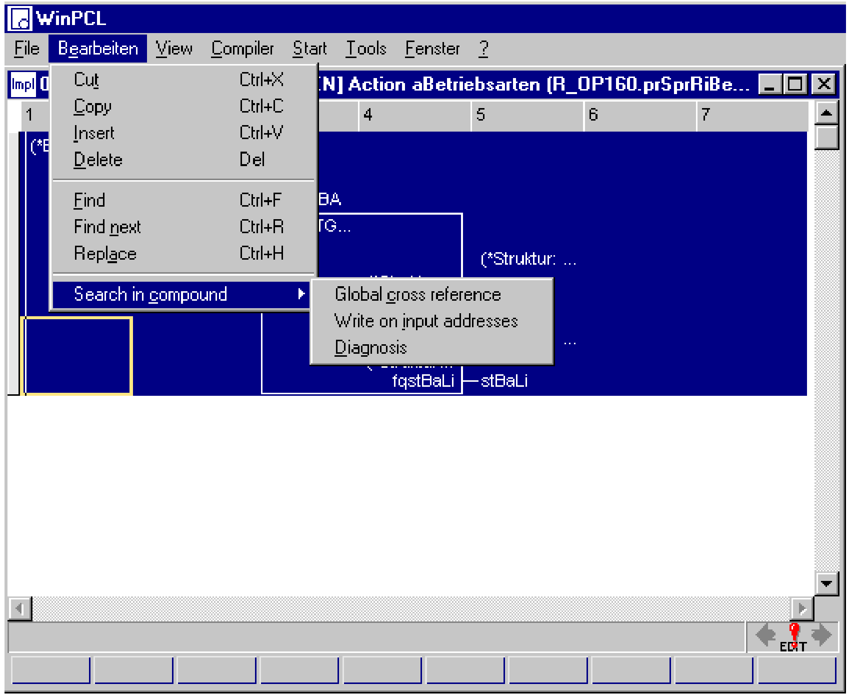
Compound comment export of current resource nn

Exit

This menu item closes the program. Open and modified files are stored after confirmation of a security prompt.

4.3 Edit

The "Edit" menu item comprises a group of block commands and the "Find / Replace" group.



menue_bearbeiten.bmp

Fig. 4-44: "Edit" menu item

Block command group

- Cut <Ctrl>+<X>
- Copy <Ctrl>+<C>
- Insert <Ctrl>+<V>
- Delete <Del>

"Find / Replace" group

- Find <Ctrl>+<F>
- Find Next <Ctrl>+<R>
- Replace <Ctrl>+<H>.

"Search in compound" group

- Global Cross Reference - Search in Compound
- Write on Input Addresses - Search in Compound
- Diagnosis, Search in Compound

Cut <Ctrl>+<X>

is a standard Windows command. This command is used to remove the particular text passage / block selected and to file it in the clipboard (also see Copy <Ctrl>+<C>).

Note: If a block is cut in the declaration editor, the data types are lost together with the variables. In the implementation, the applications then change their color to red - error!

Copy <Ctrl>+<C>

is a standard Windows command. Using this command, the text passage / block selected is filed in the clipboard, being preserved (also see Cut <Ctrl>+<X>).

Insert <Ctrl>+<V>

is a standard Windows command. This command applies the text which was filed in the clipboard with the "Copy" command to the current editor.

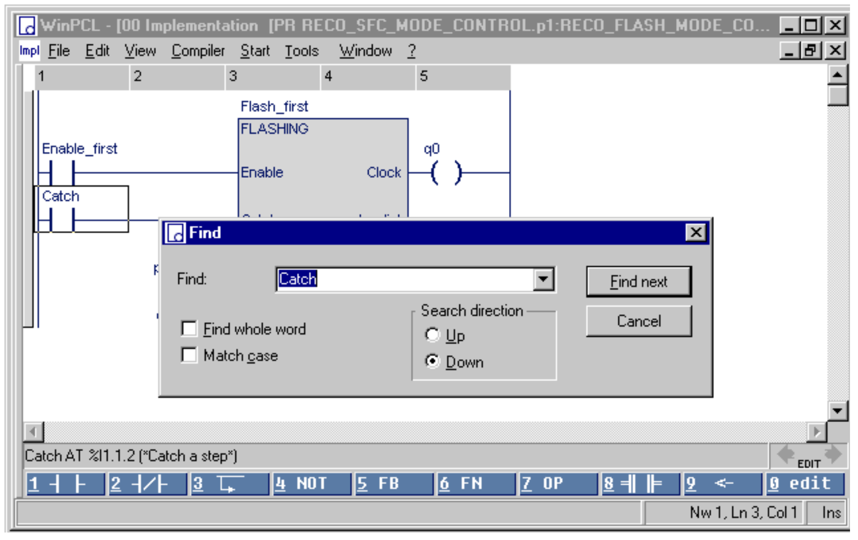
Note: If a text block is copied into the declaration editor, the added variables with same name are indicated in red - error!

Delete <Del>

is a standard Windows command for deleting the text passage / block selected.

Note: If a text block is deleted in the declaration editor, the data types are lost together with the variables. In the implementation, the applications then change their color to red - error!

Find <Ctrl>+<F>



suchen.bmp

Fig. 4-45: Finding a character string

The find function is a standard Windows command. Enter the text to be found in the "Find" field and click on the "Find next" button. The cursor stops on the search criterion.

The find function can be restricted to

- Find whole word, and
- Match case.

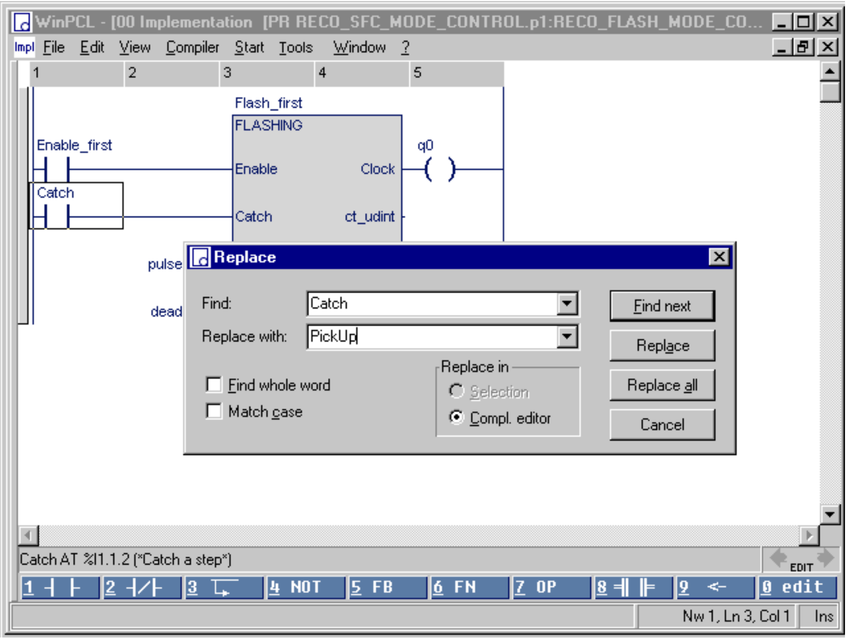
Furthermore the search direction can be defined.

Once started, any search process can be continued by pressing the Find Next <Ctrl>+<R> button.

Find Next <Ctrl>+<R>

This standard Windows command continues the already started search for a specified text. The set options are preserved (Find <Ctrl>+<F>).

Replace <Ctrl>+<H>



ersetzen.bmp

Fig. 4-46: Finding and replacing of a character string

The standard Windows command searches for a given term. The cursor stops on the search criterion. The found term is replaced by pressing the "Replace" button. Pressing the "Replace all" button replaces the found term automatically replaced at any point where it occurs.

The find function can be restricted to

- Find whole word, and
- Match case.

The area the be searched through can be defined; either

- search in the complete editor or
- only in the block selected with the "Selection" option.

Search in Compound

comprises the present options of searching for variables or instances (Global Cross Reference - Search in Compound), for input variables %I (Write on Input Addresses - Search in Compound) or for diagnosis information (Diagnosis, Search in Compound) within the entire compound.

Global Cross Reference - Search in Compound

This menu item activates the Cross Reference Help, with the user having to enter the text to be found. The cross-reference help can be used for the following items:

- File
- File and files used by it
- From the current resource

Note: At present, the selection window displaying the cross references found can only be used for loading files, but not for selecting instances of these files. For that reason, a **status display** is **not possible** in the respective editor.

Write on Input Addresses - Search in Compound

This menu item activates the Cross Reference Help for finding all inputs which are written to. The search can be used for the following items:

- File
- File and files used by it
- From the current resource

(also see Properties, allow to write on inputs, IO simulation).

Diagnosis, Search in Compound

(Menu item not yet released)

This menu item activates the search for the elements of ProVi messages and SFC diagnoses.

In the appearing selection dialog you can specify which elements of the diagnosis are to be searched for.

Then, the diagnosis elements "Module number" "Message type" and "Message number" must be exactly specified in the following input field. So, you have to specify, for example, for "Message type", if it is to be searched for "Error", "Status message", "Warning", "Start condition" or "Startup diagnosis".

During the choice of "Criterion analysis" the input field becomes blind.

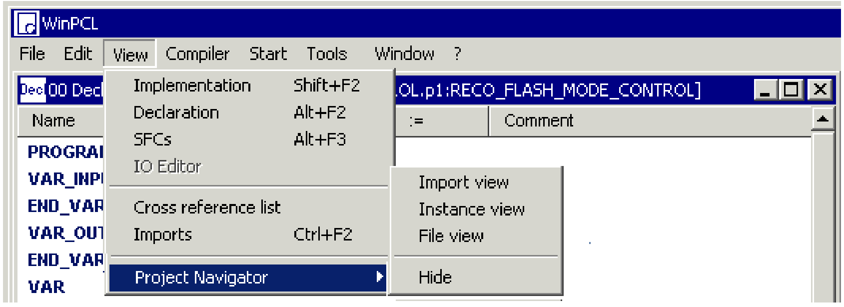
As result a window is shown, in which all uses of the "Message type" "Error" are listed. The diagnosis elements for which are searched can be found in the first column of the cross reference help.

In this window you find commands already known from the cross reference help, so for example cursor commands for the navigation and the branching to the concrete position.

The search can be used for the following items:

- File
- File and files used by it
- From the current resource

4.4 View



menue_ansicht.bmp

Fig. 4-47: "View" menu item

The "View" menu item combines the main components (editors) of the programming system. For reasons of a better understandability these editors are described in separate chapters.

Implementation Implementation <Shift>+<F2>, described in chapter >>Instruction List Editor<< and chapter >>Ladder Diagram Editor<<

Declaration Declaration <Alt>+<F2>, described in chapter 3, >>Declaration Editors<<

IO Editor IO editor, described in chapter >>IO Editor<<

SFCs SFCs <Alt>+<F3>, are described in Chapter >>Sequential Function Chart Editor << and in Chapter >>Action Block Editor<<

The menu items of the next group are described subsequent to this overview:

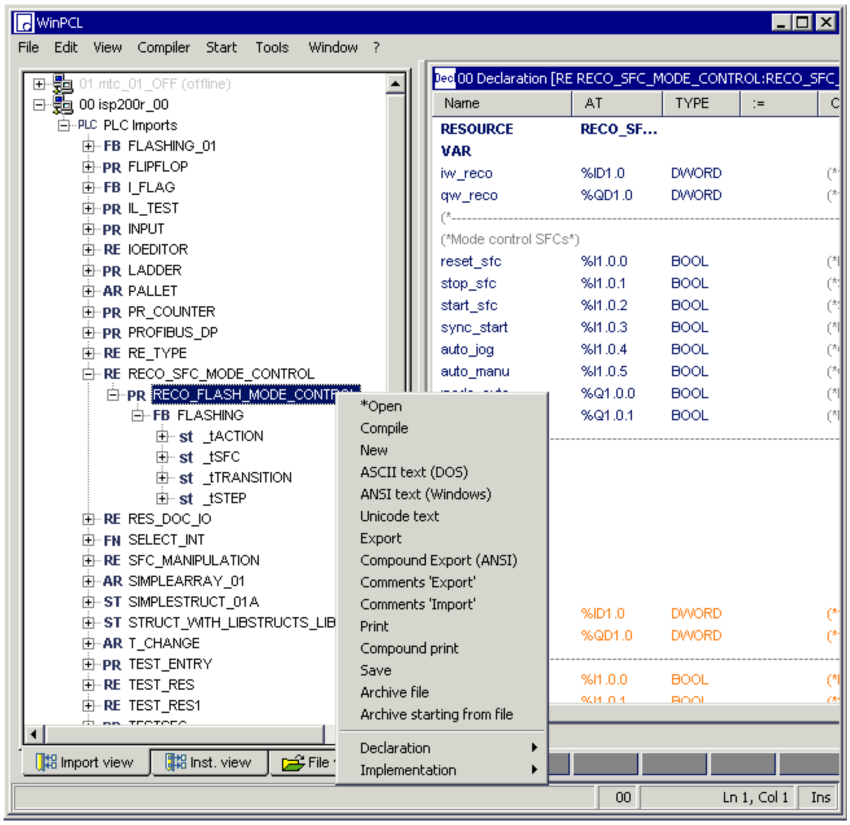
- Cross Reference List (and Cross Reference Help)
- Import <Ctrl>+<F2>.

Project Navigator

The project navigator appears as a docked window to the left of the screen. As is known from an explorer, the project navigator also displays tree structures in the "File", "Import" and "Instance" views of the POUs of the basic directory or the variant selected.

Import View

The import view shows the files used at least once by the current file.



Pronav_Import_01.bmp

RE	Resource (user)
PR	Program (user)
FB	User function block
fb	Firmware/standard function block
FN	User function
fn	Firmware/standard function
ST	User structure
st	Firmware/standard structure
AR	User array
ar	Firmware/standard array

Fig. 4-48: Import view of a variant

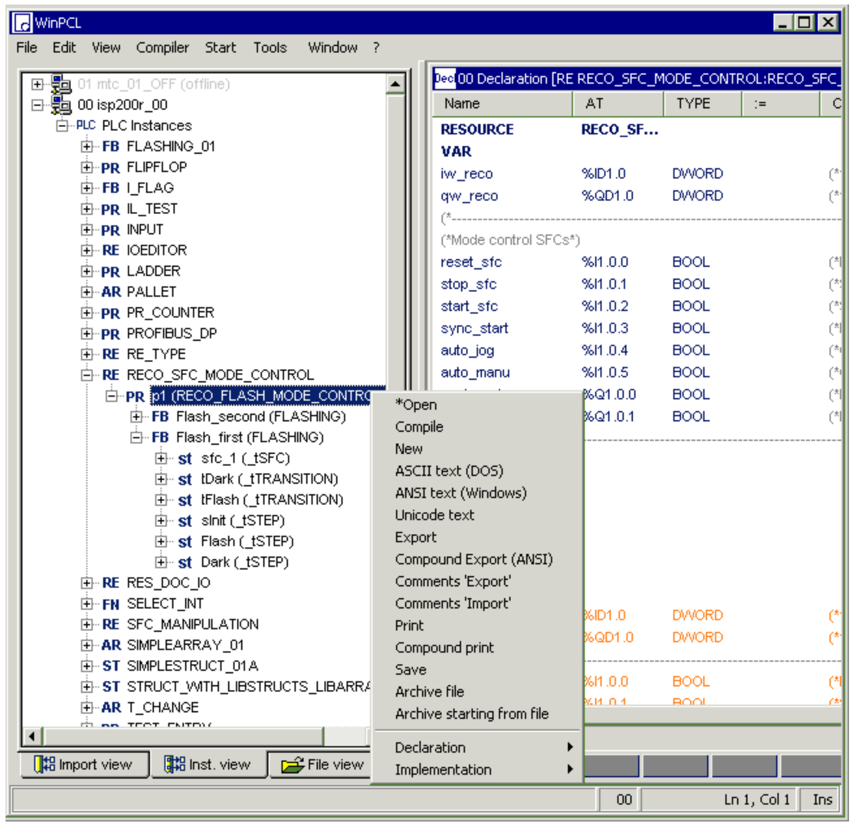
(Also see Import <Ctrl>+<F2>.)

Various pop-up functions (<Shift>+<F10>, right mouse button) are assigned to the various items within the tree.

- "Control" line: refresh; the structure is rescanned
- "PLC Imports" line: Transition to WinPCL Options
- "RE/PR/FB/FN/ST/AR" lines: Open the file (standard, double-click), further commands in the upper pop-up area, as well as transitions to possible editors in the lower area
- "fb/fb/st/ar" lines: open the declaration part for display . . .

Instance View

The instance view displays the file instances used by the currently selected instance. The instance name is always displayed first and then, in brackets, the type name. If used within one file, several instances of the same type appear independently of each other.



Pronav_Instance_01.bmp

RE	Resource (user)
PR	Program (user)
FB	User function block
fb	Firmware/standard function block
FN	User function
fn	Firmware/standard function
ST	User structure
st	Firmware/standard structure
AR	User array
ar	Firmware/standard array

Fig. 4-49: Instance view of a variant

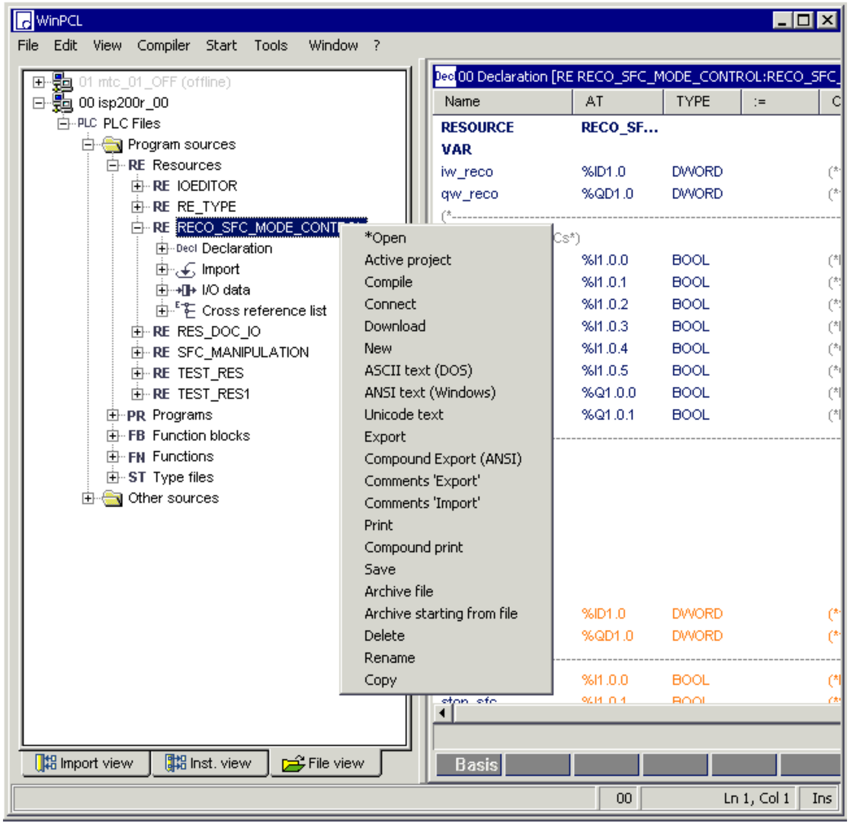
(Also see Import <Ctrl>+<F2>.)

Various pop-up functions (<Shift>+<F10>, right mouse button) are assigned to the various items within the tree.

- "Control" line: refresh; the structure is rescanned
- "Instances" line: Transition to WinPCL Options
- "RE/PR/FB/FN/ST/AR" lines: open the file instance, e.g. for displaying the status with running control (standard, double-click), further commands in the upper pop-up area, as well as transitions to possible editors in the lower area.
- "fb/fb/st/ar" lines: open the declaration part for display . . .

File View

The file view displays the files which are included in the basic directory or in the current variant, independent of the way they are used. The files are each arranged by RE/PR/FB/FN/AR/ST order. Each file is followed by the transitions to possible editors.



Pronav_File_01.bmp

RE	Resource (user)
PR	Program (user)
FB	User function block
fb	Firmware/standard function block
FN	User function
fn	Firmware/standard function
ST	User structure
st	Firmware/standard structure
AR	User array
ar	Firmware/standard array

Fig. 4-50: File overview of a variant

Various pop-up functions (<Shift>+<F10>, right mouse button) are assigned to the various items within the tree.

- "Control" line: refresh; the structure is rescanned
- "Program sources" line: permits creation of new RE/PR/FB/FN/AR/ST using the pop-up menu To make them visible in the project navigator, the tree must be rescanned in the "Control": refresh line.
- New RE/PR/FB/FN/AR/ST can also be created in the "Resources", "Programs", "Function blocks", "Functions", etc., using the pop-up menu.
- "RE/PR/FB/FN/ST/AR" lines: using the pop-up menu: open the file in the particular editor as well as further currently available commands.
- "fb/fb/st/ar" lines: open the file in the particular editor using the pop-up menu.

Cross Reference List (and Cross Reference Help)

- In the first version, the cross reference list (CRL) relates to the focused program organization unit. The CRL shows those elements of the program organization unit which are preset in the "WinPCL Options, Cross Reference List (CRL)", their declaration sites and the locations where they are used (Cross Reference on Resource Level, Cross Reference on PR / FB FN level). The cross reference list can be called up using the "View / Cross reference list" menu item.
- The Cross Reference Help relates to the element where the cursor is currently positioned. Here, the user has the option whether to search in the file, from the file, or in the complete compound. The cross reference help can be called up in the pop-up menu of the particular editor, using the right mouse button or <Shift>+<F10>.

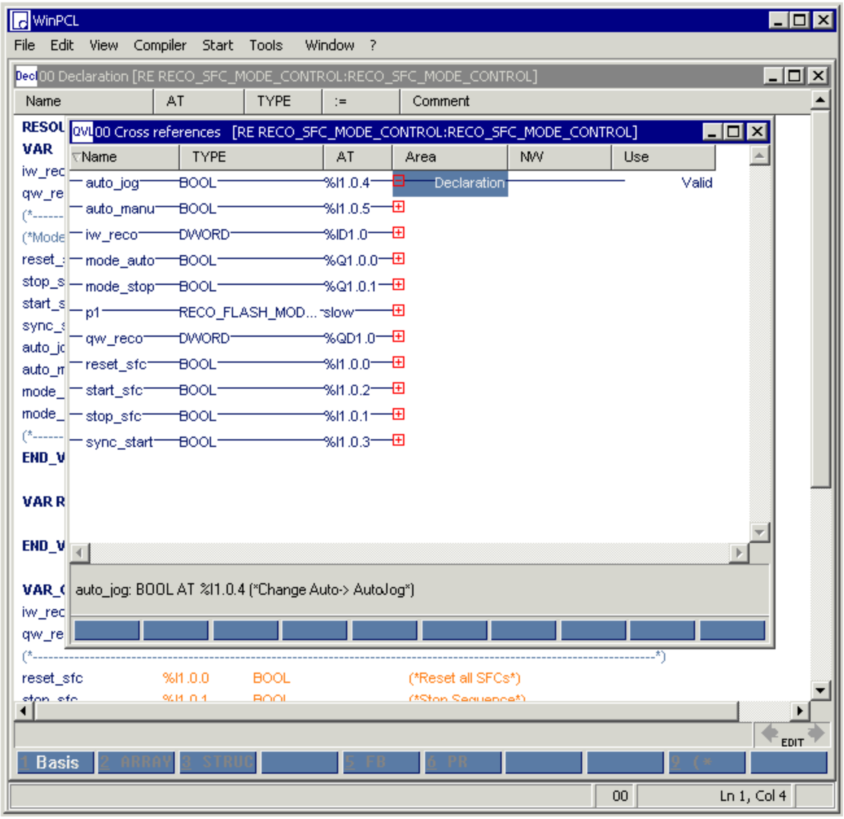
Cross Reference List Pop-up Menu <Shift>+<F10>

This pop-up menu contains the commands which are essential to this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	**Explanation**
Go to place of use	Opens the editor required; the cursor is on the desired position.
Declaration help	Description of the data type of the current element, where the cursor is positioned.
Cross reference help	List of all places where the current element is used. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force (PLC in operating mode "STATUS")	Allows the entry of a variable name. The value of the variables is displayed and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of program code execution.
Status ARRAYs / Structures	Display of the status of array and structure elements, forcing by pressing the <Shift>+<F10> keys or the right mouse button.
Options	* Optimization of the column width. * Expanding; completely opening the tree structure * Arranging the current column alphabetically, in ascending or descending order.
Print current window.... <Ctrl>+<P>	Print of the editor contents by pressing <Ctrl>+<P>.

Fig. 4-51: Cross reference list pop-up menu

Cross Reference on Resource Level



QVL_RES.bmp

Fig. 4-52: Cross reference on resource level

The cross reference list shows the names of the respective elements of the program organization unit in the "**Name" column.** The figure above shows variables (lines 1 to 7) and the instance names of the programs which run under the task manager (lines 8 to 12).

The triangle next to "Name" indicates, that the elements of this column have been sorted alphabetically with the buttons 1- A->Z and 2- Z->A in the bottom line of the active window. A sort by the elements of the adjacent columns is also possible. To achieve this, the cursor must be put on the desired column.

The **"Type" column** shows the data type of the variables (lines 1 to 7) and, for the names of the programs, the type names of the program instances (lines 8 to 12).

The **"AT" column** shows the absolute addresses for the variables, if provided. The program instances show the name of the task.

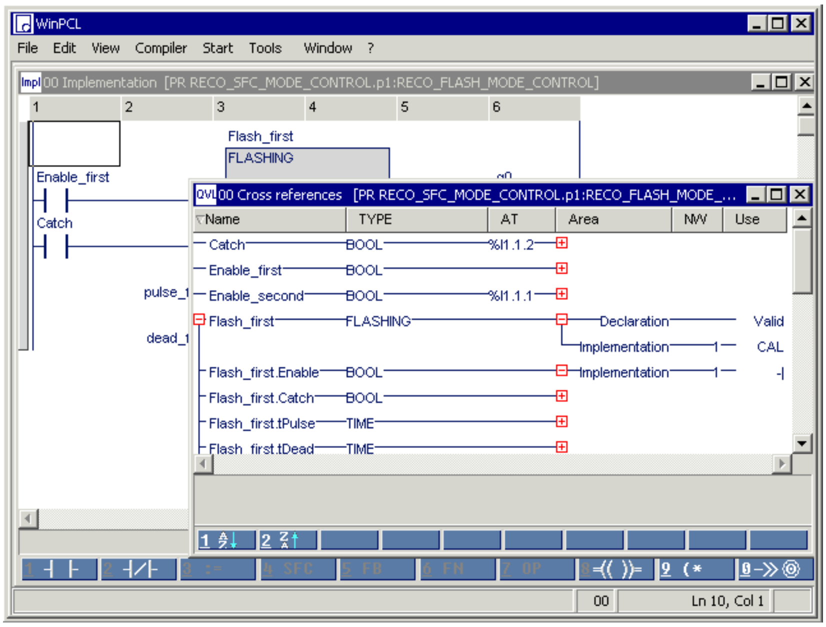
Declaration is displayed in the **"Area" column.**

The **"Use" column** indicates, that the element is valid.

Note: An information on global release is missing at the moment.

Branching to the respective place of use is possible by pressing the <ENTER> key or by double-clicking the mouse.

Cross Reference on PR / FB / FN level



QVL_FB.bmp

Fig. 4-53: Cross reference list of a function block with functional sequence

The cross reference list shows the names of the respective elements of the program organization unit in the **"Name" column.**

The **"Type" column** shows the data types of the variables and the type names of the function block instances.

The **"AT" column** shows the absolute addresses for the variables, if provided.

More information can be found in the cross reference list on program level, function block level and function level:

Area	NW	Comment	Explanation
Declaration		Valid	Valid variable declaration
Steps		Valid	Valid entry in the step table
Transitions		Valid	Valid entry in the transition list
Actions		Valid	Valid entry in the action list
SFC sfc_name		Step	The element is a step in the sfc_name sequence
SFC sfc_name		Boolean transition / NAND Boolean transition	The element is a Boolean / NAND Boolean transition in the sfc_name sequence.
STEP step_name	1	Boolean action / NAND Boolean action	The element is a Boolean / NAND Boolean action.
TRANSITION trans_name	1	LD, LDN, ST, -[ ]-, -[/]-,-[P]-, -[N]-, -( )-	The element is a variable, which is used true or not true in the trans_name transition in network 1 (IL / LD).
ACTION action_name	x	LD, LD>, LDN, LDN>, ST, STN...... -[ ]-,-[/]-,-[P]-,-[N]-,-( )-,-(/)-...	The element is a variable, which is used true or not true in the action_name action in network x (IL / LD).
Implementation	y	LD, LDN,ST, STN...... -[ ]-,-[/]-,-[P]-,-[N]-,-( )-,-(/)-...	The element is a variable, which is used true or not true in the implementation in network y (IL / LD).

Fig. 4-54: Information from the cross reference list

Note: At present, information on external use is not provided.

Rexroth Bosch Group

The triangle next to "Name" indicates, that the elements of this column have been sorted alphabetically with the buttons 1- A->Z and 2- Z->A in the bottom line of the active window. A sort by the elements of the adjacent columns is also possible. To achieve this, the cursor must be put on the desired column.

Branching to the respective place of use is possible by pressing the <ENTER> key or by double-clicking the mouse.

Cross Reference Help

The cross reference help has the same contents as the cross reference list but with the limitation that it refers to only one element or to the element and subelements, which are currently used. The cross reference help is activated by opening the "Cross reference help" pop-up menu by pressing the right mouse button or the <Shift>+<F10> keys.

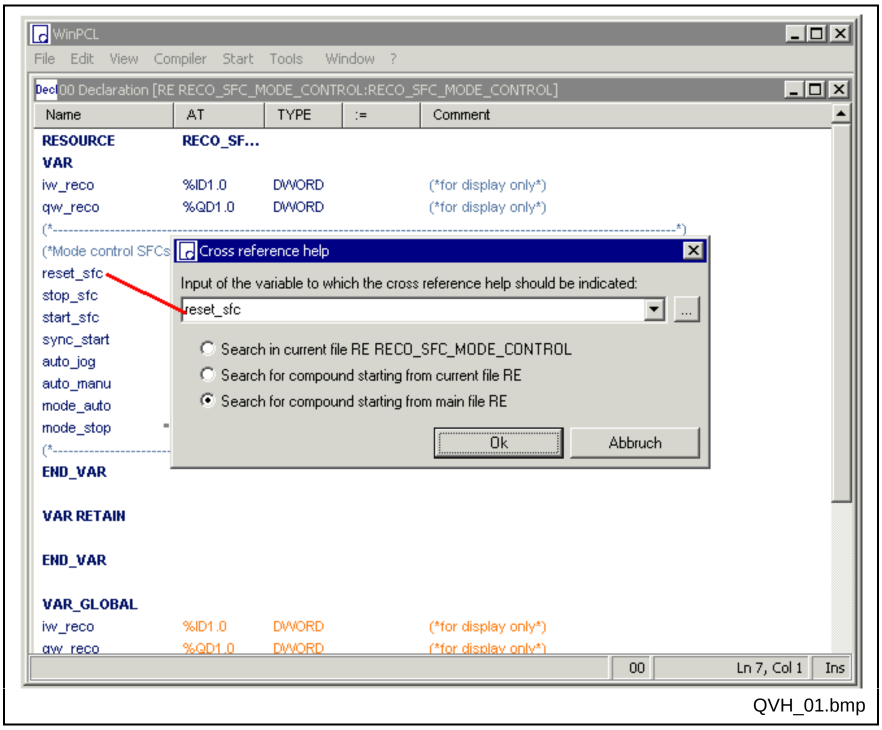


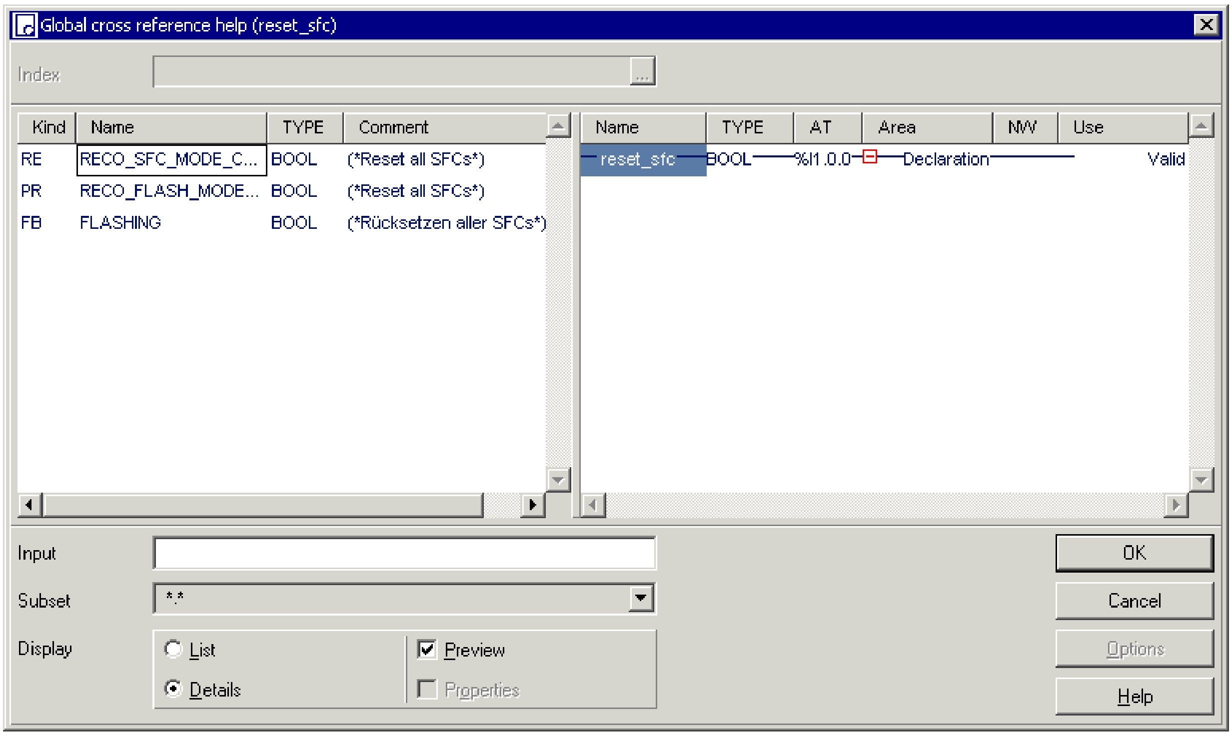
Fig. 4-55: Search for the variable "I_Start"

It is possible to select one of the following search areas:

- Search in current file
- Search in the current file and the files it is using
- Search starting from the current resource

Note: All variables of the ***same name*** are found.

For that reason, the variables may be of a different type in the various files, having nothing in common but their name!



QVH_02.bmp

Fig. 4-56: Selection of the file where the variable "I_Start" can be tracked further.

The variable "I_Start" has been found in the following files:

- RE RE_SFC_1 and
- PR PR_SFC_00.

In the example above, the variable is agreed globally and is used in the program via VAR EXTERNAL.

The program is preselected,

- in the declaration as valid line,
- in the implementation as normally open LD contact in network 1,
- in the "Message" action in connection with an IL-LD command, network 1 as well.

With the cursor at the desired position, double-clicking or pressing <Ctrl>+<Enter> opens the desired file at the desired position.

Note: At present, the selection window displaying the cross references found can only be used for loading files, but not for selecting instances of these files. For that reason, a **status display** is **not possible** in the respective editor.

The right-hand section of the screen can be activated or deactivated using "Display / Preview".

Cross Reference Help Pop-up Menu <Shift>+<F10>

The pop-up menu contains the commands which are essential to this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	Explanation
Go to application	Opens the editor required; the cursor is on the desired position.
Declaration help	Description of the data type of the current element, where the cursor is positioned.
Cross reference help	List of all places where the current element is used. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force PLC in operating mode "STATUS"	Allows the entry of a variable name. The value of the variables is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of program code execution.
Status ARRAYs / Structures	Display of the status of array and structure elements, forcing by pressing the <Shift>+<F10> keys or the right mouse button.
Options	* Optimization of the column width. * Expanding; completely opening the tree structure * Arranging the current column alphabetically, ascending or descending
Print current window <Ctrl>+<P>	Print of the editor contents by pressing <Ctrl>+<P>.

Fig. 4-57: Cross reference help pop-up menu

Edit Strategy - Variations in Font Color of Cross References

Analogously to the other editors, the user's attention is drawn to errors or invalid lines in the cross reference list of the programming system by variations in font color.

Correct information is displayed in dark-blue lines with white or gray background.

Deviations from this color combination signalize errors, either in the declaration or the place of use of the variable.

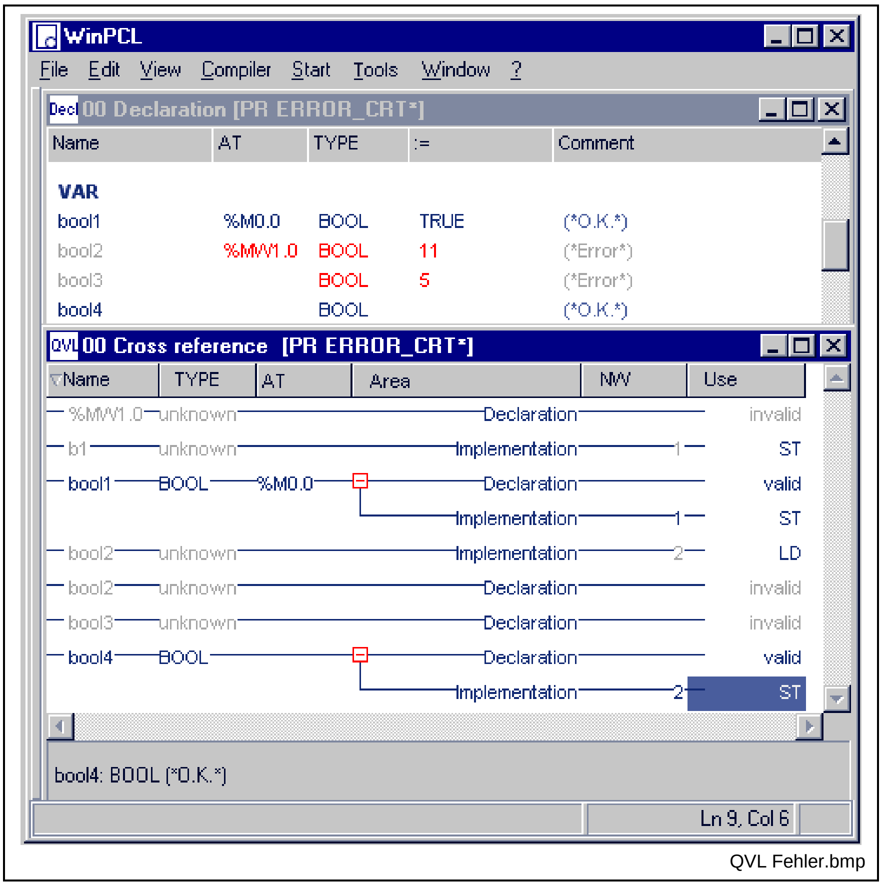


Fig. 4-58: Cross references - variations in font color of cross references

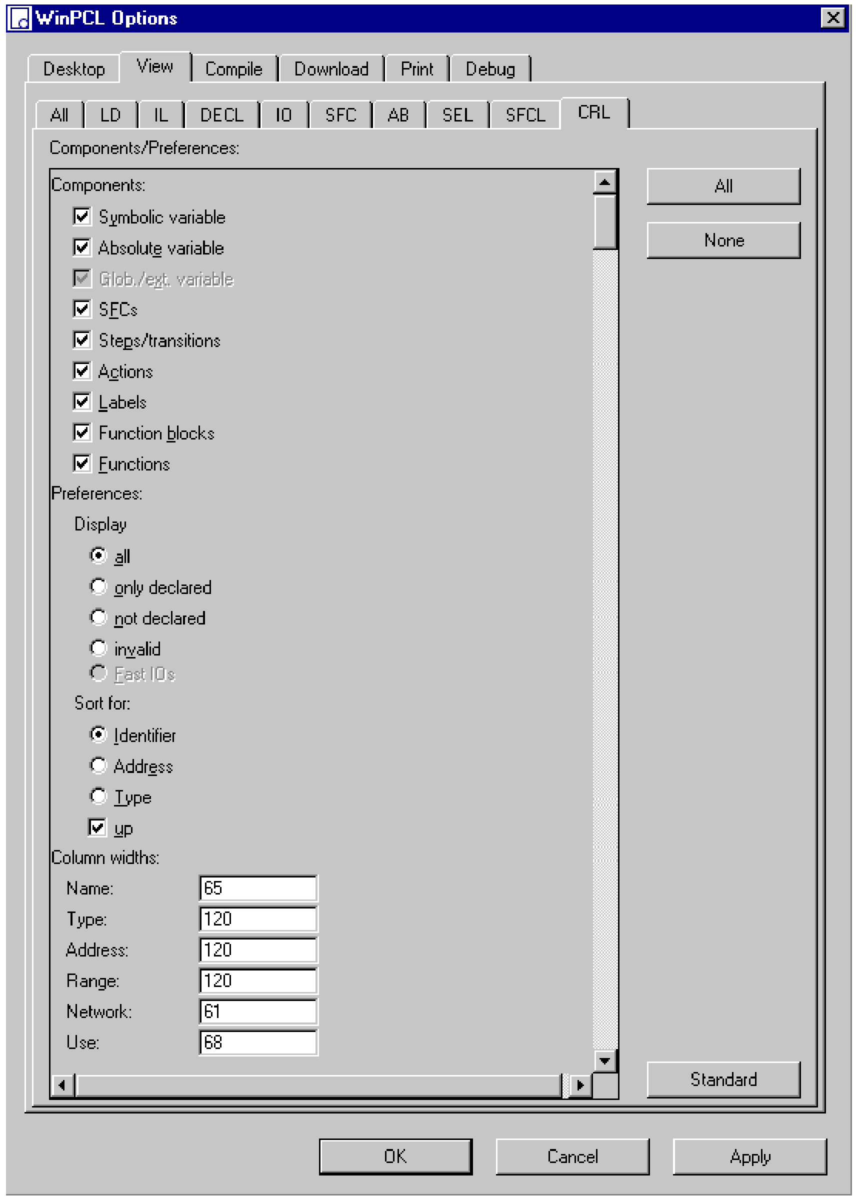
Name	Area	Comment
%MW1.0	Declaration	Invalid line, name and absolute address are displayed separately.
b1	Implementation	Variable not declared yet, unknown type.
bool1		Correct
bool2	Implementation	Used in spite of error in declaration.
bool2	Declaration	Invalid
bool4		Correct

Fig. 4-59: Display of errors in the cross reference list

Rexroth Bosch Group

Options - Cross Reference List (and Cross Reference Help)

Select "CRL" from the menu Tools / Options / View.



optionen_qvl.bmp

Fig. 4-60: Options - cross reference list

The desired components for the cross reference list can be selected as shown in the figure above.

All / only declared / not declared / invalid preferences can be displayed.

They can be sorted by ascending or descending order, by identifier / address or type.

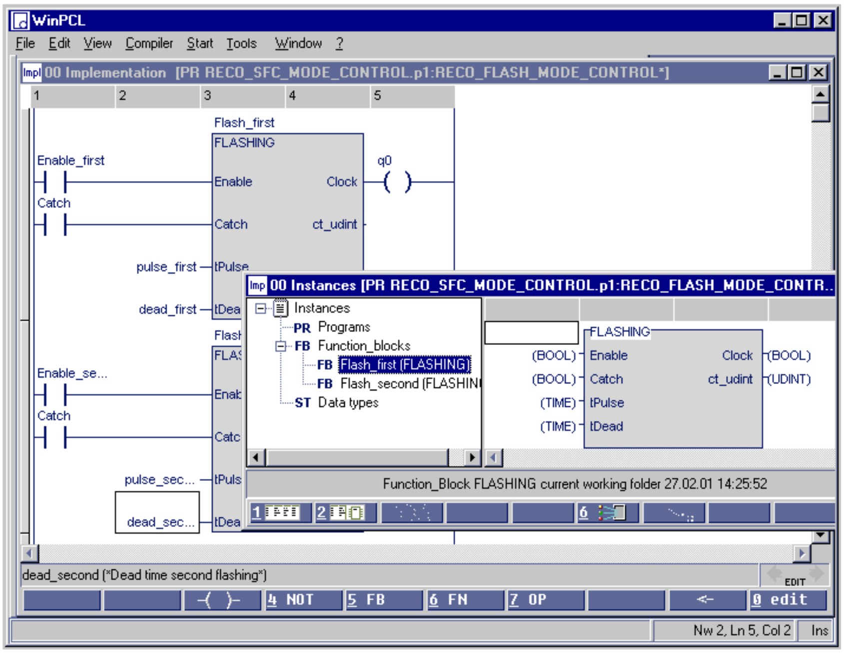
The column width can be preset.

Note: The settings also affect the printout of the cross reference list.

Import <Ctrl>+<F2>

The "Import" menu item indicates which program organization units (programs / function blocks / functions) or data types (structures / ARRAYs) have been imported by the focused program organization unit and from where the import was started (current work directory / standard library).

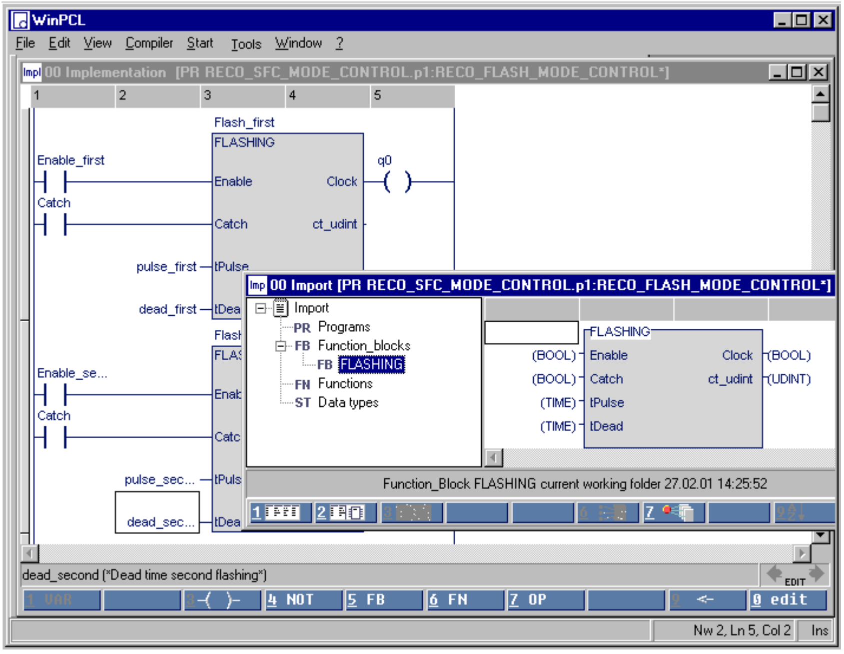
It is possible to select from different views:



import_tree_instance.bmp

Footer command: 3 - tree representation with preview
7 - Instance view

Fig. 4-61: Import window, tree representation, instance view



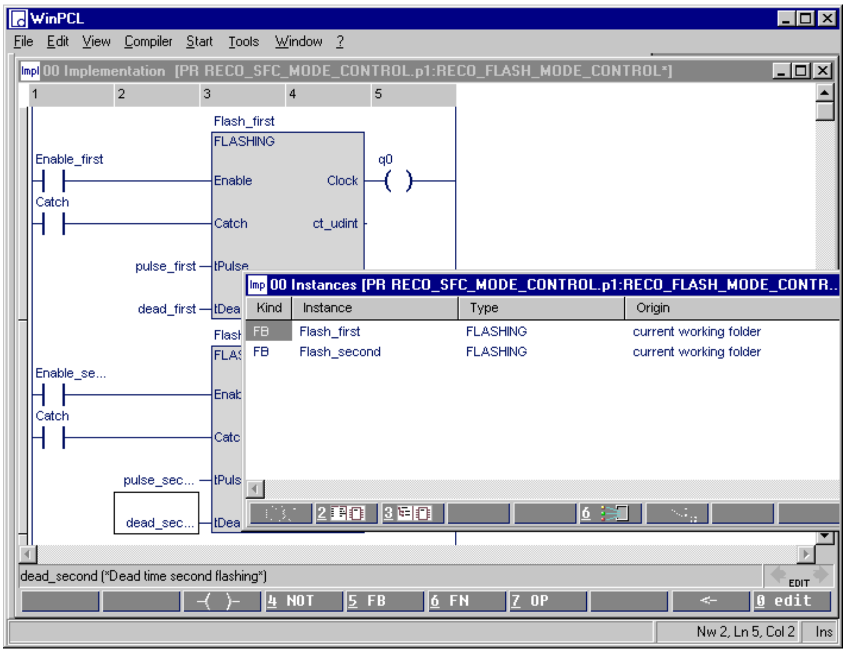
import_tree_import.bmp

Footer command: 3 - tree representation with preview,
6 - Import view

Fig. 4-62: Import window, tree representation, import view

In the instance view, all instances of a type are displayed separately, while in the import view each type is displayed only once. The buttons 6 and 7 in the footer permit toggling the views.

Furthermore a list representation with or without preview can be selected instead of the tree representation.

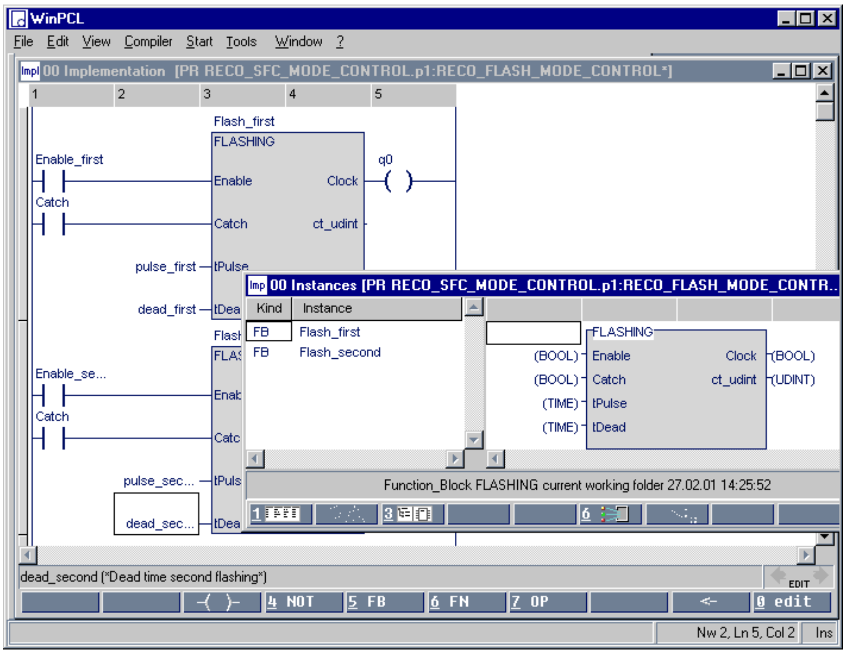


import_list_instance2.bmp

Footer command: 1 - List without preview,
7 - Instance view

Fig. 4-63: Import window, list without preview, instance view

or



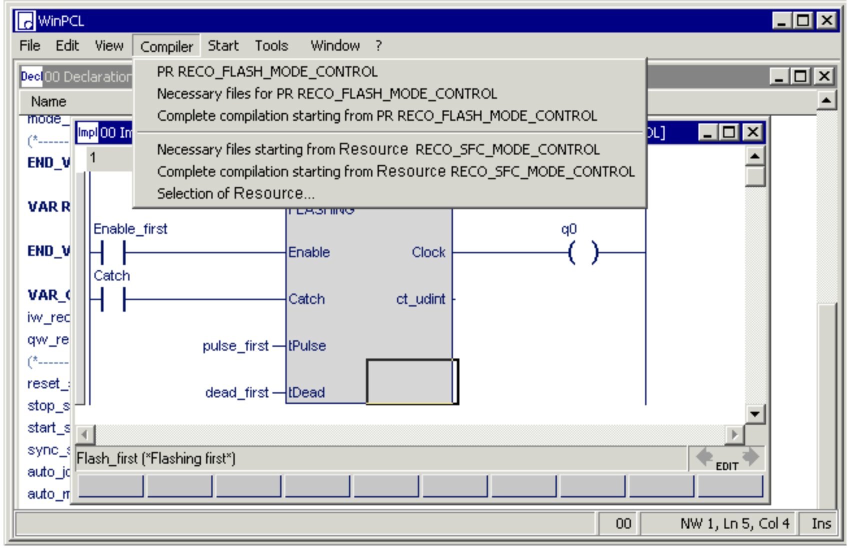
import_list_instance1.bmp

Footer command: 2 -List with preview,
7 - Instance view

Fig. 4-64: Import window, list with preview, instance view

Both windows are also available as import views.

4.5 Compiler



menue_compiler.bmp

Fig. 4-65: "Compiler" menu item

The currently possible compilation methods are combined under the "Compiler" menu item. If files included in the compound of the current resource are concerned, each of these compilation methods activates the "Edit" mode. The subsequent download <Ctrl>+<F9> causes the resource to restart. For further information on online editing, please refer to >>Online editing in the ladder diagram<< and >>Online editing in the instruction list<<.

Compilations Based on the Focused File:

Focused file "xx"

File alone, together with earlier compilation products of the files used by this file.

Necessary files for "focused file xx"

All files, which are necessary for the focused file; only those files are compiled which have been modified since the last compilation.

Complete compilation starting from "focused file xx"

All files, which are necessary for the focused file.

Note: These files do not have to reside in the control or belong to the compound of the current resource!

Compilations Based on the current resource:

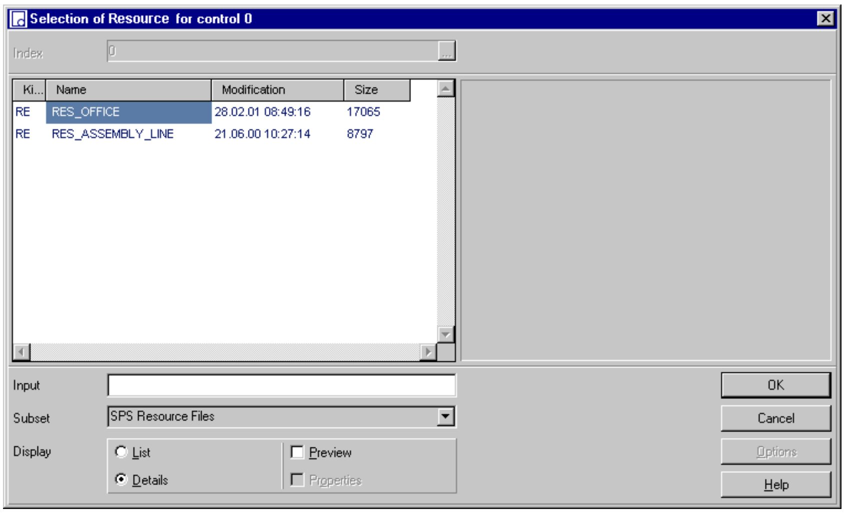
Necessary files starting from the current resource "xx"

All files, which are necessary for the current resource; only those files are compiled which have been modified since the last compilation.

Complete compilation starting from current resource "xx"

All files, which are required for the current resource.

Selection of the Current Resource



hauptdatei.bmp

Fig. 4-66: Dialog window for selecting the current resource

One of the PLC resources of the current variant in the current control can be selected as current resource.

Note: The current resource is automatically deleted, if you change the variant and/or the control. It has to be entered again for the new environment.

4.6 Start

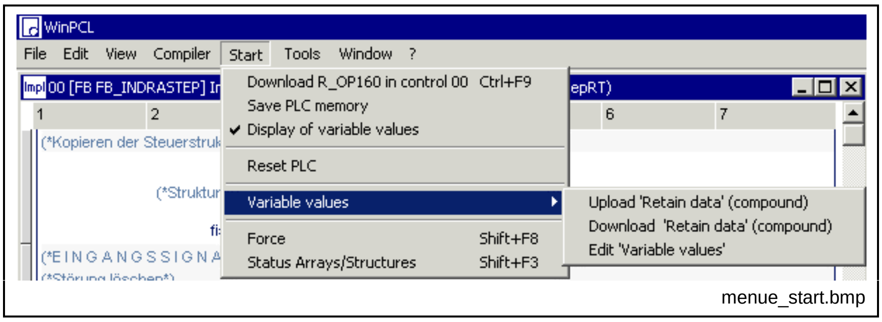


Fig. 4-67: "Start" menu item

The "Start" menu item allows to

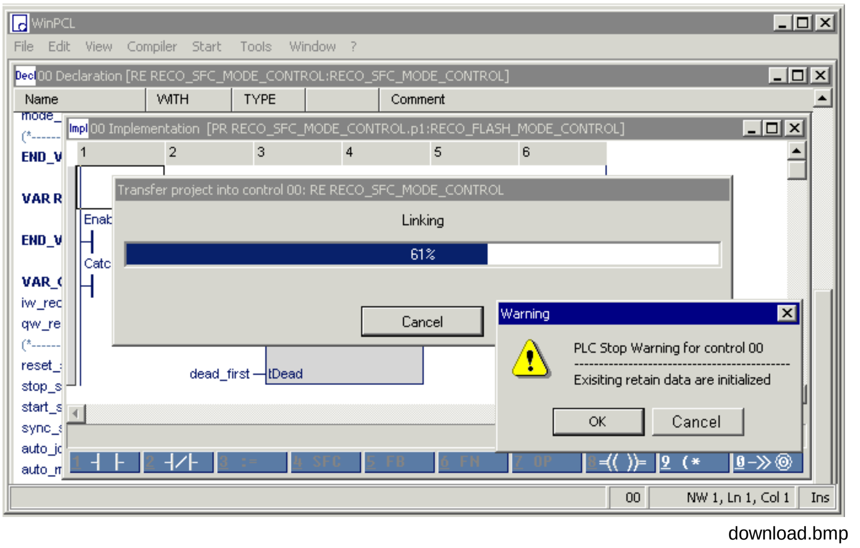
- load the programs of a resource into the control,
- copy program data from the (volatile) PLC storage to a non volatile storage of the control (only for PPC and Soft Controls)
- activate and deactivate the variable values and display the active elements in the sequential function chart. The "Display of variable values" button is in effect during "STATUS" and "ONLINE". This button is without any effect in the "EDIT" state.
- reset the control under certain circumstances (soft reset),
- access variable values (assignment of the values to the complete variable name), i.e.
  - download retain data from the control and save them as file,
  - upload the retain data from this file to the control again
  - download retain data or all data from the control, export them as text files, alter variables and/or values and upload the altered values to the control again.
- view and affect single-element variables on the control and
- view and affect multi-element variables.

Download "xx" in Control "yy" <Ctrl>+<F9>

The "Download "xx" in control "yy" <Ctrl>+<F9>" command is of a highly complex nature.

On the basis of the set current resource "xx", this command saves all files which are used by the current resource and then compiles all necessary files (also see "Compiler / Necessary files starting from the current resource").

This compilation is followed by a load procedure into the control "yy".



download.bmp

Fig. 4-68: Download of a resource

After completion of the download procedure, first the control is initialized and the multi-task system is activated, that means the resource is executed. The complete PLC program is executed subsequently, according to the priorities of the individual tasks and the order of the entries of the programs, the execution of which is controlled by a task .

Note: After completion of the download procedure, the control and, thus, the RETAIN data of the resource, the programs and the function blocks are initialized.

After completion of the download procedure, it is ensured that all source files pertaining to the PLC program running in the control, including their secondary files, are stored in the directory... \MTGUI\project_xxx\ProgramData\device_yyy\PLC\Downloaded Files.

Save PLC Memory

PPC and soft controls only have volatile memories for the PLC programs. Therefore, it is necessary to copy the corresponding data from the volatile to a non-volatile memory before switching off the control. The user decides when this "saving" takes place (see section "Options, Download").

- Save PLC memory after download
- Save PLC memory after every online changing

When switching on the control the data are loaded back into the volatile memory.

Note:	The last saved status is loaded back into the volatile memory!

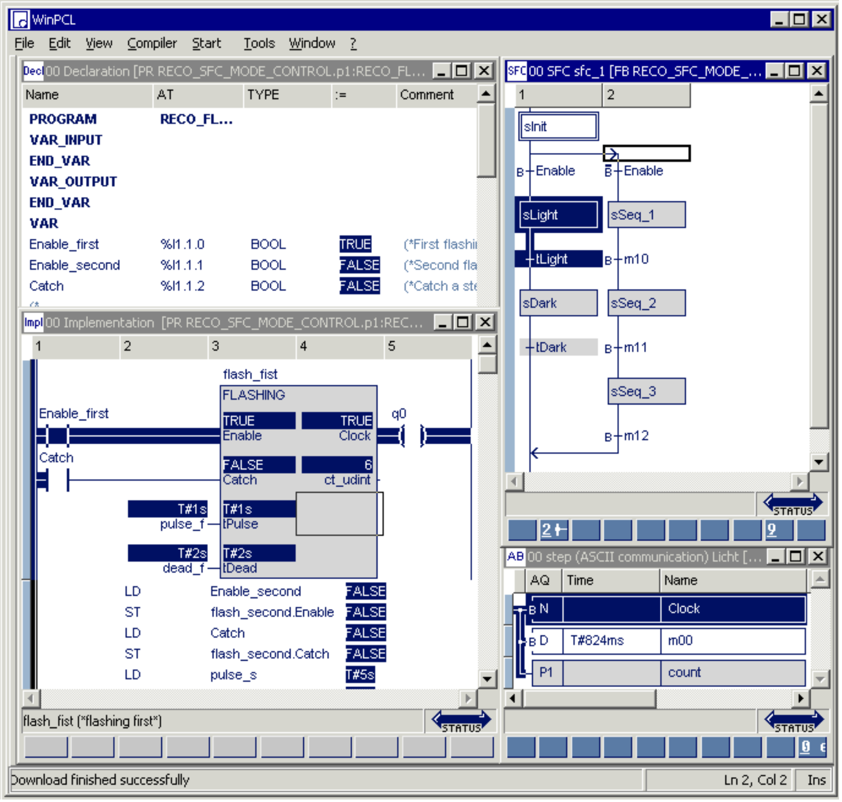
Note:	This menu item is indicated in gray for controls, that do not require this function.

Display of Variable Values

This menu item permits activation and deactivation of the variable status display in the "STATUS" and "ONLINE" states. This button is without any effect in the "EDIT" state.

The status display of a running PLC program, i.e. of the complete resource, allows viewing of the variable values (ANY_ELEMENTARY) at the end of the cycle of the resource. Each cycle was executed at least once.

Note:	The current status display is not yet able to differentiate between edited and skipped sections in LD / IL!



status.bmp

Fig. 4-69: Status displays in different editors

The figure above shows an overview of the status displays in different editors:

- Declaration editor
  - Value of single-element variables, "=" column, white on blue
  - Value of ARRAYs and structures, see status ARRAYs / Structures
- Ladder diagram editor
  - Value of non-Boolean single-element variables, white on blue
  - Value of ARRAYs and structures, see status ARRAYs / Structures
  - Reproduced signal flow according to the variable values, blue
  - Blue left margin in front of bus-bar
- Instruction list editor
  - Value of single-element variables in the column next to the operands, white and blue
  - Value of ARRAYs and structures, see status ARRAYs / Structures
  - Blue line to the left
- Sequential function chart (SFC) editor
  - Active steps and executed transitions, white on blue
  - Non-executed sequences in their initial condition, only the initial step is white on blue

- Action block editor
  - Boolean action: white on blue, variable is TRUE, otherwise blue on white, variable is FALSE
  - Other actions: white on blue, action is in process
  - Action time follows according to the definition of the action qualifier.

Note: With activated status display, executed actions show the value T#0s, not the value of the action time.

  - Blue line to the left: the step, the action blocks belong to, is executed; no blue line: action blocks are edited somewhere else, here only their status is displayed.

Reset PLC

A software reset of the control is possible with this menu item. Reset, however, depends on the adjacent control components, e.g. MTC or the like.

For that reason, it is necessary that the user informs himself on the effects and options of the hardware used.

Variable Values

This menu item permits

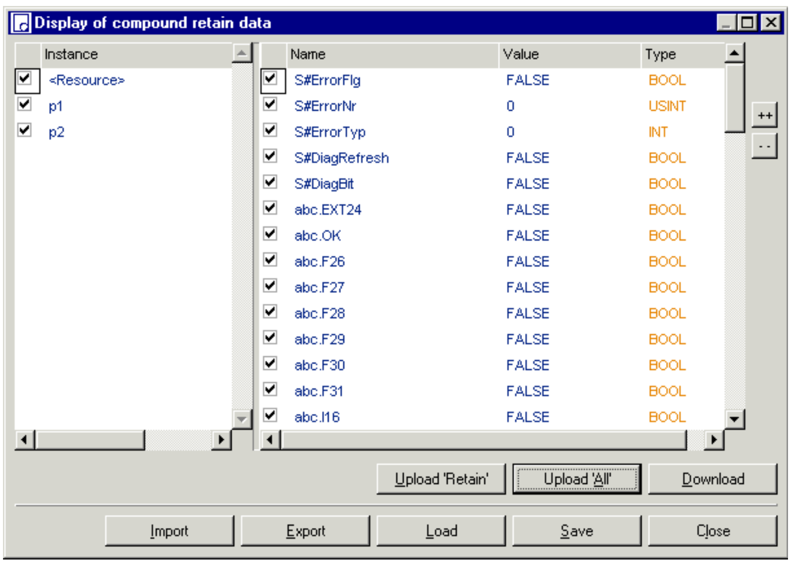
- to upload RETAIN data from the control (Upload ‘Variable values’),
- to edit the variable values, to save them in a file and to reload them,
- to download the values edited to the control again (Download ‘Variable values’).

In addition, the values are assigned to the complete variable name.

Upload Variable Values (Compound)

All RETAIN data of the compound, i.e. retain variables and data of function blocks, that have been declared under VAR_RETAIN, currently running on the control is removed from the control.

The data is automatically stored in file "ressource_name@DEF@RES.RVD". Then, the data are displayed.



Edit_Var_mit_laden_speichern.bmp

Fig. 4-70: Edit window after "Upload Retain Data (Compound)"

The buttons "**Save**" and "**Load**" in the above window allow to archive and re-call the RVD file into the window.

The buttons "***Export***" and "***Import***" enable to archive or load the file as text file under an arbitrary name.

The button "***Download***" activates the transfer of the displayed variables to the control. Variables of the same name are overwritten with the values (Also compare with "Edit Variable Values").

A download of single variables or instances can be prevented, if the 'check mark' before the respective element is removed. This property is not saved. The buttons "Select all" or "Select none" refer to the elements displayed in the right window.

The buttons "***Upload Retain***" (upload the retain variable values from the control) and "***Upload all***" (upload all variable values from the control) overwrite the window contents.

Download Variable Values (Compound)

This menu item allows to reload file "ressource_name@DEF@RES.RVD" in the control.

It is not relevant, if, at this time, the variable (instance) is still in the VAR RETAIN or already in the VAR area.

Thereby, the variable name and the corresponding instance way are taken into account.

A download of single variables or instances can be prevented, if the 'check mark' before the respective element is removed. This property is not saved. The buttons "Select all" or "Select none" refer to the elements displayed in the right window.

Note: Variable values, that exceed the size predefined by the type, are not archived.

Variable values, that have no equivalent in the control, generate an error message and are listed there.

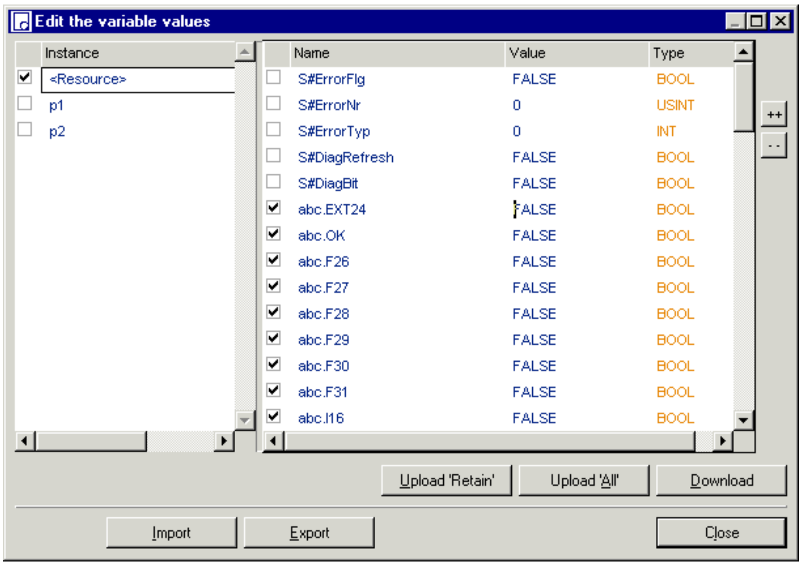
Data consistency during the transfer is not guaranteed!

Additionally, the following restrictions have to be considered:

- Variables assigned to SFC elements can not be modified (the values loaded in the control are overwritten again).
- Instances of the function blocks (selection) mentioned below are not up-dated correctly (see also section 'Limitation of the Declaration of Function Blocks in the Retain Area' in chapter 'Declaration Editors'):
  - Time stages (TON, TOFF, TP)
  - Blocks to combine PLC / CNC and PLC / SYNAX.
  - Blocks for serial communication.
  - Blocks for bus communication.

Edit Variable Values

If the menu item "Edit variable values" is opened, the edit window appears without button "***Load***" and "***Save***". All operations allowed now refer to the text files designated by the user with own names.



Edit_Var_ohne_laden_speichern.bmp

Fig. 4-71: Edit window in menu item "Edit variable values"

The button "***Upload Retain***" (upload the retain variable values from the control) and "***Upload all***" (upload all variable values from the control) serve to fill up the, at that time, empty edit window.

The buttons "***Export***" and "***Import***" allow to archive or load the file as text file under an arbitrary name. The name of this text file and its archive location can be freely selected. The file can be edited with the notepad or a comparable text editor.

The button "***Download***" activates the transfer of the displayed variables to the control. Variables of the same name are overwritten with this values (compare also with section "Edit Variable Values")

The download of single variables or instances can be prevented, if the 'check mark' before the respective element is removed. This property is not saved. The buttons "Select all" or "Select none" refer to the elements displayed in the right window.

Note: Paths and variable names have to be correctly archived. Faulty (not found) variables are displayed in gray.

While modifying variable values, please consider the limitations predefined by the variable type!

Additionally, the following restrictions have to be considered:

- Variables assigned to SFC elements can not be modified (the values loaded in the control can be overwritten again).
- Instances of functions blocks mentioned below (selection) are not correctly up-dated (see also section 'Limitation of the Declaration of Functions Blocks in the Retain Area' in chapter 'Declaration Editors'):
  - Time stages (TON, TOFF, TP)
  - Blocks to combine PLC / CNC and PLC / SYNAX.
  - Blocks for serial communication.
  - Blocks for bus communication.

Force <Shift>+<F8>

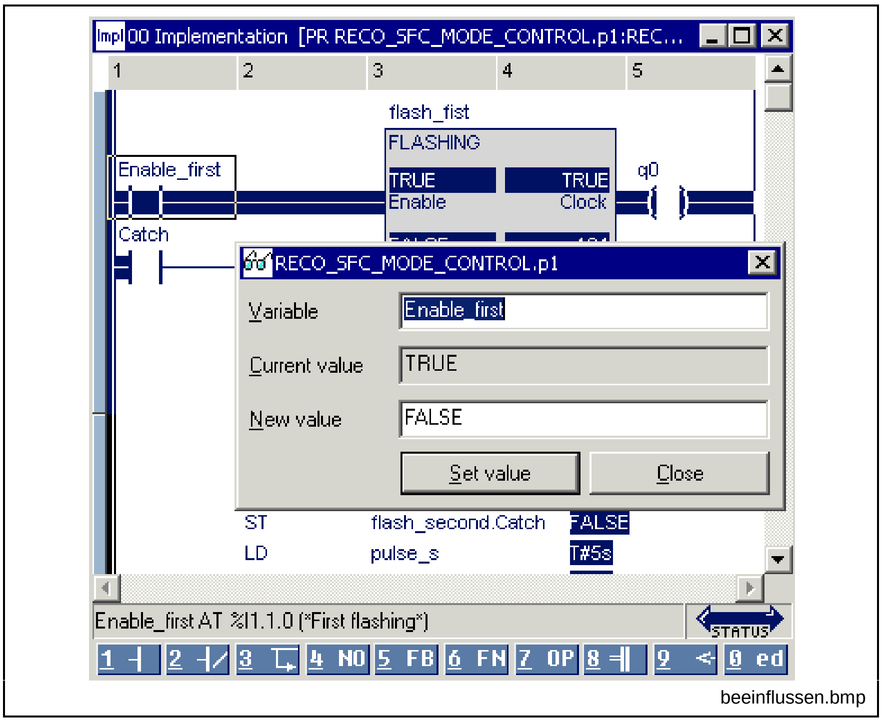


Fig. 4-72: Forcing variables, here in the ladder diagram

Using

- the menu item "Start / Force" menu item,
- the <Shift>+<F8> keys,
- the right mouse button or
- the key combination <Shift>+<F10> in the opened pop-up menu,

it is possible to display or change the value of elementary variables (ANYELEMENTARY).

If the cursor is positioned on a useful variable in the focused editor (see above), the name of this variable is applied and the value of the variable is displayed.

It is possible to define several windows for different variables.

Note: The desired variable has to be valid - here declared or agreed as external variable - in the focused program organization unit.

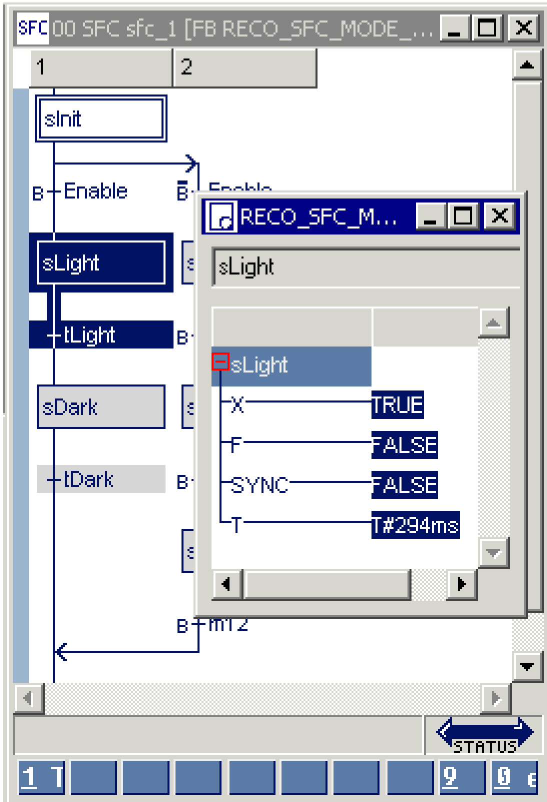
To ensure that input variables (%I...) can also be forced, a value change is inserted **once** after update of the image memory (inputs area).

After having been forced **once,** the variable itself is subject to the generally applicable processing guidelines.

ARRAYs and structures can be viewed and changed using "Status ARRAYs / Structures<Shift>+<F3>".

A deactivation of the status closes all view windows.

Status ARRAYs / Structures<Shift>+<F3>



status_arr_01.bmp

Fig. 4-73: Status of ARRAYs / structures, here system variables of a step

The values of structures and ARRAYs can be displayed and changed by means of the tool "Status ARRAYs / Structures <Shift>+<F3>".

This tool is loaded via

- the "Start / Status ARRAYs / Structures" menu item,
- the <Shift>+<F3> keys,
- the right mouse button, or
- the <Shift>+<F10> keys in the opened pop-up menu.

If the cursor is positioned on a useful variable ("step light" in the example above), the name is applied to the selection window and the respective structure is displayed. Otherwise the name can be entered manually.

The elements are shown in a tree structure, so that each element is accessible, even for nested structures or ARRAYs.

The value of the elements can be changed. To achieve this, the cursor must be positioned on the name of the element to be changed, the pop-up menu opened by pressing the right mouse button or the <Shift>+<F10> keys and the value changed in the "Force" window.

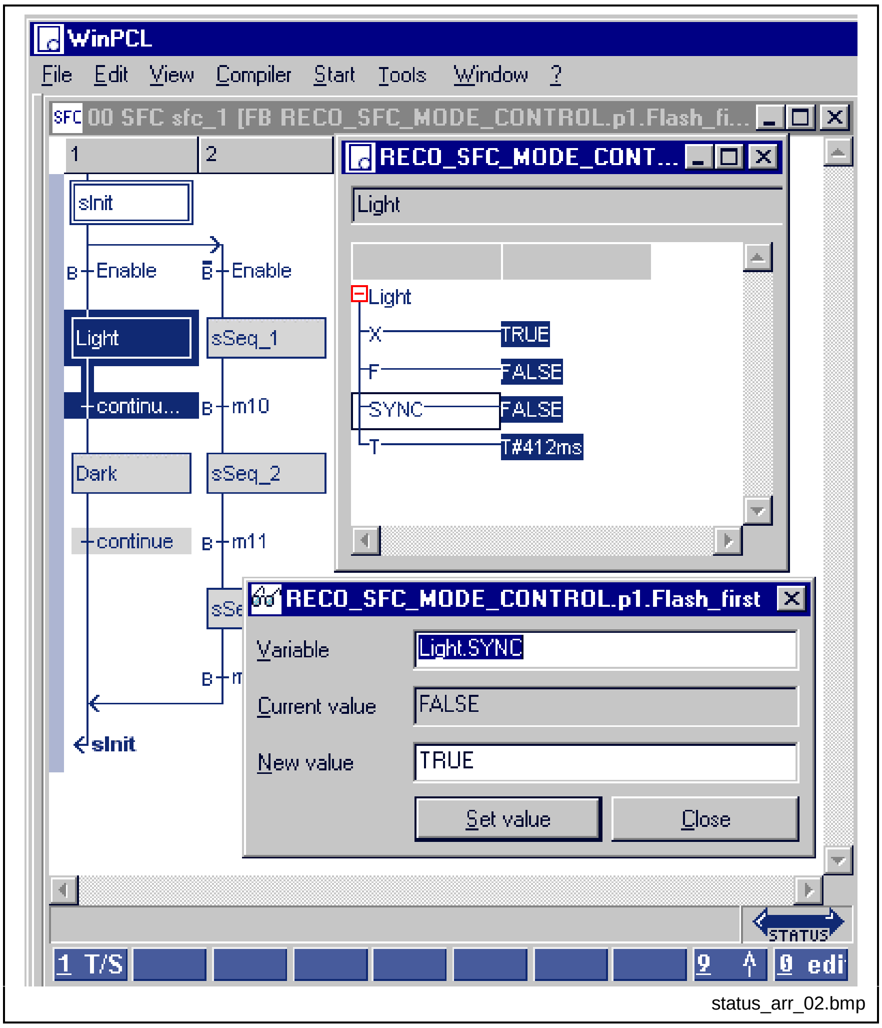
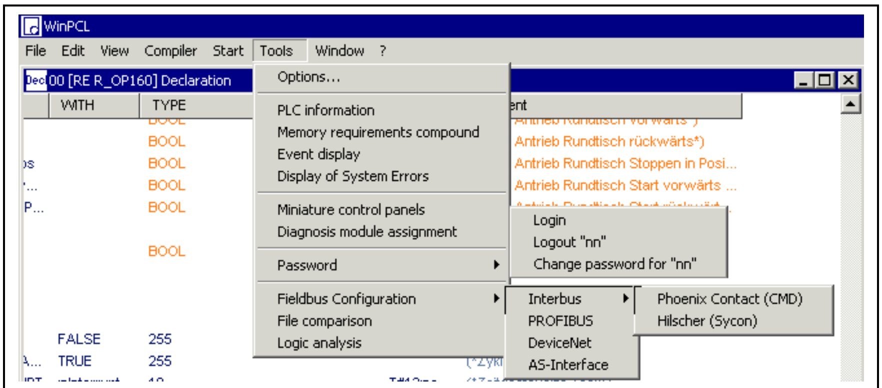


Fig. 4-74: Forcing structured data type elements

4.7 Tools



menue_Tools.bmp

Fig. 4-75: "Tools" menu item

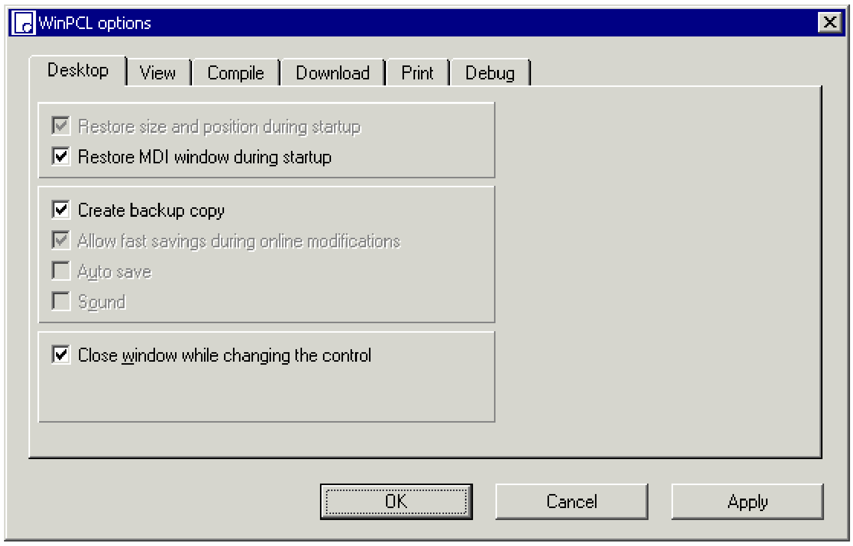
The "Tools" menu item opens the following subitems:

- Options: settings for editors, print, cross reference list, etc.
- PLC Information: data for the resource which is running in the control, display of the PLC firmware version, the components which are available in the current control, etc.
- Memory Requirements for Compound: display of the memory requirements before downloading
- Event Display: protocol of the data exchange between interface and control
- Display of System Errors: display of control states in the configuration
- Miniature Control Panels: preparation of miniature control panels for use
- Diagnosis, Module Assignment: ProVi, diagnosis in SFCs, module assignment in case of multiple instancing
- Password: login, logout of "current user", changing the password of the "current user"
- Fieldbus Configuration: preparation and start of the configurators for the respective bus system
- Logic Analysis: start of the general tool "logic analysis"
- File-File Comparison: comparison of any WinPCL files

Options

This menu item allows access to the desktop presettings, the view of all editors, the setting of the editors, the presetting for cross references, print, and the like.

WinPCL Options, Desktop



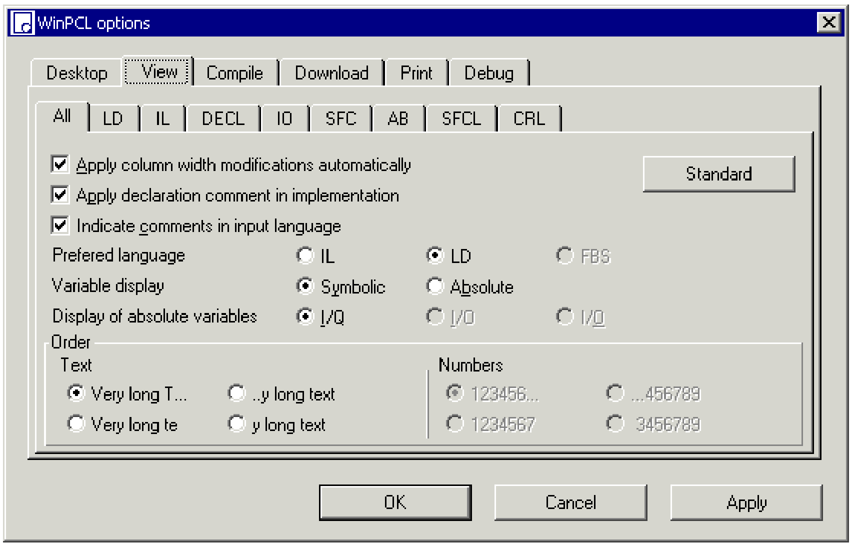
optionen_desktop.bmp

Fig. 4-76: WinPCL options, desktop

WinPCL options	Meaning
Restore size and position during startup	This setting refers to the size and position of the desktop on the screen.
Restore MDI window during startup	Windows of the last session are opened at the previous position with same size.
Other options not enabled yet	

Fig. 4-77: Explanations on WinPCL options, desktop

WinPCL Options, View of All Editors



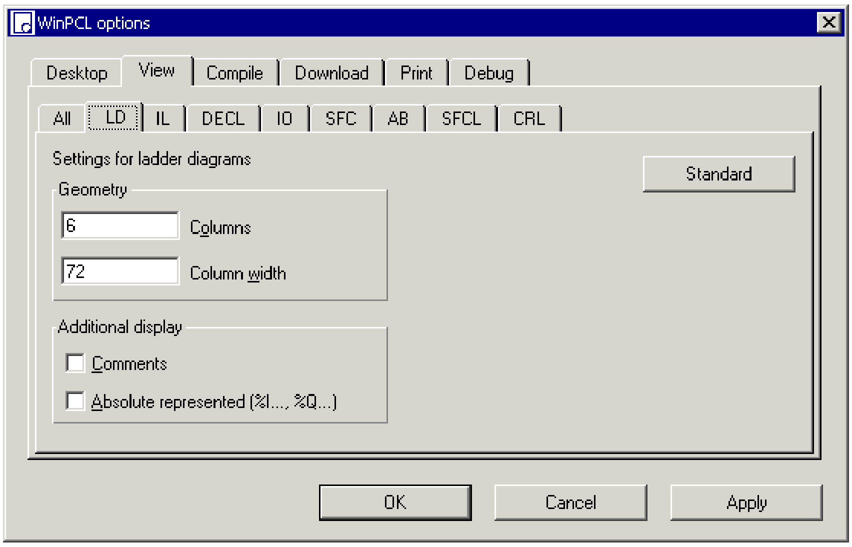
optionen_ansicht_alle.bmp

Fig. 4-78: WinPCL options, all editors

WinPCL options	Meaning
Apply column width modifications automatically	Changed column widths become automatically effective for the next window of the same editor.
Apply declaration comment in implementation	The declaration comment of variables is displayed as default comment in the implementation; it can be edited and then becomes an implementation comment. If this implementation comment is deleted, the declaration comment is restored.
Variable display symbolic / absolute	The IO address of the variable can be displayed in the stead of its name.
Indicate comments in input language	Comments are displayed in the original language. Alternatively, imported comments can be displayed.
Display of absolute variables	I/Q is released.
Truncating very long texts	Selection where to truncate, to the right or left, with or without "...".
Truncating very long numbers	Selection where to truncate, to the right or left, with or without "...".

Fig. 4-79: Explanations on WinPCL options, all editors

WinPCL Options, Ladder Diagram (LD)



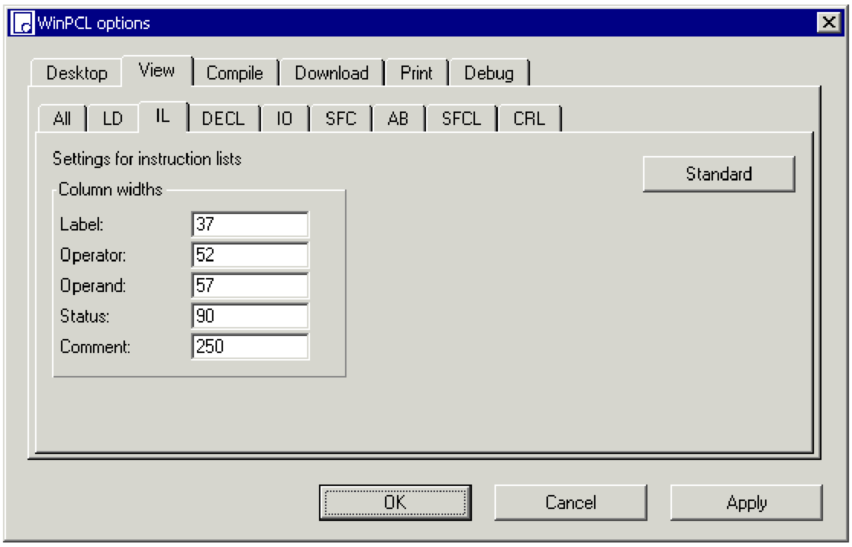
Optionen_ansicht_kop.bmp

Fig. 4-80: WinPCL options, ladder diagram

WinPCL options	Meaning
Settings for ladder diagrams:	
Geometry	This option can be used to specify the number and width of the columns.
Additional display:	
Comments	The comment on the variables is displayed above the ladder.
Absolute represented	The absolute address of the variable is displayed above the ladder.

Fig. 4-81: Explanations on WinPCL options, ladder diagram, tools

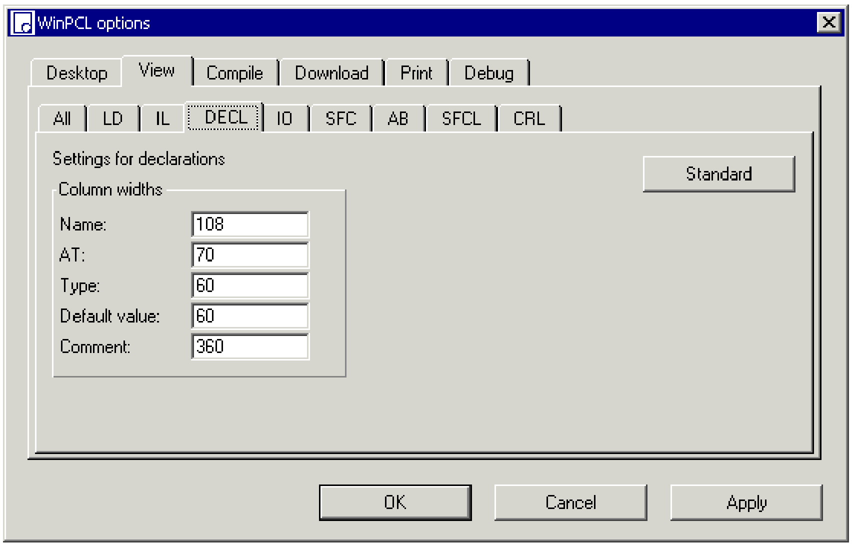
WinPCL Options, Instruction List (IL)



optionen_ansicht_awl.bmp

Fig. 4-82: WinPCL options, instruction list

WinPCL Options, Declaration Editor (DECL)



optionen_ansicht_dekl.bmp

Fig. 4-83: WinPCL options, declaration editor

WinPCL Options, IO Editor (IO)

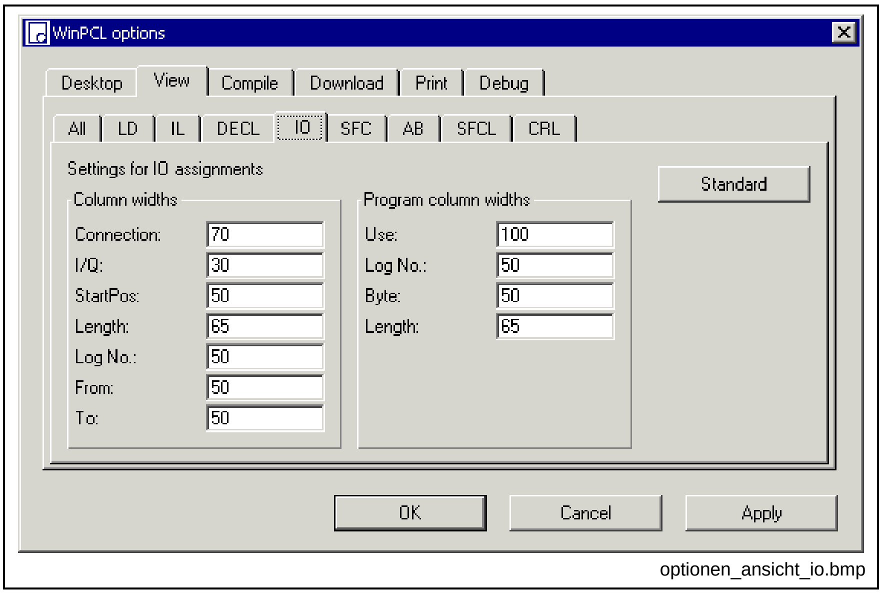


Fig. 4-84: WinPCL options, IO editor

WinPCL Options, Sequential Function Chart (SFC)

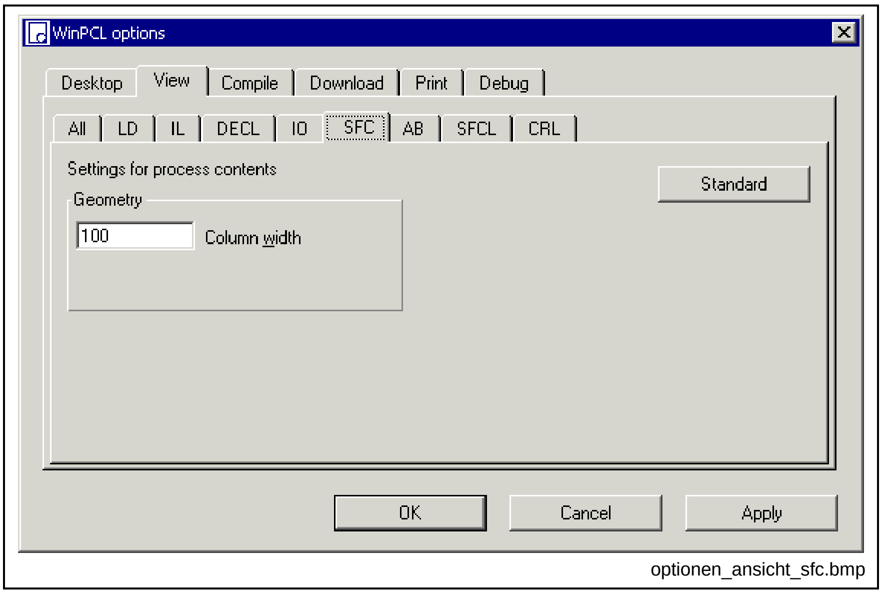
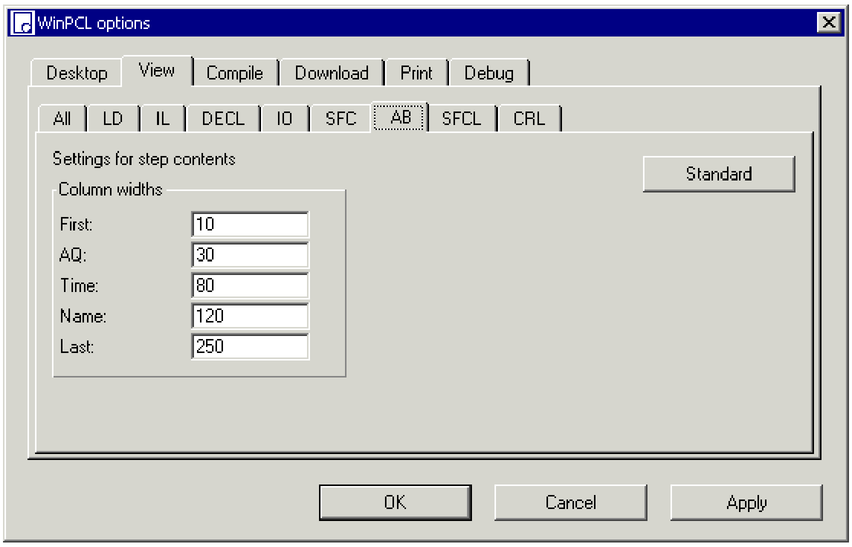


Fig. 4-85: WinPCL options, sequential function chart

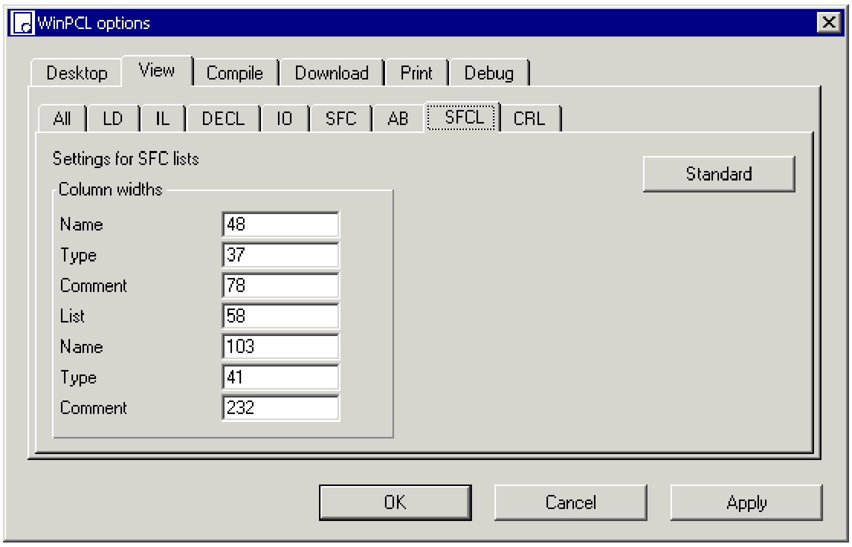
WinPCL Options, Action Block Editor (AB)



optionen_ansicht_ab.bmp

Fig. 4-86: WinPCL options, action block editor

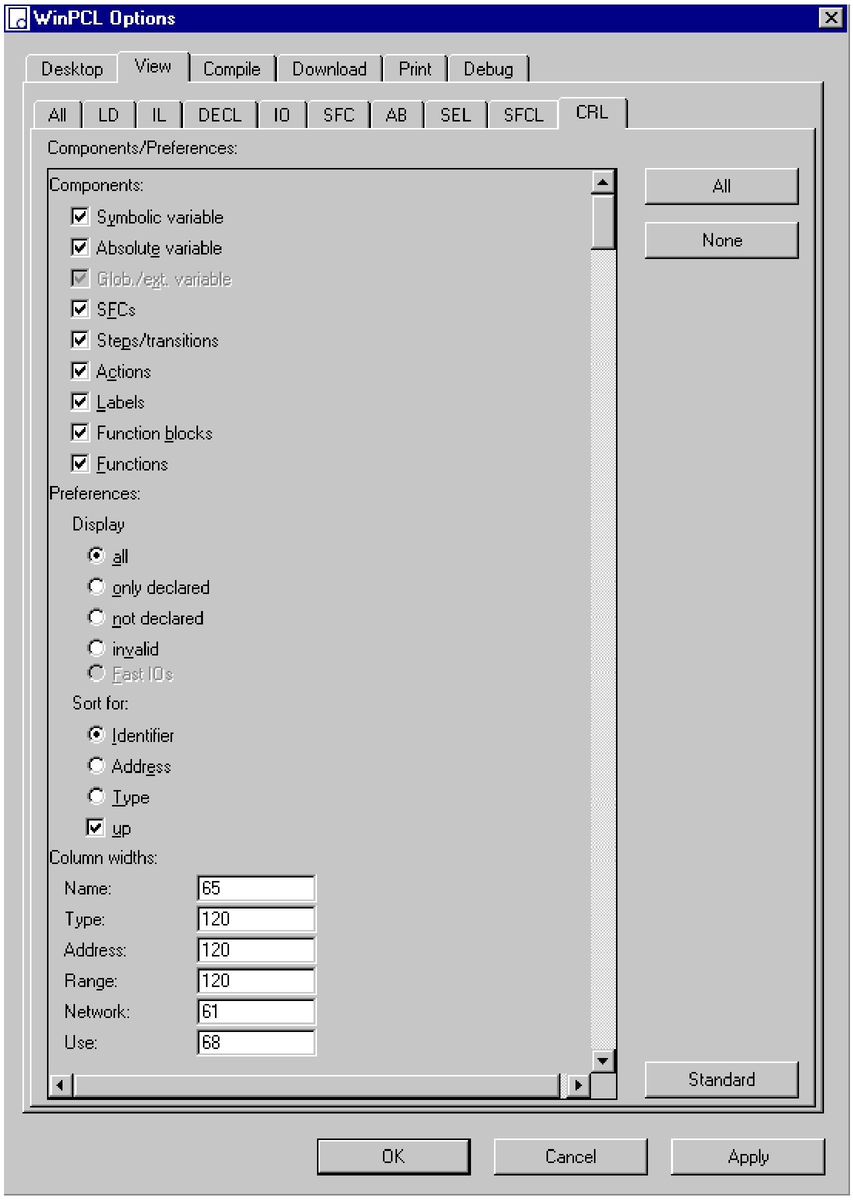
WinPCL Options, SFC List (SFCL)



optionen_ansicht_sfcl.bmp

Fig. 4-87: WinPCL options, SFC list

WinPCL Options, Cross Reference List (CRL)



optionen_qvl.bmp

Fig. 4-88: WinPCL options, Cross reference list

The desired components for the cross reference list can be selected as shown in the figure above.

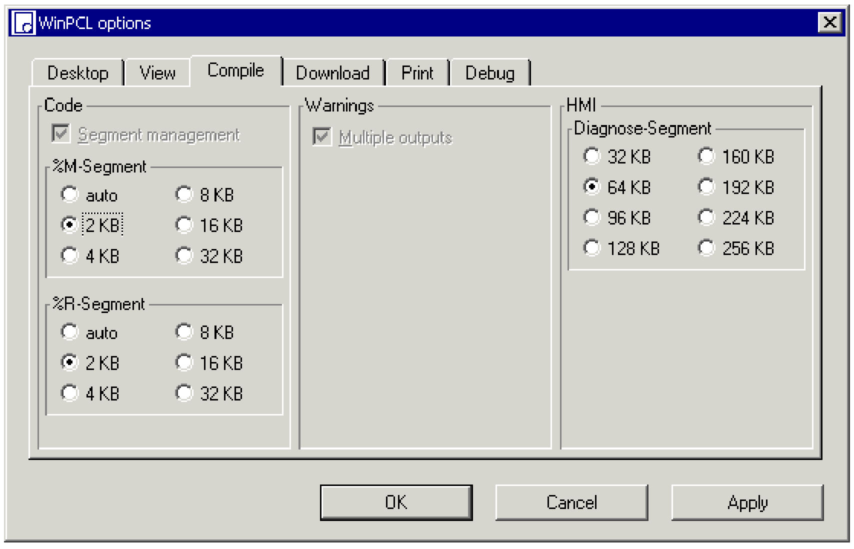
All / only declared / not declared / invalid cross references can be displayed.

They can be sorted by ascending or descending order, by identifier / address or type.

Further the column width can be preset.

Note: The settings also affect the printout of the cross reference list.

WinPCL Options, Compile



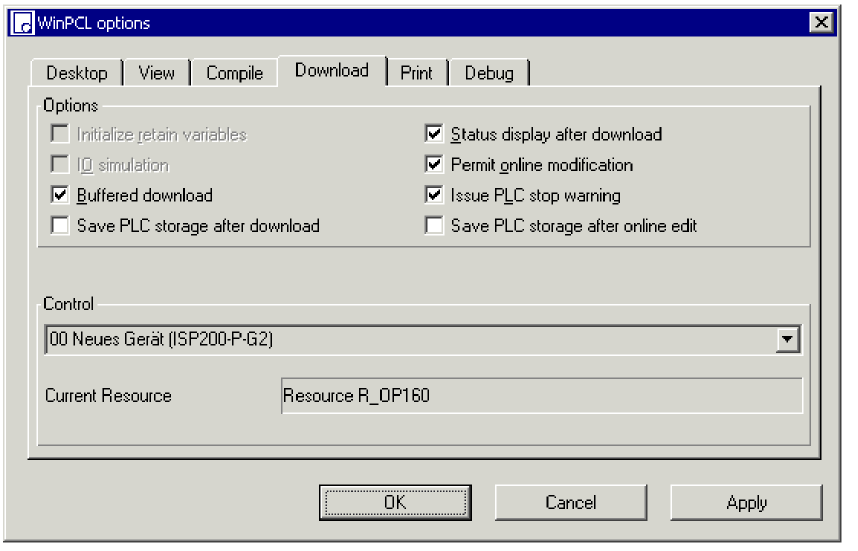
optionen_compile.bmp

Fig. 4-89: WinPCL options, Compile

WinPCL options	**Meaning**
Code:	
Segment management	
%M Segment	* auto: Size of %M memory in the control as required * otherwise fixed setting of the reserved memory
%R Segment	* auto: Size of %R memory in the control as required * otherwise fixed setting of the reserved memory
Diagnose Segment:	The reserved memory in the control for diagnosis purposes is set here.

Fig. 4-90: WinPCL options, compile

WinPCL Options, Download



optionen_download.bmp

Fig. 4-91: WinPCL options, download

WinPCL options	**Meaning**
Options	
Buffered download	The download process is accelerated if this checkbox is activated. Deactivate this function only if you are instructed to do so!
Status display after download	The status display is activated immediately after download if this setting is active.
Permit online modification	Permits that, in some of the modifications, the control still executes the program with preservation of the variable values, although the code has been changed.
Issue PLC stop warning	Major modifications may require the Edit mode. If such modifications are activated by being downloaded, variables and SFCs are re-initialized. The warning reminds the user that the system must be moved to a reasonable operating state.
Save PLC Storage after Download	Automatic backup function for controls with non volatile storage (PPC or Soft Control only)
Save PLC Storage after Online-Edit	Automatic backup function for controls with non volatile storage (PPC or Soft Control only)
Control:	Window to select the control the set current resource is to be displayed for.
Current resource:	This field displays the current resource.

Fig. 4-92: WinPCL options, download

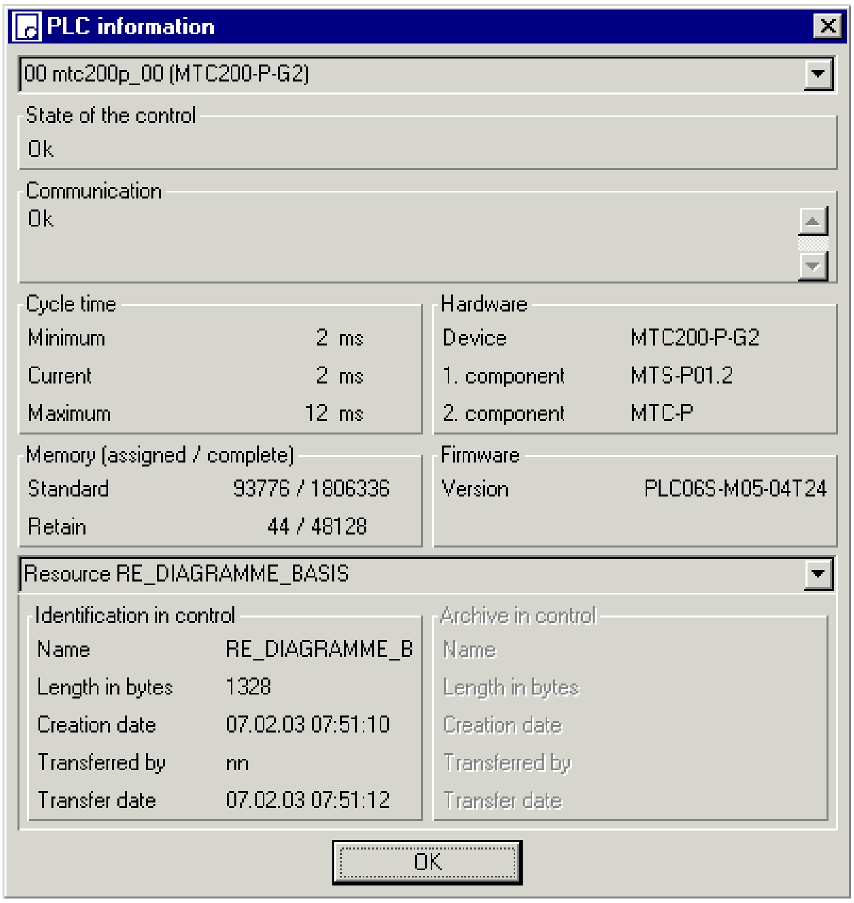
WinPCL Options, Print

The options to change the print settings are described in section "Print Options".

WinPCL Options, Debug

The options settable under Debug may only be used according to the Rexroth service personnel's instructions.

PLC Information



spsinfo.bmp

Fig. 4-93: "Tools / PLC information" menu item

This menu item can be used to fetch information on the controls pertaining to the control compound and to inquire the current data of the resource that runs on the control and its programs.

Upper part of the PLC information window

One of the controls, which was entered in the control compound by means of the system configurator, can be selected in the first line.

The following is indicated for this control:

- its current state
- information on the transmission path between the programming interface and the control
- cycle time of the resource
  - minimum cycle time: minimum occurred cycle time of the resource since program start
  - current cycle time
  - maximum cycle time: maximum time consumed by the resource; usually occurring once during the first run due to initialization processes.
- memory: available memory and unassigned (blank) memory

- hardware
  - device type
  - PLC component
  - other components e.g. CNC
- firmware version: Display of the PLC running on the firmware

Lower part of the PLC information window

The line in the middle of the window can be used to select and display the resource for the current control or one of its programs.

The following information can be loaded for the selected program / resource:

- identification in control
  - name of program / resource
  - length in bytes
  - creation date and creation time
  - transferred by: name of login
  - transfer date and creation time (to control)
- archive in the control (inactive for the moment)
  - name of archive
  - length in bytes
  - creation date and creation time
  - transferred by: name of login
  - transfer date and creation time (to control)

Memory Requirements for Compound

This window shows the memory requirements of the resource and the files it controls.

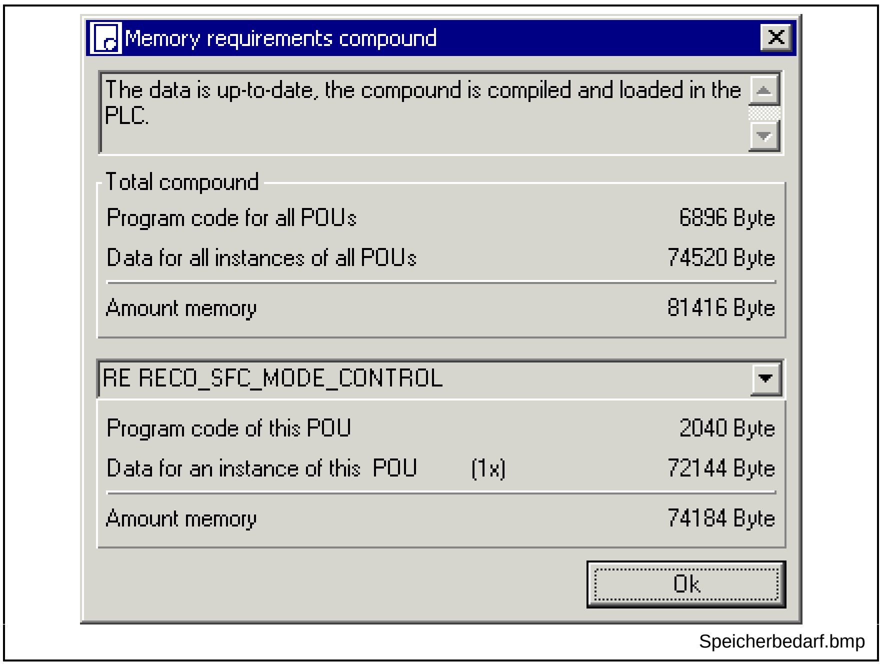


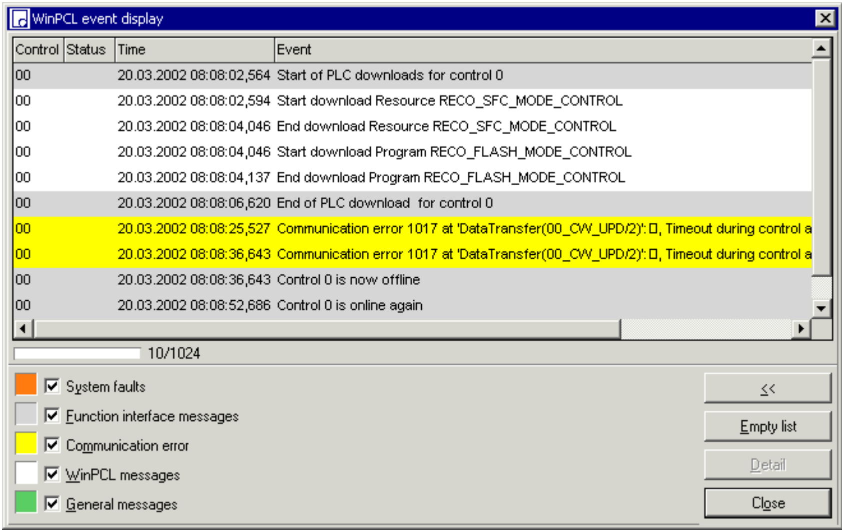
Fig. 4-94: Compound memory requirements

The upper section shows the relation of the data of the compound to the data in the PLC.

This is follows by data of the total compound, separately for data and program code memory requirements.

The lower section permits selection of the individual POUs. The code and data memory requirements are shown for individual instances.

Event Display



ereignisanzeige.bmp

Fig. 4-95: Menu item "Tools / Event display" with pop-up menu

The event display is a protocol of the message exchange between control and operating interface.

Date and time and the event are entered into a list.

A maximum of 1024 entries is allowed, the figure shows 8 of 1024 entries. If the maximum number is exceeded, the oldest entries are deleted automatically. The list can be cleared by pressing the "Empty list" button.

The content of the entries can be restricted to the items listed to the left on the screen (see figure above).

- System faults (background: light-orange)
- Function interface messages (background: gray)
- Communication error (background: yellow)
- WinPCL messages (background: white)
- General messages (background: green)

We recommend to set system faults always to active.

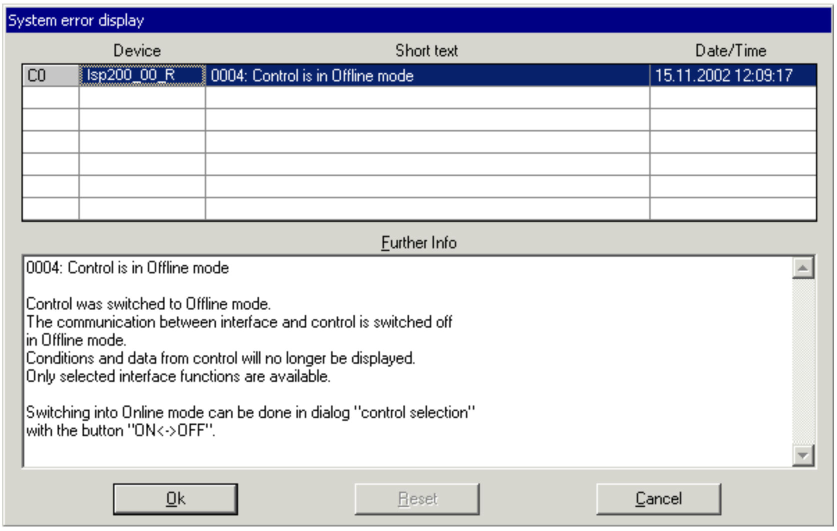
The dialog can be shown and hidden using the "<" /">" keys. It can also be activated via the pop-up menu by pressing the right mouse button or the <Shift>+<F10> keys.

In addition to the function of activating and deactivating the dialog, the pop-up menu also provides the known specification functions.

Display of System Errors

In case of a (system) error the display indicates additional information that you can’t find in the Event Display.

Thereby, the errors can concern the proper control or any other control of the respective configuration.



Systemfehleranzeige.bmp

C0: Address of the control in the configuration
Isp200_00_R: Name of the control (system configurator)

Fig. 4-96: System error display

Note: The reset key in the figure above as well as the menu item "Reset PLC" is only active during online operation.

Miniature Control Panels

If a miniature control panel displays the values of PLC variables, these variables must be downloaded to the control separately .

To achieve this, the screen manager /8/, /9/ generates a file (BTV file) for each application containing PLC variables, with this file providing the appropriate information.

The files whose data is to be downloaded to the control must be selected for each resource downloaded to the control. The appropriate file must be selected for each screen manager application intended to communicate with this control.

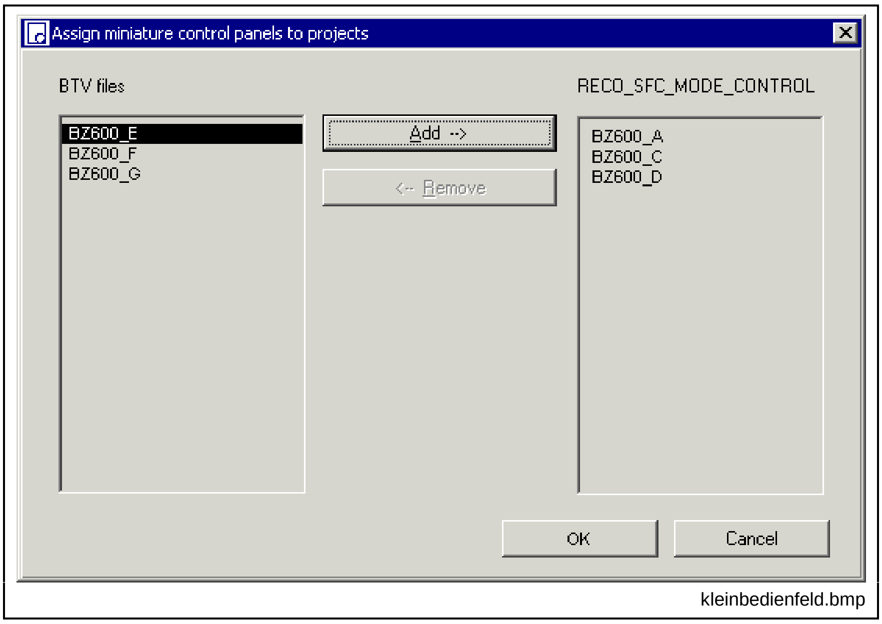


Fig. 4-97: Miniature control panel selection window

The BTV files which can be selected are shown in the left-hand window.

The right-hand window shows the files downloaded to the control.

If an application only contains the PLC diagnosis function, but not its own variables, it is not necessary to select a BTV file.

Note: This menu item is activated only if a resource file is opened in the focused editor.

Diagnosis, Module Assignment

This section provides information required for working with the diagnosis function in WinPCL:

- ProVi Messages (Diagnosis in LD / IL Networks)
- SFC diagnosis
- Module Assignment (Multiple Use of POUs)
- Diagnosis Display of I/O Addresses in and FBs

Diagnoses are subdivided in diagnoses associated with instruction list or ladder diagram networks and diagnoses tied to sequential function charts. Since a diagnosis is always filed in the program code, modules must be assigned if a function block with diagnosis generation comprises several instances.

ProVi Messages (Diagnosis in LD / IL Networks)

General

ProVi messages are messages emitted by the PLC, which can be displayed on the WinHMI GUI or on the miniature control panels by means of the screen manager.

ProVi messages are subdivided in five message types:

- Errors
- Messages
- Warnings
- Starting conditions
- Setup diagnosis functions

The message type defines the type of display on the WinHMI GUI (see /7/ WinHMI documentation).

The warning, starting condition and setup diagnosis message types are contained only once in each control.

The error and message message types are contained once in each module, but can be contained several times if there are several modules in a control (for modules see /7/ WinHMI documentation).

The text to be displayed for a ProVi message must be entered in the Message Integrator. There, the message can also be translated for multilingual diagnosis (see /7/ WinHMI documentation).

Rexroth
Bosch Group

Programming a ProVi message

ProVi messages can be emitted in each program and each FB.

In these POUs, a ProVi message can be assigned to each network with Boolean result.

Proceed as follows:

- Program the network intended to trigger the ProVi message.
- Press the right mouse button (or the <Shift>+<F10> keys) to select the ProVi messages item.

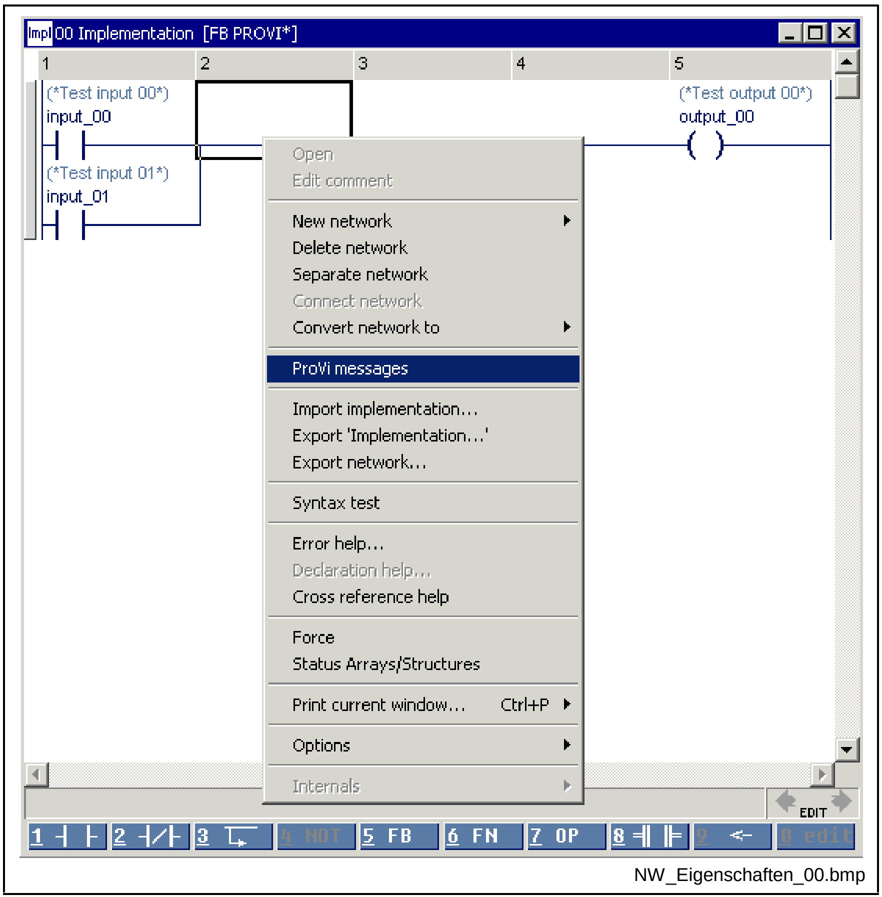
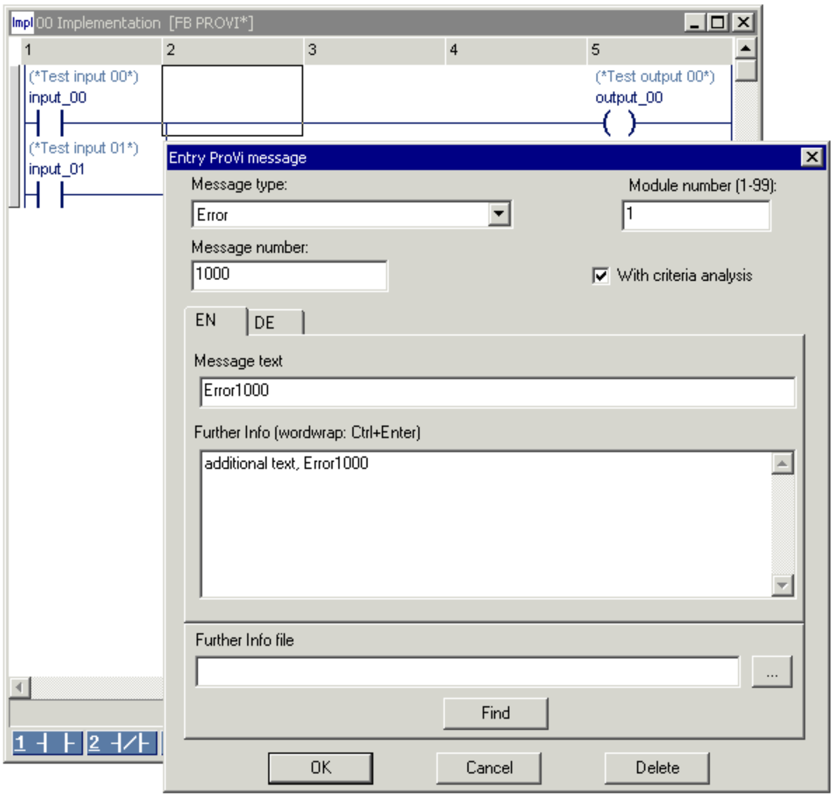


Fig. 4-98: Assignment of ProVi messages

- A dialog opens where the message type (error, message, warning, etc.), the message number and the module number can be entered.



NW_Eigenschaften_01.bmp

Fig. 4-99: Dialog for selecting the message type

Entry ProVi message dialog

This dialog can be used for multilingual entry of the text to be displayed for a ProVi message. The data entered here will be displayed in the Message Integrator and in WinHMI during diagnosis.

Texts already included in the Message Integrator can also be selected and assigned to a message in the PLC program.

Automatic selection of an unassigned message number

If the dialog opens for a network which does not contain any ProVi message, an unassigned message number is automatically suggested after the message type and module numbers have been selected (this number corresponds to the highest existing message number + 1).

This dialog can then be used to enter the message text, the continuing text and the continuing text file for this message number.

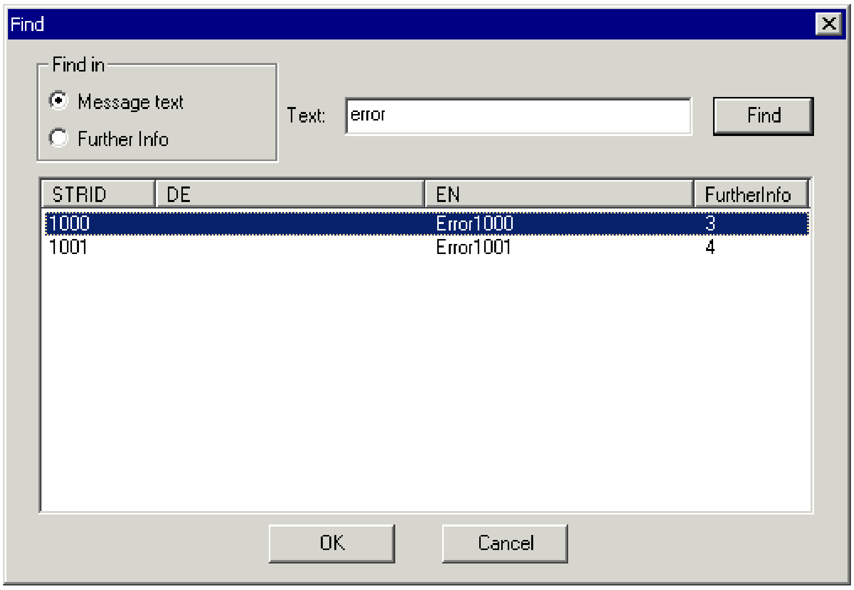
Manual selection of a message number

If the automatically selected message number fails to be the one desired, the message number can also be entered manually. If the Message Integrator already contains data for this message number, this data is displayed in the dialog where it can also be edited.

Finding a message number

It is also possible to find an already existing message text and to accept its message number.

The Find dialog can be called up using the Find button.



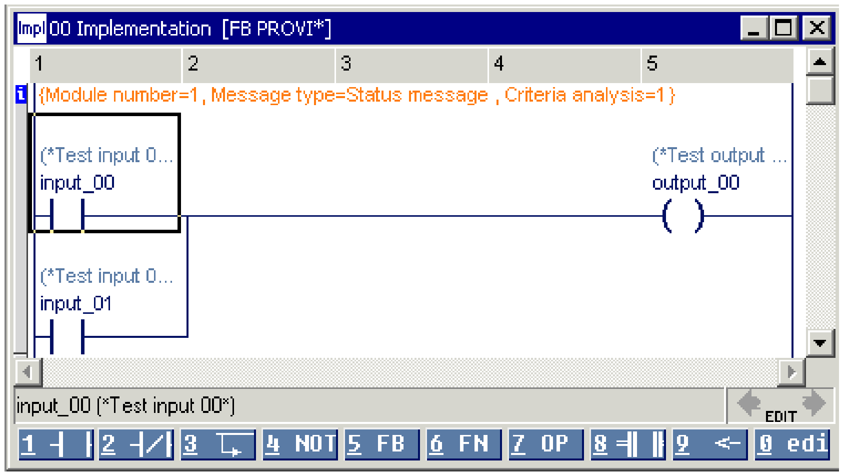
NW_Eigenschaften_03.bmp

Fig. 4-100: Find dialog of the Entry ProVi message window

It is possible to search in the message texts or in the continuing texts, always in all existing languages.

Exiting the dialog by clicking on OK applies the selected message number in the Entry ProVi message window.

- Any ProVi message assigned to the network is indicated by the "blue i" to the left on the status bar of the network. The orange color of the characters characterizes the message.
- Use "Edit\Find <Strg F>" to search for parts of the orange writing.



NW_Eigenschaften_02.bmp

Fig. 4-101: The "blue i" indicating a ProVi message

Note: The maximum length of the message texts must not exceed 80 characters.

If the ProVi message texts for operator terminals are to be transferred to the control, the following restrictions for ProVi messages have to be considered:

- Maximum 999 messages can be transferred to the control.
- The number of ProVi messages is limited to the range of 1 to 65535.
- Message number 0 has a special status. It is used to generate a default text in the operator terminal, if no message text is defined for a requested message number (e.g. no message text defined for ProVi message 123). For this reason, message number 0 must not be used in this case.

Output of ProVi messages

If the result of a ProVi network is TRUE, the message is emitted; the message is applied until the result of this network is FALSE again.

Here, the status of the result of the network is decisive, not the status of the variable at the end of the cycle. This permits to make use of the same variable in different ProVi networks.

Note: If a network is not edited any longer (e.g. it is skipped, or the action is not active any longer), the ProVi message cannot change. In other words, if the requirement for output of a message is not met any longer and the network of this message is not executed any longer, this message is nevertheless applied.

Analysis of ProVi criteria

The default setting of a ProVi message is without criteria analysis. However, the criteria analysis can be activated separately for each message.

To save the error status, a latch can be programmed. If a latch is in the network, the whole latch branch (even if it still contains other contacts) is not calculated in the criteria analysis and does not appear on the display.

To achieve a reasonable diagnosis, specific programming guidelines must be observed, as otherwise the criteria analysis indicates all elements of the network as faulty:

- Principally, only operations of Boolean variables can be analyzed. These can be globally and locally declared. They can be part of a multi-element variable (ARRAYs and structures as well as their hybrid form).
- It is not allowed to create an intermediate result.
- Inclusive-XOR operations are not permitted.
- There’s no tracking of the error cause via several networks (dummy flag calculation).
- Operations, functions and function blocks can not be analyzed.

Exceptions:

It is possible to use function blocks with a Boolean input as upper input and at least one Boolean output as only output that is wired.

Such a FB is not calculated in the criteria analysis, i.e. the criteria analysis recognizes the upper input and the output as connected. Therefore, e.g. timers can be used for a time delay of the message during limit switch monitoring. To generate a useful error indication in case of an error the used function block must switch a positive network result (VKE=TRUE) from its upper input to its upper output.

If the ProVi criteria analysis is carried out, the I/O addresses of the variables are displayed (see Diagnosis Display of I/O Addresses in PRs and FBs).

SFC diagnosis

General information

Each sequential function chart with operating modes (see IndraStep documentation /2/) can generate a diagnosis on the WinHMI GUI.

The criteria analysis can be called up for this sequential function chart (see WinHMI documentation /7/). The criteria analysis then displays one or more ladders which have caused the error in the sequential function chart.

The criteria analysis is carried out automatically for each disturbed sequential function chart with diagnosis function, so that it is not necessary to program an additional code. It is, however, necessary to observe some programming guidelines so as to obtain a reasonable criteria analysis.

A sequential function chart must be assigned to a module. This defines the position of indication on the WinHMI GUI.

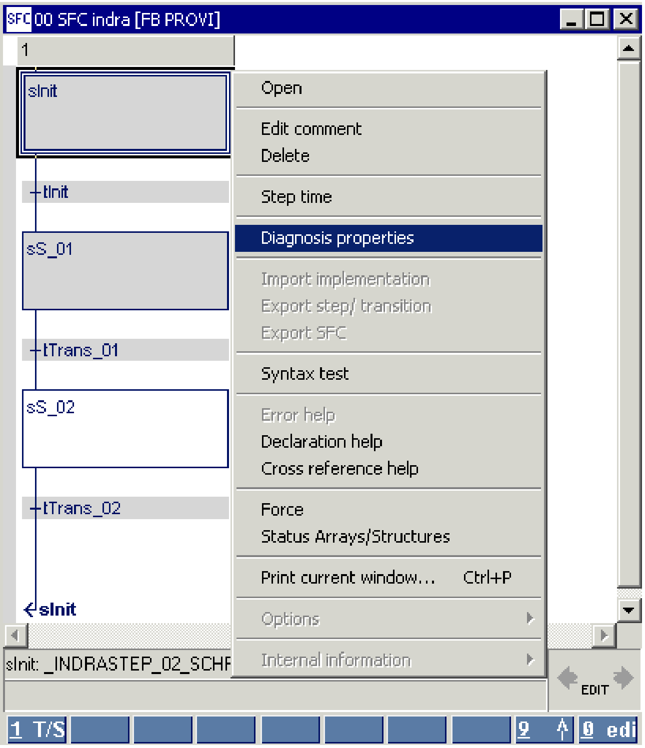
The comments of the sequential function chart (sequence, action, transition, step, variable, IL / LD) can be translated in the Message Integrator for a multilingual display of the diagnosis.

Programming a SFC diagnosis

By assigning a module to the sequential function chart, the diagnosis for this sequential function chart is achieved automatically.

Proceed as follows:

- Program the SFC intended to trigger the diagnosis message.
- Press the right mouse button (or the <Shift>+<F10> keys) to select the Diagnosis properties item.



SfcDiag_00.bmp

Fig. 4-102: Assigning the SFC properties

Rexroth Bosch Group

- A dialog opens where the module number can be entered.

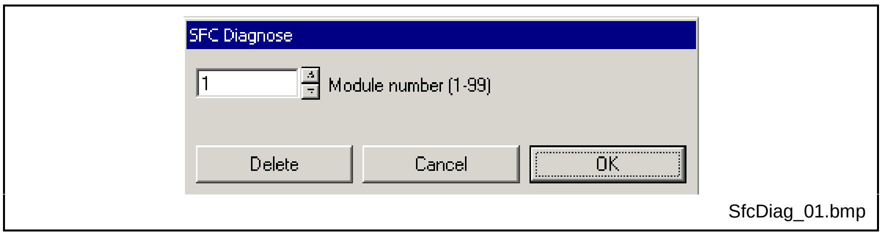


Fig. 4-103: Dialog for entering the module number

- The to the left on the SFC status bar indicates that the diagnosis function is assigned to this sequential function chart. The orange color of the characters characterizes the message.
- Use "Edit\Find <Strg F>" to search for parts of the orange writing.

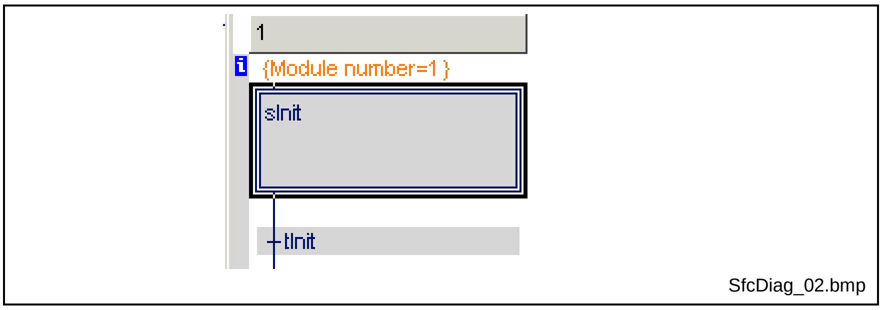


Fig. 4-104: The "blue i" indicates a sequential function chart with diagnosis

Output of SFC diagnosis messages

In case of a failure in the sequential function chart (see "Programming with IndraStep - SFCs with Mode Control and Diagnosis" /2/), a corresponding message is emitted.

This message specifies the SFC name, the failed step and the SFC error type. This message is applied until the error in the sequential function chart is cleared.

It is also possible to call up the criteria analysis for this failed sequential function chart on the WinHMI GUI (see WinHMI documentation /7/).

Programming guidelines

In order to achieve a reasonable diagnosis on the basis of the criteria analysis, the following programming guidelines for the actions and transitions of the SFC must be observed.

These guidelines are applicable to programming of the sequential function chart. In other words, only those actions and transitions are affected which are contained in a sequential function chart with diagnosis.

The implementations, actions and transitions, which are not used in the SFC diagnosis, will not be displayed in the criteria analysis. This means that these guidelines do not apply to the code contained therein.

- It is not permitted to use a Boolean transition or a Boolean action.
- The code in the sequential function chart may contain Boolean variables only.
- Inclusive-XOR operations are not permitted.
- It is not permitted to call up functions or function blocks.
- Temporary results, i.e. assignments within one network, are not permitted (see example below).

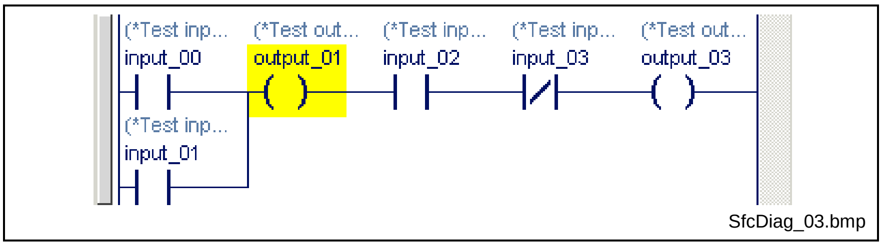


Fig. 4-105: Impermissible use of the temporary flag "Output_01" (yellow).

Exception:

- There are variables which, when used, cause the network to be removed from the diagnosis.
  These variables must be defined as described in "Programming with IndraStep - SFCs with Mode Control and Diagnosis" /2/.
  Networks in which these variables are assigned are not displayed in the diagnosis.
- There are variables which, when used, cause a function block to be removed from the diagnosis.
  These variables must be defined as described in " Programming with IndraStep - SFCs with Mode Control and Diagnosis" /2/. If a function block is used in networks in which one of these variables is assigned, then this function block is not displayed in the diagnosis.
  This exception is valid for function blocks containing a Boolean input as upper input and a Boolean output as only output that is wired. The network may only contain one FB, otherwise the FB’s are still displayed in the criteria analysis.

  Such a FB is not calculated in the criteria analysis, i.e. the criteria analysis recognizes the upper input and the wired Boolean output as connected. Therefore, e.g. timers can be used for the time delay of the message during limit switch monitoring. To generate a useful error indication in case of an error the used function block must switch a positive network result (VKE=TRUE) from its upper input to its upper output (see also Variable "Output_03" in the following example).

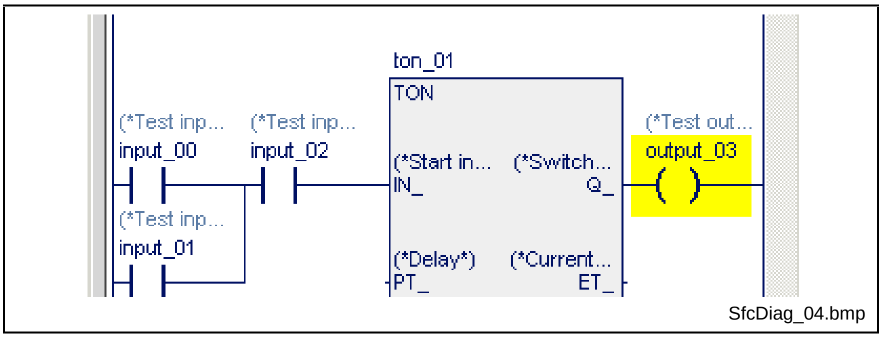


Fig. 4-106: Hiding FBs from the diagnosis by defined variables (yellow)

Diagnosis display:

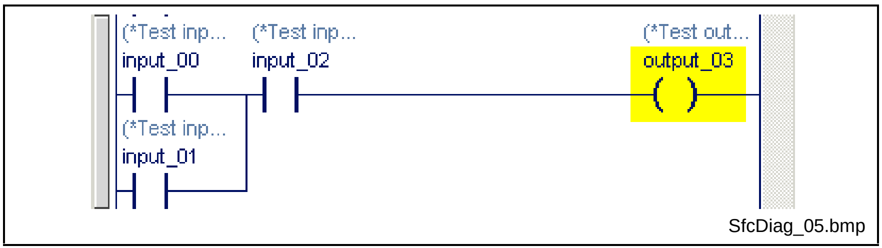


Fig. 4-107: Hidden function block

Note: There's a tracking of the error cause via several networks within an action (dummy flag calculation).

Diagnosis Display of I/O Addresses in PRs and FBs

In the criteria analysis, the I/O address is displayed for variables corresponding to absolutely addressed inputs and outputs.

Under the following conditions, this is applicable even if the real I/O variable is not defined in the POU of the sequential function chart:

1. The I/O variable is defined globally and is used as VAR_EXTERNAL variable in the POU.
2. The variable displayed is an input or output of the POU, and an I/O variable is directly programmed at this input or output. The variable may neither be negated nor linked to other variables.

Note: The POU must always be invoked because, otherwise, the status of the internal variable does not correspond to the status of the input or output.

Example:

The variable Input_01 is used in the sequential function chart of the SfcFB POU. In the criteria analysis, the address %I1.4.6 is displayed for this variable.

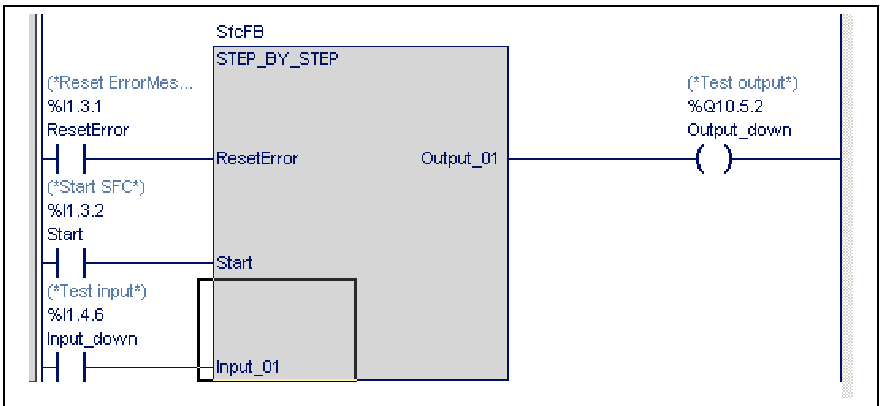


Fig. 4-108: Diagnosis display of absolute addresses in FBs

Module Assignment (Multiple Use of POUs)

If an FB (or a program) with diagnosis is declared several times, the instance in which the diagnosis is to be displayed must be defined.

Example:

One of the function blocks (DRILL_FB) completely controls a drill and also contains the diagnosis messages of the drill. A control should control two modules each of which contains one drill. Instances of the same function block are used for either drill.

```
VAR
Drilling_Module1        FB_DRILLING        (*Drilling station 1*)
Drilling_Module2        FB_DRILLING        (*Drilling station 2*)
```

Diagnose_Bohrwerk_00.bmp

Fig. 4-109: Declaration of two instances of DRILL_FB

ProVi messages (errors and messages) and a sequential function chart (drill) are programmed in the FB.

During programming, module numbers had to be specified for these diagnoses (SFC diagnosis). In the example, module number 1 has been programmed. However, the diagnosis for one of the drills should be displayed in module 1 and that of the other drill in module 2.

In the resource, a separate module number can be assigned to each use of a diagnosis.

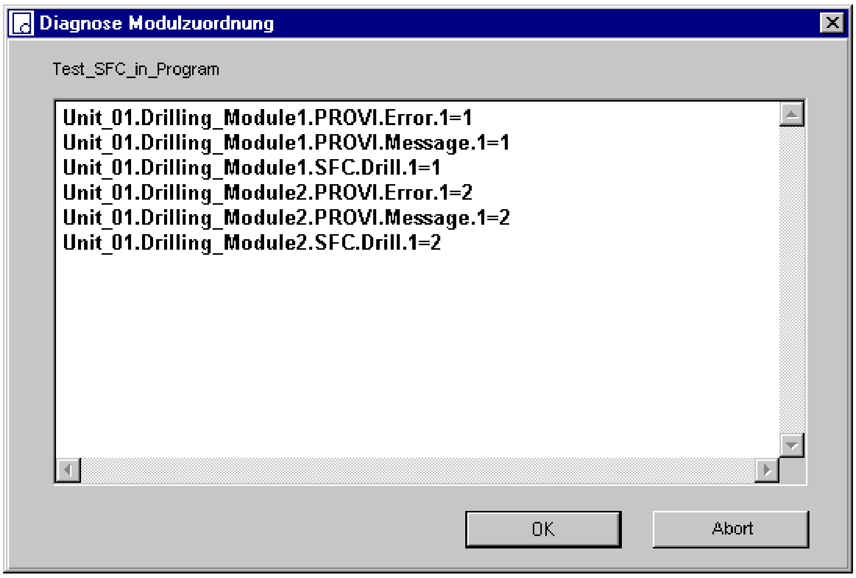
The dialog for the module assignment can only be called up for the resource of the PLC program. This can be achieved using the "Tools \ Diagnosis module assignment" menu item.

This dialog contains an ASCII editor where the module assignments can be entered with the appropriate syntax:

Rexroth Bosch Group

Example:
The program where the two drills (see above) are declared is declared as Device_01 in the resource.

In this dialog, module number 1 is assigned to all diagnoses of Drill_Modul1 and module number 2 to all diagnoses of Drill_Modul2.



Diagnose_Bohrwerk_01.bmp

Fig. 4-110: Diagnosis module assignment

If no entry is made in this dialog, the original module number is assigned, i.e. the entry for Drill_Modul1 in the above figure is not necessary because the original module number is 1.

Syntax:

- The various specifications are always separated by a dot.
- The following are defined keywords: **SFC, PROVI, ERROR, MESSAGE.**
- The complete instance name must be entered for the POU. The individual instances of the POUs must be separated from each other by a dot. Example: Device_01.Drill_Modul1
- Sequential function charts are specified by means of **SFC**.*SfcName.ModulNo.*
  - *SfcName* = name of the sequential function chart in the POU
  - *ModulNo* = module number programmed in the original (SFC diagnosis)
- ProVi messages are specified by **PROVI**.*MessageType.ModulNo.*
  - *MessageType* = **ERROR** or **MESSAGE**
  - *ModulNo* = module number programmed in the original (ProVi Messages (Diagnosis in LD / IL Networks))

  There is no module number for the other ProVi message types. For that reason, they cannot be assigned here.
- New module numbers are specified by =X:
  - X is the new module number.

The syntax of a line is as follows:

- For sequential function chart:
  *InstanceName.***SFC***.SfcName.ModulNo=X*
- For ProVi:
  InstanceName.**PROVI**.MessageType.ModulNo=X

The assignment to the module number is possible at any position in the path. It is, therefore, not necessary to enter the entire string:

Character string	**Meaning**
Device_01.Drill_Modul2.PROVI.Error.1=3	Only this message type is displayed in module 2.
Device_01.Drill_Modul2.PROVI.Error=3	All ProVi errors in this instance are displayed in module 3.
Device_01.Drill_Modul2.PROVI=3	All ProVi messages in this instance are displayed in module 3.
Device_01.Drill_Modul2.SFC.Drill=4	Only this sequential function chart is displayed in module 4.
Device_01.Drill_Modul2.SFC=4	All sequential function charts of this instance are displayed in module 4.
Device_01.Drill_Modul2=5	All diagnoses of this instance of the POU are displayed in module 5. This also applies to all instances of the POUs declared in this POU.
Device_01=5	All diagnoses appearing in this program are displayed in module 5. This also applies to all instances of the POUs declared in this POU (in this example: Drill_Modul1 and Drill_Modul2).

Fig. 4-111: Examples of module number assignments

It is always the last module assignment in an instance path that is the decisive one. If, for example, the following assignments have been made:

Device_01.Drill_Modul2.SFC.Drill=4

Device_01.Drill_Modul2=3

Device_01=2

the diagnoses of the example are displayed in the following modules:

- Instance Device_01.Drill_Modul1
  - ProVi error 1 in module 2
  - ProVi message 1 in module 2
  - Sequential function chart of drill 1 in module 2
- Instance of Device_01.Drill_Modul2
  - ProVi error 1 in module 3
  - ProVi message 1 in module 3
  - Sequential function chart of drill 1 in module 4

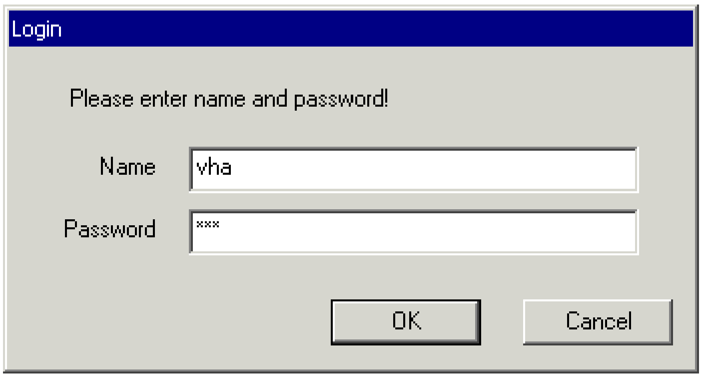
Password

The user logs in the system using the "Password / Login" menu item. By logging in (see: User Management, WinPCL Rights, Remote Programming), the actual user’s defined access rights are enabled.

The menu item also allows logout of the current user.

In addition to that, the current user can change the password with this menu, but the defined authorizations cannot be changed.

Login



login.bmp

Fig. 4-112: "Tools / Password / Login" menu item

Login has to take place with the agreed user name and the corresponding password.

Login is necessary to start the programming system and after expiry of the password. The old password expires and has to be replaced by a new one.

Note: Name and password must be case-sensitive!

Logout

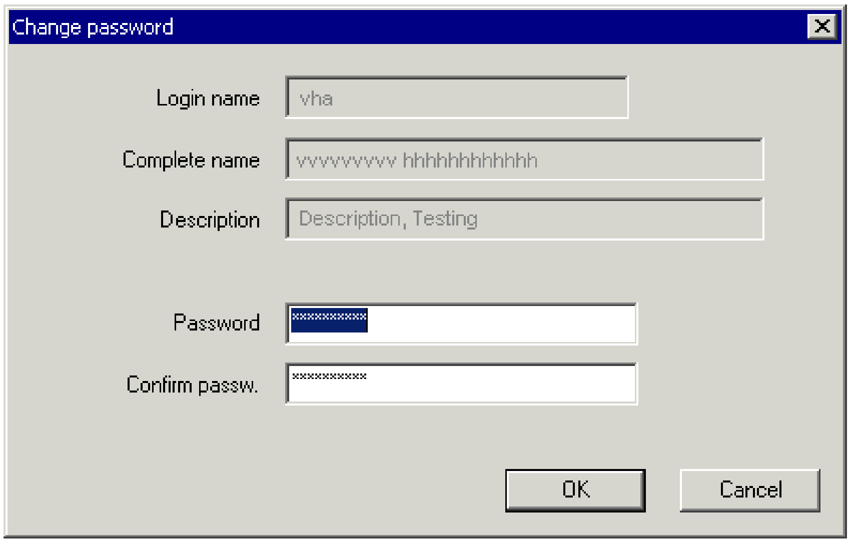
logout.bmp

Fig. 4-113: "Tools / Password / Logout" menu item

The password is no longer effective after its expiration date and when the user is logging out .

The "logout" is confirmed with the message window shown above.

Change Password for "***"



ändern pw.bmp

Fig. 4-114: "Tools / Password / Change" menu item

The menu item "Change password for "***" " allows a password to be changed for a user while the access rights defined in the user management are kept valid.

The window shows the login name, the complete name and a description of the user's function after "Login"; this information cannot be changed.

A new password can be entered.

The new password is accepted after confirmation and after pressing of the OK button.

Note: Name and password must be case-sensitive!

Fieldbus Configuration

It is possible to call the fieldbus configurators from the WinPCL interface at two different locations:

- Menu Tools / Fieldbus configuration
- Using the IO editor's menu called by the right mouse button with "Fieldbus configuration"

It is only possible to call up the fieldbus configuration, if a resource window is active.

Presently, two different fieldbus configurators can be used. The configurator to be used depends on the bus master to be configured.

- CMD (Phoenix) – Configuration of the Phoenix INTERBUS card
    - Master_1
    - Master_2, connection of a second INTERBUS master, whose bus is not yet PCP-compliant. An extension by the second bus master is only available after releasing version 06V02 / 23V02.
- SyCon (Hilscher) – Configuration of the Hilscher cards; here, INTERBUS, PROFIBUS, DeviceNet and AS Interface are supported.

It is not possible to start the configurator several times. If the configurator shall be opened for a further control, resource or another bus type, the currently running version has to be terminated.

But it is possible to start CMD and SyCon at the same time.

Data Management

The data of the fieldbus configuration are always saved for the selected resource. Thus, an own configuration for every resource can be saved.

This data are saved and restored when storing in an PLC archive.

If a configurator is started form WinPCL, the two configuration files "BG4 and INI" are generated in the root directory of drive C. The configurator works with this files. After terminating the configurator the files are deleted again and the changed configuration data are taken over in the data of the resource.

Note: If the configurator is started stand alone, the data of a resource can not be processed. There's no possibility to open the corresponding configuration files directly from the hard disc.

The files generated in the root directory bear the name DeviceXX.BG4 and DeviceXX.INI, whereby XX corresponds to the control number, with which communication takes place.

Remote Programming

If WinPCL is started as remote programming system, the configurators must be only installed on the programming computer. It is not necessary to install the configurators on the control computer.

In the case of remote programming WinPCL must be opened as long as it is worked with the configurator. As soon as WinPCL is closed, the current configuration can be no longer stored.

Note: This applies only to remote programming. If WinPCL runs on the control computer, you can close WinPCL after the start of the configurator and save the configuration anyway.

Standard Settings

If the configurator for a resource is started for the first time, an empty configuration is automatically generated with the communications path BRC.

But you can also use self-created configurations as standard. Please consider the following aspects:

1. Don't open the configurator from WinPCL.
2. Create the desired standard configuration.
3. Save this configuration in directory *..\Project_000\CustomData\Ressource* (This directory is to be found in the installation directory of WinPCL). The file name depends on the used configurator and the used bus type (see Phoenix Contact (CMD), Hilscher (SyCon), PROFIBUS, DeviceNet, AS Interface).

Note: As for the remote programming the standard configuration files are searched on the programming PC and not on the control PC.

INTERBUS

To configure the INTERBUS two configurators can be used:

- Phoenix Contact (CMD)
- Hilscher (SyCon)

Depending on the used bus card the corresponding configurator has to be selected.

Phoenix Contact (CMD)

Communications Path

If in the CMD tool BRC is set as communication path, a direct communication (without serial cable) with up to two INTERBUS controller boards is possible.

Note: If the CMD tool is opened from WinPCL this is the standard setting. But this setting can be changed anytime and is still available when the configuration of this resource is opened again.

As the CMD tool needs the information, with which INTERBUS controller board it is to communicate, direct communication with the INTERBUS controller boards is only possible when they are called from WinPCL.

Note: Communication with the INTERBUS controller board is only possible, if a valid PLC program with programmed INTERBUS I/Os is executed in the control.

Own Standard Configuration

The CMD tool generates besides the BG4 file also an INI file with the same name. This file must be copied in the corresponding directory as default file (PlcCmdDefaultConfig.ini).

Note: The CMD tool only supports 8 signs as directory and file names. Therefore, the files have to be stored under another name and then, have to be renamed.

If "Preprocessing variables" is used in the standard configuration, the CMD tool supports only 32 signs as complete path and file name In this case the files have to be stored in another directory, at first, and then, have to be copied.

Hilscher (SyCon)

Driver

If the BRC driver has been selected in the system configurator SyCon, a direct communication (without additional cable) is possible by using the fieldbus card.

As the SyCon needs the information, with which controller board it is to communicate, direct communication with the controller board is only possible when it is called from WinPCL.

Note: Communication with the Hilscher system configurator is only possible, if a valid PLC program with progammed I/Os is executed in the control.

Own Standard Configuration

The own standard files have to be saved under the name PlcSyConDefault. Depending on the used bus type the ending is different.

- INTERBUS – PlcSyConDefault.ib
- PROFIBUS – PlcSyConDefault.pb
- DeviceNet – PlcSyConDefault.dn
- AS Interface – PlcSyConDefault.as

PROFIBUS

From WinPCL only Hilscher PROFIBUS cards can be configured with SyCon.

DeviceNet

From WinPCL only Hilscher DeviceNet cards can be configured with SyCon.

AS Interface

From WinPCL only Hilscher AS Interface cards can be configured with SyCon.

Logic Analysis

This menu item starts the general "Logic analysis" tool.

The online help contains a detailed description of this tool.

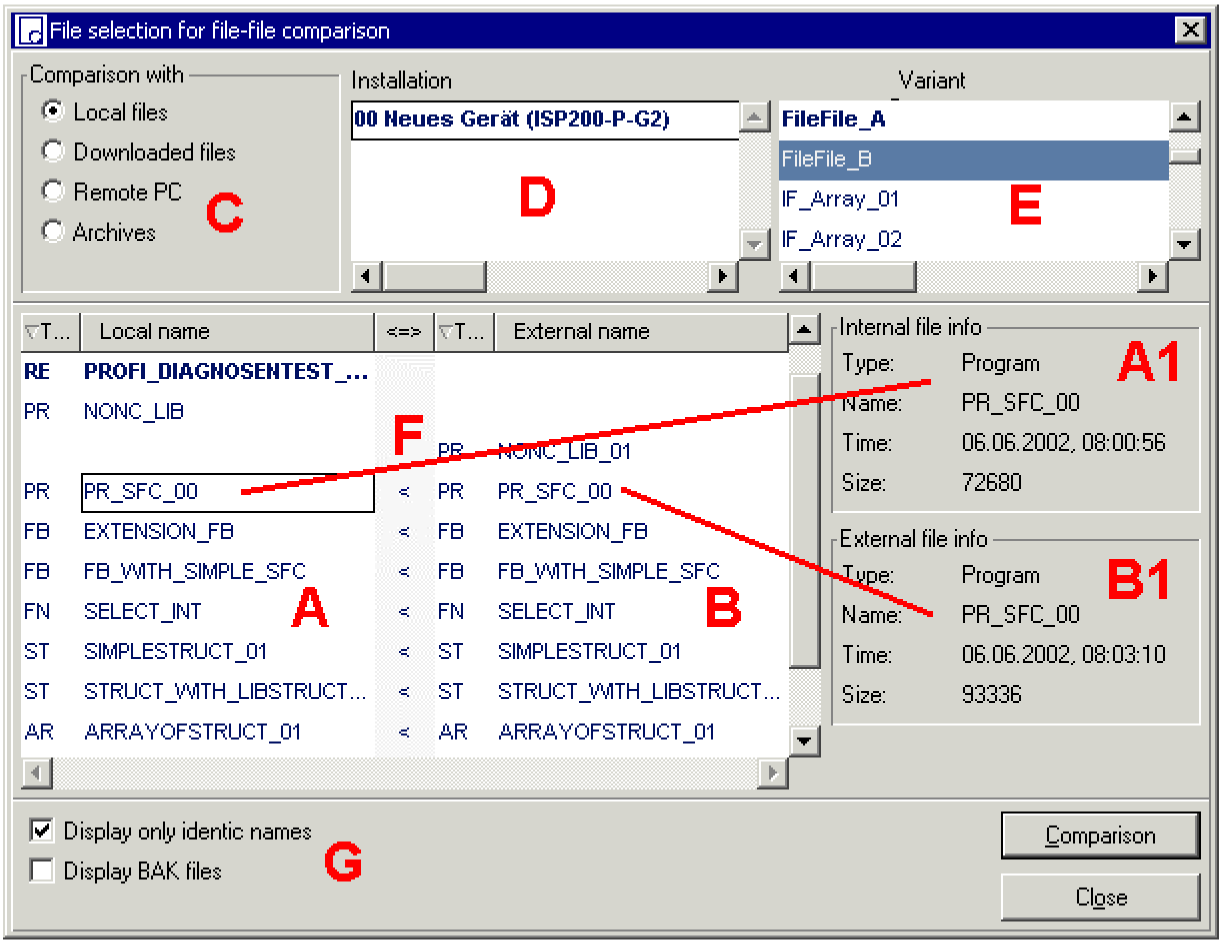
Note: Now, the logic analysis is also available for remote programming.

File-File Comparison

This menu item allows to compare WinPCL files, whereby the selected files are compared in textual form. The differences are marked in terms of colors.

With the option "Display only identic names" it is determined which of the two following windows appear for the immediate file selection:

Option "Display only identic names"



FileFile_01.bmp

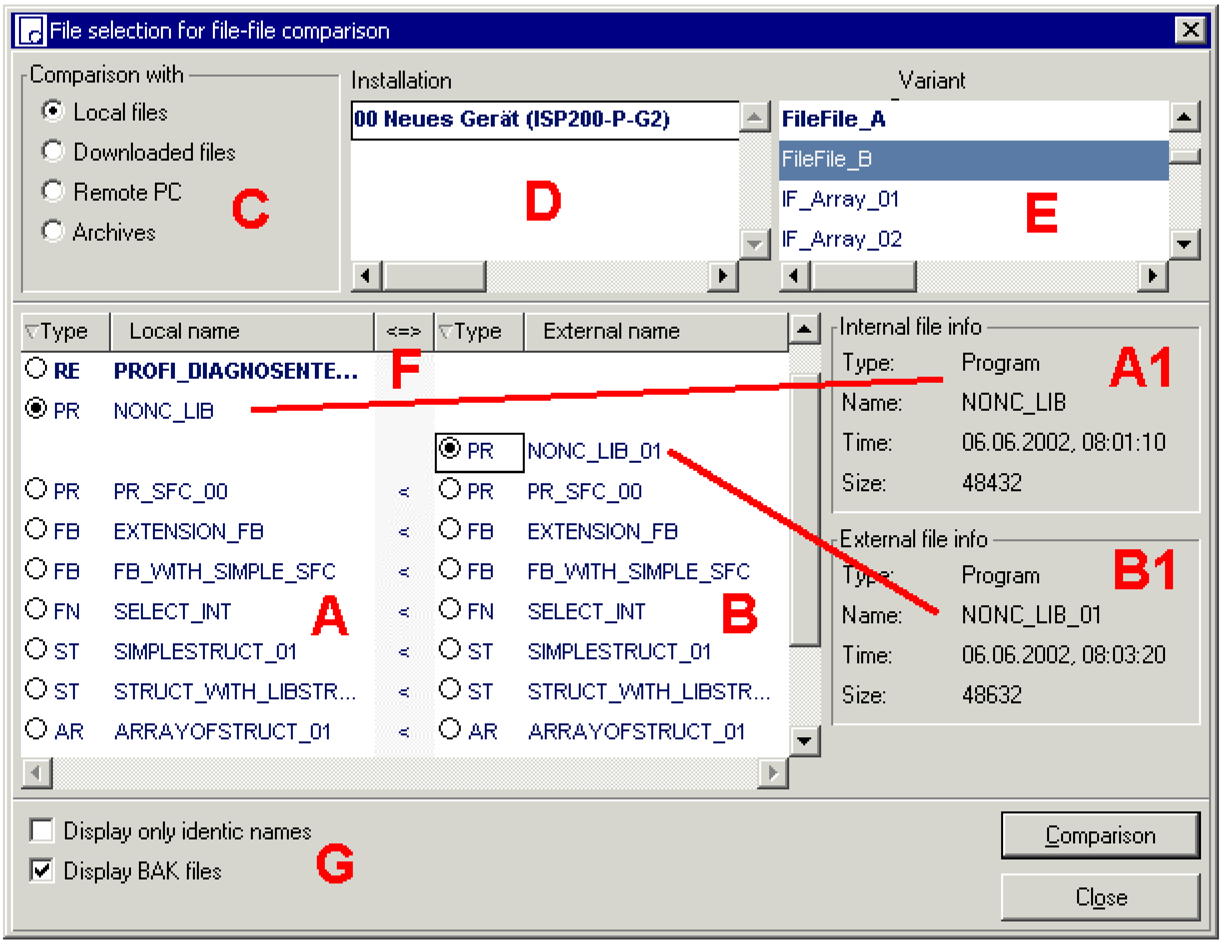
A:	Files of the current variant, selected file PR PR_SFC_00
A1:	File info on the selected file
B:	Files of the variant selected for comparison
B1:	File info on files with identic name of the second variant
C:	Selection possibilities for second variant (variant of any control **or** downloaded files of any control **or** remote connection **or** WinPCL archive at any location)
D, E:	Additional selection possibilities corresponding to field C
F:	Rough comparison depending on time if equal "=" or unequal "<", ">"
G:	Options: "Display only identic names" / With and without BAK files"

Fig. 4-115: Selection with option "Display only identic names"

With this option only files with identic names (and of the same type) can be compared. The rough comparison in column **F** allows a pre-estimation on the basis of the file time, if the file pairs are identic.

The file information is displayed for any selected pair (**A1/B1**).

Option "Display only identic names" is deactivated



FileFile_00.bmp

A:	Files of the current variant, selected file PR NONC_LIB
A1:	File info on the selected file
B:	Files of the variant selected for comparison PR NONC_LIB_01
B1:	File info on files of the second variant
C:	Selection possibilities for second variant (variant of any control **or** downloaded files of any control **or** remote connection **or** WinPCL archive at any location)
D, E:	Additional selection possibilities corresponding to field C
F:	Rough comparison depending on time if equal "=" or unequal "<", ">"
G:	Options: "Display only identic names" / With and without BAK files"

Fig. 4-116: Selection with **option "Free name assignment"** ("Display only identic names" is deactivated)

The forced pair formation for the identic name is canceled, if you deselect option "Display only identic names".

In this case, it is possible to compare files with their BAK files **or** files with renamed files **or** files which e.g., have been converted by a program in a function block.

Selection of Files of the Second Variant

Basically, the comparison starts from files of the current variant (or from the basic directory).

Files of the second variant can be taken from different sources:

- **Comparison with local files**

FileFile_02.bmp

C: Selection possibilities for second variant (variant of any control)
D: Selection of one of the available installations
E: Selection of a variant of the selected installation

Fig. 4-117: Selection window for local files

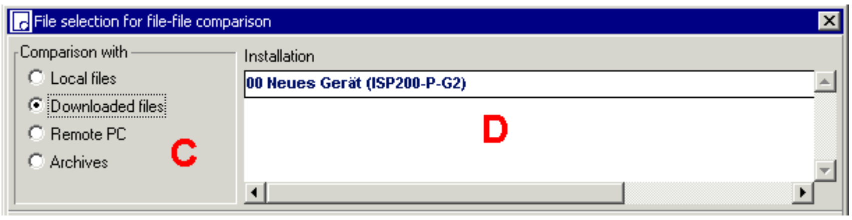
You can select the controls entered in the system configurator in field **D.** The current control is entered in bolt letters.

Field **E** shows the variants available for the selected control:

- FileFile_A: bold letters signify current variant (files on the left side)
- FileFile_B: blue bar signifies second variant (files on the right side)

The files can belong to the same control and in this case to the same variant **or** to the same control and another variant **or** to another control.

- **Comparison with "Downloaded files"**



FileFile_03.bmp

C: Selection possibilities for second variant (folder "Downloaded files" of any control)
D: Selection of one of the available installations

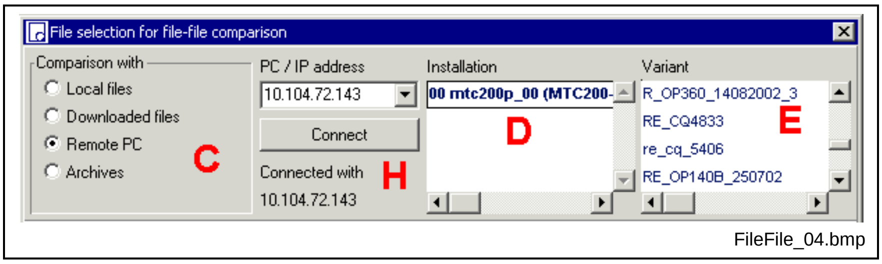
Fig. 4-118: Selection window for downloaded files

If a compound (resource with their programs...) is loaded in the PLC, the source files compiled to this compound are automatically stored in folder "Downloaded files" of the respective control.

Thus, the selection is restricted on the installation (control) because it has only one suchlike folder.

- **Comparison with remote PC**

Remote operation requires that a/any server (BTV) is controlled by a client (notebook).



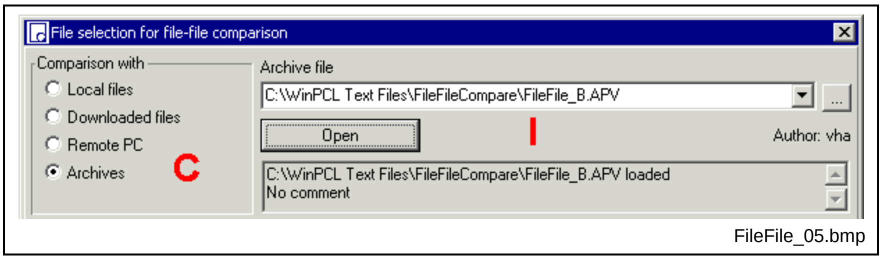
C:	Selection possibilities for second variant (remote operation to any control)
H:	Input of the PC name or its IP address, then "Connecting"
D:	Selection of one of the available installations on the PC/IP reached by remote connection
E:	Selection of a variant on this installation

Fig. 4-119: Selection window for remote PC

Note: The selected server (BTV) must work with activated remote PG interface.

- **Comparison with archived files**

WinPCL files can be combined as archives. The comparison can be executed with files of such an archive.



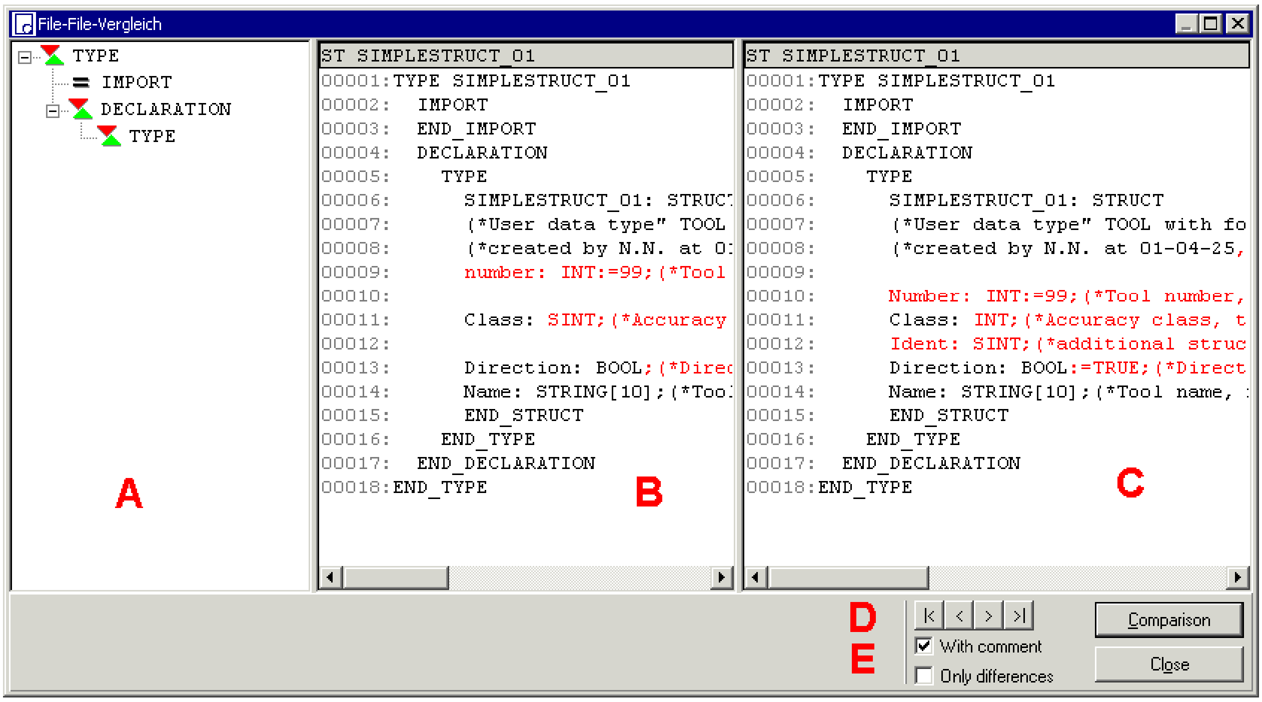
C:	Selection possibilities for second variant (WinPCL archive)
I:	Input or selection of archive name

Fig. 4-120: Selection window for archives

After setting of the archive file's name, at first, the right window is still empty. The archive files appear on the right side after clicking on "Open".

Comparator Window

After starting the comparator the preset files are compared. The selected files are exported as text files and compared line by line according to content groups. The following window contains the comparison results:



FileFile_06.bmp

A:	Rough structure of the file
B:	Display of file selected on the left side of the starting window
C:	Display of file selected on the right side of the starting window
D:	Go to the start / to the previous difference / to the next difference / to the end
E:	Additional options

Fig. 4-121: Comparison result

The rough structure **A** contains the logic structure of the file as tree.

The pictograms before the sections have the following meaning:

Pictogram	Meaning
=	Section identic, no differences
	Differences in section
	Missing details on the right side
	Missing details on the left side

Fig. 4-122: Pictograms and their meaning

Columns **B** and **C** contain the textual description of the selected files. Within the blocks specified by column **A** the texts are compared line by line and sign by sign.

Rexroth
Bosch Group

The crossover from difference to difference can be accelerated by button group D:

Buttons	Meaning
|<	Go to start
<	Go to the previous difference
>	Got to the next difference
>|	Go to the end

Fig. 4-123: Buttons and its meaning

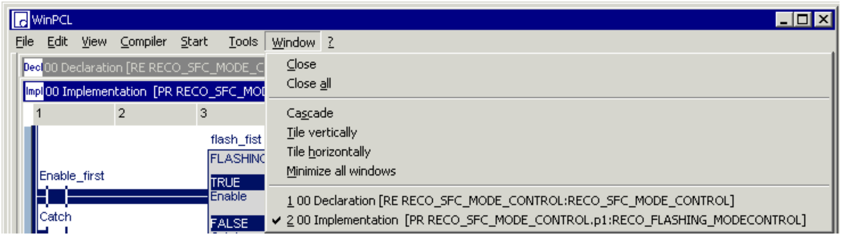
The options of group **E** "With comment" and "Only differences" allow an actualization of the comparison result without leaving the window for the identic files. Additionally to this, you have to activate button "Comparison".

If a comparison is executed "Without comment", the comments are not included in the comparison and are displayed as inactive (gray).

If a comparison is executed with "Only differences", identic sections are hidden.

For longer files you will find a slider at the right margin of field **B** and **C**. The slider in field **C** allows to move field **B** and **C** synchronously, the slider in field **B** allows to move the text in field **B** relative to field **C**.

4.8 Window



menue_fenster.bmp

Fig. 4-124: "Window" menu item

The "Window" menu item allows direct access to opened windows in addition to the Windows standard commands:

- <Ctrl>+<F4>: Close the focused window.
- <Ctrl>+<F6>: Go to and focus the next window.
- <Shift>+<Ctrl>+<F6>: Go to and focus the previous window.

Close

As is the case with <Ctrl>+<F4>, the focused window is closed. Before the last window of a program organization unit is closed, the file is checked to determine whether it has been modified; if yes, a safety prompt asks whether the changed status is to be saved or not.

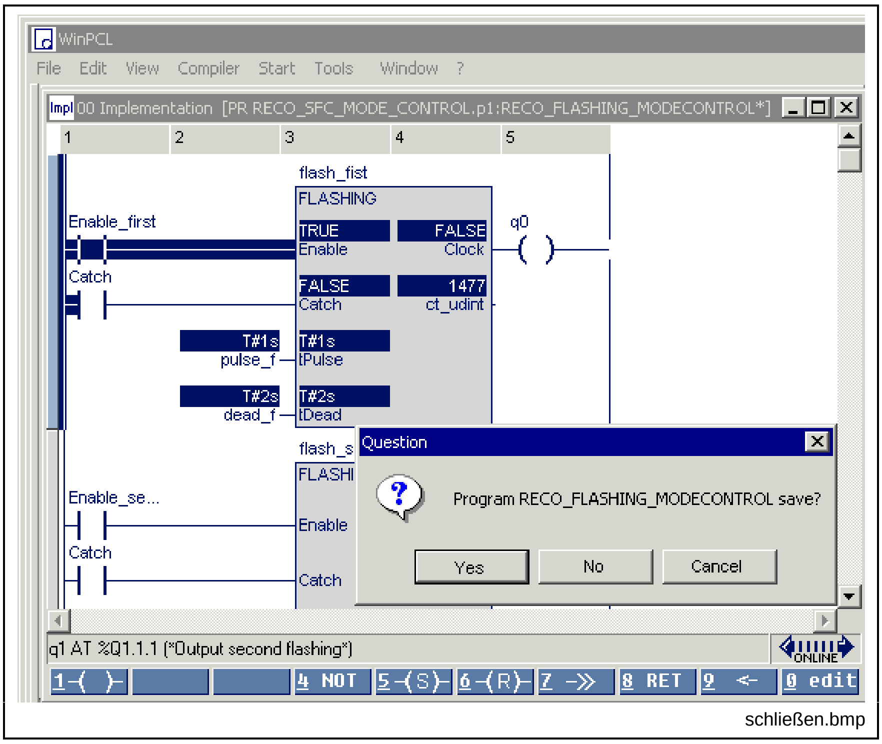


Fig. 4-125: Closing of the last window of a file

Close All

All opened windows are closed.

Before the last window of a program organization unit is closed, the file is checked to determine whether it has been changed; if yes, a safety prompt asks whether the changed status is to be saved or not.

Cascade

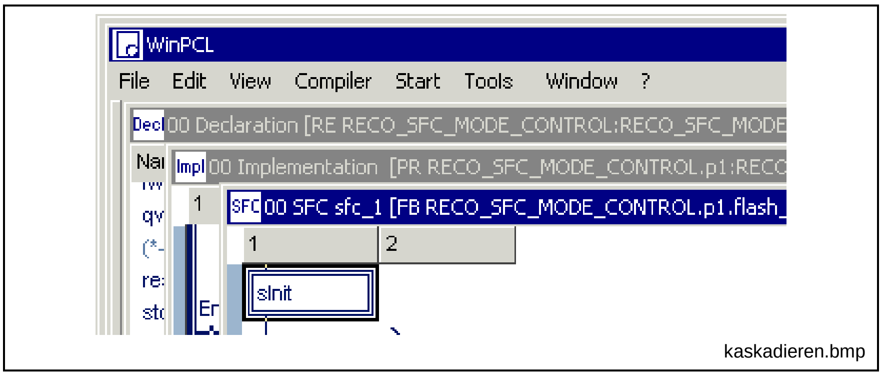
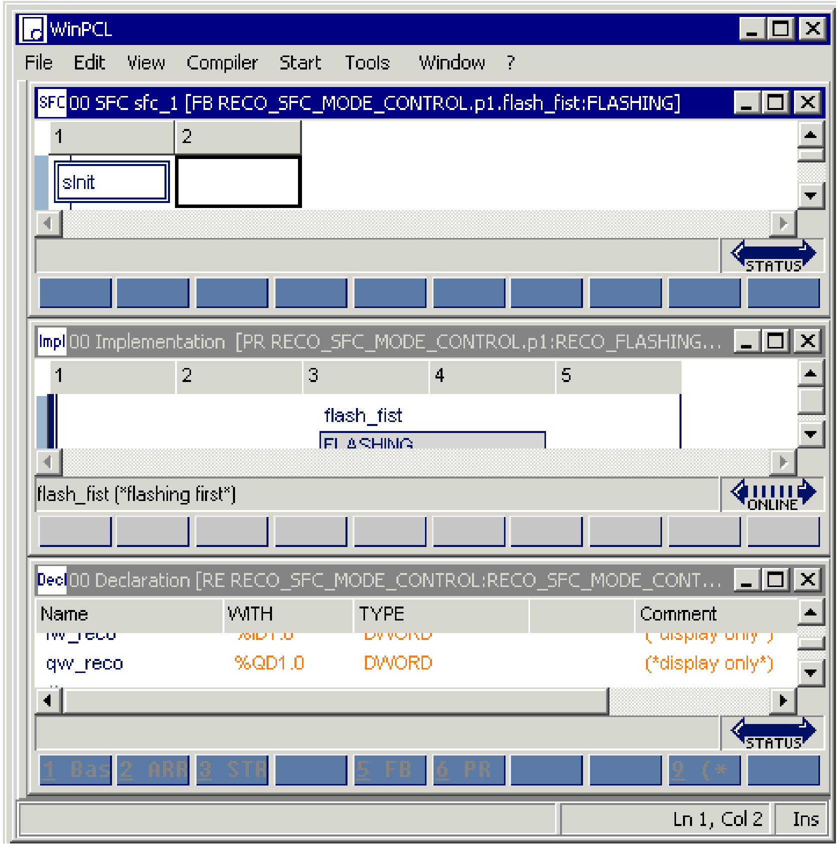


Fig. 4-126: Cascade windows

The opened windows are arranged one behind the other. The front window is focused.

Rexroth Bosch Group

Tile Horizontally

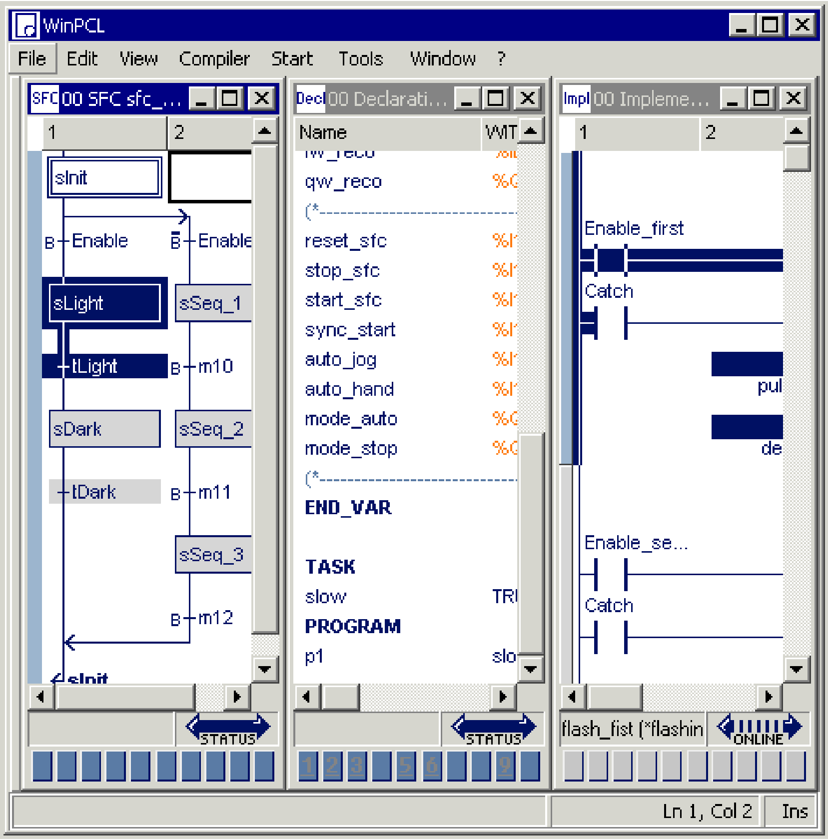


horizontal.bmp

Fig. 4-127: Tile windows horizontally

The whole area that is available is equally divided among the opened windows. The focused window is on top.

Tile Vertically



vertikal.bmp

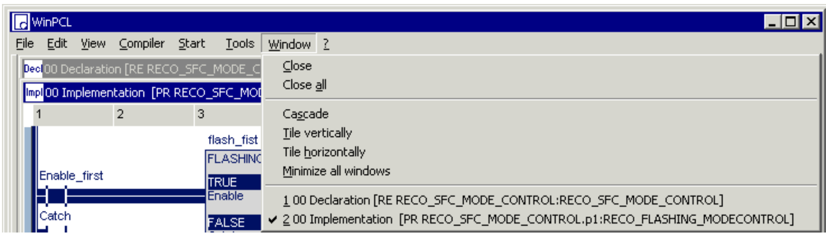
Fig. 4-128: Tile windows vertically

The whole area that is available is equally divided among the opened windows. The focused window is on the left.

Minimize All Windows

With this command, all opened files are reduced to their minimum size and are visible in the left area of the WinPCL window.

List of Windows



menue_fenster.bmp

Fig. 4-129: Minimize all windows

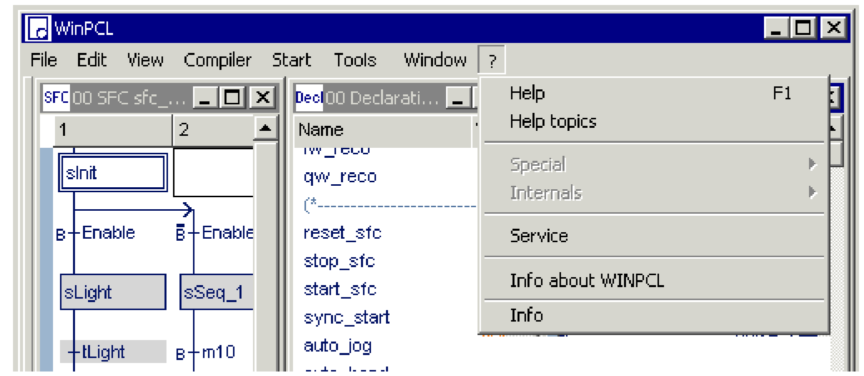
The list of opened windows shows for each window

- the number of the control,
- the editor and
- the type name or the complete instance name of the file.

The window number of the focused window is marked with a checkmark.

Any window can be accessed by double-clicking the mouse or using the window number..

4.9 ? Help



menue_hilfe.bmp

Fig. 4-130: "? Help" menu item

The "? Help" menu item provides access to the online help.

Using Help <F1>, the term where the cursor is positioned is applied as the search criterion and is then searched in the online help.

Using the Help Topics (Contents & Index), the search can take place in the WinPCL online help through a structured table of contents or any search criterion can be entered.

The "Special" and "Internals" submenus are only for service purposes.

In addition, a file password can be entered in the "Service" submenu. All files connected with this password, can be opened and edited without any restrictions.

System information on currently active components can be obtained by activating the "Info about WinPCL" and "Info" submenus.

Help <F1>

The <F1> help function represents the help on the search criterion. This search criterion can be defined as follows:

- The term the cursor is currently positioned on is taken as search criterion. (For example, the cursor may be positioned on the type name of a function block in the declaration editor or on a function name in the IL editor.)

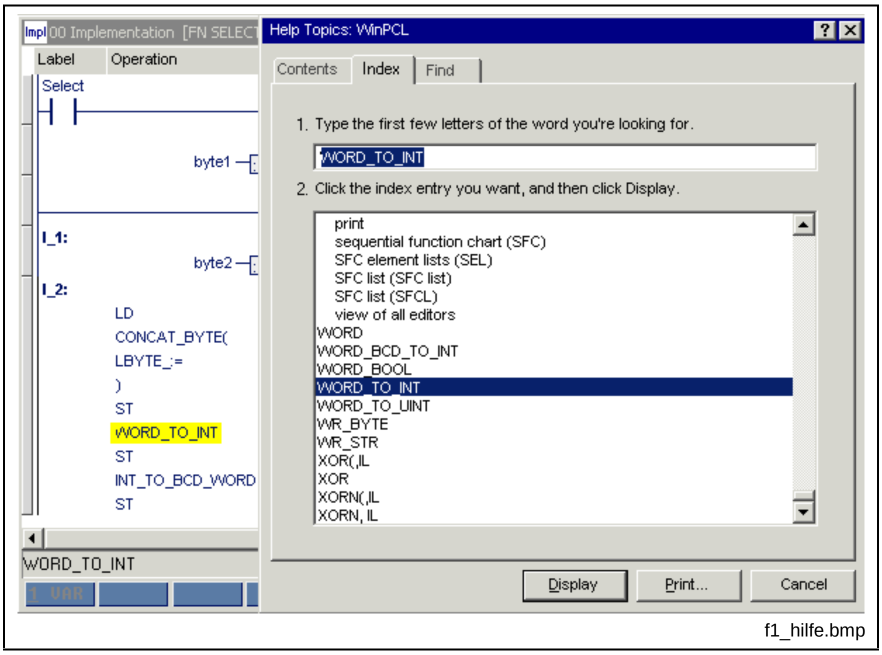


Fig. 4-131: <F1> help on cursor position

- A blank position in an editor may be used as search criterion. In this case, the online help refers to the editor description.
- A dialog where the <F1> key is pressed may be used as search criterion. In this case, the online help refers to the window description.
- The value of the status display of S#ErrorTyp and S#ErrorNr may be used as search criterion. In this case, the cause of the error and the error type are displayed (cursor positioned on the numerical value of the status display of the variable).

Help Topics (Contents & Index)

This menu item can be used to call up the general online help.

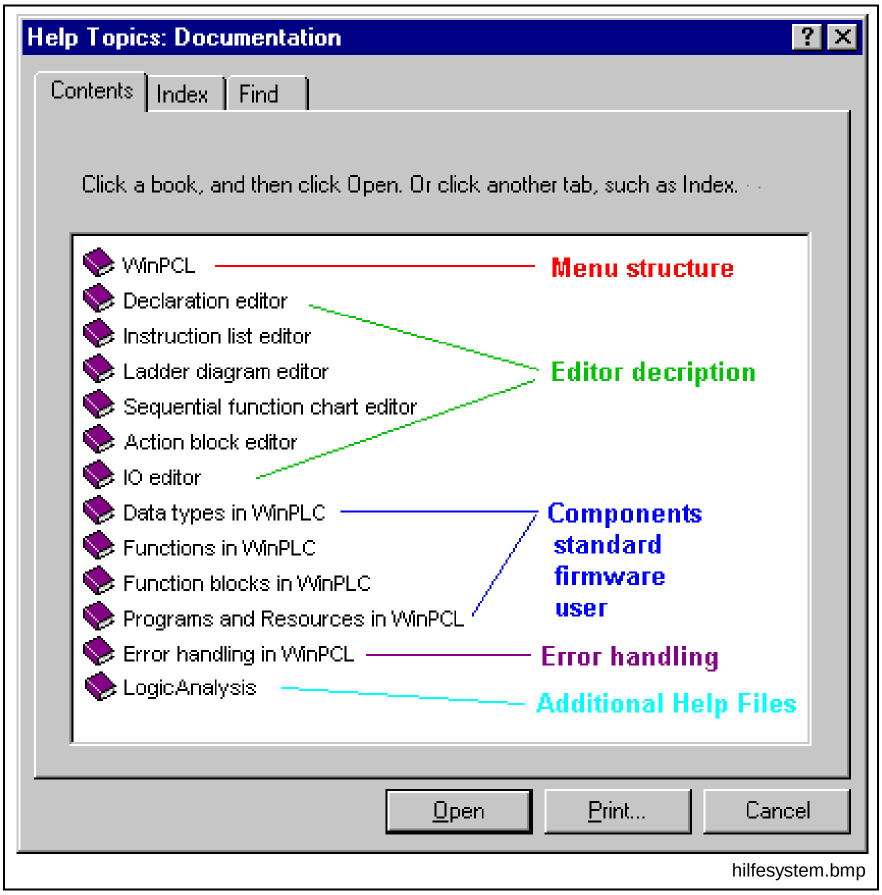


Fig. 4-132: Help topics on WinPCL

The WinPCL online help contains the following chapters:

The chapter "WinPCL" book provides an overview about the currently available functions of the WinPCL menu.

The editors are treated subsequently.

Data types, functions and function blocks give help with regard to the contents and to the standard and firmware elements.

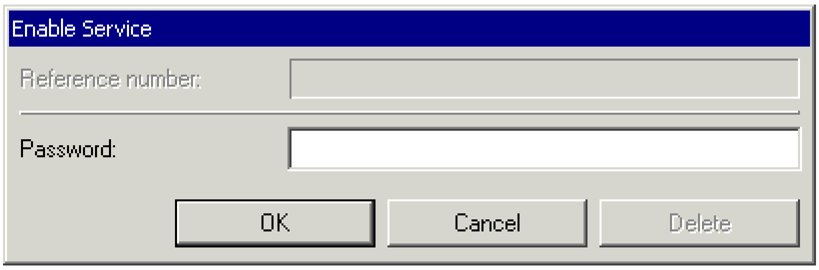
Special

The "Special" menu item is for service purposes and does not contain any menu items which are useful for the user.

Internals

The "Internals" menu item is for service purposes and does not contain any menu items which are useful for the user.

Service



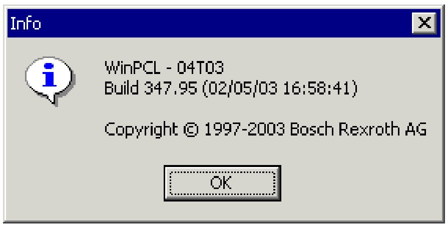
Service.bmp

Fig. 4-133: Enable service

Entry of the password provides the corresponding access (edit / view) to all files, which are protected with this password, without the password window opening for each file.

Whenever a password is entered, its deletion is also enabled.

Info About WinPCL



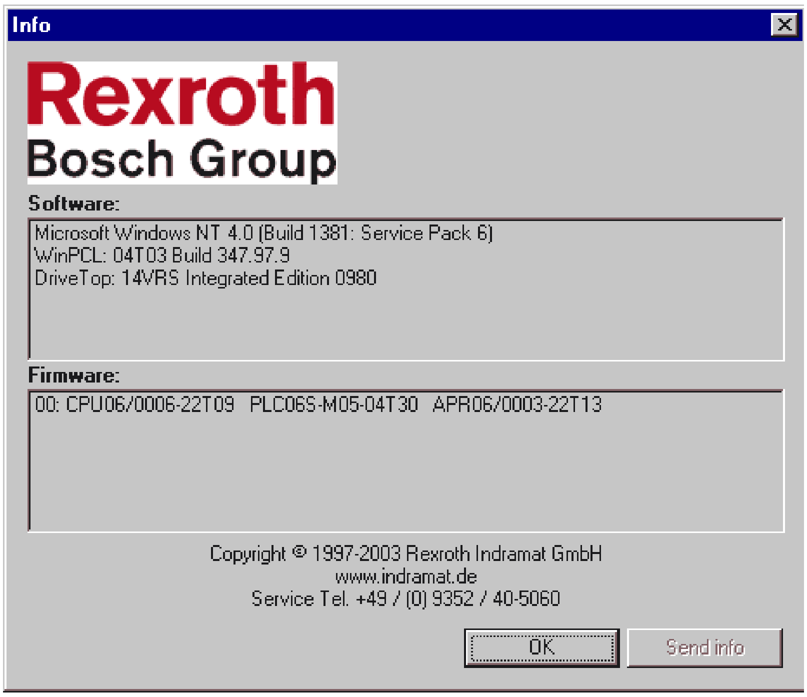
Info zu WinPCL.bmp

Fig. 4-134: Info about WinPCL

This submenu contains detailed information about the version with date and time; this allows a unique identification of the used version.

Information on the operating system of the respective control (there may be several ones) is provided by the "PLC Information" menu item.

Info



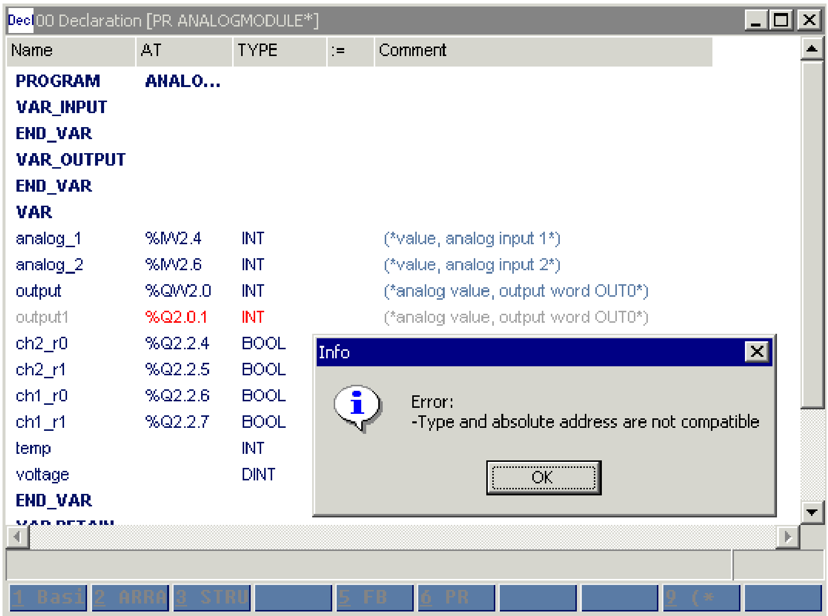
info_mtgui.bmp

Fig. 4-135: Info about "Multi Task Graphic User Interface (MTGUI)"

This window also contains copyright, Internet address and service phone numbers.

Help on a Particular Error <Ctrl>+<F1>

If one of the editors shows an error by changing the color from blue to red or gray - invalid, a plain text help can be called up by pressing the <Ctrl>+<F1> keys. The cursor has to be placed on the respective line first:



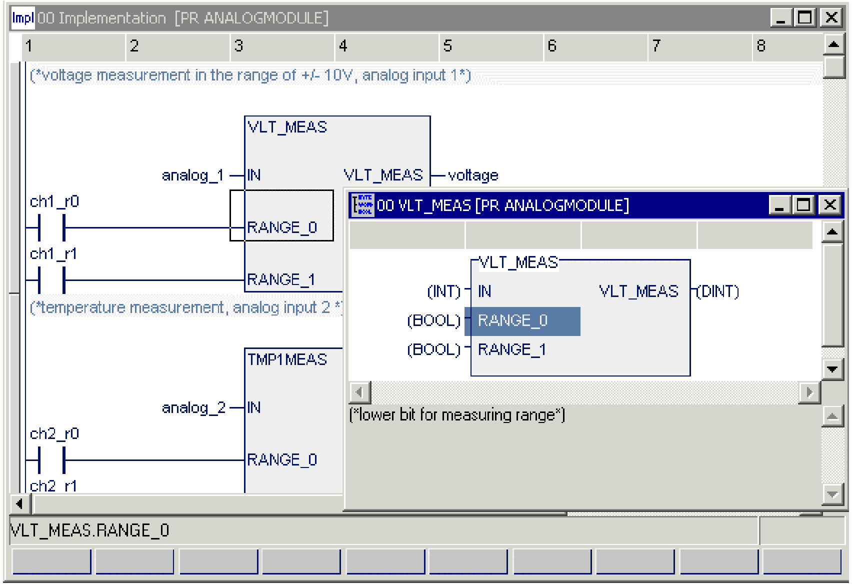
Fehlerhilfe.bmp

Fig. 4-136: Example of "Help on a particular error <Ctrl>+<F1>"

Help on Declaration <Shift>+<F1>

If provided, the help on declaration <Shift>+<F1> can be used to call up declaration information on the item where the cursor is positioned.

Example: the cursor is positioned on a function block type in declaration.



dekl_hilfe.bmp

Fig. 4-137: Example of "Help on declaration <Shift>+<F1>"

Rexroth
Bosch Group

4.10 Miscellaneous

Language Conversion

The language can be set in the HMI main menu via Start_Setup\Language Selection.

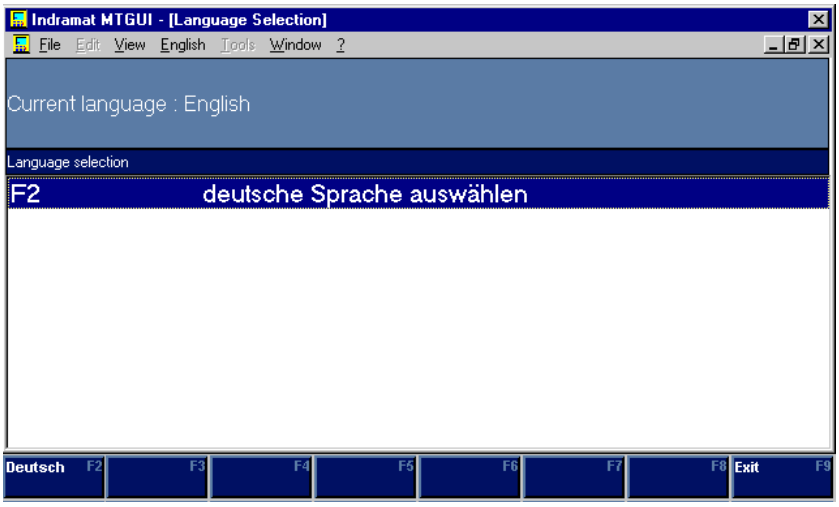


Fig.. 4-138: Language selection

The current language is altered by changing a parameter in the "**Language.ini"** file.

Note: Before changing the parameter, the Rexroth GUI must be exited. The changes made will only be effective when the GUI is restarted.

The **"Language.ini"** file resides in the installation path of the Rexroth GUI in the "..\MTGUI\Config" (\WINPCL\Config) directory. The file can be edited directly with a unicode-compatible text editor (e.g. Notepad).

The "ActLanguage = **XX**" parameter in the "General" section must be changed, with XX being the appropriate language identification.

Excerpt from the Language.ini file:

```
[General]
ActLanguage=DE
DefaultLanguage=EN
```

Language identifications comply with those specified in DIN / INN.

Excerpt from the DIN / INN language identification:

Language	Language identification
German	DE
English	EN

Fig. 4-139: Language identification according to DIN / INN

Remote Programming

Purpose

If decentralized control systems are used, the machines concerned, such as transfer machines, machining centers, etc., must be equipped with several PCs. One or several units (controls, such as MTC200, ISP200, etc.) may be assigned to each PC. An Ethernet network is used to establish the interconnected system.

If, e.g. for service or startup purposes, a notebook is incorporated in this network, the PLC component of one of these controls can be remote-controlled from the notebook.

Note: During installation, a dummy user is created initially, which should be replaced by a complete installation.

The remote-controlled control itself can be operated in a different software component (main menu, HMI, parameters, or the like), but not in WinPCL as an alternative to the notebook.

Activation requires Activities on the Control Side (Server) and Activities on the Notebook Side (Client).

Activities on the Control Side (Server)

Activation on the control side is effected on the basis of the Setup Menu item of the main menu of the HMI GUI.

The respective user must have logged in as administrator in his system.

The AddON teleservice must be installed ("Start Setup / Start Teleservice Setup").

The RAS server (WinNT component) must be installed.

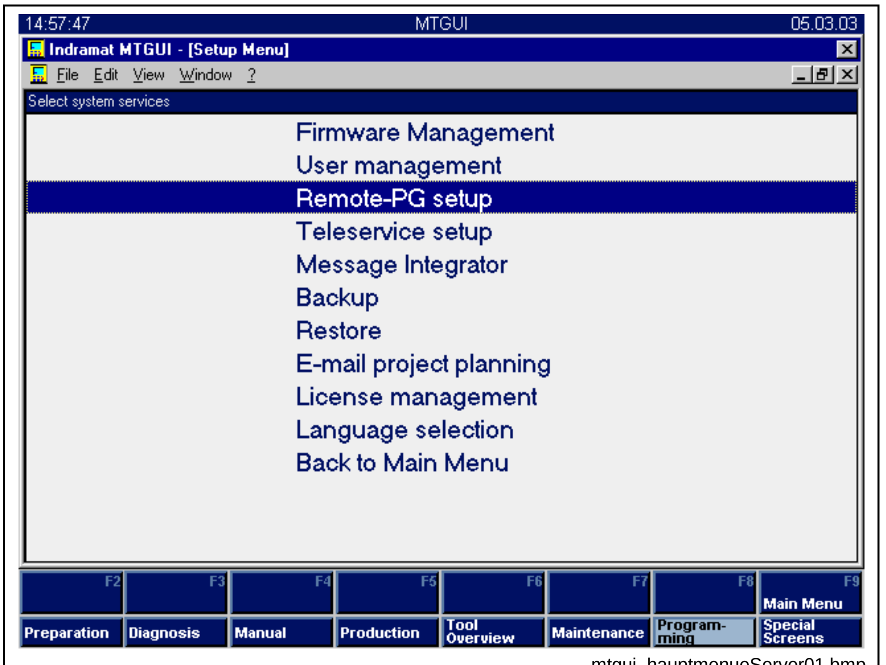
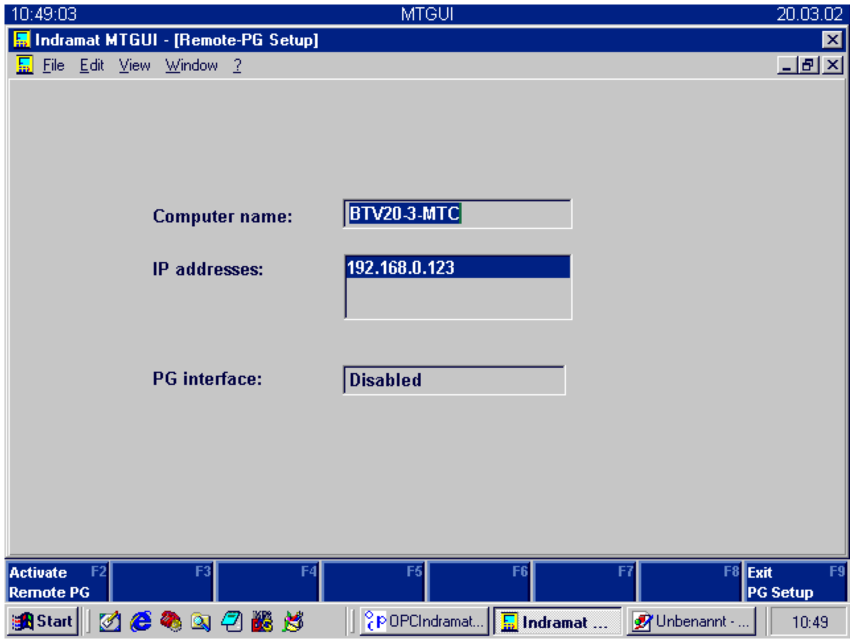


Fig. 4-140: Setup menu of an HMI GUI (WinMTC or WinISP)

The menu item selected serves

- to define the computer name and the IP address of the control,
- to activate / deactivate the remote control (<F2>),
- to display the state of the PG interface.

The information on "computer name" (and "IP address") must be entered in the client-PC (notebook) to be used for remote programming.



mtgui_hauptmenueServer02.bmp

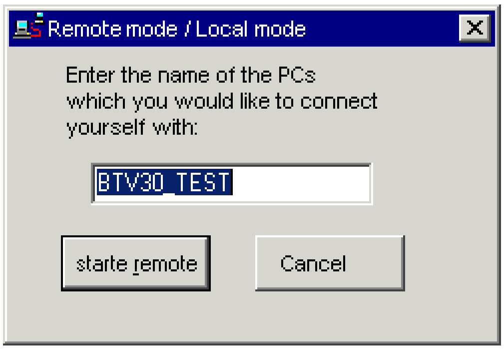
Fig. 4-141: Defining the computer name and the IP address

If the remote programming mode is activated, the user's work in his own WinPCL of the control is disabled. All other activities may be carried out with the remote programming mode activated on the control.

Activities on the Notebook Side (Client)

Activation of the remote programming mode requires an appropriate WinPCL from an ISPPG installation on the client side.

Calling the ISPPG opens the following dialog:



remote_auswahl01.bmp

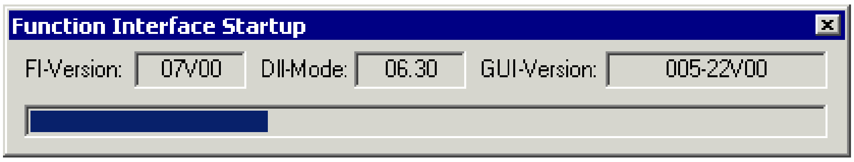
Fig. 4-142: Dialog for entering the desired server name

After the server name (computer name of the control in the network, see Activities on the Control Side (Server)) has been entered, the remote connection to the desired server can be activated.

Note: At that time, the server must already be running with an activated remote interface, otherwise an error message will be emitted.

Version check in case of remote programming

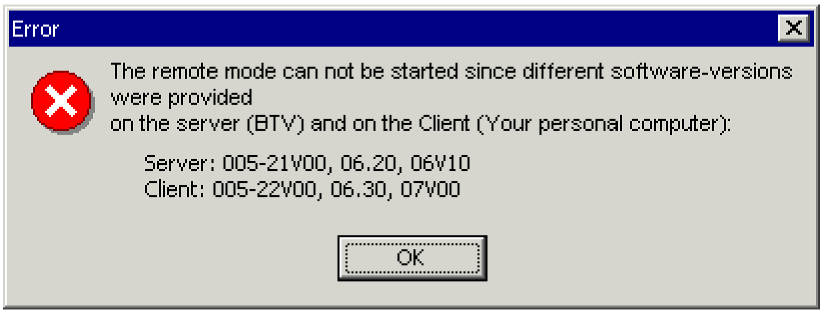
The remote connection requires GUI and function interface versions which are compatible with each other.



FI_Server_Hochlauf_01.bmp

Fig. 4-143: Function interface startup with display of test criteria

Thereafter, the password (Password) is requested, including a check of the rights required for remote operation. If the version check is completed successfully, WinPCL is started; if not, the startup procedure is stopped with display of an error message.



Versionsfehler_Remote.bmp

Fig. 4-144: Error message in case of an unsuccessful version check

Remote Programming Rules

Remote programming requires the following:

- WinPCL, processing the sources (RE/PR/FB/FN…) of the server (your BTV) and saving same on that server, is running on the client (your notebook).
- The client causes download of the compilation products to the PLC of the server.

Archives

The situation described affects the archive of files and compounds.

When selecting the archive destination (Archive), the programmer decides whether the archive resides

- in the server (your BTV), or
- in the client (your notebook), or
- on a floppy disk in one of your disk drives, or
- at any other location in your network.

Firmware download

A firmware download, if becoming necessary on the server (your BTV), is not possible during remote operation.

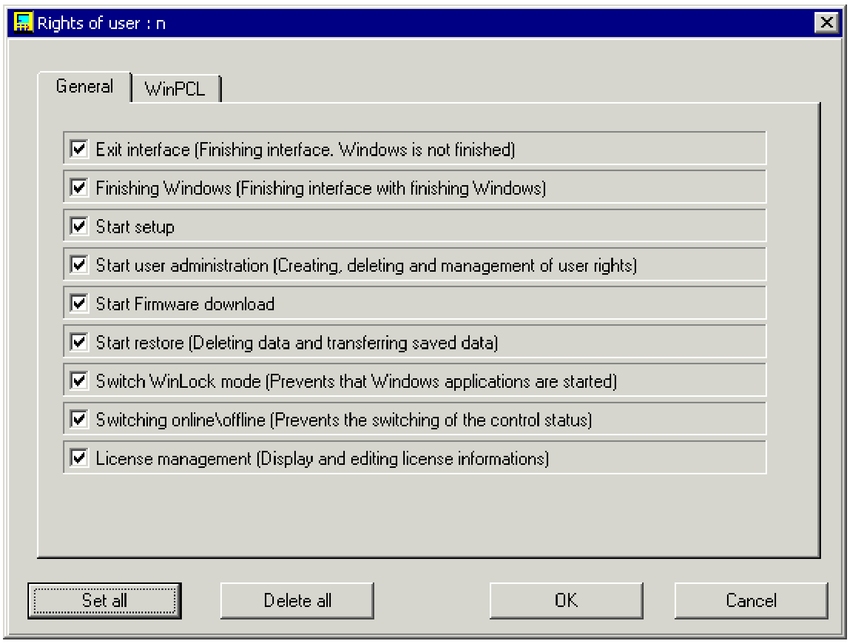
The required firmware must first be filed in an archive on the server (your BTV) and then be transferred to the control using a server tool (Setup Menu / Start Firmware Management).

Password Rights for Remote Operation

Remote operation requires that the user is accordingly enabled in the user management (Setup Menu / Start User Management).

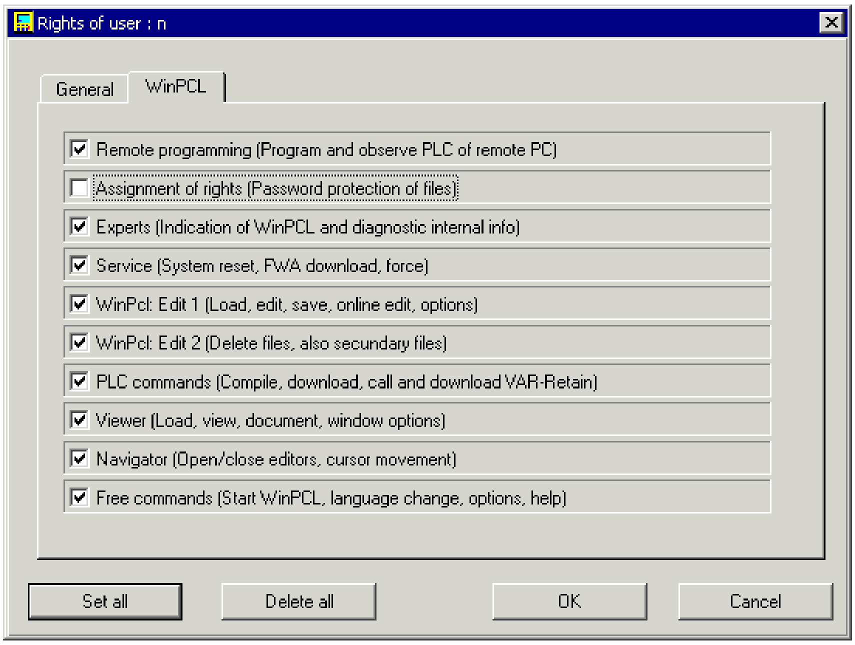
User Management, WinPCL Rights, Remote Programming

The user management provides a dialog for assigning various rights, in addition to the "General" rights to "WinPCL". To view the rights of any user "n", login is only permitted with "supervisor" status (only he may view and alter any assigned rights).



Rechte_allgemein.bmp

Fig. 4-145: General rights of a user "n"



Rechte_WinPCL.bmp

Fig. 4-146: WinPCL rights of user "n"

4.11 Keys and Key Combinations

This sections contains a summary of all key combinations currently used in WinPCL.

<Ins>

Toggling of insert / overwrite.

<Del>

Common delete key, the exact usage is described in the editors.

<Esc>

- Escape key for running commands,
- for closing temporary windows.

<Tab>, <Shift>+<Tab>

Toggling of ladder diagram and instruction list for the current network.

F Keys and Their Alt / Ctrl / Shift Combinations

	Key	<Shift>+key	<Ctrl>+key	<Alt>+key
F1	General help on cursor position	Declaration help, interface of data types, functions or function blocks, display of declaration comment	Calls up of the "Error help"; additional information on color-coded error	
F2		Goes to the implementation of the file, which is opened in the focused editor window	Goes to the import overview, which belongs to the file, that is opened in the focused editor window	Goes to the declaration of the file, which is opened in the focused editor window.
F3		Status display of multi-element variables (array or structure)		Goes to "SFCs" (list of all SFCs and SFC elements)
F4			Closes the focused window	Closes the programming system; prompts whether changed files have to be stored
F5				
F6			Moves to the next window	
F7				
F8		Permits viewing and forcing of variables		
F9			Download of the current resource (Compiler / Selection of the current resource) including the files pertaining to it to the preset control (File / Selection of current control)	
F10		Goes to the local pop-up menu		

Fig. 4-147: List of F key combinations

New keys / key combinations are gray.

Alt-Key Combinations

<Alt>

Goes to the first menu item (file) of the main menu.

<Alt>+<Space>

Opens the system icons.

<Alt>+<C>

Opens the compiler menu item:

- Compilation for file / necessary files / complete compilation starting from the focused file
- Compilation of necessary files / complete compilation starting from the current resource
- Selection of the current resource

<Alt>+<E>

Opens the "Edit" menu item:

- Copy, cut, insert blocks, delete
- Find, find next, replace

<Alt>+<F>

Opens the "File" menu item:

- Create new file, open existing files
- Selection of current control, variant selection for control 00
- Save, save as, save all
- Properties of the focused file
- Print, archive
- Import, export
- Exit

<Alt>+<S>

Opens the "Start" menu item:

- Download "xx" in control "yy"
- Status display on / off
- Reset of the control
- Forcing variables
- Status display: ARRAYs / Structures

<Alt>+<T>

Opens the "Tools" menu item:

- Options
- PLC information
- Event display
- System Error Display
- Miniature control panels
- Diagnosis module assignment
- Password

<Alt>+<V>

Opens the "View" menu item; transition to the individual editors:

- Implementation, declaration
- On resource level: IO editor
- SFC list, step list, transition list, action list
- Cross reference list, import list
- Tree representation for the current system

<Alt>+<W>

Opens the "Window" menu item:

- Close focused window, close all windows; if file contents have been modified, a saving prompt appears.
- Cascade windows, tile windows horizontally, vertically, minimize all windows.
- List of windows

<Alt>+Number

Goes to the respective command in the footer.

<Alt>+<Enter>

Opens a selection window instead of entering a name. ...

<Alt>+<?>

Opens the "Help" menu item:

<Alt>+<TAB>

Toggling the Windows applications (to the right).

<Shift>+<Alt>+<TAB>

Toggling the Windows applications (to the left).

Ctrl-Key Combinations

<Ctrl>+<C>

Block command, copies the block selected to the Windows clipboard.

<Ctrl>+<V>

Calls up the project navigator.

<Ctrl>+<F>

Finds for a character string

<Ctrl>+<E>

Enters or modifies the implementation comment on a particular LD element, with the cursor positioned on that element in the ladder diagram.

<Ctrl>+<H>

Replaces a character string by another string.

<Ctrl>+<P>

Prints the editor contents.

<Ctrl>+<R>

Repeats the last search, the last replacement.

<Ctrl>+<S>

Saves the focused file, the file time is the time of the last modification.

<Ctrl>+<V>

Block command, pastes a block from the Windows clipboard.

<Ctrl>+<X>

Block command, cuts the block selected and stores it to the Windows clipboard.

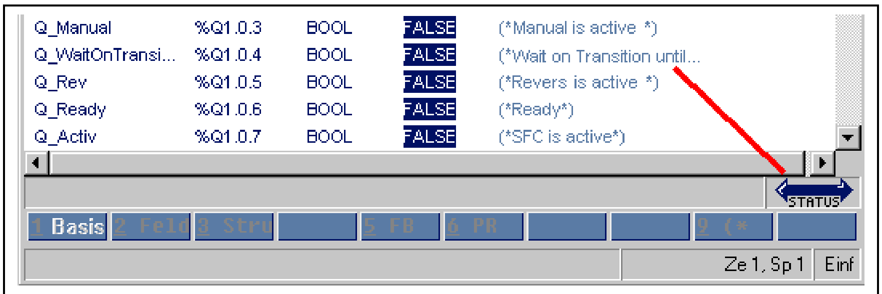
<Ctrl>+<Enter>

Branches to a function block, a function, an SFC

4.12 Pictograms

Essential information is provided in the form of pictograms which are constantly kept on a current level.

Operating modes:



Pikto_01.bmp

Fig. 4-148: Operating modes

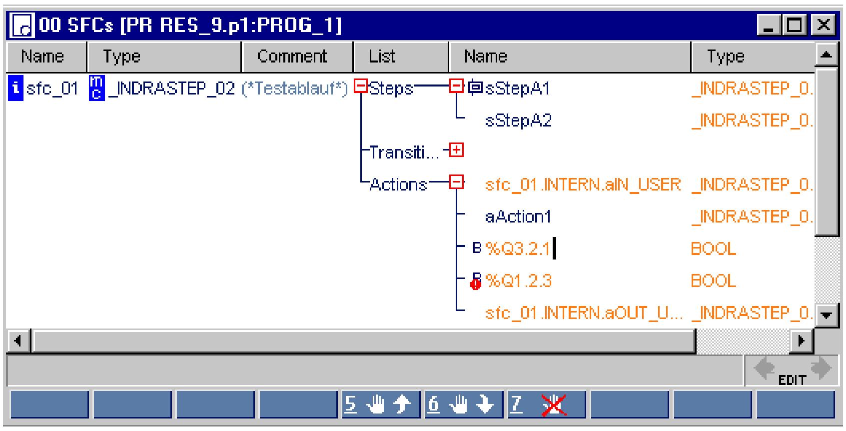
The following operating modes are possible:

Pictogram	Status	Comment
	No	The file is in the Edit mode; online edit limit is already exceeded; variables and SFCs are initialized after the download.
	Yes	The file complies with the code running in the control; the status of the variable is displayed.
	In part	Single networks have been changed; their status cannot be displayed; the other ones comply with the code in the control, their status is available.
	No	Edit mode; transmission to control disturbed.
	No	Online mode; transmission to control disturbed.
	No	Status; transmission to control disturbed.
	No	Edit mode; branching path (from the resource to the current instance) unknown, or file loaded directly. The file belongs to the current compound.
	No	Single networks have been changed. Branching path (from the resource to the current instance) unknown, or file loaded directly. The file belongs to the current compound.
	No	The file complies with the code running in the control. Branching path (from the resource to the current instance) unknown, or file loaded directly. The file belongs to the current compound.
	No	The file does not belong to the current compound.

Fig. 4-149: Operating modes - indication in relation to the particular editor

Properties:

The pictograms listed below are used in SFCs (list of SFCs / steps / transitions / actions):



pikto_02.bmp

Fig. 4-150: SFCs with pictograms

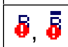
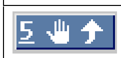
Pictogram	Comment
	Initial step of an SFC
	IndraStep operating modes (mode control)
	Network and SFC properties are used (see ProVi Messages (Diagnosis in LD / IL Networks) and SFC diagnosis)
B, $\bar{B}$	Boolean and negated Boolean actions
	Invalide absolut addressed Boolean action, Invalide negated absolut addressed Boolean action, (Declaration of the Boolean variable has been deleted.)
	Arrange action processing order manually, move current action upwards
	Arrange action processing order manually, move current action downwards
	Action processing order rearranged manually, so that the order does not correspond to the SFC graphic any longer, Reset action processing order

Fig. 4-151: Properties - indication in networks and SFCs

Program organization units and data types:

The pictograms listed below are used to differentiate between program organization units and data types.

The first column lists the symbols for the particular user POU / type and the symbol for a standard or firmware POU / type.

The color-coding in the second column indicates that the particular file fails to comply with the current WinPCL version or is defective in any other way.

Pictogram of POU / type	Pictogram of defective POU / type	Comment
RE, re	RE	Resources (RESOURCE)
PR, pr	PR	Programs (PROGRAM)
FB, fb	FB	Function blocks (FUNCTION BLOCK)
FN, fn	FN	Functions (FUNCTION)
AR, ar	AR	Arrays (ARRAY)
ST, st	ST	Structures (STRUCT)
TY, ty	TY	Type - general files

Fig. 4-152: POUs and data types

5 Declaration Editors

5.1 General Notes on the Declaration Editors

The programming system allows the usage of the following program organization units (POU) in compliance with EN 61131-3:

- Resource, Declaration, Resource
- Program, Declaration, Program
- Function block, Declaration, Function Block
- Function, Declaration, Function

and

- Agreement and utilization of data types, i.e. the Declaration of Structures (STRUCT) and the Declaration of ARRAYs.

Dependent on the different performance of the program organization units, the declaration editor has to fulfill different requirements with regard to declaration of

- variables,

  that means the name of the variable, the data type, if necessary an initial value specified by the user, a comment pertaining to the name and - on program and resource level - the address, where the variable is to be found or starts in the storage of the control.

In addition, the following instances must be declared within resources, programs and function blocks:

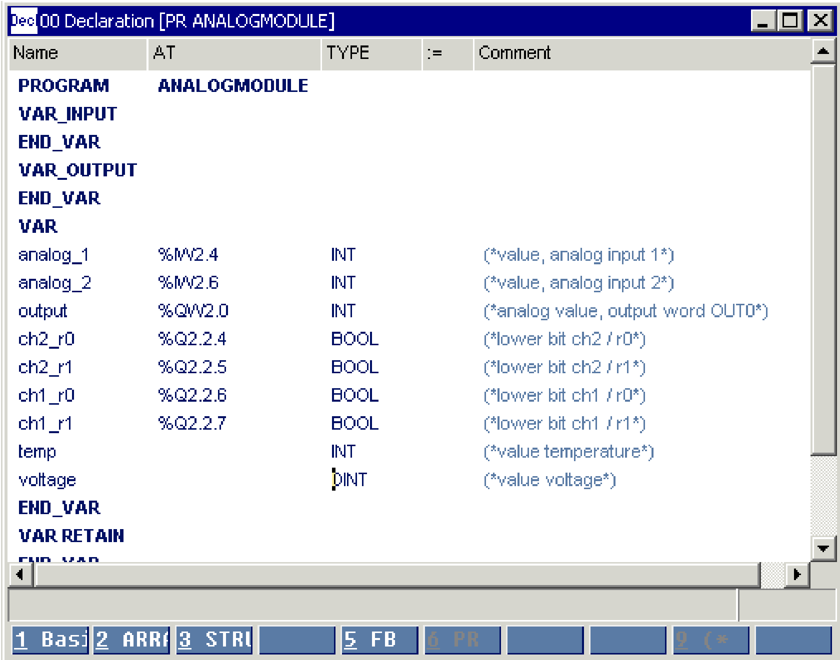
- instances of function blocks (within programs and function blocks) and
- instances of programs (within resources),

that means, the storage space required for archiving the data set of the respective instance must be reserved.

In addition to other functions, all declaration tools allow the access to options, provide the possibility of displaying the status while the program is running, and have common editing features. This effects a change of the font color in case of a faulty entry and assists the user actively with online help.

5.2 Structure of the Declaration Part

The declaration part serves for declaring variables and function block assignments before they are used in the implementation.



Deklaration_01.bmp

Fig. 5-1: Declaration part of a program

The declaration editor is structured in columns. The contents of the individual columns is stated in the gray header of the declaration editor.

1. Column: Name
Here are the names of the declared variables or the names of the function blocks applied for usage (assignment names). They may not exceed a length of 9 characters and must start with a letter.

2. Column: AT
The column is only used for programs and resources and contains the absolute address which is assigned to the name in the first column.
For single-element variables, the variable is below the given address, for multi-element variables (structures, arrays, character strings) the variable starts at the given location.

3. Column: Type
Enter the type for variables and the function block type for function blocks.
If the address clearly identifies the variable type (Boolean inputs and outputs, BYTE, WORD, DWORD), it is not necessary to expressly specify the type. But it may be specified, if desired.

4. Column: :=
This column is provided for default values specified by the user. This value must be compatible with the variable type. Moreover, this column is used to display the status of elementary variables, when a program is running on the PLC; in this case, the default value is cross-faded (Status Display in the Declaration Editor).

5. Column: Comment
This column is intended for entry of the declaration comment which is allocated to the variable name or to the assignment name.
For a better structure, complete comment lines or empty lines can be inserted, regardless of the actual declarations.

A declaration line must not be absolutely complete at once or must not instantly have the correct syntax. The user is guided to this target by means of variations in the font color (described below), because an executable program requires a correct declaration.

Editing Features, Varying Font Color in the Declaration Editor

Operating mode: Edit

When the entry is made, the font line first appears in white on a blue background. The line is untested at that moment.

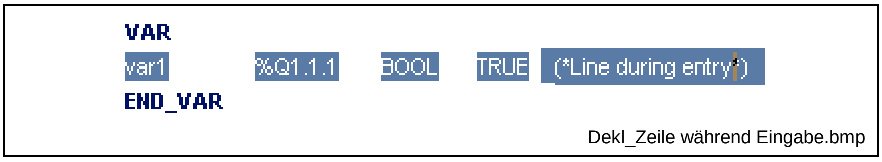


Fig. 5-2: Declaration line during the entry

The line is checked for correctness when exiting the line. The font color changes in case of an error:
If there is no error, the basic color of the font is dark blue, the comment is middle-blue.

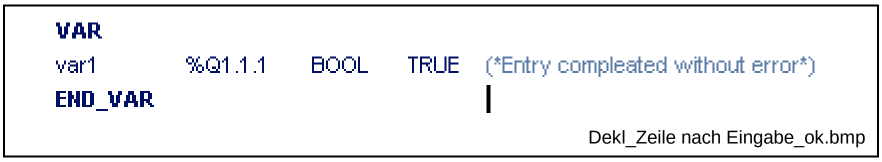


Fig. 5-3: Declaration line after the entry is completed - without errors

Incorrect names, data types, initial values or their combination are shown in red. The remaining text within such an incorrect line is shown in gray. Essential combinations are shown below:

Fig. 5-4: Conflict between absolute address of variable and data type

If the error is not detected directly, position the cursor on the line and press <Ctrl>+<F1> for online help:

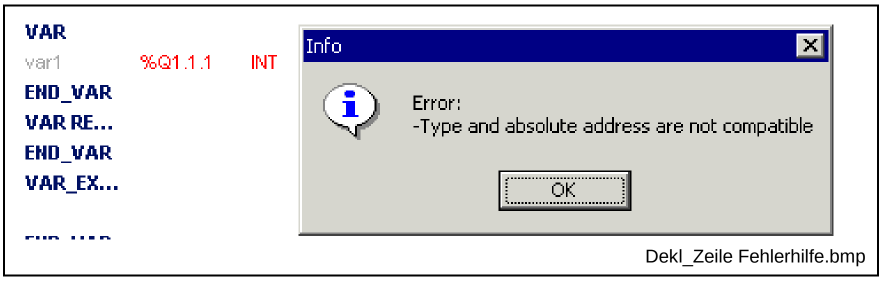
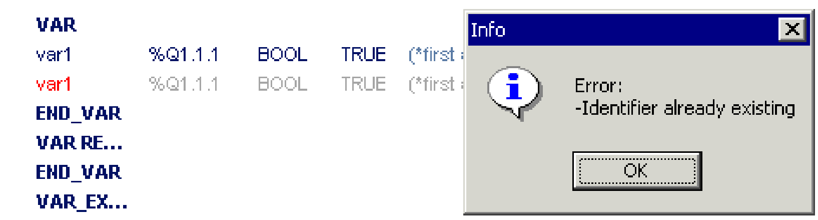


Fig. 5-5: Online help

Rexroth Bosch Group

Multiple use of a name during the entry or after copying of a block results in an error. The first use is accepted, the second and all other uses are marked as an error. Help can be called up by pressing <Ctrl>+<F1>:



Dekl_Zeile Name mehrfach.bmp

Fig. 5-6: Error caused by multiple use of a name

Declaration Footer Commands

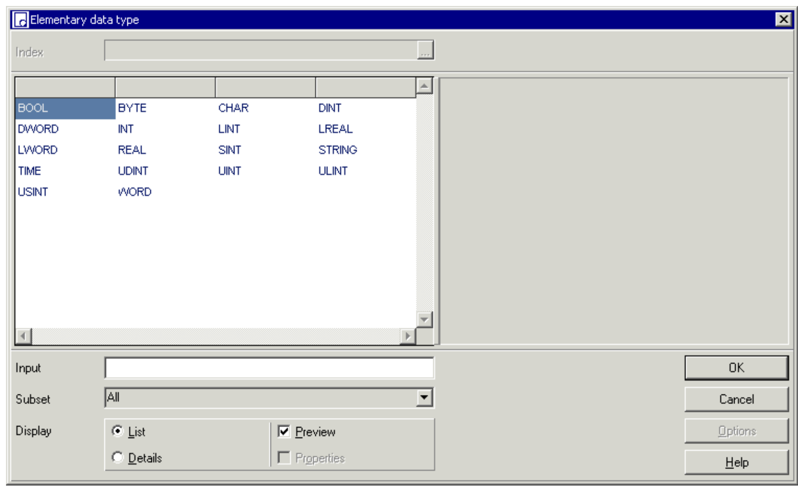
The footer commands support the work in the declaration editors. They can be activated by pressing <ALT>+numeric key.

Keys	Name	Column	Selection window	Used in:
<ALT>+<1>	Basis	TYPE	Selection Window, Elementary Data Types	RE, PR, FB FN, AR, ST
<ALT>+<2>	Array	TYPE	Selection Window, ARRAYs	RE, PR, FB FN, AR, ST
<ALT>+<3>	Structure	TYPE	Selection Window, Structures	RE, PR, FB FN, AR, ST
<ALT>+<5>	FB	TYPE	Selection Window, Function Block Types	PR, FB
<ALT>+<6>	PR	TYPE	Selection Window, Programs	RE

You can open the respective selection windows by pressing the above mentioned key combinations.

Selection Window, Elementary Data Types

The footer command "1-Basis" opens the selection window for elementary data types. They can be represented as list (space saving) or with detail information.



Basis_Auswahl.bmp

Fig. 5-7: Selection window "Basis types"

The name of the desired data type can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

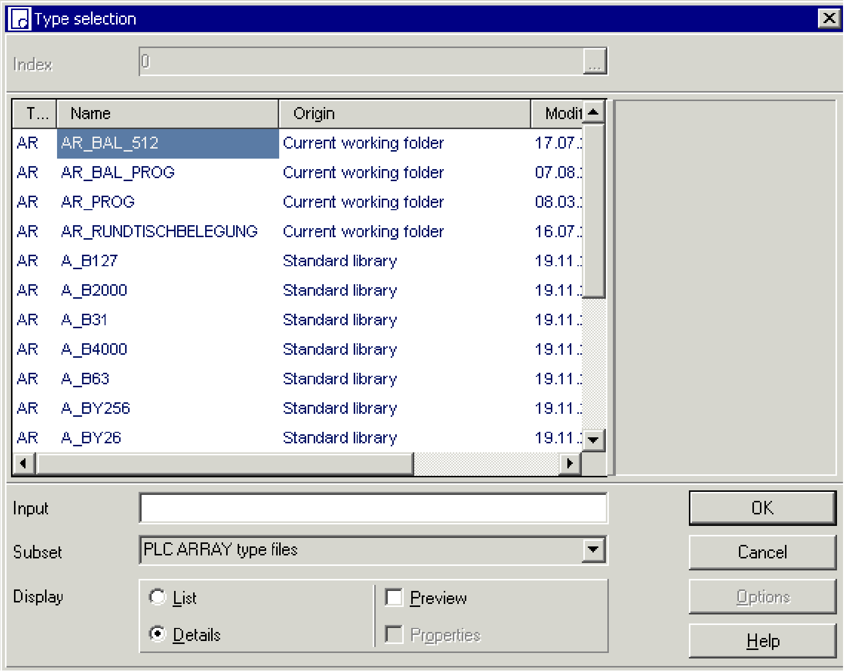
Chapter "Data Types in WinPCL" contains information about the data type contents.

Selection Window, ARRAYs

The footer command "<Alt>+<2>-ARRAY" opens the selection window for the ARRAYs. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the ARRAY (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



Felder.bmp

Fig. 5-8: Selection window, ARRAYs

The name of the desired ARRAY can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

With "Subset" you can also select structures or all data types instead of ARRAYs.

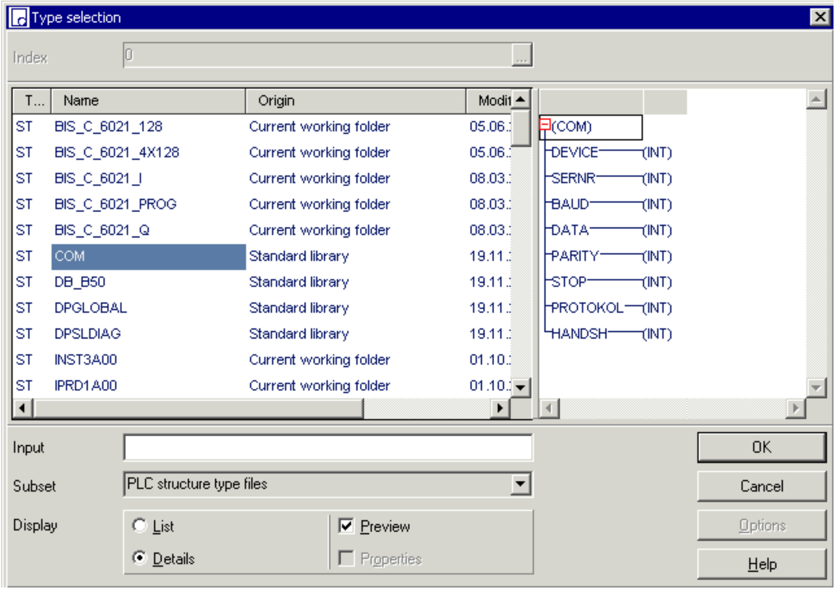
Chapter "Data Types in WinPCL" contains information about the data type contents.

Selection Window, Structures

The footer command "<Alt>+<3>-Structures" opens the selection window for the structures. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the structure (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



Strukturen.bmp

Fig. 5-9: Selection window, structures

The name of the desired structure can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

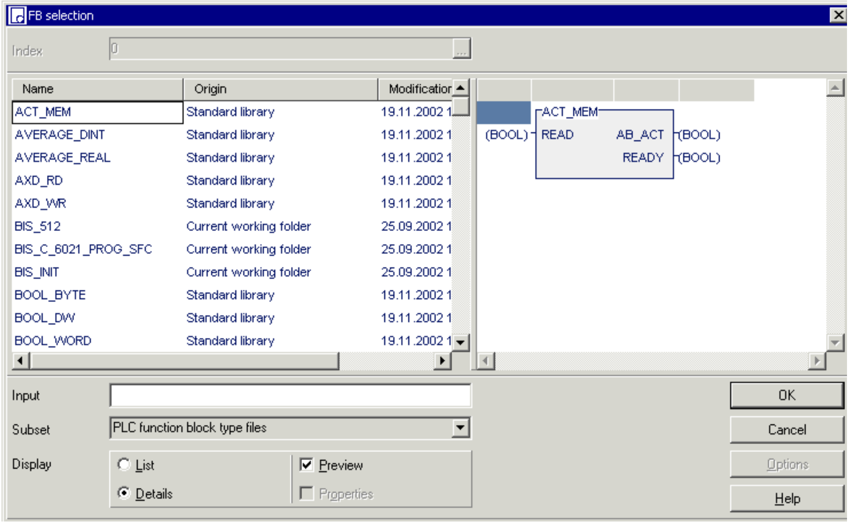
With "Subset" you can also select ARRAYs or all data types instead of structures.

Selection Window, Function Block Types

The footer command "<Alt>+<5>-FB" opens the selection window for the function block types. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the function block type (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



FB_Typ_Auswahl.bmp

Fig. 5-10: Selection window, function block types

The name of the desired block type can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

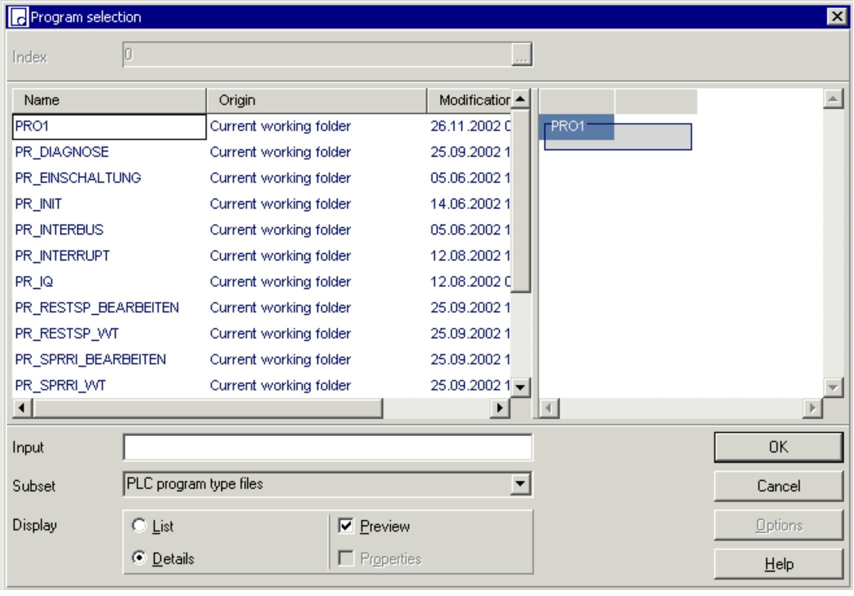
Note: If there are FBs in the library and if they also exist as user function block, it is possible to select one of them. Using the one excludes using the other.

Selection Window, Programs

The footer command "<Alt>+<6>-PR" opens the selection window for the program types. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the program (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



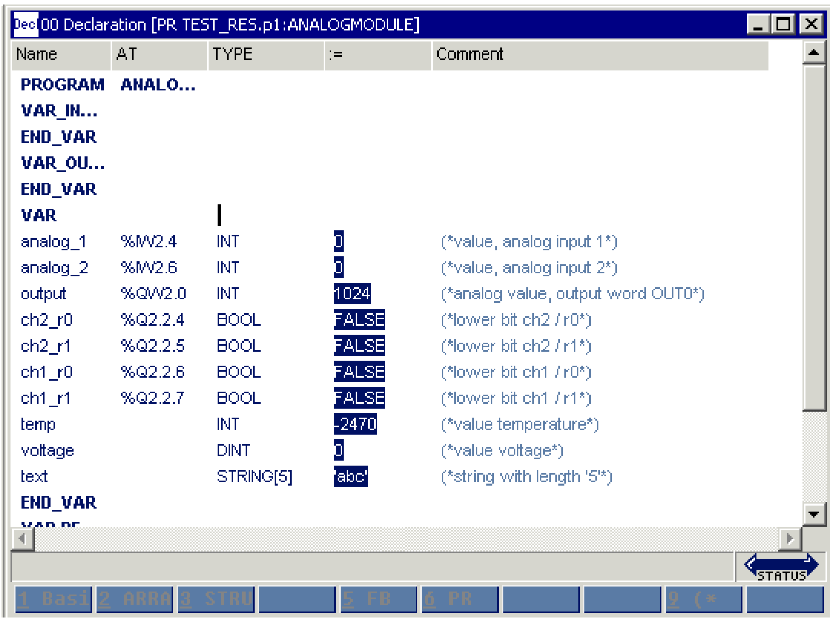
PR_Auswahl.bmp

Fig. 5-11: Selection window, programs

The name of the desired program can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Status Display in the Declaration Editor

Status information is shown for all elementary variables in the declaration editor in the combined column for initial value / status:



Dekl_Zeile Status.bmp

Fig. 5-12: Status in the declaration editor

Status information of function blocks can be called up by a double click or <Ctrl>+<Enter> on the instance.

Further ways to get status information are:

- Start / Force <Shift>+<F8> for elementary variables (ANY_ELEMENTARY)
- Start / Status ARRAYs / Structures <Shift>+<F3>

Declaration Editor Options

The appearance of the declaration editor can be changed using the "Extras / Options" menu item. The following options are available:

Group	Option	Meaning	
Desktop	Restore size and position during startup	The desktop is restored in the same size and position.	
	Restore MDI window during startup	MDI windows are opened in same order when restarting the system	
	Auto save	Allows the automatic saving of the current file in presettable time intervals without any prompt.	
	Sound	Allows the activation or deactivation of a beep sound.	
View / All	Automatic take-over of column width change	Restoring of the column with same width.	
	Apply column width modifications automatically	Comments, that have been entered in the respective declaration line are also indicated in the implementation. The implementation can be changed; the comment is then doubled, the declaration line remains unaffected.	
	Variable display	Is not relevant for the declaration.	
	Display of absolute variables	The user can select from I/Q, E/A and I/O for absolute addresses.	
	Truncating very long texts	Texts and numbers can be truncated to the right or left, and can be represented with or without "..." marking.	
	Truncating very long numbers		
View / DECL	Column width for the individual columns (with default values)	Name	100
		AT	60
		Type	60
		Default value	60
		Comment	230

Pop-up Menu, Declaration Editor <Shift>+<F10>

This pop-up menu contains the essential commands for this editor. It can be called up by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	**Explanation**
Open	Branch, also <Ctrl>+<Enter>
Delete unused identifier	Finding and Deleting Unused Declarations (variables and function block instances)
Sort	Sorting the declaration lines marked in the block by names/AT/TYPEs/default values/comments
Import declaration	The ASCII file selected from the "WINPCL" text files is attached in sections to a possibly existent declaration.
Export declaration	The content of the declaration editor is exported as ASCII file and stored in the folder "WINPCL text files".
Syntax text	List of all errors in the current editor. You can move to the location where the error occurred by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys. Subsequent import of types, which were not available at the time of declaration. (The data type or function block was created after the declaration line has been written. The type is indicated in red; error; the type is searched again in the syntax test. If found, the error is corrected.)
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the interface of the data type or of the function block type of the current line.
Cross reference help	List of all places where the variable is used (Cross Reference List, Declaration Editor). You can move to the place of use by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Force	Allows the entry of a variable name. The value of the variable is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of the program code execution.
Status ARRAYs / Structures	Status display for the elements of an array or a structure. Selection is done through a tree structure till the specific element is reached.
Print current window	Printing of the editor contents(<Ctrl>+<P>)
Options	Presetting of the editor appearance (Declaration Editor Options)
Internals	Search for errors in the programming system, to be used only if approved by the service.

Block Commands, Declaration Editor

Select the text by pressing and holding the SHIFT KEY while using the appropriate arrow key or by pressing and holding the left mouse button while dragging it across the text.

Extending the selection	**Key combination**
One character to the right	<Shift>+ arrow key <to the right>
One character to the left	<Shift>+ arrow key <to the left>
To the end of the line	<Shift>+<End>
To the beginning of the line	<Shift>+<Home>
Down by one line	<Shift>+ arrow key <downward>
Up by one line	<Shift>+ arrow key <upward>
Down by one page	<Shift>+<Page down>
Up by one page	<Shift>+<Page up>

Deletion of text	**Keys**
Deleting the character to the left of the cursor	BACKSPACE KEY
Deleting the character to the right of the cursor	<Del>

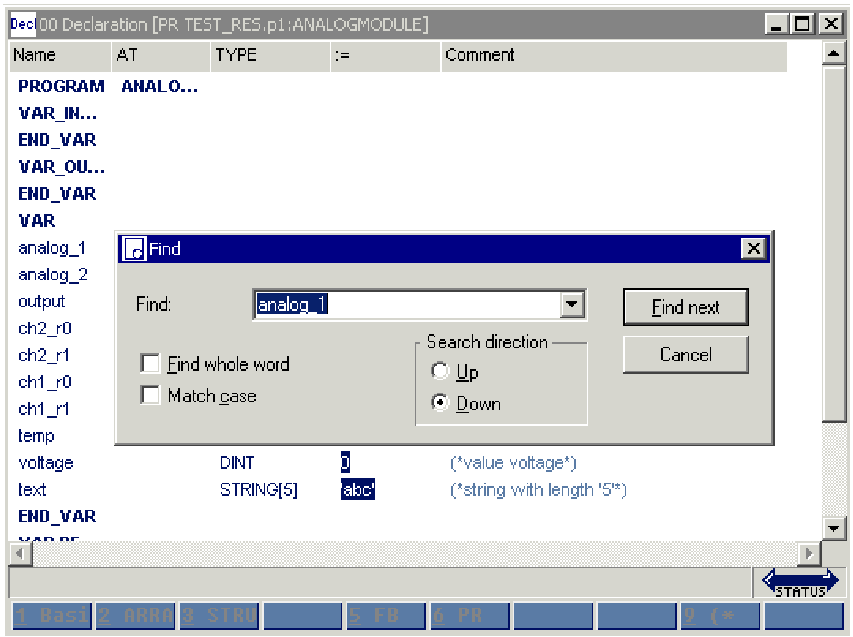
Copying and moving of text	**Key combination**
Copying the text selected to the clipboard	<Ctrl>+<C>
Moving the text selected to the clipboard	<Ctrl>+<X>
Pasting the contents from the clipboard	<Ctrl>+<V>

Note: When using the copy and paste function, the names of the variables or instances are doubled. This is a faulty condition. Result: Varying font color.

Note: When using the cut function, variables or instances are missing, at least temporarily. This can result in a faulty condition in the implementation. Result: Varying font color.

Search and Replace, Declaration Editor

This function is in the first version and provides the features of a text editor:

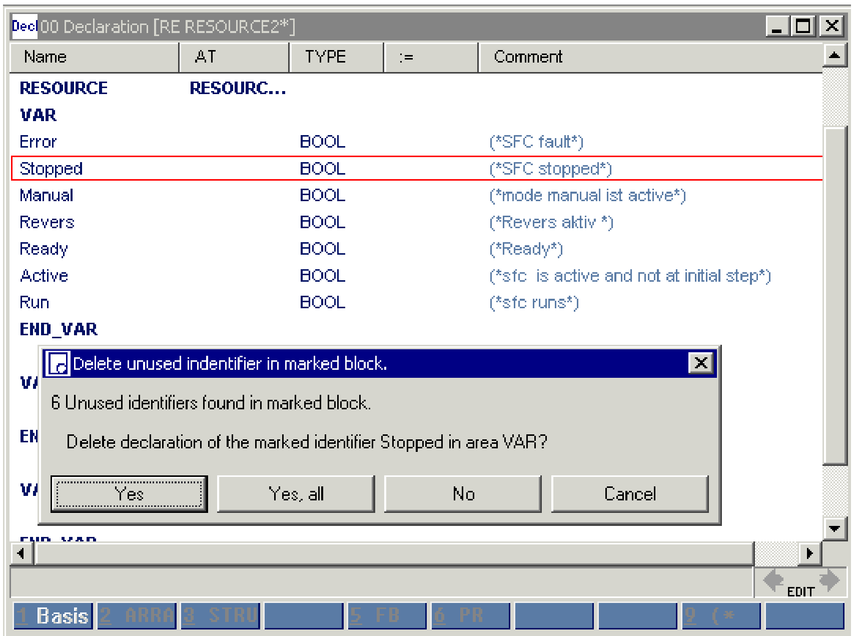


Dekl_Suchen.bmp

Fig. 5-13: Find function in the declaration editor

Finding and Deleting Unused Declarations

Using the pop-up menu (<Shift>+<F10> or right mouse button) in the declaration editor, unused variables and function block instances can be found and then deleted.



Decl_ungenutzt.bmp

Fig. 5-14: Finding and deleting unused declarations

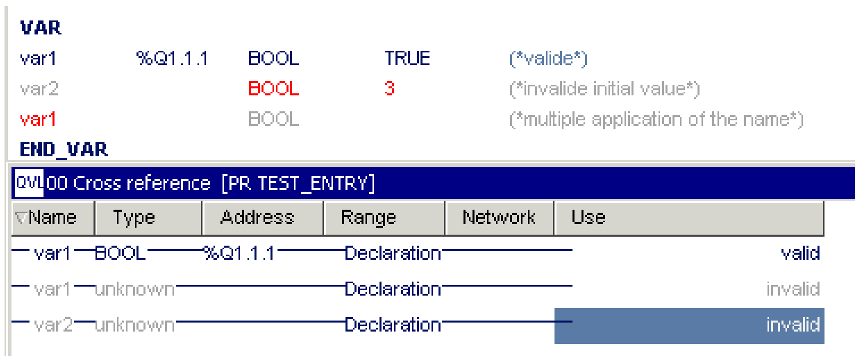
Here, the (current) total number of unused declarations is defined. Any declaration in a red frame is offered for deletion. As an alternative, all unused declarations can be deleted.

Note: The "Yes, all" button refers to the following unused declarations. Declarations already having been confirmed with "No" will remain preserved.

The following areas are searched: "VAR...END_VAR", "VAR RETAIN...END_VAR" and "VAR EXTERNAL...END_VAR".

Cross Reference List, Declaration Editor

Contrary to the cross references of the pop-up menu, the overview activated via "View / cross reference list" lists all of the variables. Of course, only variables from lines with the correct syntax can be resolved by their place of use. However, all faulty names or names with double declaration are displayed and can, thus, be reached with by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.

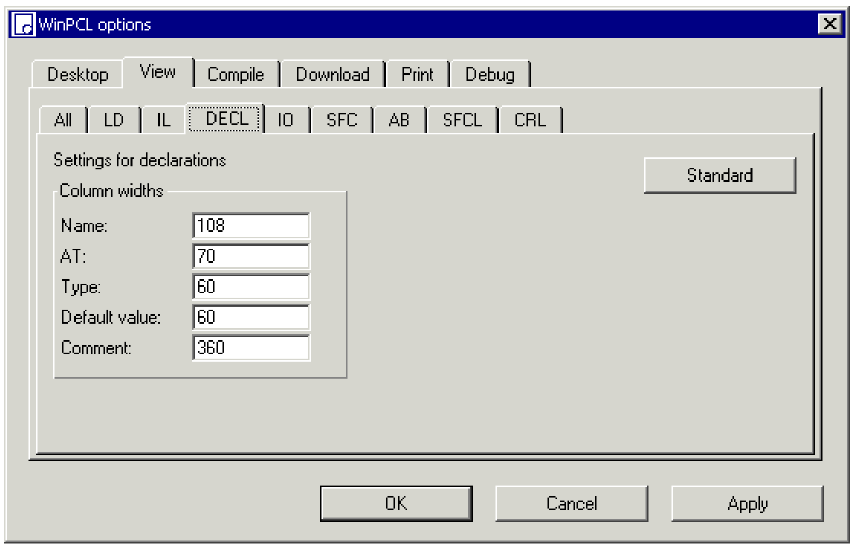


Dekl_Zeile Querverweisliste.bmp

Fig. 5-15: List of cross references to the declaration

Documentation, Declaration

The documentation (printed from the editor, <Ctrl>+<P>) is created using the column widths specified under Extras \ Declaration Editor Options \ View \ Declaration.



optionen_ansicht_dekl.bmp

Fig. 5-16: Options, declaration

The "Apply" button activates the column widths set for the declaration editor. The width of the column can be entered either in the window shown above or preset in the editor by dragging the headers.

The "Standard" button resets the default values.

The "OK" button applies the setting and closes the dialog window.

The "Cancel" button closes the window; the previous values are kept.

Detailed information on the real print process and the features is to be found in the main chapter on WinPCL.

5.3 Declaration, Resource

The declaration editor on resource level serves for

- declaring variables, if necessary with retain properties; the variables can be linked to an absolute address;
- releasing variables as global variables, which can be used in the program instances and their FB instances which belong to the resource;
- defining tasks;
- declaring program instances, which have to run within the resource under management of the tasks defined above.

Note: The name of a resource may not exceed a length of 32 characters.

If this length is exceeded, excess characters may be cut off outside of WinPCL.

Areas in the Declaration Editor (Resource)

Area	Comment
Declaration comment	The lines between resource nn and VAR are intended for comments on the resource for defining the intended use, changes etc.
VAR ... END_VAR	Variables with standard properties, bound to an absolute address, if necessary
VAR RETAIN ... END_VAR	Variables with retain properties, bound to an absolute address, if necessary
VAR_GLOBAL ... END_VAR	Variables released for global use in the complete resource
TASK	Definition of the necessary tasks by specifying
Cyclic task	• Name, enable (release), priority, comment
Time-controlled task	• Name, enable (release), priority, interval, comment
Edge-controlled task	• (Not enabled yet)
PROGRAM	Definition of program instances and assignment to the tasks
	• Name of the instance, with (task assignment), type (program type), comment

Structure of the Declaration Lines

VAR END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Absolute address, if necessary: %I, %Q, %M	Data type	Initial value , not for %I-variable	Comment

Variables with %I-binding must not be overwritten.

VAR RETAIN ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Absolute address, if necessary: %R	Data type	Initial value	Comment

VAR_GLOBAL ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable which is declared in VAR or VAR RETAIN	Echo of the entries specified at the original place, not changeable			

TASK

Name	Enable	Priority	Interval	Comment
Name of the task	Enable (variable or TRUE)	Lowest: 65535	Empty	Cyclic task
Name of the task	Enable (variable or TRUE)	Highest: 0 Lowest: 65534	Constant (TIME)	Cyclic task, restart according to interval

Limitations:

- No more than 8 tasks are permitted, with at least one cyclic task being necessary.
- The highest priority of a task is zero, the lowest one 65 535.

PROGRAM

Name	With	Type		Comment
Name of the instance	Task name	Program type		

Limitations:

The total number of programs is limited to 120.

Declaration Footer Commands, Resource Level

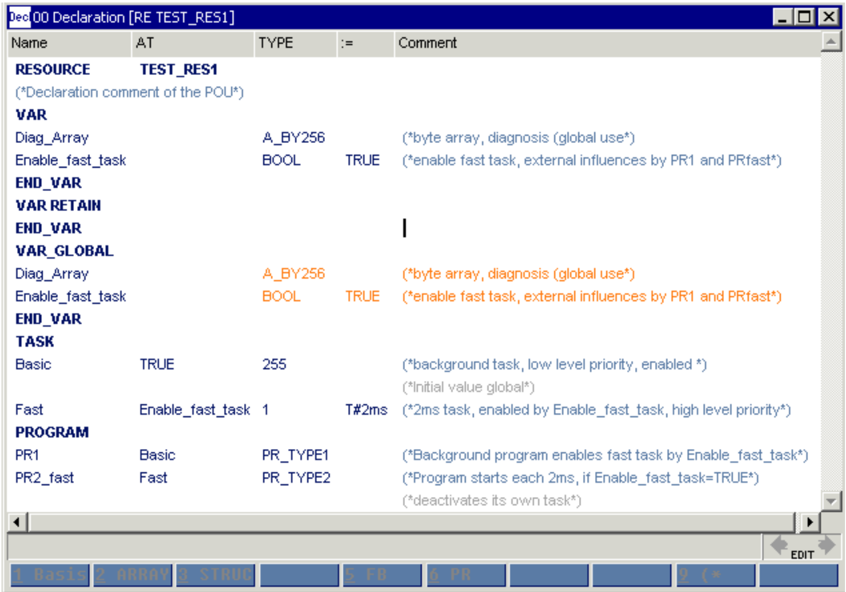
can be activated with <ALT>+numeric key.

Keys	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of ARRAYs
<ALT>+<3>	Structure	TYPE	Selection of structures
<ALT>+<6>	PR	TYPE	Selection of program types

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F1>	all	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help about the type

Structure of the Declaration Part of a Resource (Example)



Dekl_Ressource.bmp

Fig. 5-17: Declaration part of a resource

Rexroth Bosch Group

Comments on the entries:

Area	Entry	Comment
		Declaration comment
VAR	Diag_Array	Array, declared on resource level
	FastEnable	Boolean variable with initial value TRUE defined on resource level
VAR_GLOBAL	Diag_Array	Globally enabled, can be used in all PR and FB via VAR-external access
	FastEnable	Globally enabled, can be used in all PR and FB via VAR-external access, here especially for enabling the 2-ms tasks
TASK	Basic	No interval, runs in cycles, lowest priority, background task, always enabled
	Fast	2-ms interval, is started every 2 ms, if enabled with FastEnable , high priority, interrupts basic task
PROGAM	PR1	Background program, can activate and deactivate the Fast Task via VAR-external FastEnable
	PR2_fast	Program started every 2 ms, can deactivate the Fast Task via VAR_External FastEnable, but not reactivate it!

Fig. 5-18: Comments on figure 2-11

5.4 Declaration, Program

On program level, the declaration editor serves

- for declaring input and output variables of the program (not enabled),
- for declaring variables, which can be used within the program; the variables can be bound to absolute addresses and can have additional retain properties,
- for declaring pointers for internal use,
- for declaring function block instances, which can be immediately used in the program; the instances can have additional retain properties,
- for declaring external variables which were enabled as global variables on resource level and are to be used in the program.

Areas in the Declaration Editor (Program)

Area	Properties	Comment
Declaration comment	The lines between PROGRAM nn and VAR_INPUT are intended for comments on the program for defining the intended use, changes etc.	
VAR_INPUT ... END_VAR (not enabled)	External supply no writing	Variables, absolute address binding forbidden
VAR_OUTPUT ... END_VAR (not enabled)	Delivers information to the outside	Variables, absolute address binding forbidden
VAR END_VAR	Standard	Function block instances - standard, Variables - standard, absolute address binding possible Pointer
VAR RETAIN ... END_VAR	Retain properties	Function block instances - with retain properties; variables with retain properties, absolute address binding possible
VAR_EXTERNAL ... END_VAR	On resource level defined as global with standard or retain properties	Variables, absolute address binding on resource level possible

Structure of the Declaration Lines

VAR_INPUT ... END_VAR (disabled)

Name	AT	TYPE	:=	Comment
Name of a variable (not enabled)	Disabled	Data type	Initial value possible	Comment

VAR_OUTPUT ... END_VAR (disabled)

Name	AT	TYPE	:=	Comment
Name of a variable (not enabled)	Disabled	Data type	Initial value possible	Comment

VAR END_VAR

Name	AT	TYPE	:=	Comment
Name of a function block instance	Disabled	Function block type	Disabled	Comment
Name of a variable	Absolute address, if necessary: %I, %Q, %M	Data type	Initial value , not for %I-variable	Comment
Name of the pointer	Disabled	Data type	Disabled	Comment

Variables with %I-binding must not be overwritten.

VAR RETAIN ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a function block instance	Disabled	Function block type	Disabled	Comment
Name of a variable	Absolute address, if necessary: %R	Data type	Initial value	Comment

VAR_EXTERNAL ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Disabled	Comment

The name and type of the external variable must comply with a global variable at the time when the currently compiled components are connected. Any possible retain properties or binding to an absolute address, with % write protection, are applied from the original declaration to the resource.

Declaration Footer Commands, Program Level

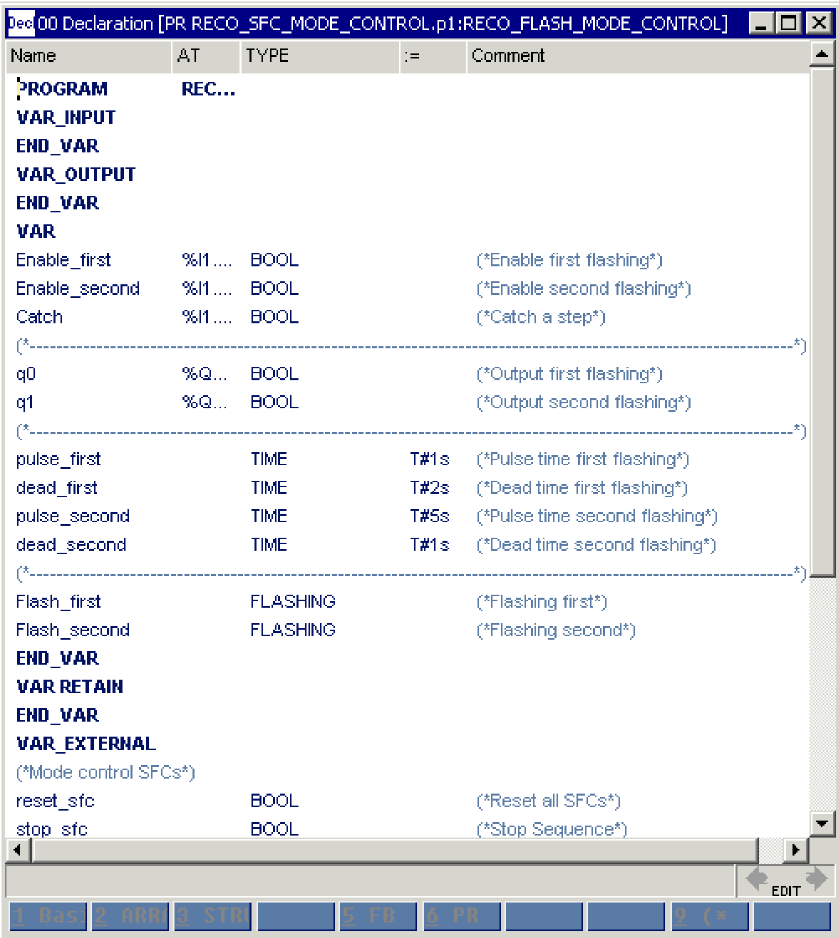
can be activated with <ALT>+numeric key.

Key combination	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of fields / ARRAYs
<ALT>+<3>	Structure	TYPE	Selection of structures / STRUCTs
<ALT>+<5>	FB	TYPE	Selection of function block types

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F1>	All	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help on the type

Structure of the Declaration Part of a Program (Example)



Dekl_Programm.bmp

Fig. 5-19: Declaration part of a program

5.5 Declaration, Function Block

On function block level, the declaration editor serves

- for declaring the input and output variables of the function block,
- for declaring variables, which can be used within the function block; the variables can have additional retain properties,
- for declaring function block instances, which can be immediately used in the function block; the instances can have additional retain properties,
- for declaring pointers to be applied from outside and for internal use,
- for declaring external variables which were enabled as global variables on resource level and are to be used in the function block.

Areas in the Declaration Editor (Function Block)

Area	Properties	Comment
Declaration comment	The lines between function block nn and VAR_INPUT are intended for comments on the function block for defining the intended use, changes etc.	
VAR_INPUT ... END_VAR	External supply, no writing	Variables, no absolute address binding possible, Pointer
VAR_OUTPUT ... END_VAR	Delivers information to the outside	Variables, no absolute address binding possible
VAR END_VAR	Standard	Function block instances, Variables Pointer
VAR RETAIN ... END_VAR	Retain properties	Function block instances, Variables
VAR_EXTERNAL ... END_VAR	On resource level defined as global with standard or retain properties	Variables, absolute address binding and initial value definition on resource level possible

Structure of the Declaration Lines

VAR_INPUT ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Initial value possible	Comment
Name of a pointer	Disabled	Data type	Disabled	Comment

VAR_OUTPUT ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Initial value possible	Comment

VAR END_VAR

Name	AT	TYPE	:=	Comment
Name of a function block instance	Disabled	Function block type	Disabled	Comment
Name of a variable	Disabled	Data type	Initial value possible	Comment
Name of a pointer	Disabled	Data type	Disabled	Comment

VAR RETAIN ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a function block instance	Disabled	Function block type	Disabled	Comment
Name of a variable	Disabled	Data type	Initial value possible	Comment

VAR_EXTERNAL ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type, elementary or defined by the user	Disabled	Comment

The name and type of the external variable must comply with a global variable at the time when the currently compiled components are connected. Any possible retain properties or binding to an absolute address, with % write protection, are applied from the original declaration to the resource.

Declaration Footer Commands, Function Block Level

can be activated with <ALT>+numeric key.

Key combination	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of fields / ARRAYs
<ALT>+<3>	Structure	TYPE	Selection of structures / STRUCTs
<ALT>+<5>	FB	TYPE	Selection of function block types

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F1>	All	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help on the type

5.6 Declaration, Function

On function level, the declaration editor serves

- for declaring the input and output variables of the function,
- for declaring variables which can be used within the function.

Areas in the Declaration Editor (Function)

Area	Properties	Comment
Function value (main output)	Delivers information to the outside	To be connected to a network, header of the function
Declaration comment	The lines between function nn and VAR_INPUT are intended for comments on the function for defining the intended use, changes etc.	
VAR_INPUT ... END_VAR	External supply, no writing	First variable to be connected to a network, further variables only to be connected to a value
VAR_OUTPUT ... END_VAR	Delivers information to the outside	Variable only to be connected to a value
VAR END_VAR	Standard	Variables - standard

Structure of the Declaration Lines

Header

FUNCTION	Name	TYPE
Code word	Name of the function = name of the main output	Type of the function = type of the main output

VAR_INPUT ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Disabled	Comment

VAR_OUTPUT ... END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Disabled	Comment

VAR END_VAR

Name	AT	TYPE	:=	Comment
Name of a variable	Disabled	Data type	Disabled	Comment

Declaration Footer Commands, Function Level

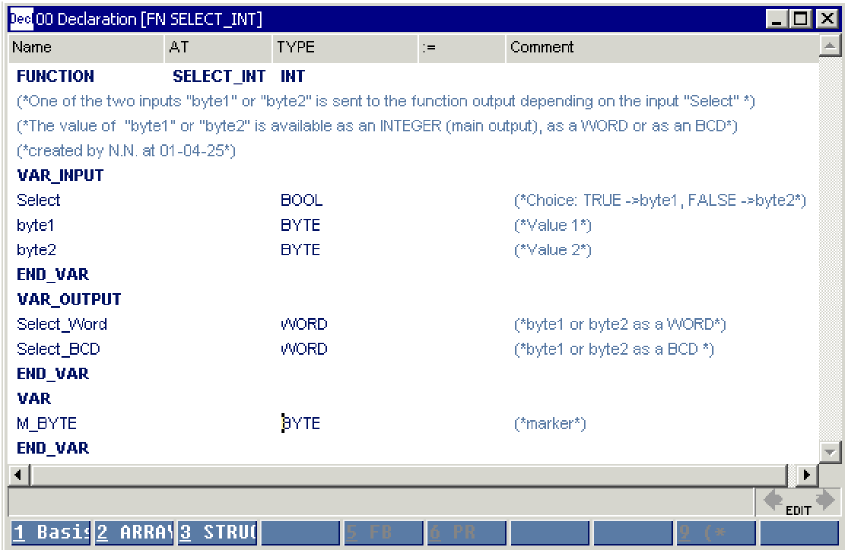
can be activated with <ALT>+numeric key.

Key combination	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of fields / ARRAYs
<ALT>+<3>	Structure	TYPE	Selection of structures / STRUCTs

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F>	All	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help on the type

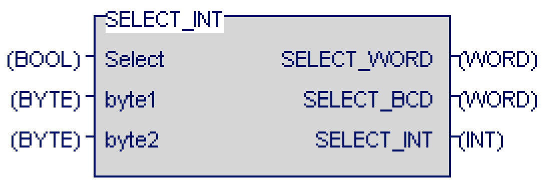
Structure of the Declaration Part of a Function (Example)



Dekl_Funktion.bmp

Fig. 5-20: Function "SELECT_INT"

The function has the following interface:



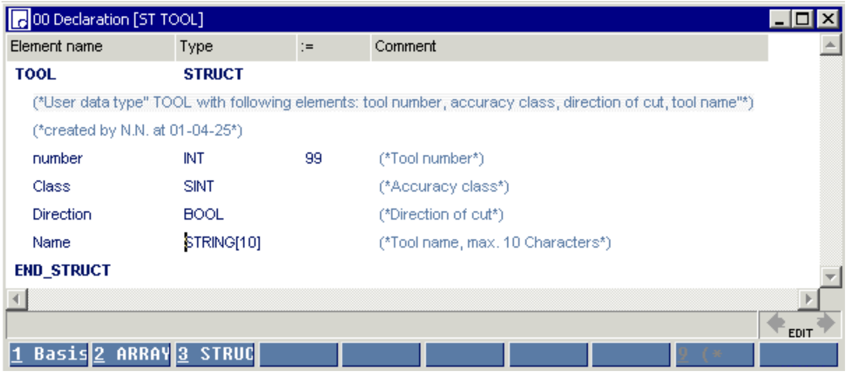
Interface SELECT_INT.bmp

Fig. 5-21: Interface of the function according to the declaration part

5.7 Declaration of Structures (STRUCT)

A structure consists of one or several elements, which can be of the elementary type or can be a structure or an array. Each element has its own name and, if it is of the elementary type, can have a user-defined initial value. Structures and fields have their own initial values. In addition to the declaration comment of the structure, each element can have its own comment.

Structure of the Declaration of Structures (Example)



ST Werkzeug.bmp

Fig. 5-22: Declaration part of the structure "TOOL"

The declaration comment is added to the line specifying the name. Of the four elements of the structure, "number" is defined with "99" by the user; the standard value "0" or "FALSE" is assigned the other elements. The name is ' ' (empty) .

Declaration Footer Command, Structure

can be activated with <ALT>+numeric key.

Key combination	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of arrays
<ALT>+<3>	Structure	TYPE	Selection of structures
<ALT>+<9>	(*	Element name	Full line comment

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F1>	All	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help on the type

Pop-up Menu, Structure Editor <Shift>+<F10>

The popup menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	**Explanation**
Import declaration	The ASCII file selected from the "WINPCL" text files is attached in sections to a possibly existent declaration.
Export declaration	The content of the declaration editor is exported as ASCII file and stored in the folder "WINPCL text files".
Syntax test	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys. Subsequent import of types, which were not available at the time of declaration.
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the data type interface of the current line.
Force	Allows the entry of a variable name. The value of the variable is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of the program code execution.
Status ARRAYs / Structures	Status display for the elements of an array or a structure. Selection is done through a tree structure till the specific element is reached.
Print current window	Printing of the editor contents(<Ctrl>+<P>)

5.8 Declaration of ARRAYs

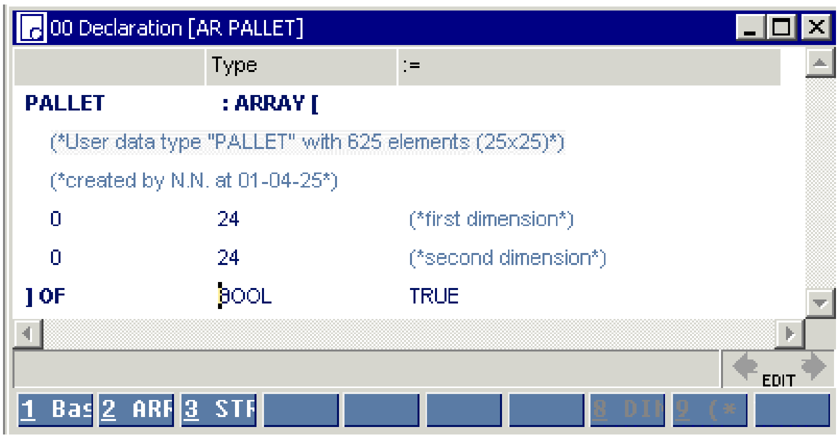
The elements of a field have a unique data type, which can be of the elementary type or can be a structure or even a field itself. The user can assign a unique initial value to all elements, if they are elementary. Structures and arrays have their own initial values. The elements of an array are arranged in dimensions (1 to 4 dimensions).

The lowest limit is always zero!

In addition to the declaration comment of the array, a comment can be given for each dimension.

Structure of the Declaration of ARRAYs (Example)

Array with elementary elements

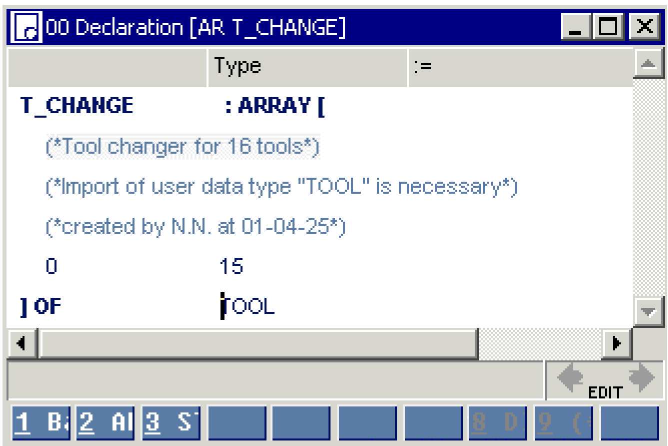


AR palette.bmp

Fig. 5-23: Declaration part of the elementary array "PALLET"

The declaration comment is added to the line with the name. All dimensions start with the zero element. The unique data type is BOOL The user sets the value for each element to TRUE.

Array with structured elements (type "TOOL")



AR Wechsler.bmp

Fig. 5-24: Declaration part of the structured array "T_CHANGER"

The declaration comment is added to the line with the name. All elements are structured data types which have four elements.

Declaration of Structures (STRUCT).
The elements of the structured data type have their own initial values. "Number" is preset to "99" by the user; the standard value "0" or "FALSE" is assigned to all other elements. The name is ' ' (empty) .

It is recommended to create the data type "TOOL" before declaring "T_CHANGER". It is then automatically imported.

If it is missing, "TOOL" in "T_CHANGER" is changed into red color (type unknown...). The import can be done subsequently, after "TOOL" has been declared, by activating "Syntax test" in the pop-up menu of the editor (<Shift>+<F10>).

Declaration Footer Command, ARRAYs

can be activated with <ALT>+numeric key.

Key combination	Name	Column	Meaning
<ALT>+<1>	Basic	TYPE	Selection of elementary data types
<ALT>+<2>	Array	TYPE	Selection of fields
<ALT>+<3>	Structure	TYPE	Selection of structures
<ALT>+<8>	DIM	TYPE	Entry of dimensions
<ALT>+<9>	(*	Element name	Full line comment

Other Keys and Key Combinations

Key combination	Column	Meaning
<Ctrl>+<Enter> or double-click	Name	Branch to instance
<Ctrl>+<Enter> or double-click	TYPE	Branch to instance
<Ctrl>+<F1>	All	Online help in case of syntax errors
<Shift>+<F1>	TYPE	Declaration help on the type

Pop-up Menu - ARRAY / Editor <Shift>+<F10>

The pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	Explanation
Import declaration	The ASCII file selected from the "WINPCL" text files is attached in sections to a possibly existent declaration.
Export declaration	The contents of the declaration editor is exported as ASCII file and stored in the folder "WINPCL text files".
Syntax text	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys. Additional import of types, which were not available at the time of declaration.
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the data type interface of the current line.
Force	Allows the entry of a variable name. The value of the variable is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of the program code execution.
Status ARRAYs / Structures	Status display for the elements of an array or a structure. Selection is done through a tree structure till the specific element is reached.
Print current window	Printing of the editor contents(<Ctrl>+<P>)

5.9 Limitation of the Declaration of Function Blocks in the Retain Area

In principle, the declaration of function blocks in the retain area is permitted according to IEC-61131-3.

However, there are limitations which must be observed whenever Rexroth system function blocks are used. The following list specifies the function blocks which may not be declared in the retain area.

Timer function blocks

Programming of timer stages is not permitted in the retain area.

- TP
- TON
- TOFF
- FLASH

Serial interface

Function blocks for supporting serial interfaces in the PLC application program:

- OPEN_COM
- CLOS_COM
- BTXX
- BTXX_2

Communication with other control components

Data exchange with other control components is achieved using function blocks via special data channels in the common dual-ported RAM. Usually, more than one communication cycle is required for data exchange with the CNC.

The progress made in data exchange is mapped as state-machine in the particular function block concerned. In other words, the internal state of the function block depends on the state of the DPR. However, the DPR is always re-initialized after a reset.

Current communication cycles will be lost. As a result, the state-machine in the FB is invalid, if this FB has been declared to be remanent.

The following function blocks may not be declared to be remanent:

MTCNC	**SYNAX**
NC memory selection	MC_CHANGE_PHASE
SEL_MEM	MC_RD_PARAMETER
ACT_MEM	MC_WR_PARAMETER
	MC_WR_LISTDATA
Process data channel	MC_DIAGNOSIS
AXD_WR	MC_RD_LISTDATA
AXD_RD	MC_RD_DATASTATUS
DCD_RD	MC_ABORT_TRANSMISSION
DCD_WR	MC_RW_PTR_TLG
MTD_WR	MC_RD_PHASE
MTD_RD	MC_RD_ATTRIBUTE
NCVAR_RD	MC_RD_NAME
NCVAR_WR	MC_RD_UNIT
OTD_WR	MC_RD_MIN_VALUE
OTD_RD	MC_RD_MAX_VALUE
TL_DELETE	MC_RD_ELEMENT
TL_ENABLE	MC_WR_ELEMENT
TL_MOVE	MC_RD_ARRAY
TL_RESET	MC_WR_ARRAY
TLBD_RD	MC_RW_ARRAY_TLG
TLBD_WR	
TLD_WR	
TLD_RD	
TLED_RD	
TLED_WR	

Fig. 5-25: Coupling with other control components

Note: These blocks may neither be declared directly in the retain area nor indirectly via blocks which are declared in the retain segment themselves.

6 Instruction List Editor

6.1 General Notes on the Instruction List Editor

The instruction list editor serves for entering and modifying the program code (implementation) in programs, function blocks and functions in the Edit and Online-Edit modes, as well as for indicating variable values at the end of a PLC cycle in the Status mode and for unchanged networks in the Online-Edit mode.

It allows instructions to be used in compliance with EN 61131-3 and indicates text as an alternative to the graphic ladder diagram editor of the system. Most of the instruction list constructs can be converted into ladder diagram networks and vice versa with the <TAB> key.

6.2 Structure of an Instruction List Line

The IL line in the edit mode is divided into four columns (grid):

- Label
- Operation, also see "Instructions and Approved Data Types"
- Operands, upon request
- Comments on the current line

Label	**Operation**	**Operand**	**Comment**
mLABEL:	LD	var_01	(*Fully used IL line with comment*)
(*Single or multi-line comment on the complete IL line*)			

The following contents of IL lines are also possible:

- Empty line
- Label in empty line, at the beginning of a network (marginal marking)

If the status display is activated, a fifth column appears which is used to indicate the value of the variable.

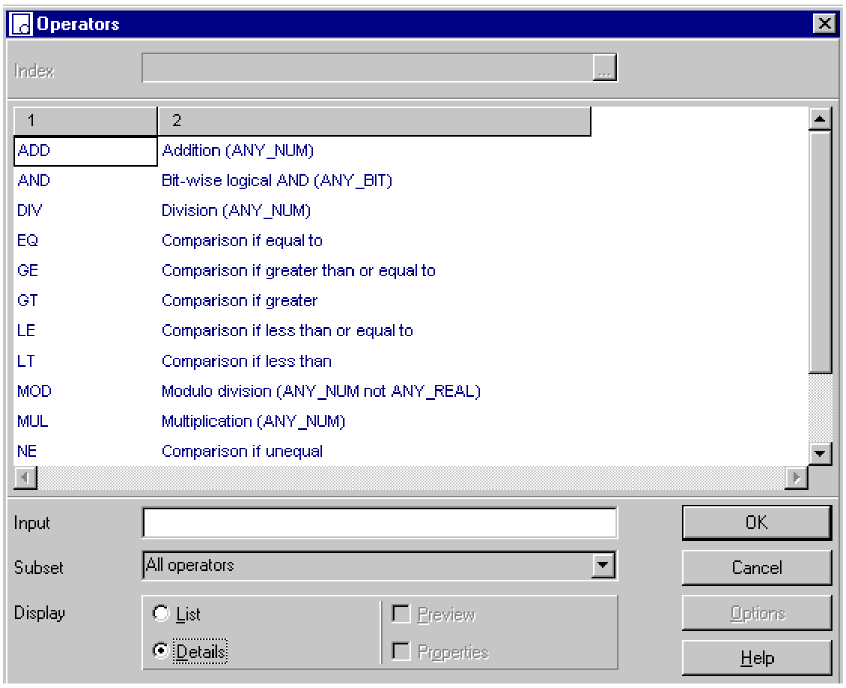
Depending on the position at which the cursor is located as well as the preceding operators, it is possible to call the selection window via the footer commands (with <ALT>+<number>).

Selection Window, Operators

The footer command "<Alt>+<7>-OP" appears, if the cursor is positioned in column 2, operators. The selection window for operators opens. They can be represented as list (space saving) or with detail information.

Furthermore, subsets can be selected:

- All operators
- Logic operators
- Boolean operators
- Arithmetic operators
- Comparison operators
- Jumps



OP_Auswahl.bmp

Fig. 6-1: Selection window, operators

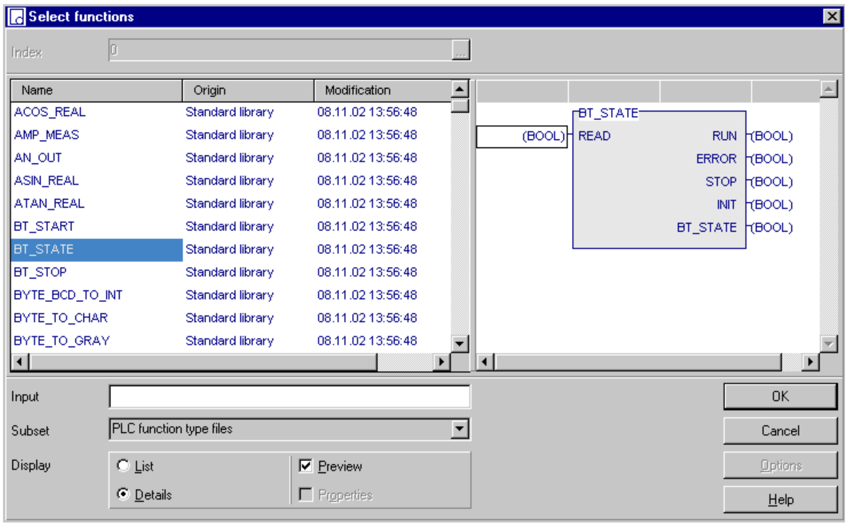
The name of the desired operator can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Functions

The footer command "6-FN" appears, if the cursor is positioned in column 2. The selection window for functions opens. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the function (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



FN_Auswahl.bmp

Fig. 6-2: Selection window, functions

The name of the desired function can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Instances of Function Blocks

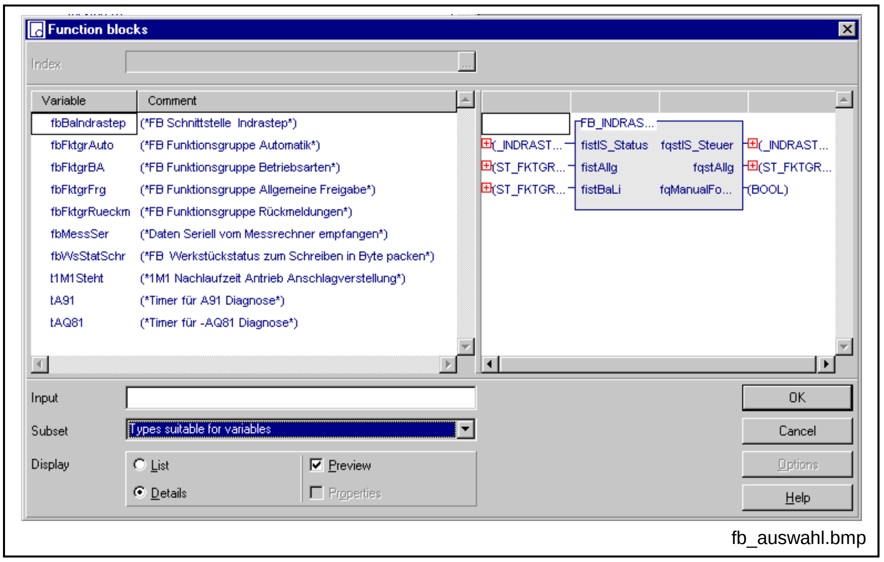
The footer command "<Alt>+<5>-FB" appears, if the cursor is positioned in column 3, operands, and the operation in column 2 is a subprogram call.

- CAL, unconditional call
- CALC, conditional call, if the operation result is TRUE.
- CALCN, conditional call, if the operation result is FALSE.

The selection window for already declared instances of function blocks opens. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the function block instance (interface) appears.

In addition to the name of the instance its declaration comment is indicated.



fb_auswahl.bmp

Fig. 6-3: Selection window, instances of function blocks

The name of the desired function block instance can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, SFCs

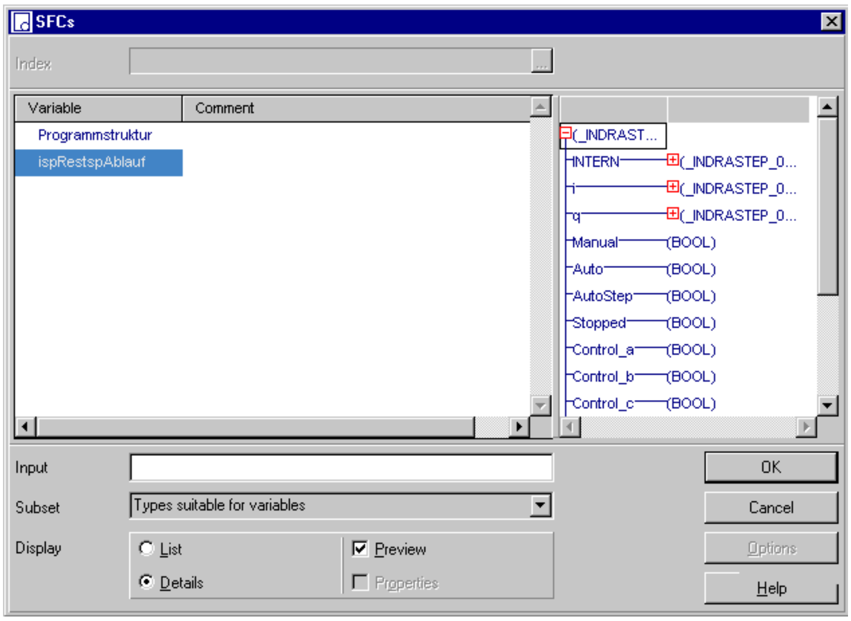
The footer command "<Alt>+<4>-SFCs" appears, if the cursor is positioned in column 3, operands, and the operation in column 2 is an unconditional call.

- CAL, unconditional call.

The selection window for already declared SFCs opens. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the SFCs appears.

In addition to the name of the SFC its declaration comment is indicated.



SFC_Auswahl.bmp

Fig. 6-4: Selection window, SFCs

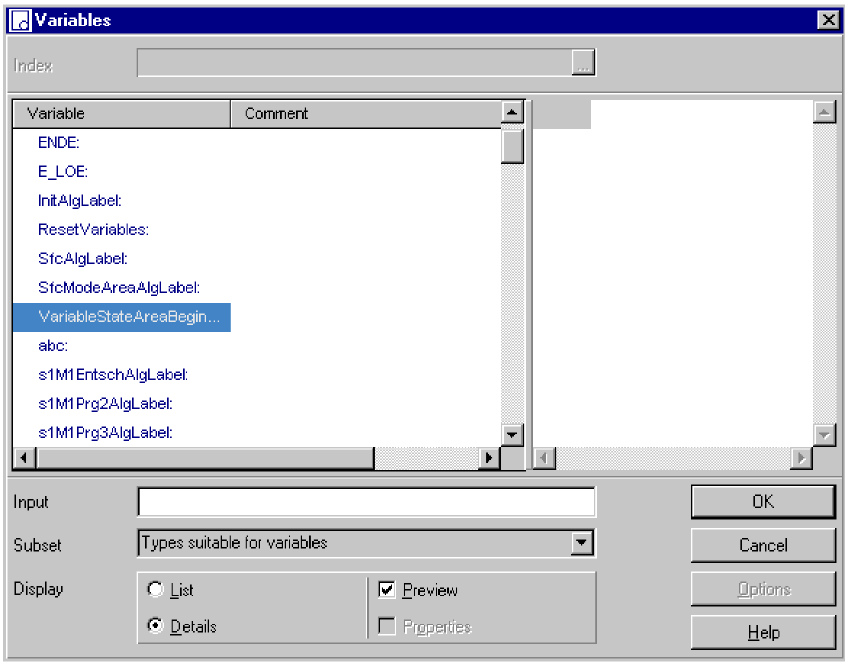
The name of the desired SFC can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Labels

The footer command "0-=>>(o)" appears, if the cursor is positioned in column 3, operands, and the operation in column 2 is a jump.

- JMP, unconditional jump.
- JMPC, conditional jump, if the condition is fulfilled.
- JMPCN, conditional jump, if the condition is not fulfilled.

The selection window for already declared labels opens. They can be represented as list (space saving) or with detail information.



Auswahl_Marke.bmp

Fig. 6-5: Selection window, labels

The name of the desired label can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Variables

The footer command "1-VAR" appears, if the cursor is positioned in column 3, operands, and an appropriate operation is listed in column 2 (e.g. LD, ST, ADD....). The selection window for operands opens.

Alternatively, the window can be called by pressing <ALT>+<Enter>. They can be represented as list (space saving) or with detail information.

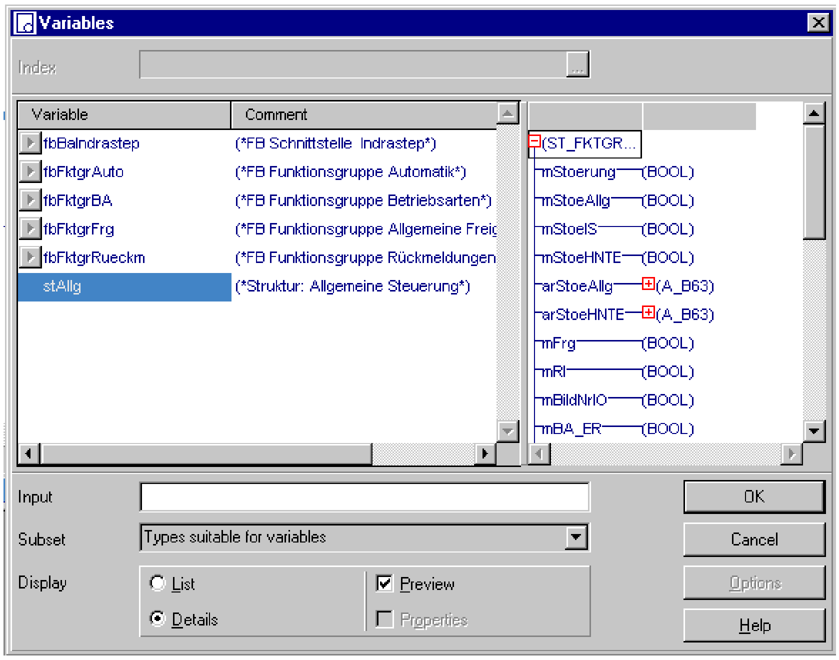
As subset you can select type-specific or all variables.

If the option "Preview" is selected, a graphic representation of the variables appears.

In addition to the name of the variable its declaration comment is indicated.

Multi-element variables (instances of structures, ARRAYs or FBs) are marked by a preceded gray arrow. You can open this variables by double-clicking or pressing <Enter> on the respective variable. The elements can be (recursively) selected.

In future, a switch for the return path upwards to the instance name is planned.



VAR_Auswahl.bmp

Fig. 6-6: Selection window, variables

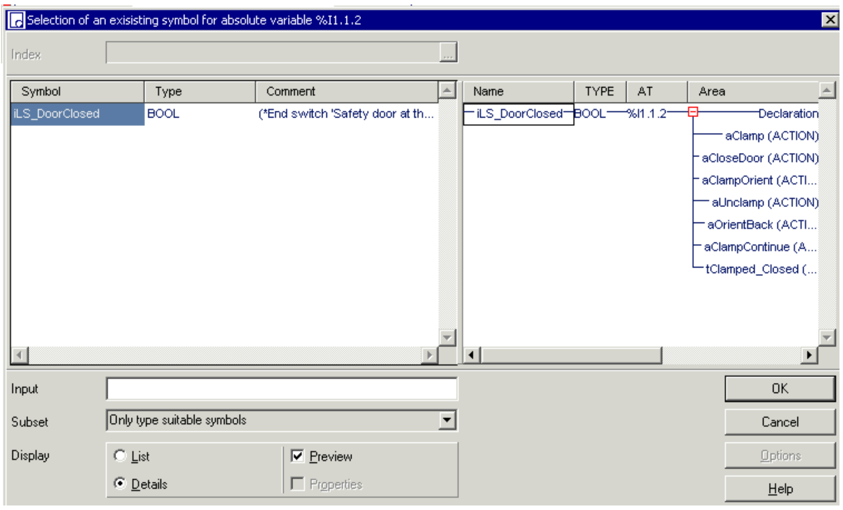
The name of the desired variable can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Absolute Addressed Variables

When entering an absolute address – e.g. %I1.1.2 – WinPCL verifies, if a name, that is linked with the address via a declaration exists in the validity area of the file. If a name is found, it is indicated in the selection window.

When confirming the element obtains this name, when rejecting the absolute address remains at the element.

The window is used in the IL editor to enter operands.



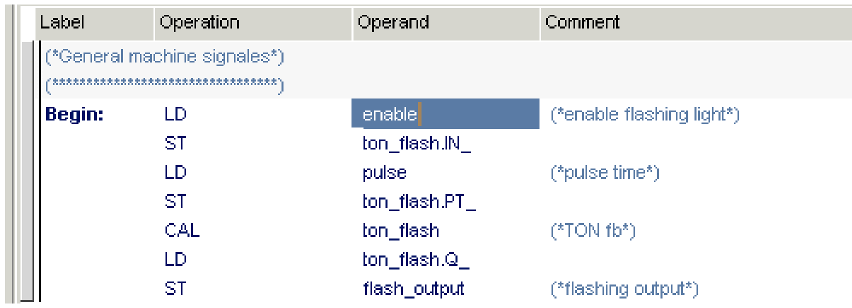
Auswahlfenster_Abs_Var_singl.bmp

Fig. 6-7: Window to select the absolute address

6.3 Editing Features, Varying Color in the IL Editor

Operating modes: edit and online-edit.

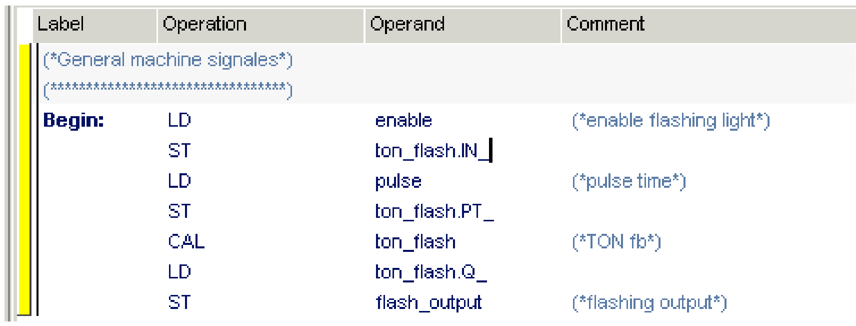
The section of the line where editing takes place is first white on a blue background while you do the entry.



AWL_Eingabe_00.bmp

Fig. 6-8: Entry of an IL line, editing of a variable

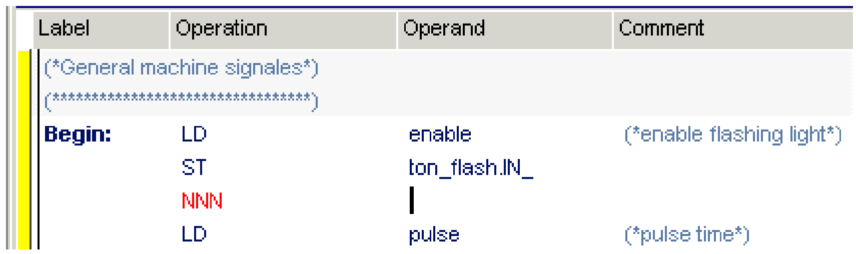
The color of the marginal marking on the left side changes from "gray", e.g. correct normal condition to "yellow", that means the network was modified. At this time the section is still untested.



AWL_Eingabe_01.bmp

Fig. 6-9: Entry of an IL line, variables identified as being correct, network still untested, yellow marginal marking

Faulty operations, undeclared names or a combination of this, change their color into red when you exit the edited section or the edited line. The remaining text within such an incorrect line is shown in gray. If the error is not detected directly, position the cursor on the line and press the <Ctrl>+<F1> keys for online help.



AWL_Eingabe_03.bmp

Fig. 6-10: Entry of an IL line, operator identified as being faulty, network still untested, yellow marginal marking

The test whether neighboring lines in this network match is not carried out unless you exit the network, e.g. by moving the cursor out of the network. The marking at the left margin changes from yellow to gray, that means everything is ok or to red, that means there is an error in the network. The basic font color is dark-blue (no errors), the comment is middle-blue, the left margin is gray.

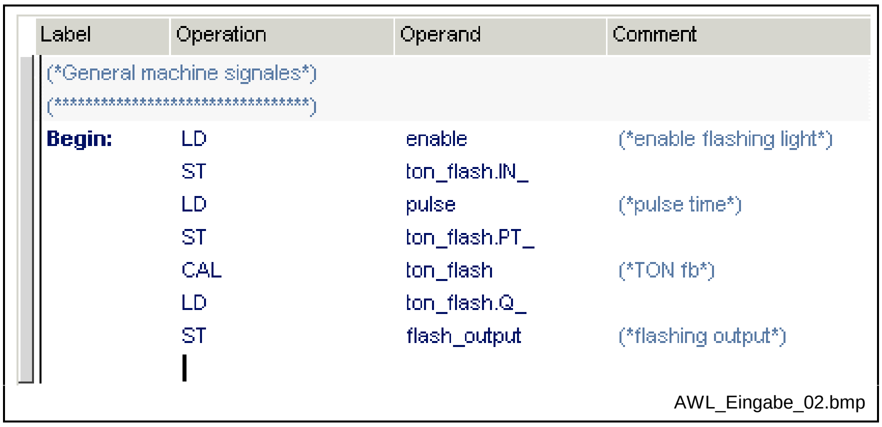


Fig. 6-11: Network without errors after editing, marginal marking is gray

If faulty variables were not corrected before the network test, they are indicated in red and the marginal marking is indicated in red too.

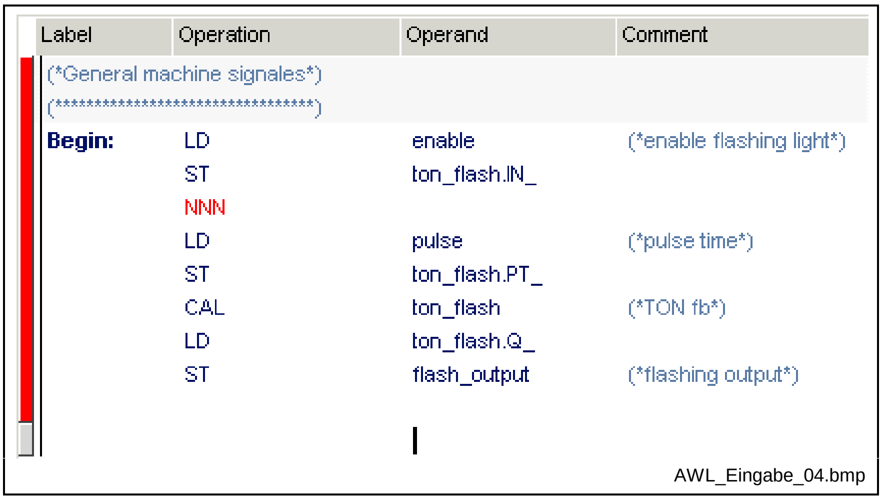


Fig. 6-12: Network faulty after editing, marginal marking is red

Errors remain visible in this manner. They must be eliminated before a successful compilation run can be started, but do not affect normal operation. A complete check, including labels and jumps, is carried out during the compilation attempt or the syntax test (Pop-up Menu, IL Editor <Shift>+<F10>).

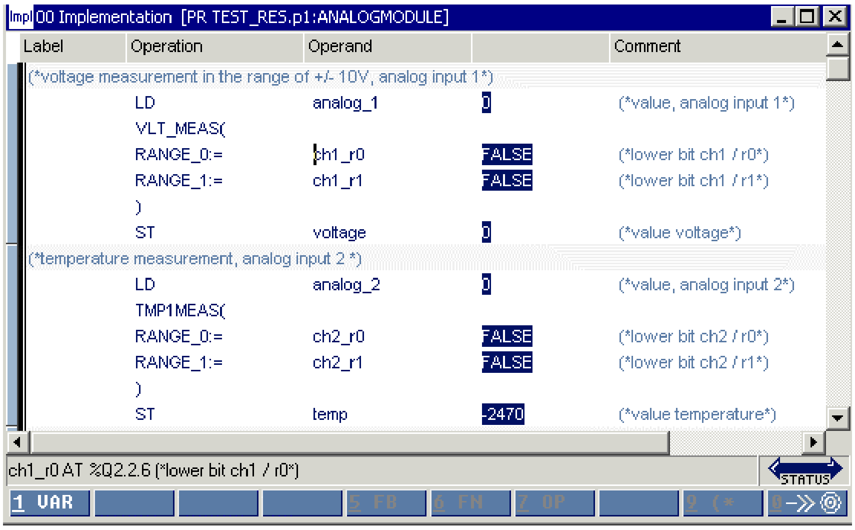
6.4 Options, IL Editor

The options , relevant for the instruction list editor can be selected by means of the "Extras / Options" menu item:

<table>
<tr><th>Group</th><th>Option</th><th colspan="2">Meaning</th></tr>
<tr><td rowspan="4">Desktop</td><td>Restore size and position during startup</td><td colspan="2">The desktop is restored in the same size and position.</td></tr>
<tr><td>Restore MDI window during startup</td><td colspan="2">MDI windows are opened in the same order when restarting the system.</td></tr>
<tr><td>Auto save</td><td colspan="2">Allows the automatic saving of the current file in presettable time intervals without any prompt.</td></tr>
<tr><td>Sound</td><td colspan="2">Activation or deactivation of a beep sound.</td></tr>
<tr><td>View / All</td><td>Apply column width modifications automatically</td><td colspan="2">Restoring of the column with same width.</td></tr>
<tr><td rowspan="5"></td><td>Apply declaration comment in implementation</td><td colspan="2">Comments, that have been entered in the respective declaration line are indicated in the implementation. The implementation can be changed; the comment is then doubled, the declaration line remains unaffected.</td></tr>
<tr><td>Variable display</td><td colspan="2">With symbols (name) or absolute (address).</td></tr>
<tr><td>Display of absolute variables</td><td colspan="2">The user can select from I/Q, E/A and I/O for absolute addresses.</td></tr>
<tr><td>Truncating very long texts</td><td colspan="2" rowspan="2">Texts and numbers can be truncated to the right or left, and can be represented with or without "..." marking.</td></tr>
<tr><td>Truncating very long numbers</td></tr>
<tr><td rowspan="5">View / IL</td><td rowspan="5">Column width for the individual columns (with standard values)</td><td>Label</td><td>70</td></tr>
<tr><td>Operation</td><td>70</td></tr>
<tr><td>Operand</td><td>110</td></tr>
<tr><td>Status</td><td>90</td></tr>
<tr><td>Comment</td><td>250</td></tr>
</table>

Fig. 6-13: IL editor options

6.5 Status Display in the IL Editor



AWL_Status.bmp

Fig. 6-14: Status display in the instruction list

Variable values at the end of a PLC cycle are indicated in the status mode and for unchanged networks in the online edit mode.

For this, an additional column is inserted between the operand column and comment column.

Further ways to get status information are:

- Start / Force <Shift>+<F8> for elementary variables (ANY_ELEMENTARY)
- Start / Status ARRAYs / Structures <Shift>+<F3>

6.6 Online Editing in the Instruction List

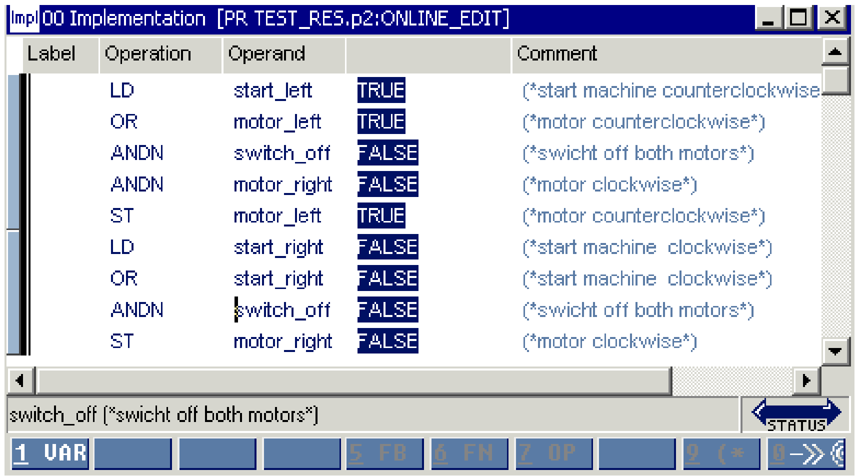
The online edit feature in the present version permits to exchange the code for a program, a function block or a function, without changing the data of the particular POU concerned. A machine or plant can continue its working cycle although the program code was changed.

Changes in the implementation of **one** program organization unit, which require neither declaration nor imports, are allowed at present. All other changes are not online capable.

Status of the POU >>Status<<

A running PLC program with activated status display is the initial condition for online editing. In the following example, the instruction "ANDN motor_left" for locking is to be added online.

Identification: "Status" is indicated in the right bottom corner.



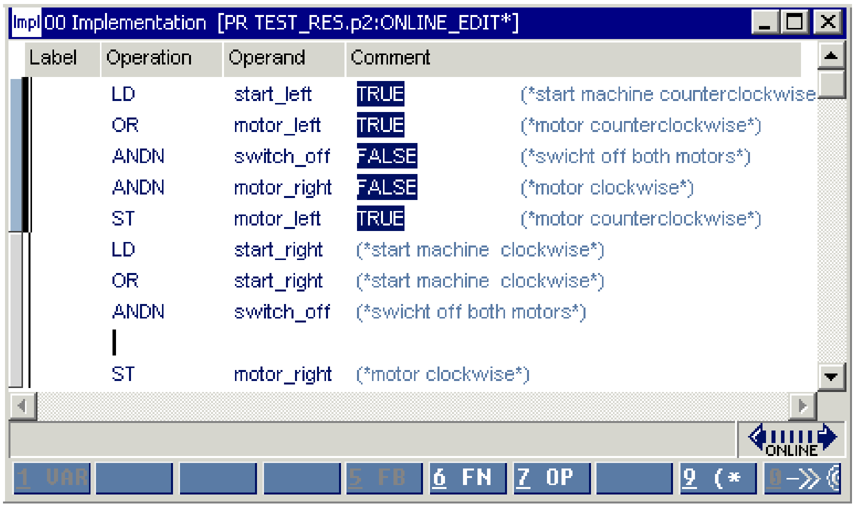
online_awl_01.bmp

Fig. 6-15: Before the online editing

Status of the POU >>Online<<

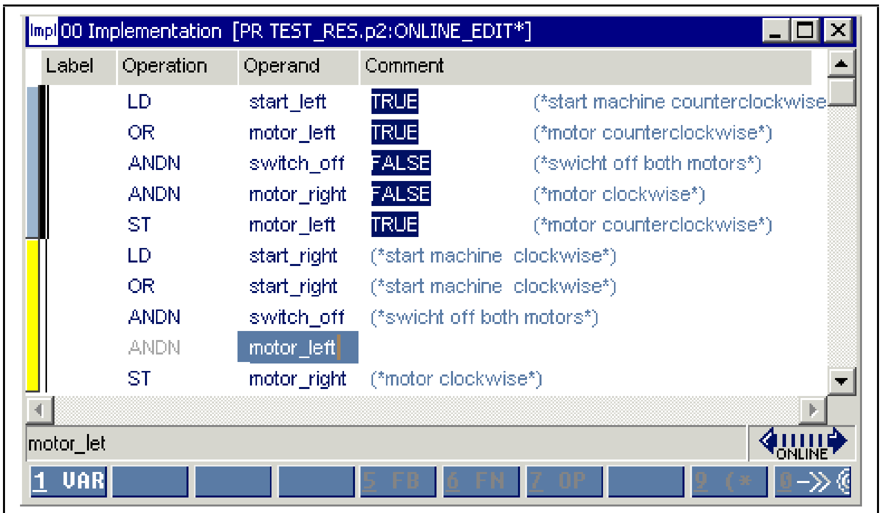
The online editing mode is initiated by inserting a blank line. The color of the marginal marking for the actual network changes from blue to yellow (for colors see Chapter "Editing Features, Varying Color in the IL Editor").

Identification: "Online" is indicated in the right bottom corner.



online_awl_02.bmp

Fig. 6-16: Online editing, inserting an empty line



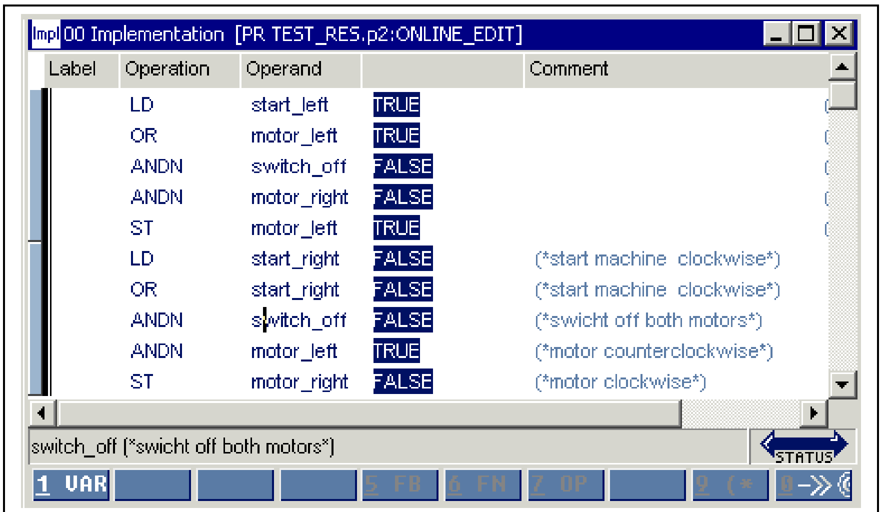
online_awl_03.bmp

Fig. 6-17: Online editing, filling in a line

Download of the change

Online changing is completed by downloading the changed code to the control, using the "Start / Download "xx" in control "xx"" menu item or by pressing <Ctrl>+<F9>.

Identification: "Status" is indicated in the right bottom corner.



online_awl_04.bmp

Fig. 6-18: Online editing completed with the download

Changes which are not online capable

At present, changes, which require neither declaration nor imports, are allowed as online changes in the implementation of **one** program organization unit .

Kind of change	Operating mode
Deletion of a network	Online
Deletion of a contact or an IL line	Online
Insertion of a new network or a new IL line	Online
Insertion or change of a contact, a coil, an instruction or the like	Online
Insertion of labels or jumps	Online
Insertion of an currently declared function block	Online
Insertion of a function which was already used in the POU (new imports are not online capable as yet!)	Online
Change of network properties of existing networks (at least one network of the POU has to provide activated network properties!)	Online
Change of network comments in IL / ladder diagram	Online
New declaration or change of declaration of variables or instances of programs / function blocks, as well as in all lists	Edit
Insertion or deletion of steps, transitions and actions	Edit
First application of network properties or deletion of the last network properties	Edit
Change of the action qualifier of an action block	Edit
Change of comments in all lists, in SFCs, action blocks and in all declarations	Edit
Changes in the IO editor	Edit
Changes in a second file, if there is already one file in the online edit mode	Edit

Fig. 6-19: Overview of online capable changes (selection)

Edge Evaluation in the Instruction List

In analogy to the ladder diagram, networks in the IL can often be simplified or at least represented more clearly by using transition operations.

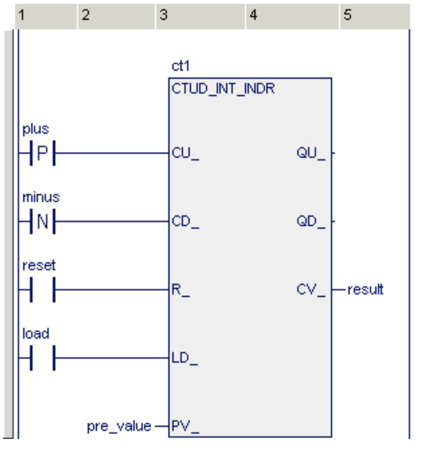
Here, the transition from one operation to an operation with edge evaluation and vice versa represents an online change.

LD, OR, AND with Detection of Transitions (Edges)

For edge evaluation purposes, the LD, OR, AND operations can be expanded to LD>, OR>, AND> (positive edge, "P" ladder) or LD<, OR<, AND< (negative edge, "N" ladder).

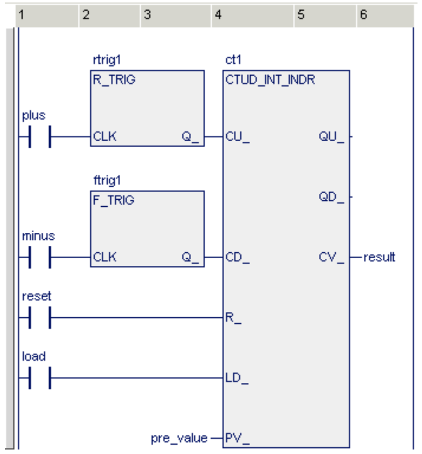
The figure below shows examples of use in a ladder diagram.

Edge contacts



Label	Operator	Operand	Comment
	LD>	plus	(*increase
	ST	ct1.CU_	(*Count up
	LD<	minus	(*decreas
	ST	ct1.CD_	(*Count dc
	LD	reset	(*reset va
	ST	ct1.R_	(*Reset in
	LD	load	(*load pre
	ST	ct1.LD_	(*Preset ir
	LD	pre_value	(*preset v
	ST	ct1.PV_	(*Preset v
	CAL	ct1	
	LD	ct1.CV_	(*Counter
	ST	result	(*current

with R_TRIG / F_TRIG



Label	Operator	Operand	Comment
	LD	plus	(*increase result*
	ST	rtrig.CLK	(*Input for edge ic
	CAL	rtrig	
	LD	rtrig.Q_	(*Output pulse w
	ST	ct1.CU_	(*Count up with r
	LD	minus	(*decrease result
	ST	ftrig.CLK	(*Input for edge ic
	CAL	ftrig	
	LD	ftrig.Q_	(*Output pulse w
	ST	ct1.CD_	(*Count down wi
	LD	reset	(*reset value*)
	ST	ct1.R_	(*Reset input, dor
	LD	load	(*load preset valu
	ST	ct1.LD_	(*Preset input*)
	LD	pre_value	(*preset value*)
	ST	ct1.PV_	(*Preset value, ap
	CAL	ct1	
	LD	ct1.CV_	(*Counter positior
	ST	result	(*current value*)

Fig. 6-20: Comparison of edge contacts and R_TRIG / F_TRIG in LD and IL

Note: Each IL line with edge evaluation has its own old value!

The pulse is active for exactly one PCL cycle, i.e. the network must be executed at least twice.

ST for Detection of Transitions (Edges)

For edge evaluation purposes, the ST operation can be expanded to ST> (positive edge, "P" coil) or ST< (negative edge, "N" coil).

Note: Each IL line with edge evaluation has its own old value!

The pulse is active for exactly one PCL cycle, i.e. the network must be executed at least twice.

Online Changes and Edge Evaluation - IL

If, by online changing , LD, OR or AND and/or ST, STN operations are converted to operations for detection of edge transitions, then this results in the following transitions:

Rules for LD<, LD>, OR<, OR>, AND<, AND> by the example of LD>, LD<:

- If an operation is converted into an operation with edge evaluation, it assumes an old value which is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the operation.
- If an operation with "P" edge evaluation is converted into an operation with "N" edge evaluation, the old value is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the operation. (This also applies to conversion of N into P!)
- If an operation with "P" edge evaluation is converted into another operation with "P" edge evaluation (or N into N), ***the current old value remains unchanged!*** This value and the current value of the variable form the basis of the behavior of the operation.
- A new variable assumes a new old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the variable.
- An inserted new IL line with operation for edge evaluation assumes an old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the line.

Changing the operation and the value of the variable

New	Same variable				NEW variable			
Old	LD>	LD<	LD>	LD<	LD>	LD<	LD>	LD<
LD	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LDN	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LD>	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LD<	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LD	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LDN	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
LD>	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0-1-0	FALSE
LD<	FALSE	0-1-0	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE

Fig. 6-21: Online change in case of LD operations

Explanation by the example of the LD operation (OR, AND analogously):

LD Load operation, state of the variable FALSE

LD> Load operation, positive edge, state of the variable TRUE, etc. ..

Rules for ST operations (ST>, ST<):

- If an ST operation is converted into an ST operation with edge evaluation, it assumes an old value which is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the operation.
- If an ST operation with "P" edge evaluation is converted into an ST operation with "N" edge evaluation, the old value is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the operation. (This also applies to conversion of N into P!)
- If an ST operation with "P" edge evaluation is converted into another ST operation with "P" edge evaluation (or N into N), ***the current old value remains unchanged!*** This value and the current value of the variable form the basis of the behavior of the operation.
- ***A new variable assumes the old value of its predecessor.*** This value and the current value of the variable form the basis of the behavior of the variable.
- An inserted new coil for edge evaluation assumes an old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the coil.

Changing the operation type and the value of the variable

New	Same variable				NEW variable			
Old	ST>	ST<	ST>	STN	ST>	ST<	ST>	ST<
ST	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
STN	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
ST>	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
ST<	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
ST	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
STN	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
ST>	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE	FALSE
ST<	FALSE	0-1-0	0-1-0	FALSE	FALSE	0-1-0	0-1-0	FALSE

Fig. 6-22: Online change in case of coils

Explanation:

ST ST operation, state of the variable FALSE

ST> ST operation, positive edge, state of the variable TRUE, etc. ..

6.7 Pop-up Menu, IL Editor <Shift>+<F10>

This pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	**Explanation**
Open	Branch, also <Ctrl>+<Enter>
New network	- by **adding** an empty **IL line** before the current network - by **adding** an empty **LD network** before the current network - by **adding** an empty **IL line** behind the current network - by **adding** an empty **LD network** before the current network
Delete network	Deletion of the current network (see marginal marking).
Separate network	Service command
Connect network	Service command
Convert network to	- a **ladder diagram** network - convert the complete contents of the editor into ladder diagram networks with **"LD (all)"** - convert the complete contents of the editor into IL networks with **"IL (all)"**
ProVi messages	Display and modification of the diagnosis properties.
Import implementation	The ASCII file selected from the "WinPCL text files" is added to the current IL line.
Export implementation	The complete contents of the IL editor is exported as an ASCII file and stored in the folder "WINPCL text files".
Export network	The complete IL network is exported as an ASCII file and stored in the folder "WINPCL text files" (marginal marking).
Syntax text	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the interface of the data type or of the function block type of the current line.
Cross reference help	List of all places where the variable is used. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force	Allows the entry of a variable name. The value of the variables is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variable and the start of the program code execution.
Status ARRAYs / Structures	Display of the status of array and structure elements, forcing by pressing the <Shift>+<F10> keys or the right mouse button.
Print current window	Print of the editor contents with <Ctrl>+<P>)
Options	Optimization of the column width
Internals	Search for faults in the programming system, to be used only if approved by the service.

Fig. 6-23: Pop-up menu of the instruction list editor

6.8 Block Commands, IL Editor

Select the text by pressing and holding the SHIFT KEY while using the appropriate arrow key or by pressing and holding the left mouse button while dragging it across the text.

You can select IL for every network by clicking the gray bar on the left outer side.

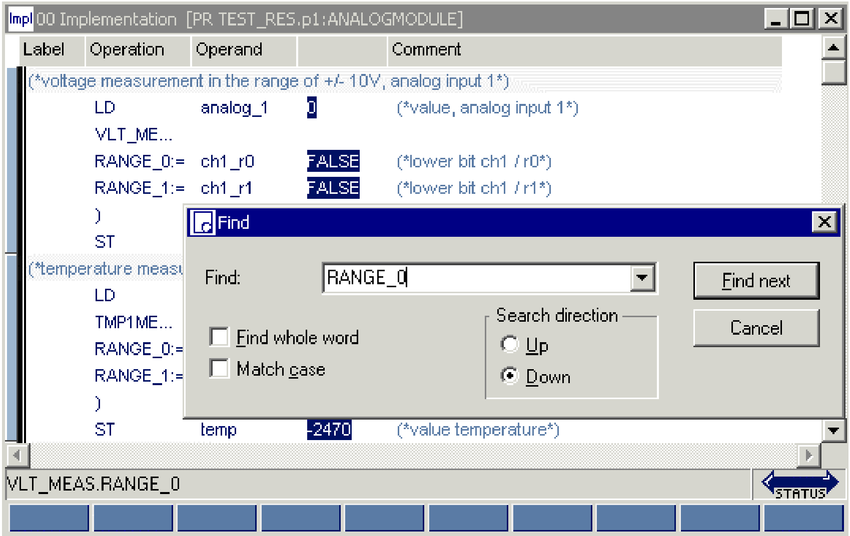
Extending the selection	**Key combination**
One character to the right	<Shift>+ arrow key <to the right>
One character to the left	<Shift>+ arrow key <to the left>
To the end of the line	<Shift>+<End>
To the beginning of the line	<Shift>+<Home>
Down by one line	<Shift>+ arrow key <downward>
Up by one line	<Shift>+ arrow key <upward>
Down by one page	<Shift>+<Page down>
Up by one page	<Shift>+<Page up>

Deletion of text	**Keys**
Deleting the character to the left of the cursor	BACKSPACE KEY
Deleting the character to the right of the cursor	<Del>

Copying and moving of text	**Key combination**
Copying the text selected to the clipboard	<Ctrl>+<C>
Moving the text selected to the clipboard	<Ctrl>+<X>
Pasting the contents from the clipboard	<Ctrl>+<V>

6.9 Search and Replace, IL Editor

is in the first version and provides the features of a text editor:

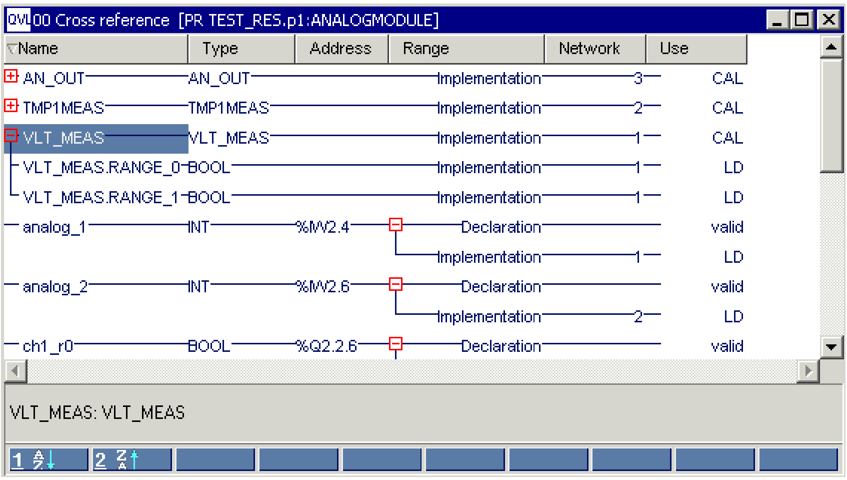


AWL_Suchen.bmp

Fig. 6-24: Search function in the IL editor

6.10 Cross Reference List, IL Editor

In contrast to the cross references of the pop-up menu, the overview obtained via "View / Cross reference list" displays all variables. Of course, only variables from lines with the correct syntax can be resolved by their place of use. However, all faulty names or names with double declaration are displayed and can, thus, be reached with by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.



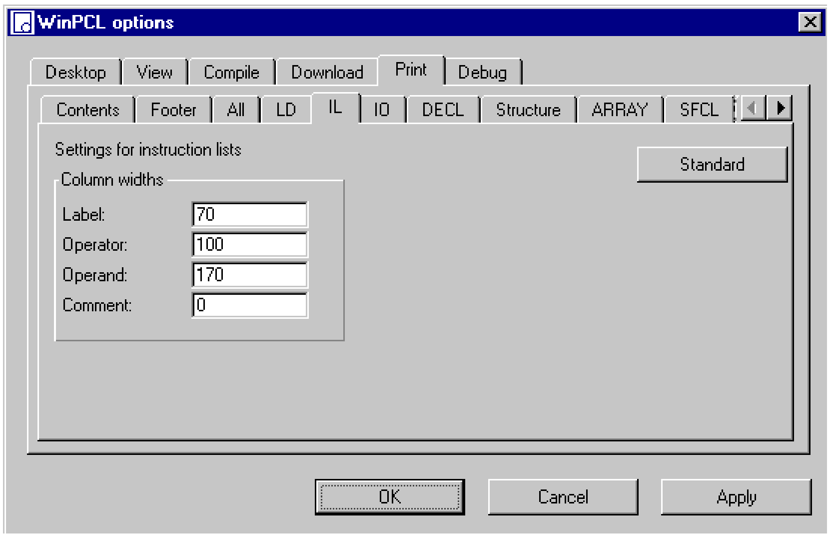
AWL_QVL.bmp

Fig. 6-25: Cross reference list with IL applications

Pictogram	Meaning
LD, LDN	Read access, negated read access
CAL	Block call
ST, STN	Write access, negated write access
SET, RES-	Set, reset, write access

6.11 Documentation, IL Editor

Documentation must be implemented by using the column widths specified under Extras / Options, IL Editor / Print / IL.



print_awl.bmp

Fig. 6-26: Print options in Instruction list

The button "Apply" activates the column width set for the instruction list editor. The width of the column can be entered either in the window shown above or preset in the editor by dragging the headers.

The button "Standard" resets the default.

The "OK" button applies the setting and closes the dialog window.

The "Cancel" button closes the window; the previous values are kept.

Detailed information on the real print process and the features is to be found in the main chapter on WinPCL.

6.12 Instructions of the IL, Table Overview

The following instructions are supported in compliance with EN 61131-3:

- Loading and Storing Operations
- Set and Reset Commands (Bit Operands Only)
- Logic Instructions
- Jumps, Calls, Return (Conditional and Unconditional)
- Arithmetic Instructions and
- Comparators

Note: All instructions which are used for a linking of data only apply to operands with same data types!

Loading, storing, setting and resetting instructions		Logic instructions		Jumps, calls, return (conditional and unconditional)		Comparison instructions		Arithmetic instructions	
LD	SET	AND	AND(	JMP	CAL	EQ	EQ(	ADD	ADD(
LDN	SETC	ANDN	ANDN(	JMPC	CALC	EQN	EQN(	SUB	SUB(
ST	SETCN	OR	OR(	JMPCN	CALCN		NE(	MUL	MUL(
STN	RES	ORN	ORN(		RET	NEN	NEN(	DIV	DIV(
	RESC	XOR	XOR(		RETC	GT	GT(	MOD	MOD(
	RESCN	XORN	XORN(		RETCN	GE	GE(		
						LT	LT(		
						LE	LE(		

6.13 Instructions and Approved Data Types

Note: 64 Bit data types are only supported by PPC and Soft Controls (LWORD, LINT, ULINT, LREAL).

Loading and Storing Operations

The following operations are supported

- LD with extension to the LD> and LD< edge evaluation, LDN
- ST with extension to the ST> and ST< edge evaluation, STN

LD

(Loading and Storing Operations), (Instructions of the IL, Table Overview)

Operation	Activity	Approved for data type
LD	Loads the value of the operand.	All, pointer, P# (address of)
LD>	Transfer of the 0-1 edge of the operand as pulse	BOOL
LD<	Transfer of the 1-0 edge of the operand as pulse	BOOL

LDN

(Loading and Storing Operations), (Instructions of the IL, Table Overview)

Operation	Activity	Approved for data type
LDN	Loads the bitwise negated value of the operand.	BOOL; BYTE; WORD; DWORD, LWORD

ST

(Loading and Storing Operations), (Instructions of the IL, Table Overview)

Operation	Activity	Approved for data type
ST	Stores the current value to the operand.	All, pointer
ST>	Transfer of the 0-1 edge as pulse	BOOL
ST<	Transfer of the 1-0 edge as pulse	BOOL

STN

(Loading and Storing Operations), (Instructions of the IL, Table Overview)

Operation	Activity	Approved for data type
STN	Stores the bitwise negated value to the operand.	BOOL; BYTE; WORD; DWORD, LWORD

Set and Reset Commands (Bit Operands Only)

The following operations are supported:

- SET, SETC, SETCN
- RES, RESC, RESCN.

SET

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
SET	Unconditional setting of the bit operand.

SETC

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
SETC	Setting of the bit operand if the previous result is TRUE, otherwise no activity.

SETCN

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
SETCN	Sets the bit operand if the previous result is FALSE, else no activity.

RES

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
RES	Unconditional resetting of the bit operand.

RESC

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
RESC	Resets the bit operand if the previous result is TRUE, else no activity.

RESCN

(Set and Reset Commands (Bit Operands Only)), (Instructions of the IL, Table Overview)

Operation	Activity
RESCN	Resets the bit operand if the previous result is FALSE, else no activity.

Logic Instructions

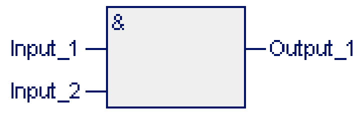
The following logic operations are supported

- AND, bit-by-bit AND operation, with extension to the AND> and AND< edge evaluation
- OR, bit-by-bit OR operation, with extension to the OR> and OR< edge evaluation
- XOR, bit-by-bit XOR operation

AND

The AND operation (Logic Instructions, Instructions of the IL, Table Overview) is carried out bit by bit.
If the AND operation is applied to BYTE, WORD, DWORD, bit positions of the same order are linked.

Operation	**Activity**
AND	AND operation of the current value with the value of the operand
AND>	AND operation of the current value with a pulse, with a 0-1 transition of the value of the operand (for Boolean variables only)
AND<	AND operation of the current value with a pulse, with a 1-0 transition of the value of the operand (for Boolean variables only)
ANDN	AND operation of the current value with the bitwise negated value of the operand
AND(	AND operation of the current value with the value of the following expression
ANDN(	AND operation of the current value with the bitwise negated value of the following expression
)	Termination of an expression

AND operation	**Representation in the instruction list**
	LD Input_1 AND Input_2 ST Output_1

AND for	**Boolean variable (in LD shown as contact)**
Input_1 Input_2 ------------- Output_1	1 0 1 0 1 1 0 0 --------------- 1 0 0 0

AND for	**BYTE- variable**
Input_1 Input_2 ------------- Output_1	10100101 16#A5 11000101 16#C5 --------------- -------- 10000101 16#85

AND for	**WORD variable**
Input_1 Input_2 ------------- Output_1	16#A5F0 16#C5C3 --------------- 16#85C0

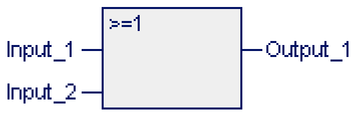
AND for	**DWORD variable**
Input_1 Input_2 ------------- Output_1	16#A5F0A5F0 16#C5C3C5C3 --------------- 16#85C085C0

AND for	LWORD variable
Input_1 Input_2 ------------- Output_1	16#A5F0 A5F0 A5F0 A5F0 16#C5C3 C5C3 C5C3 C5C3 --------------------------------------- 16#85C0 85C0 85C0 85C0

OR

The OR operation (Logic Instructions, Instructions of the IL, Table Overview) is carried out bit by bit.
If the OR operation is applied to BYTE, WORD, DWORD, bit positions of the same order are linked.

Operation	Activity
OR	OR operation of the current value with the value of the operand
OR>	OR operation of the current value with a pulse, with a 0-1 transition of the value of the operand (for Boolean variables only)
OR<	OR operation of the current value with a pulse, with a 1-0 transition of the value of the operand (for Boolean variables only))
ORN	OR operation of the current value with the bit-serial negated value of the operand
OR(	OR operation of the current value with the value of the following expression
ORN(	OR operation of the current value with the bit-serial negated value of the following expression
)	Termination of an expression

<table>
<tr><th>OR operation</th><th>Representation in the instruction list</th></tr>
<tr><td>
</td><td>LD   Input_1
OR   Input_2
ST   Output_1</td></tr>
</table>

OR for	Boolean variable (in LD shown as contact)
Input_1 Input_2 ------------- Output_1	1 0 1 0 1 1 0 0 --------------- 1 1 1 0

OR for	BYTE- variable
Input_1 Input_2 ------------- Output_1	10100101 16#A5 11000101 16#C5 --------------- -------- 11100101 16#E5

OR for	WORD- variable
Input_1 Input_2 ------------- Output_1	16#A5F0 16#C5C3 --------------- 16#E5F3

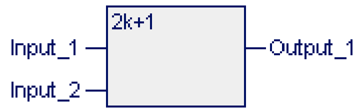
OR for	DWORD- variable
Input_1 Input_2 ------------- Output_1	16#A5F0A5F0 16#C5C3C5C3 -------------------- 16#E5F3E5F3

OR for	LWORD- variable
Input_1 Input_2 ------------- Output_1	16#A5F0 A5F0 A5F0 A5F0 16#C5C3 C5C3 C5C3 C5C3 --------------------------------------- 16#E5F3 E5F3 E5F3 E5F3

XOR

The XOR operation (Logic Instructions, Instructions of the IL, Table Overview) is carried out bit by bit.
If the XOR operation is applied to BYTE, WORD, DWORD, bit positions of the same order are linked.

Operation	Activity
XOR	XOR operation of the current value with the value of the operand
XORN	XOR operation of the current value with the bitwise negated value of the operand
XOR(	XOR operation of the current value with the value of the following expression
XORN(	XOR operation of the current value with the bitwise negated value of the following expression
)	Termination of an expression

XOR operation	Representation in the instruction list
	LD Input_1 XOR Input_2 ST Output_1

XOR for	Boolean variable
Input_1 Input_2 ------------- Output_1	1 0 1 0 1 1 0 0 --------------- 0 1 1 0

XOR for	BYTE variable
Input_1 Input_2 ------------- Output_1	10100101 16#A5 11000101 16#C5 --------------- -------- 01100000 16#60

XOR for	WORD variable
Input_1 Input_2 ------------- Output_1	16#A5F0 16#C5C3 --------------- 16#6033

XOR for	DWORD variable
Input_1 Input_2 ------------- Output_1	16#A5F0A5F0 16#C5C3C5C3 -------------------- 16#6033 6033

XOR for	LWORD variable
Input_1 Input_2 ------------- Output_1	16#A5F0 A5F0 A5F0 A5F0 16#C5C3 C5C3 C5C3 C5C3 --------------------------------------- 16#6033 6033 6033 6033

Jumps, Calls, Return (Conditional and Unconditional)

The programming system supports:

- The unconditional jump: JMP label
- Conditional jumps: JMPC label and JMPCN label (see JMP)
- The unconditional function block call CAL FBinstance
- The conditional function block calls CALC FBinstance and CALCN FBinstance (see CAL).
- The unconditional return from programs, function blocks and functions (see RET)
- The conditional return from programs, function blocks and functions RETC and RETCN RET)

(Instructions of the IL, Table Overview)

JMP

Jumps are elementary instructions for branching to the instruction list. They always lead to a jump destination, a label. With regard to the transition between instruction list, ladder diagram and function block language, a label can be only at the beginning of a network, before an LD or LDN instruction.

Jumps can be '*conditional*' or '*unconditional*'.

(Instructions of the IL, Table Overview, Jumps, Calls, Return (Conditional and Unconditional))

Operation		**Activity**
JMP	mLabel	Unconditional jump to the 'mLabel' label.
JMPC	mLabel	Jump to the 'mLabel' label, if the current value is TRUE.
JMPCN	mLabel	Jump to the 'mLabel' label, if the current value is FALSE.

Note: Jumps used in an instruction list must not result in endless loops!

The cancel condition for upward jumps has to be checked!

CAL

A CAL instruction allows a function block type assignment which was declared before to be called up within an instruction list.

The call can be '*conditional*' or '*unconditional*'.

(Instructions of the IL, Table Overview, Jumps, Calls, Return (Conditional and Unconditional))

Operation	**Activity**	
CAL	fb1	Unconditional call of the assignment 'fb1' of the function block of type xx.
CALC	fb1	Call of the assignment 'fb1' of the function block of type xx, if the current value is TRUE.
CALCN	fb1	Call of the assignment 'fb1' of the function block of type xx, if the current value is FALSE.

Example Call

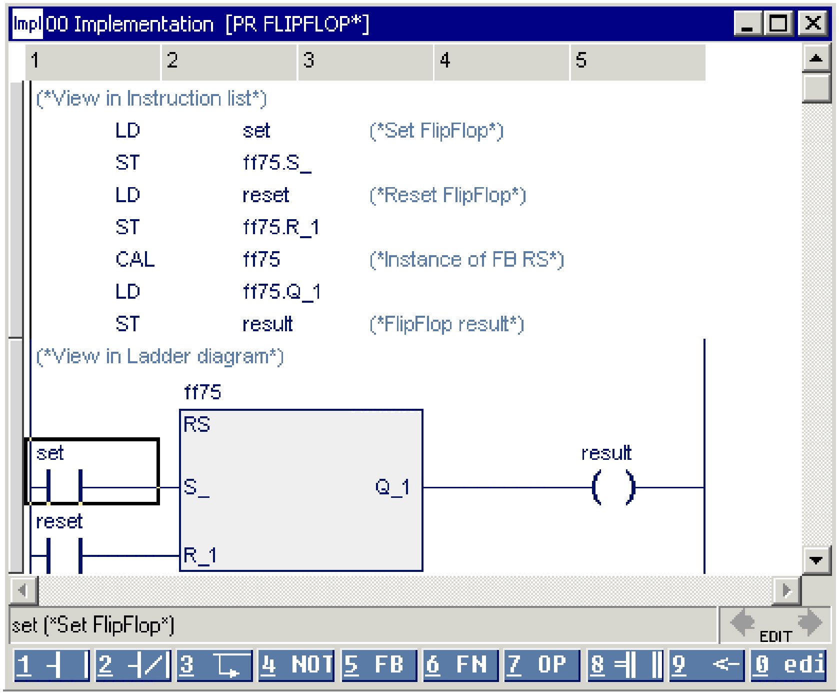
Declaration part of the program or the function block intended to use the RS flipflop ff75.

```
VAR
set        BOOL   (*Set FlipFlop*)
reset      BOOL   (*Reset FlipFlop*)
enable     BOOL   (*enable FlipFlop*)
result     BOOL   (*FlipFlop result*)
ff75       RS     (*Instance of FB RS*)
END_VAR
```

dekl UP Rufe.bmp

Fig. 6-27: Call types of function blocks in the declaration part

Unconditional call of a function block with ‘CAL’



Impl UP Rufe1.bmp

Fig. 6-28: Unconditional call of a function block in LD and IL

The required inputs were loaded before the call CAL ff75 (IL). A more complicated instruction list can be placed instead of the LD.

The executed outputs are available for retrieval after the call. If outputs of the ff75 are read already before being called, the user gets the old calculated value, possibly the initial value, if the block was not edited before.
If the call of ff75 is made conditional on a condition (ENABLE variable in the example), there are two possibilities:

Conditional call of a function block with 'CALC / CALCN'

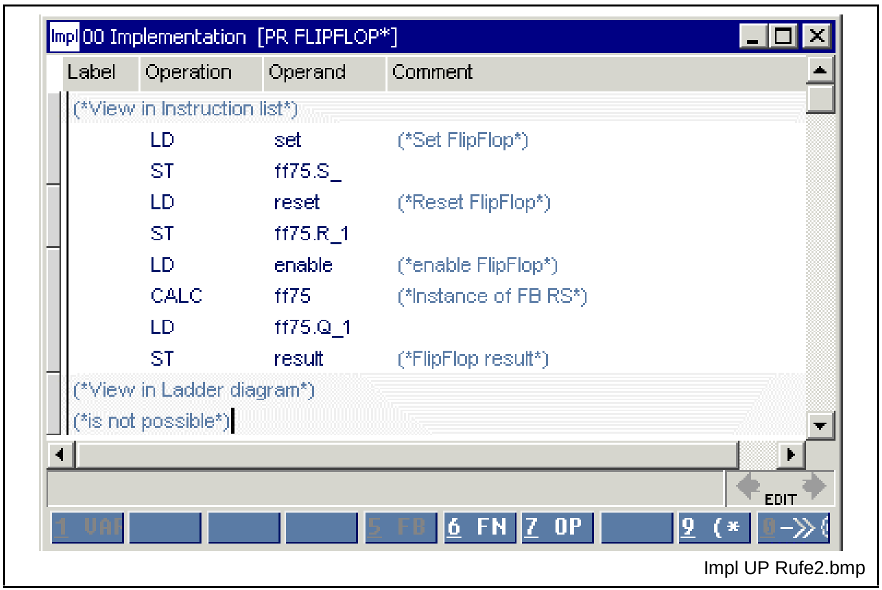


Fig. 6-29: Conditional call of a function block with "CALC / CALCN"

In the figure shown above the inputs are loaded, irrespective of whether the FB is called or not. The call takes place in dependence on the condition that stands before CALC / CALCN.
The entered IL cannot be represented graphically in the ladder diagram.
The second way of a conditional call is shown in the following figure. Loading of the inputs, the call and the supply of the outputs is skipped depending on the ENABLE requirement. The entered instruction list can be shown graphically in the LD.

Conditional call of a function block by skipping

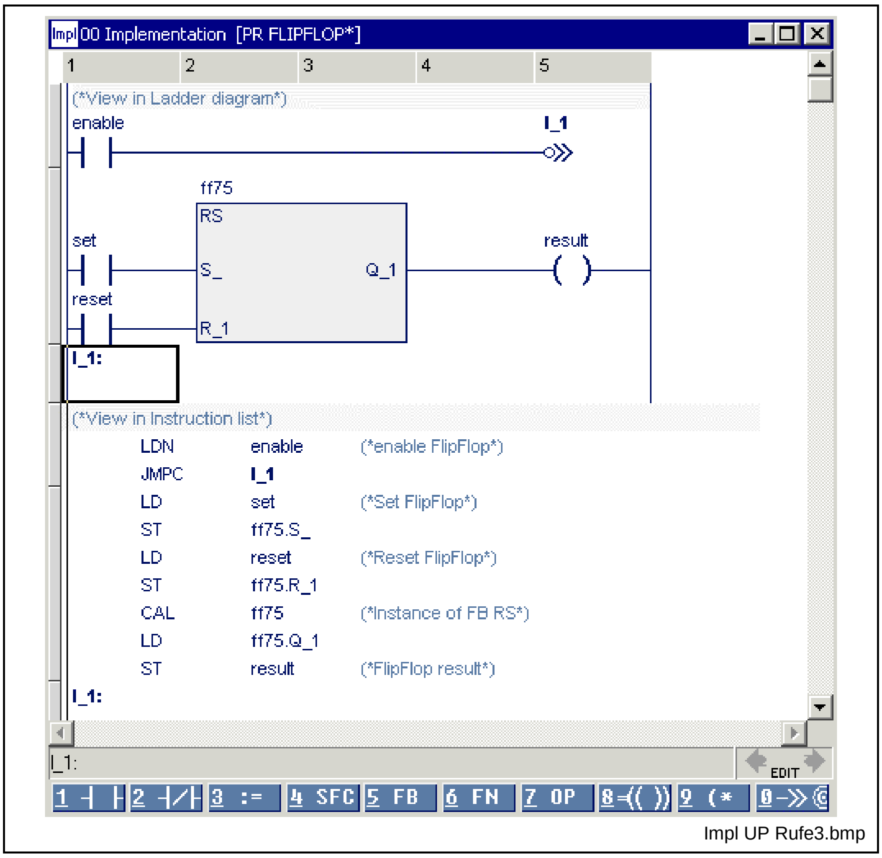


Fig. 6-30: Conditional call of a function block by skipping

RET

The execution of programs, function blocks or functions normally ends with the last instruction list line of its implementation without requiring a RET command.

However, the user can define in the IL another return destination, conditional or unconditional, with the following commands.

(Instructions of the IL, Table Overview, Jumps, Calls, Return (Conditional and Unconditional))

Note: A return command within an action block terminates the execution of the complete program organization unit (program or function block).

Operation	Activity
RET	Unconditional return from a program, function block or a function.
RETC	Conditional return from a program, function block or a function if the current value is TRUE.
RETCN	Conditional return from a program, function block or a function, if the current value is FALSE.

Arithmetic Instructions

Arithmetic instructions serve for linking numbers of the same type.

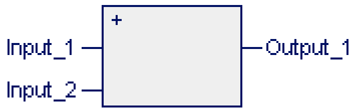
(Instructions of the IL, Table Overview)

Operation	Activity	Approved for data type
ADD	Value of an operand added to the current value.	All numbers, TIME, attaching of CHAR and STRING
ADD(	Value of the following expression added to the current value	All numbers, TIME
SUB	Value of an operand subtracted from the current value.	All numbers, TIME
SUB(	Value of the following expression subtracted from the current value	All numbers
MUL	Value of an operand multiplied by the current value.	All numbers
MUL(	Value of the following expression multiplied by the current value	All numbers
DIV	Current value divided by the value of the operand.	All numbers
DIV(	Current value divided by the value of the following expression	All numbers
MOD	Modulo division of the current value by the value of the operand.	All numbers except REAL
MOD(	Modulo division of the current value by the value of the following expression.	All numbers except REAL
)	Termination of the current expression.	

ADD

The arithmetic instruction Addition - ADD allows numbers of the same type to be added. The result is of the summand type.

(Instructions of the IL, Table Overview, Arithmetic Instructions)

ADD - Addition	**Representation in the instruction list**
	LD Input_1 ADD Input_2 ST Output_1

Example of the results (data types: **SINT / INT / DINT / LINT**)

Input_1	Input_2	Output_1	Error
	Sum less than pmin	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 3
-35	-7	-42	S#ErrorFlg: 0
+35	-7	28	S#ErrorFlg: 0
	Sum greater than pmax	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 2

Limits and S#ErrorTyp differ for SINT / INT / DINT **/** LINT:

Type	pmax	Pmin	S#ErrorTyp
SINT	127	-128	-10004
INT	32767	-32768	-10005
DINT	2147483647	-2147483648	-10006
LINT	9223372036854775807	-9223372036854775808	-10007

Example of the results (data types: **USINT / UINT / UDINT / ULINT**)

Input_1	Input_2	Output_1	Error
35	7	42	S#ErrorFlg: 0
	Sum greater than pmax	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 2

Limits and S#ErrorTyp differ for USINT / UINT / UDINT / LINT:

Type	pmax	S#ErrorTyp
USINT	255	-10000
UINT	65535	-10001
UDINT	4294967295	-10002
ULINT	18446744073709551615	-10003

Addition of REAL or LREAL numbers

The addition of REAL numbers occurs analogous to the ANY-INT numbers.

Overflow or other errors are indicated with S#ErrorFlg, S#ErrorTyp and S#ErrorNr as described in the help index "Errors with REAL Operations in Borderline Cases".

Addition of variables / constants of type TIME

If the time basis is 2 ms the physical limit for data type TIME corresponds to 99d10h5m34s590ms. Presently, this value is also indicated, if the variable of data type TIME contains the bit pattern 16#FFFFFFFF. Presently, the input is limited on 23d23h59m59s999ms.

If the limit is exceeded, you will receive:

S#ErrorFlg = TRUE, S#ErrorTyp = -10080 and S#ErrorNr = 2.

SUB

The arithmetic instruction Subtraction - SUB allows numbers of the same type to be subtracted. The result is of the input variable type.

(Instructions of the IL, Table Overview, Arithmetic Instructions)

SUB - Subtraction	Representation in the instruction list
	LD Input_1 SUB Input_2 ST Output_1

Example of the results (data types: **SINT / INT / DINT / LINT**)

Input_1	Input_2	Output_1	Error
	Difference less than pmin	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 3
-35	+7	-42	S#ErrorFlg: 0
+35	-7	+42	S#ErrorFlg: 0
	Difference greater than pmax	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 2

Limits and S#ErrorTyp differ for SINT / INT / DINT / LINT:

Type	pmax	pmin	S#ErrorTyp
SINT	127	-128	-10014
INT	32767	-32768	-10015
DINT	2147483647	-2147483648	-10016
LINT	9223372036854775807	-9223372036854775808	-10017

Example of the results (data types: **USINT / UINT / UDINT / ULINT**)

Input_1	Input_2	Output_1	Error
35	7	28	S#ErrorFlg: 0
	Difference less than 0	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 3

Limits and S#ErrorTyp differ for USINT / UINT / UDINT / ULINT:

Type	S#ErrorTyp
USINT	-10010
UINT	-10011
UDINT	-10012
ULINT	-10013

Subtraction of REAL or LREAL numbers

The subtraction of REAL numbers occurs analogous to the ANY-INT numbers.

Overflow or other errors are indicated with S#ErrorFlg, S#ErrorTyp and S#ErrorNr as described in the help index "Errors with REAL Operations in Borderline Cases".

Subtraction of variables / constants of type TIME

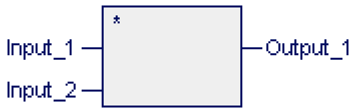
If the limit falls below 0 ms, you will receive:

S#ErrorFlg= TRUE, S#ErrorTyp= -10081 and S#ErrorNr= 3.

MUL

The arithmetic instruction Multiplication - MUL allows numbers of the same type to be multiplied. The result is of the factor type.

(Instructions of the IL, Table Overview, Arithmetic Instructions)

MUL - Multiplication	Representation in the instruction list
	LD Input_1 MUL Input_2 ST Output_1

Example of the results (data types: **SINT / INT / DINT / LINT**)

Input_1	Input_2	Output_1	Error
	Product less than pmin	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 3
-5	+7	-35	S#ErrorFlg: 0
+5	-7	-35	S#ErrorFlg: 0
-5	-7	35	S#ErrorFlg: 0
+5	+7	35	S#ErrorFlg: 0
	Product greater than pmax	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 2

Limits and S#ErrorTyp differ for SINT / INT / DINT / LINT:

Type	Pmax	Pmin	S#ErrorTyp
SINT	127	-128	-10024
INT	32767	-32768	-10025
DINT	2147483647	-2147483648	-10026
LINT	9223372036854775807	-9223372036854775808	-10027

Example of the results (data types: **USINT / UINT / UDINT / ULINT**)

Input_1	Input_2	Output_1	Error
5	7	35	S#ErrorFlg: 0
	Product greater than pmax	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 2

Limits and S#ErrorTyp differ for USINT / UINT / UDINT / ULINT:

Type	pmax	S#ErrorTyp
USINT	255	-10020
UINT	65535	-10021
UDINT	4294967295	-10022
ULINT	18446744073709551615	-10023

Multiplication of REAL or LREAL numbers

The multiplication of REAL numbers occurs analogous to the ANY-INT numbers.

Overflow or other errors are indicated with S#ErrorFlg, S#ErrorTyp and S#ErrorNr as described in the help index "Errors with REAL Operations in Borderline Cases".

DIV

The arithmetic instruction Division - DIV allows numbers of the same type to be divided. The result is the integer component and is of the input variable type.

(Instructions of the IL, Table Overview, Arithmetic Instructions)

Note: An assignment of the division with standard-initialized variables causes an error (division by zero)!

DIV - Division	Representation in the instruction list
	LD Input_1 DIV Input_2 ST Output_1

Input_1	Input_2	Output_1	Error
-35	+6	-5	S#ErrorFlg: 0
+35	-6	-5	S#ErrorFlg: 0
-6	+35	0	S#ErrorFlg: 0
-6	-35	0	S#ErrorFlg: 0
	Division by 0	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 5

S#ErrorTyp differs for the data types SINT / INT / DINT / LINT:

Type	S#ErrorTyp
SINT	-10034
INT	-10035
DINT	-10036
LINT	-10037

Example of the results (data types: **USINT / UINT / UDINT / ULINT**)

Input_1	Input_2	Output_1	Error
35	6	5	S#ErrorFlg: 0
6	35	0	S#ErrorFlg: 0
	Division by 0	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 5

S#ErrorTyp differs for the data types USINT / UINT / UDINT / ULINT:

Type	S#ErrorTyp
USINT	-10030
UINT	-10031
UDINT	-10032
ULINT	-10033

Division of REAL or LREAL numbers

The division of REAL numbers occurs analogous to the ANY-INT numbers.

Overflow or other errors are indicated with S#ErrorFlg, S#ErrorTyp and S#ErrorNr as described in the help index "Errors with REAL Operations in Borderline Cases".

MOD

The arithmetic instruction Modulo division - MOD allows the modulo division of two numbers with same type (modulo of the division DIV).

The result is of the input variable type.

(Not defined for REAL!)

(Instructions of the IL, Table Overview, Arithmetic Instructions)

Note: An assignment of the MOD division with standard-initialized variables causes an error (division by zero)!

MOD - Modulo division	Representation in the instruction list
	LD Input_1 MOD Input_2 ST Output_1

Input_1	Input_2	Output_1	Error
-35	+6	-5	S#ErrorFlg: 0
+35	-6	5	S#ErrorFlg: 0
-6	+35	-6	S#ErrorFlg: 0
-6	-35	-6	S#ErrorFlg: 0
	MOD division by 0	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 5

S#ErrorTyp differs for the data types SINT / INT / DINT / LINT:

Type	S#ErrorTyp
SINT	-10044
INT	-10045
DINT	-10046
LINT	-10047

Example of the results (data types: **USINT / UINT / UDINT /ULINT**)

Input_1	Input_2	Output_1	Error
35	6	5	S#ErrorFlg: 0
6	35	0	S#ErrorFlg: 0
	MOD division by 0	Not calculated	S#ErrorFlg: 1 S#ErrorNr: 5

S#ErrorTyp differs for the data types USINT / UINT / UDINT / ULINT:

Type	S#ErrorTyp
USINT	-10040
UINT	-10041
UDINT	-10042
ULINT	-10043

Comparators

Comparators are instructions for comparing operands and expressions with the current value:

- of numbers of the same type with regard to their size,
- of bit strings of the same type with regard to equality / inequality,
- of two characters (CHAR) or strings (STRING) with regard to their alphabetic order or
- of two time values (TIME) with regard to their size.

The result is a Boolean value.

The error variables S#ErrorFlg, S#ErrorNr, S#ErrorTyp are not affected, as no error may occur.

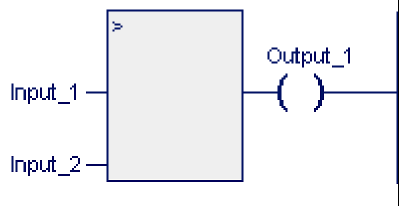
(Instructions of the IL, Table Overview)

Operation	Result			
GT GT(GE GE(EQ EQ(NE(LT LT(LE LE( )	Current value greater than operand: Current value greater than expression: Current value greater than or equal to operand: Current value greater than or equal to expression: Current value equal to operand: Current value equal to expression: Current value not equal to operand: Current value not equal to expression: Current value less than operand: Current value less than expression: Current value less than or equal to operand: Current value less than or equal to expression: Termination of an expression	Current value: Current value: Current value: Current value: Current value: Current value: Current value: Current value: Current value: Current value: Current value: Current value:	= = = = = = = = = = = =	TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE TRUE, else FALSE

GT

The comparator Greater than - GT supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is greater than the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Greater than	Representation in the instruction list
	LD Input_1 GT Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	1
5	5	0
3	5	0

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	0
’A‘	’a‘	0
’5‘	’3‘	1
’5‘	’5‘	0

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	0
’ABC‘	’‘	1
’ABC‘	’AB‘	1
’ABC‘	’ABC‘	0

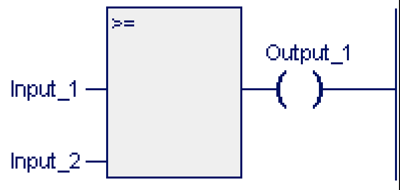
Comparison of times

Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	0
T#2ms	T#2ms	0
T#3ms	T#2ms	1

GE

The comparator Greater than or equal to - GT supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is greater than or equal to the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Greater than or equal to	Representation in the instruction list
	LD Input_1 GE Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	1
5	5	1
3	5	0

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	0
’A‘	’a‘	0
’5‘	’3‘	1
’5‘	’5‘	1

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	0
’ABC‘	‘‘	1
’ABC‘	’AB‘	1
’ABC‘	’ABC‘	1

Comparison of times

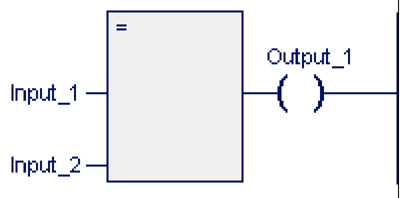
Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	0
T#2ms	T#2ms	1
T#3ms	T#2ms	1

EQ

The comparator Equal to - EQ supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is equal to the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Note: This comparison is also available for Boolean variables, BYTE, WORD, DWORD and LWORD!

Equal to	Representation in the instruction list
	LD Input_1 EQ Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	0
5	5	1
3	5	0

Note: Avoid comparisons of real numbers with number "0" since it is ambiguous!

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	0
’A‘	’a‘	0
’5‘	’3‘	0
’5‘	’5‘	1

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	0
’ABC‘	’‘	0
’ABC‘	’AB‘	0
’ABC‘	’ABC‘	1

Comparison of times

Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	0
T#2ms	T#2ms	1
T#3ms	T#2ms	0

Bit comparison

Input_1: BOOL	Input_2: BOOL	Output_1: BOOL
1	0	0
1	1	1
0	1	0
0	0	1

Comparison of bit strings (BYTE, WORD, DWORD, LWORD)

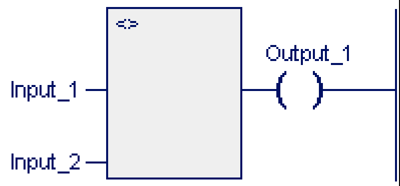
Input_1: BYTE	Input_2: BYTE	Output_1: BOOL
16#00	16#01	0
16#01	16#01	1
16#02	16#01	0

NE

The comparator Not equal to - NE supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is equal to the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Note: This comparison is also available for Boolean variables, BYTE, WORD, DWORD and LWORD!

Not equal to	Representation in the instruction list
	LD Input_1 NE Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	1
5	5	0
3	5	1

Note: Avoid comparisons of real numbers with number "0" since it is ambiguous!

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	1
’A‘	’a‘	1
’5‘	’3‘	1
’5‘	’5‘	0

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	1
’ABC‘	’‘	1
’ABC‘	’AB‘	1
’ABC‘	’ABC‘	0

Comparison of times

Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	1
T#2ms	T#2ms	0
T#3ms	T#2ms	1

Bit comparison

Input_1: BOOL	Input_2: BOOL	Output_1: BOOL
1	0	1
1	1	0
0	1	1
0	0	0

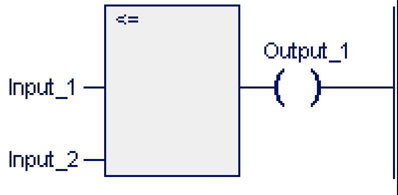
Comparison of bit strings (BYTE, WORD, DWORD, LWORD)

Input_1: BYTE	Input_2: BYTE	Output_1: BOOL
16#00	16#01	1
16#01	16#01	0
16#02	16#01	1

LE

The comparator Less than or equal to - LE supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is less than or equal to the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Less than or equal to	**Representation in the instruction list**
	LD Input_1 LE Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	0
5	5	1
3	5	1

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	1
’A‘	’a‘	1
’5‘	’3‘	0
’5‘	’5‘	1

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	1
’ABC‘	‘‘	0
’ABC‘	’AB‘	0
’ABC‘	’ABC‘	1

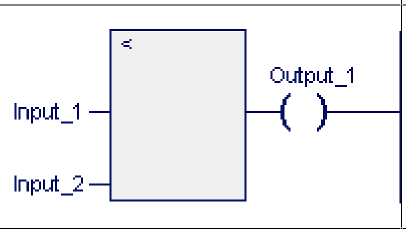
Comparison of times

Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	1
T#2ms	T#2ms	1
T#3ms	T#2ms	0

LT

The comparator Less than - LT supplies 1 at the Boolean output if the variable / constant at the upper input (current value) is less than the variable / constant at the lower input. Else, 0 is applied at the output.

(Instructions of the IL, Table Overview, Comparators)

Less than	**Representation in the instruction list**
	LD Input_1 LT Input_2 ST Output_1

Comparison of numbers (USINT, UINT, UDINT, ULINT, SINT, INT, DINT, LINT, REAL, LREAL)

Input_1: ANYNUM	Input_2: ANYNUM	Output_1: BOOL
5	3	O
5	5	0
3	5	1

Comparison of characters (alphabetic order)

Input_1: CHAR	Input_2: CHAR	Output_1: BOOL
’A‘	’B‘	1
’A‘	’a‘	1
’5‘	’3‘	0
’5‘	’5‘	0

Comparison of character strings (alphabetic order)

Input_1: STRING	Input_2: STRING	Output_1: BOOL
’ABC‘	’aBC‘	1
’ABC‘	‘‘	0
’ABC‘	’AB‘	0
’ABC‘	’ABC‘	0

Comparison of times

Input_1: TIME	Input_2: TIME	Output_1: BOOL
T#2ms	T#3ms	1
T#2ms	T#2ms	0
T#3ms	T#2ms	0

7 Ladder Diagram Editor

7.1 General Notes on the Ladder Diagram Editor

The ladder diagram editor serves for entering and modifying the program code in programs, function blocks and functions in the Edit and Online-Edit modes, as well as for indicating variable values at the end of a PLC cycle in the Status mode and for unchanged networks in the Online-Edit mode.

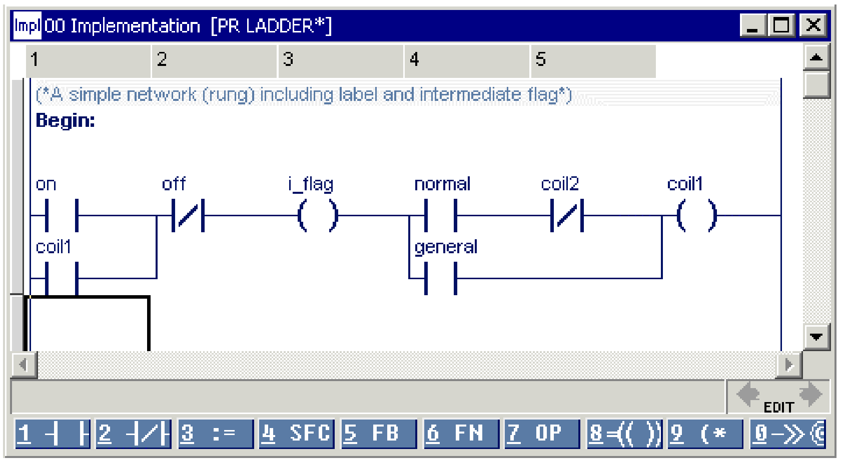
It allows ladder diagram symbols to be used in compliance with EN 61131-3 and indicates the text language "Instruction list" in graphic form. Most of the instruction list constructs can be converted into ladder diagram networks and vice versa with the <TAB> key.

7.2 Structure of a Ladder Diagram

The network is the smallest independent unit in the ladder diagram. It can consist of ladder diagram elements, such as relays, normally open contacts and normally closed contacts. Moreover, it can contain set and reset instructions and jumps.

A network can include temporary flags, functions and/or function blocks.

A network can further consist of a single-line or multi-line comment.



KOP_Netzwerk BSP_KOP.bmp

Fig. 7-1: Ladder diagram network with comment and label

Each network is limited by power rails to the right and left.

Horizontal connection lines

transfer the status from the immediately left to the immediately right neighboring element.

Vertical connection lines

can be reached by one or several connection lines coming from the left or from the right. A vertical connection line is logic "1" if one of the lines coming from the left is "1". The vertical line is "0" if all lines coming from the left are "0".

- Off: all lines coming from the left are "off".
- On: at least one line coming from the left is "on".

The status of the vertical line is passed on to the right.

Each network can be provided with a **(network) label** . This label has to start with a letter and ends with a ':' to the right.

Note: Bridge contacts cannot be realized!

7.3 Editing Ladder Diagrams

The starting point is an empty network. It is created by pressing the <ENTER> key:

- Placing the new network in front: <Enter> on left upper corner of the current network.
- Placing the new network behind: <Enter> on any other position in the network.

The footer with the active commands is indicated if you position the cursor on the grid position next to the left power rail. This footer is updated according to the position

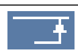
KOP_Fusszeile leeres Netzwerk.bmp

Fig. 7-2: Footer command in an empty network

Number	Pictogram	Commands
1	-| |-	Insert a normally open contact (1)
2	-|/|-	Insert a normally closed contact (1)
3	:=	Assignment in case of non-Boolean variables (1)
4	SFC	Call of an SFC
5	FB	Insert a function block instance , selection window (1)
6	FN	Insert a function, selection window (1)
7	OP	Insert an operation, selection window (1)
8	-(( ))-	Footer with additional terminating elements
9	(*	Edit a comment
0	->> (o)	Jump destination, insert / edit label

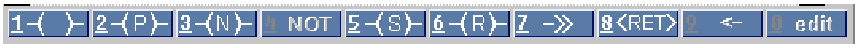
(1) In the enter mode, the graphic element selected is inserted to the right of the current position; in the overwrite mode, the current graphic element is replaced by the selected one.

The footer commands can be used to open branches to the networks and to close them at the required positions.

Number	Pictogram	Commands
3		Open a branch
3		Close a branch

Since more than 10 commands are available for individual positions in the ladder diagram, the following commands can be used to switch the footer:

Number	Pictogram	Command
8	-(( ))-	Footer with additional terminating elements
8	-|| ||-	Footer with additional contacts



KOP_Fusszeile zusätzliche Spulen.bmp

Fig. 7-3: Footer commands "additional terminating elements"

Number	Pictogram	Command
1	-( )-	Insert a coil (1)
2	-(P)-	Insert a coil which reacts to the positive edge (1)
3	-(N)-	Insert a coil which reacts to the negative edge (1)
4	NOT	Negates the current element (normally open contact ←→ normally closed contact)
5	-(S)-	Set variable in case of a 0->1 transition (1)
6	-(R)-	Reset variable in case of a 0->1 transition (1)
7	->>	(Conditional) jump (1)
8	<RET>	(Conditional) return jump from PR / FB or FN
9	<-	Place before (2)
0	Edit	Edit the current variable name

(1) In the enter mode, the graphic element selected is inserted to the right of the current position; in the overwrite mode, the current graphic element is replaced by the selected one.

(2) The graphic element to be selected is inserted before the current position.

KOP_Fusszeile zusätzliche Kontakte.bmp

Fig. 7-4: Footer with additional contacts

Number	Pictogram	Command
1	-| |-	Insert a normally open contact (1)
2	-|/|-	Insert a normally closed contact (1)
3	-( )-	Insert a temporary flag (coil) (1)
4	NOT	Negates the current element (normally open contact <-> normally closed contact)
5	-|P|-	Positive transition-sensing contact (positive edge) (0->1 transition) (1)
6	-|N|-	Negative transition-sensing contact (negative edge) (1->0 transition) (1)

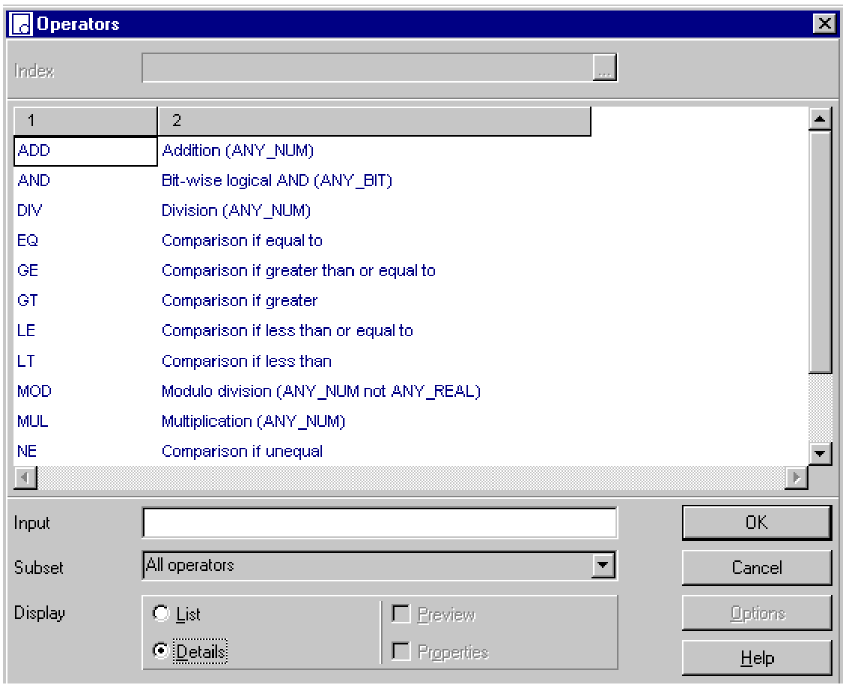
(1) In the enter mode, the graphic element selected is inserted to the right of the current position; in the overwrite mode, the current graphic element is replaced by the selected one.

Selection Window, Operators

The footer command "7-OP" opens the selection window for operators. They can be represented as list (space saving) or with detail information.

Furthermore, subsets can be selected:

- All operators
- Logic operators
- Boolean operators
- Arithmetic operators
- Comparison operators
- Jumps



OP_Auswahl.bmp

Fig. 7-5: Selection window, operators

The name of the desired operator can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

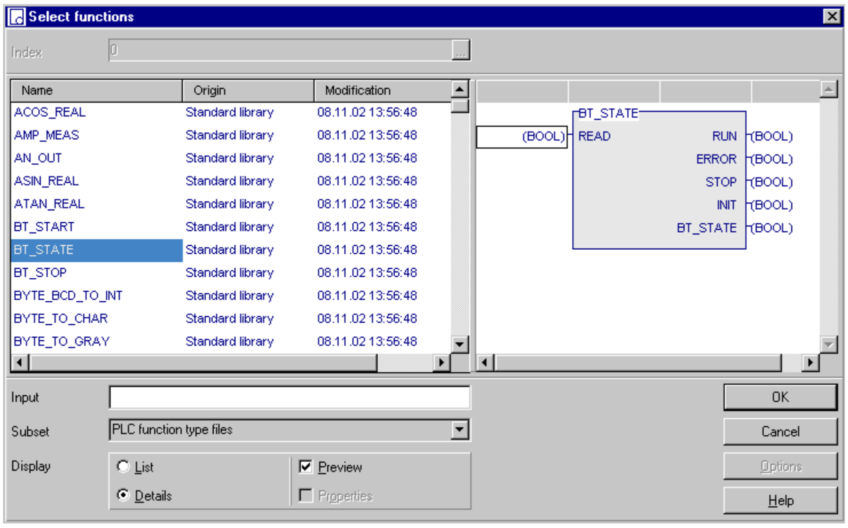
Within rungs the window can be restricted to operators with suitable syntax.

Selection Window, Functions

The footer command "6-FN" opens the selection window for functions. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the function (interface) appears.

As source the information "Standard library" / "Current work directory" is indicated.



FN_Auswahl.bmp

Fig. 7-6: Selection window, functions

The name of the desired function can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Within rungs the window can be restricted to functions with suitable syntax. Respectively, the upper input and the lowest output (main connections) of the function are used.

Selection Window, Instances of Function blocks

The footer command "5-FB" opens the selection window for already declared instances of function blocks. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the function block instances (interface) appears.

In addition to the name of the instance the declaration comment is indicated.

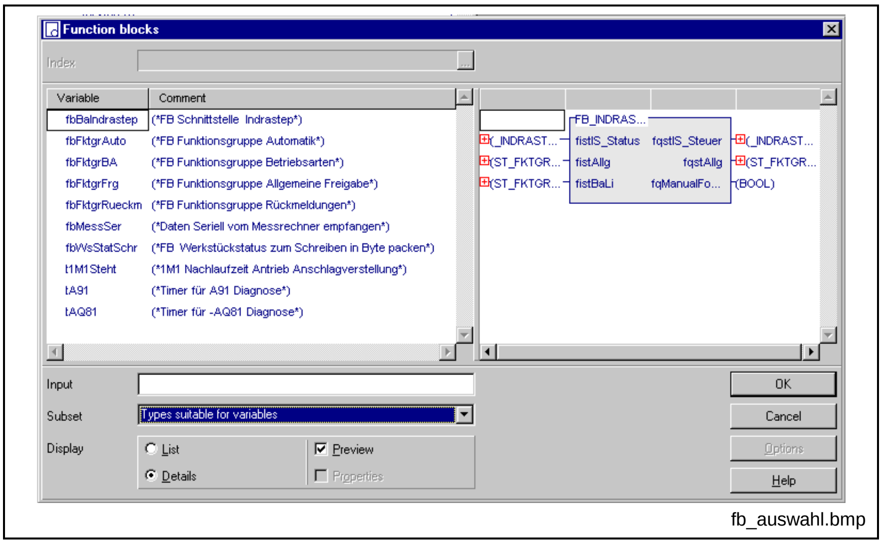


Fig. 7-7: Selection window, instances of function blocks

The name of the desired function block instance can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

As contrary to a function each input and output can be in connection with a network, one further selection window pops up, if there are several usable inputs and outputs of the same type.

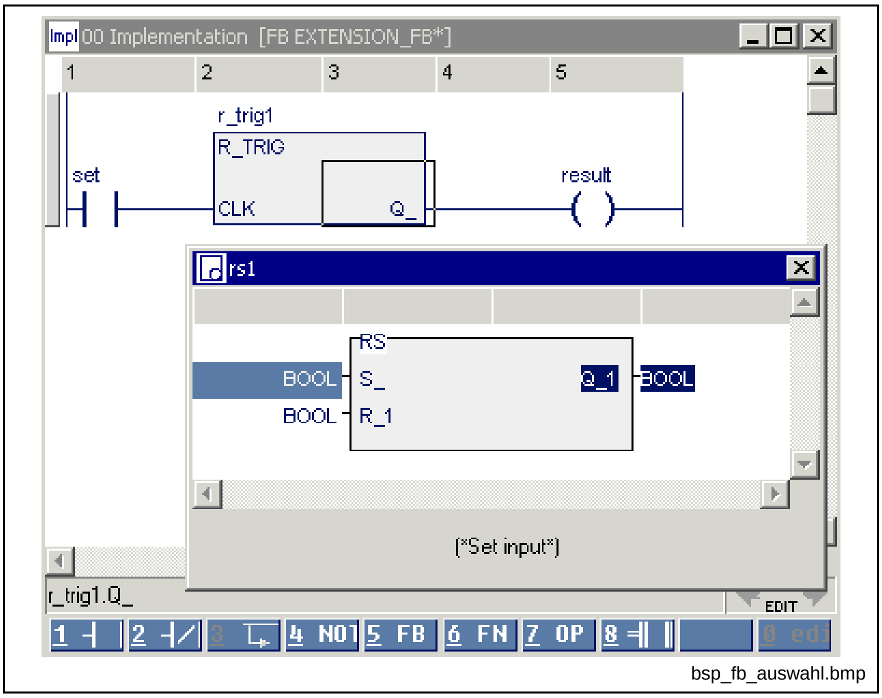
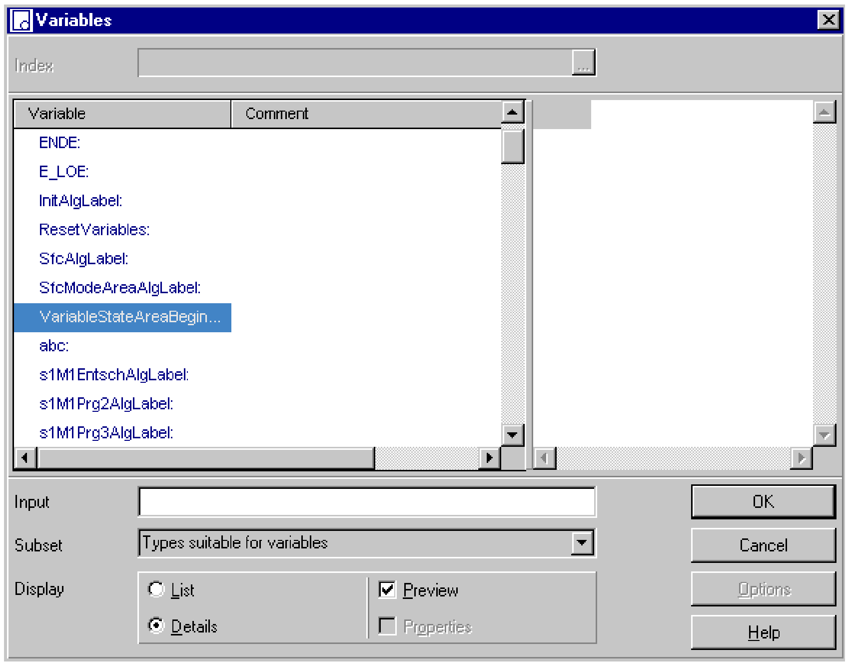


Fig. 7-8: Selection of the right connection

Selection Window, Label

The footer command "0-=>>(o)" appears, if the cursor in the right column is positioned on a jump symbol or a negated jump symbol.

The selection window for already declared labels opens. They can be represented as list (space saving) or with detail information.



Auswahl_Marke.bmp

Fig. 7-9: Selection window, label

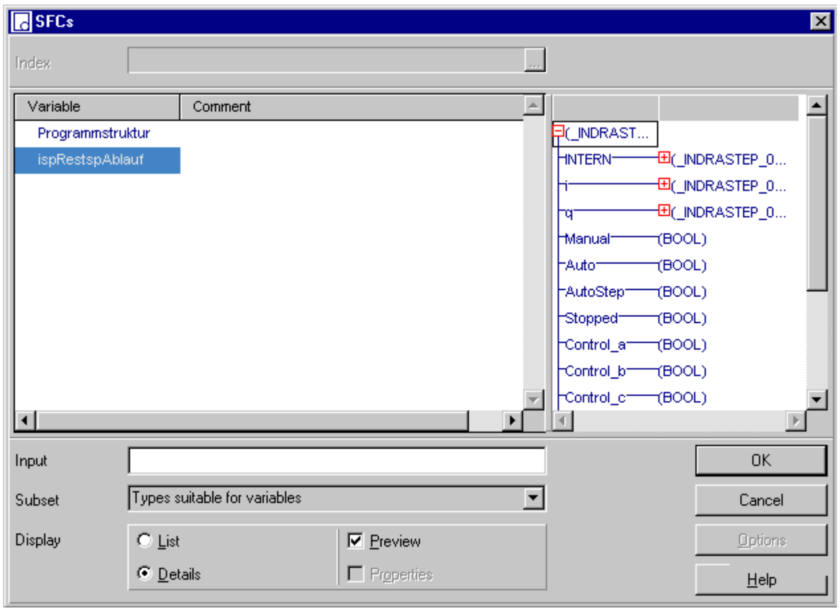
The name of the desired label can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, SFCs

The footer command "<4>-SFCs" opens the selection window for already declared SFCs. They can be represented as list (space saving) or with detail information.

If the option "Preview" is selected, a graphic representation of the SFC appears.

In addition to the name of the SFC its declaration comment is indicated.



SFC_Auswahl.bmp

Fig. 7-10: Selection window, SFCs

The name of the desired SFC can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Variables

Variable selection windows can only be called by contacts or coils or at the inputs or outputs of function blocks or functions by pressing <ALT>+<Enter>. They can be represented as list (space saving) or with detail information.

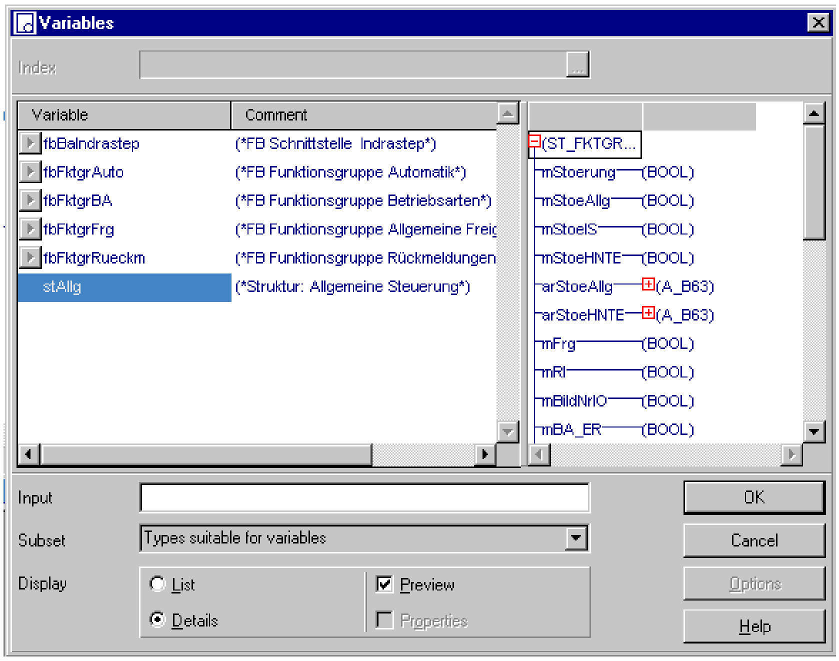
As subset you can select type-specific or all variables.

If the option "Preview" is selected, a graphic representation of the variable appears.

In addition to the name of the variable its declaration comment is indicated.

Multi-element variables (instances of structures, ARRAYs or FBs) are marked by a preceded gray arrow. You can open this variables by double-clicking or pressing <Enter> on the respective variable. The elements can be (recursively) selected.

In future, a switch for the return path upwards to the instance name is planned.



VAR_Auswahl.bmp

Fig. 7-11: Selection window, variables

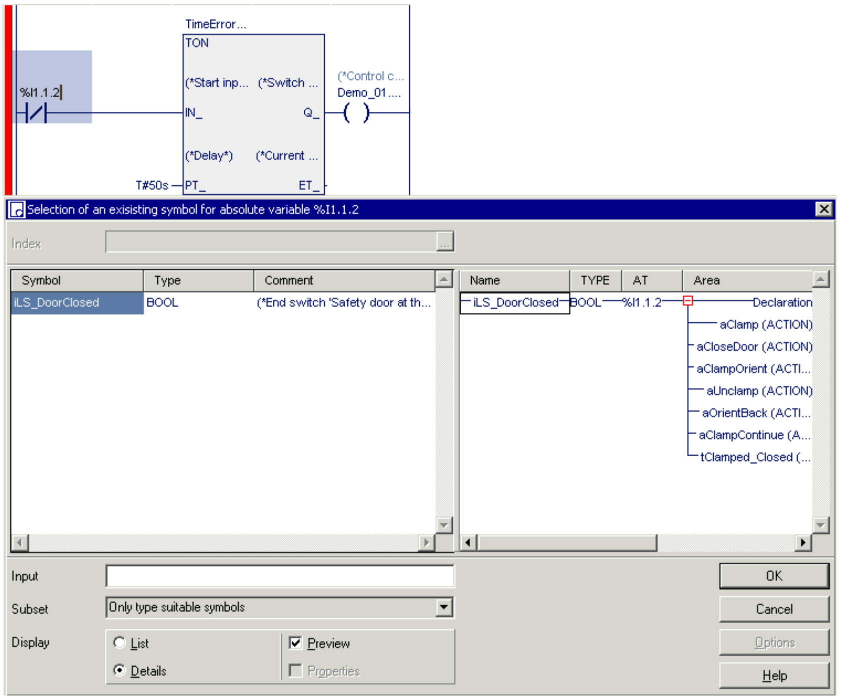
The name of the desired variable can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Rexroth Bosch Group

Selection Window, Absolute Addressed Variables

When entering an absolute address – e.g. %I1.1.2 – WinPCL verifies, if a name, that is linked with the address via a declaration exists in the validity area of the file. If a name is found, it is indicated in the selection window.

When confirming the element obtains this name, when rejecting the absolute address remains at the element.



Auswahlfenster_Abs_Var.bmp

Fig. 7-12: Window to select the absolute address

7.4 Deletion in the Ladder Diagram

The <Del> key can be used for deletions in the ladder diagram.

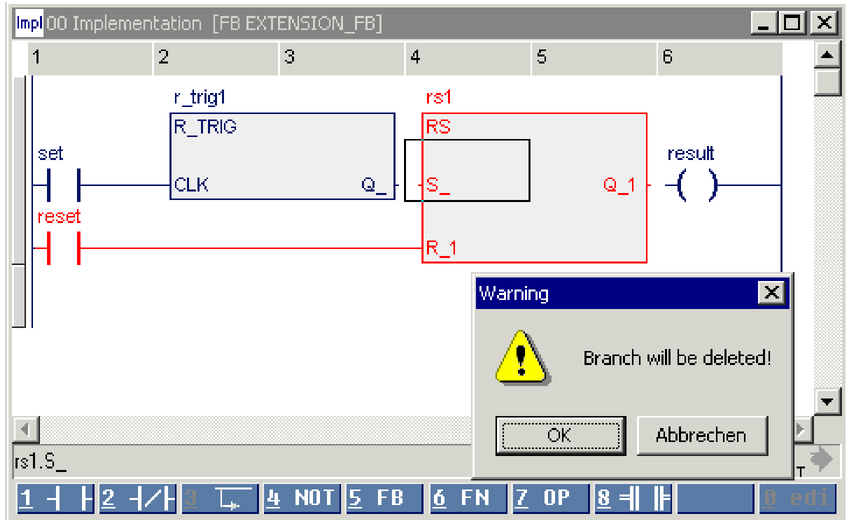
Deleting elements

In the network, any element currently selected by the cursor is deleted.

Deletion takes place as follows:

- <Del> of an element -> contact becomes the connection line,
- <Del> of a connection line -> the connection line, i.e. the branch is deleted.

The warning shown below is displayed if the cursor is positioned on a connection line, a function block or a function, that means if a serious "damage" is to be expected. The color of the elements affected changes to red.



KOP_Löschen im Netzwerk.bmp

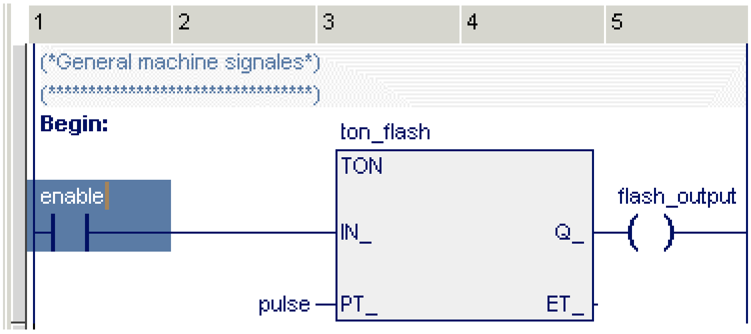
Fig. 7-13: Deletion in the network with the <Del> key

Deleting one or several networks

It is also possible to delete complete networks. To achieve this, the network(s) has(have) to be selected and then deleted with the <Del> key. The part to be deleted is clearly highlighted by the block selection.

7.5 Editing Features, Varying Color in the Ladder Diagram Editor

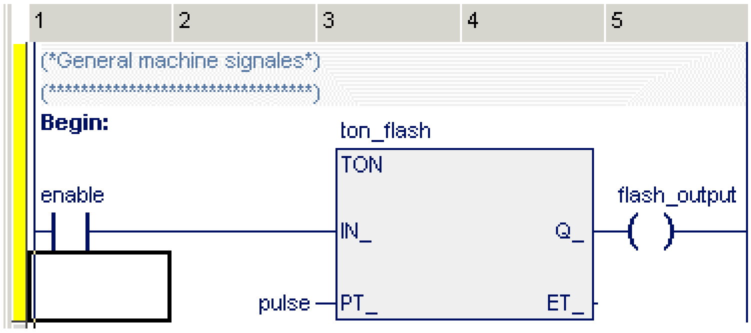
At first, that section of the network where editing takes place is white on a blue background during entry.



KOP_Eingabe_00.bmp

Fig. 7-14: Entry of an LD network, editing of a variable

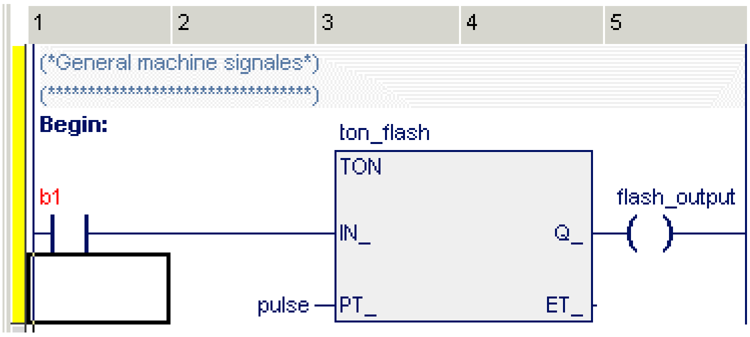
The color of the marginal marking to the left side changes from "gray", e.g. correct normal condition to "yellow", that means the network was modified. At this time the section is still untested.



KOP_Eingabe_01.bmp

Fig. 7-15: Entry of a an LD network, variables identified as being correct, network still untested, yellow marginal marking

Faulty networks, undeclared names or a combination of this, change their color to red when you exit the edited section or the edited line. If the error is not detected directly, position the cursor on the network and press <Ctrl>+<F1> for online help:



KOP_Eingabe_03.bmp

Fig. 7-16: Entry of a an LD network, operand identified as being faulty, network still untested, yellow marginal marking

The test whether neighboring lines in a network match is not carried out unless you exit the network, e.g. by moving the cursor out of the network. The marginal marking to the left changes its color from yellow to gray, that means everything is ok or to red, that means there is an error in the network. The basic font color is dark-blue (no errors), the comment is middle-blue, the left margin is gray.

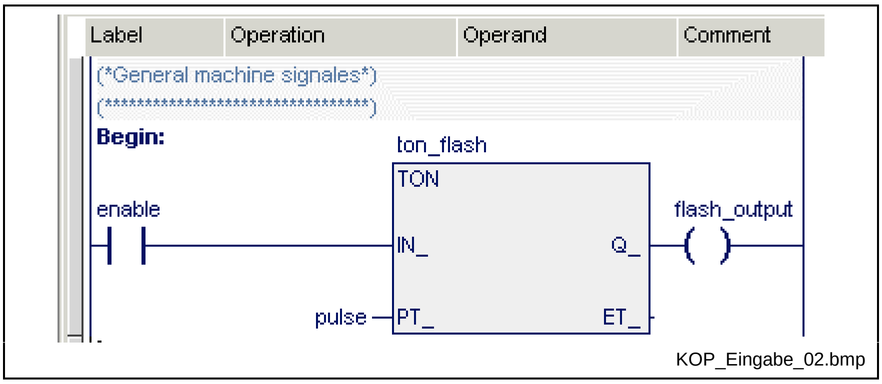


Fig. 7-17: Network without errors after editing, marginal marking is gray

If faulty networks or variables were not corrected before the network test, they are indicated in red and the marginal marking is indicated in red too.

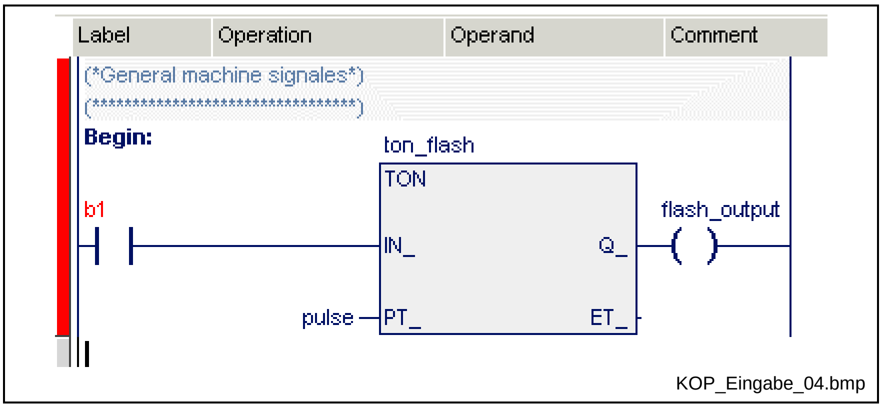
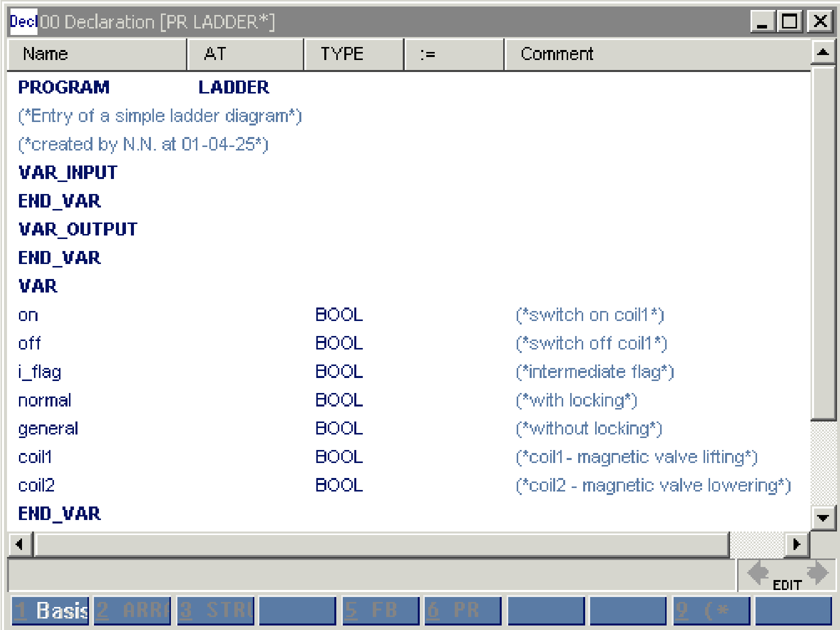


Fig. 7-18: Network faulty after editing, marginal marking is red

This ensures that errors remain visible. They must be eliminated before a successful compilation run can be started, but do not affect normal operation. A complete check, including labels and jumps, is carried out during the compilation attempt or the syntax test (Pop-up Menu, LD Editor <Shift>+<F10>).

Entry of a Simple Ladder Diagram

A simple ladder diagram network was chosen as an example for the entry. An appropriate declaration part containing the necessary variables is required:



kop_bsp_00.bmp

Fig. 7-19: Declaration part (example)

The input sequence below has to be followed (input mode).

Input sequence	**Comment**
1***on***<Enter>***coil1***<Enter>	Normally open contact "on", relay "coil1", cursor automatically behind normally open contact "on"
33***coil1***<Enter>	Normally open contact of "coil1" connected in parallel
<CursorUp><CursorRight>2***off***<Enter>	Normally closed contact "off"
1***normal***<Enter>	Normally open contact "normal"
2***coil2***<Enter>	Normally closed contact "coil2"
<CursorLeft>3<CursorRight>3***general***<Enter>	Parallel branch with normally open contact "general"
<CursorUp><CursorLeft>83***i_flag***<Enter>	Temporary flag at this point only, because otherwise it would be above the relay!
	Cursor on contact "on"
90***Begin***<Enter>	Label "Begin"

Similar to the instruction list, the declaration comment of the variable can also be made visible in connection with the elements of the ladder diagram. For this, position the cursor on the corresponding ladder diagram element.

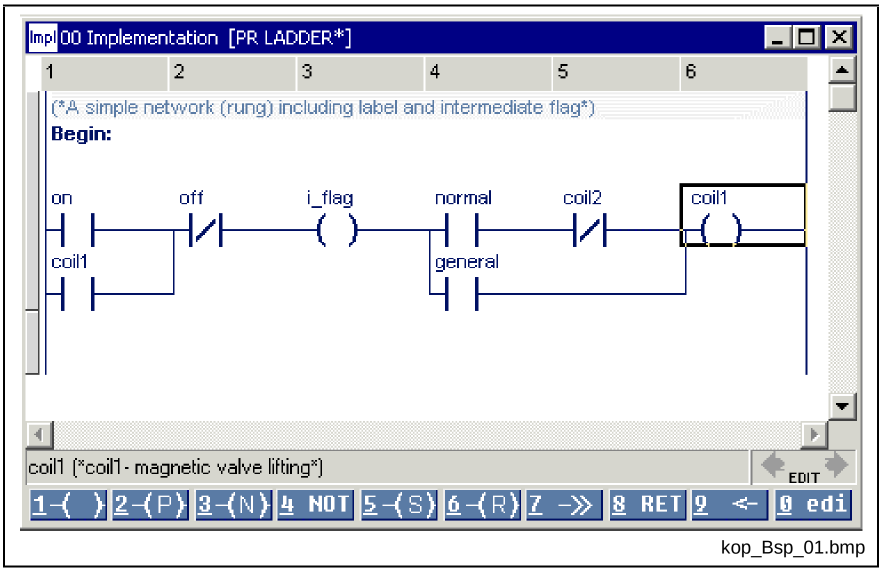


Fig. 7-20: LD with display of the declaration comment coil "coil1"

As already described in the chapter "Instruction list editor", the comment can be used unchanged or can be overwritten for the current position. To achieve this, press the <Ctrl>+<E> keys (edit comment) on the ladder diagram element. The original declaration comment becomes visible after deletion of this new comment.

Input sequence	**Comment**
	Cursor on relay coil1
<Ctrl>+<E>***coil1 locked with coil2*** <Enter>	Input of the implementation comment

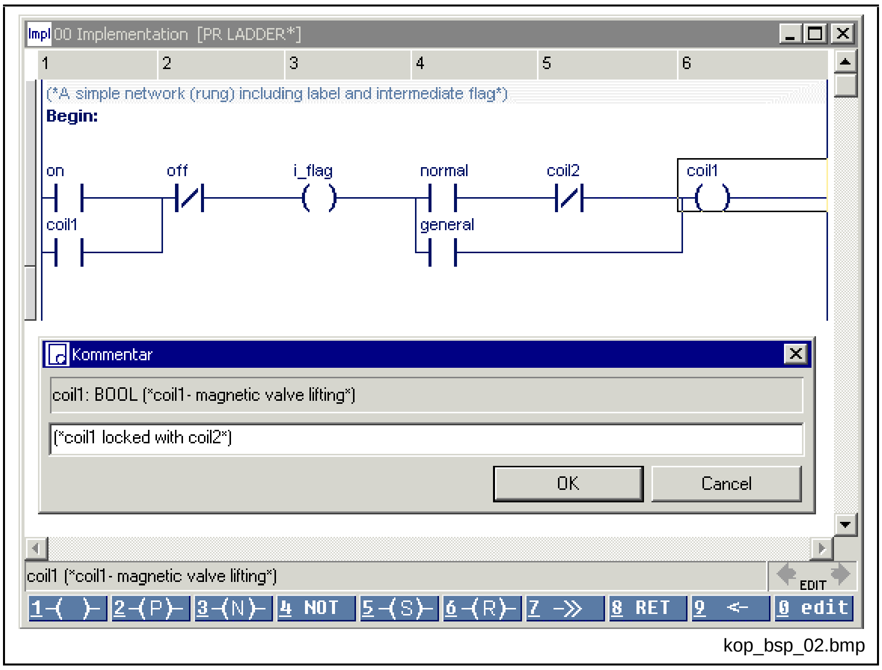


Fig. 7-21: LD, declaration comment for coil1 overwritten

The old declaration comment can still be seen in the upper part while the new comment is entered in the lower input window. Further methods to make comments visible in the LD editor are to be found under menu item "Extras / Options / View / LD".

Subsequent Modifications and Extensions in the Ladder Diagram

The following elements can be replaced by toggling the enter and overwrite mode:

Contacts (selection)

`1-¦ +- 2-¦/+-   4-NOT`

are mutually exchangeable. To achieve this:

- Position cursor on the required element and press the button in the footer (overwrite mode)

- or -

- Press the ’4-NOT’ key in the footer (mode: any).

Output elements (selection)

`1-( )- 4-NOT 5-(S)- 6-(R)- 7->> 8-<RET>`

are mutually exchangeable. To achieve this:

- Position cursor on the required element and press the button in the footer (overwrite mode);
  the ’4-NOT’ key modifies the current element.

The current name with the old symbol is removed if you enter a jump or return.

Supplementation of additional terminating elements

- Position the cursor on the first element and
  select the new element from the footer.

Overwrite mode:	The new element is entered at the place of the old one.
Enter mode:	The new element is placed to the right of the old one.
Enter mode:	’9-Place before. The new element is placed to the left of, i.e. before the old one.

Warning when using temporary flags

Temporary flags allow the determination of intermediate results from ladder diagram networks. The result of the branch positioned before the temporary flag is applied. The temporary flag ‘**i_flag**’ assumes the value of contact ‘**kC**’ (see instruction list!).

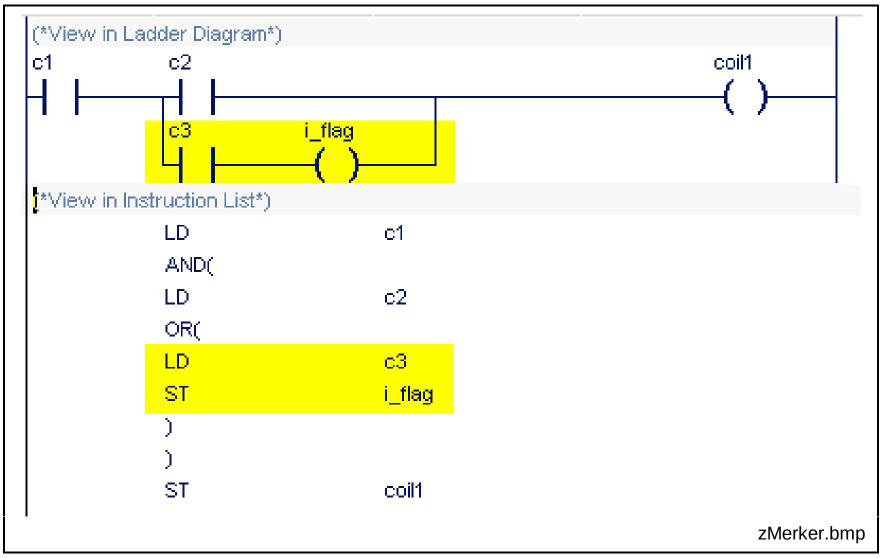


Fig. 7-22: Temporary flag

Note: This statement also applies to functions and function blocks which are used instead of the temporary flag!

If instead of the temporary flags functions or function block intances are used, the same unexpected behavior occurs:

Instead of triggering the contacts connected in series, the time stages are permanently triggered via 2#1 (TRUE).

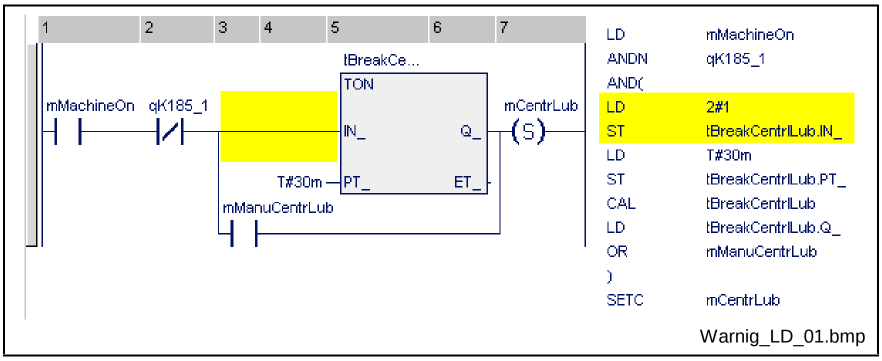


Fig. 7-23: Warning when using FN/FB in OR branches

This problem can be solved, when the switching is changed corresponding to the following figure:

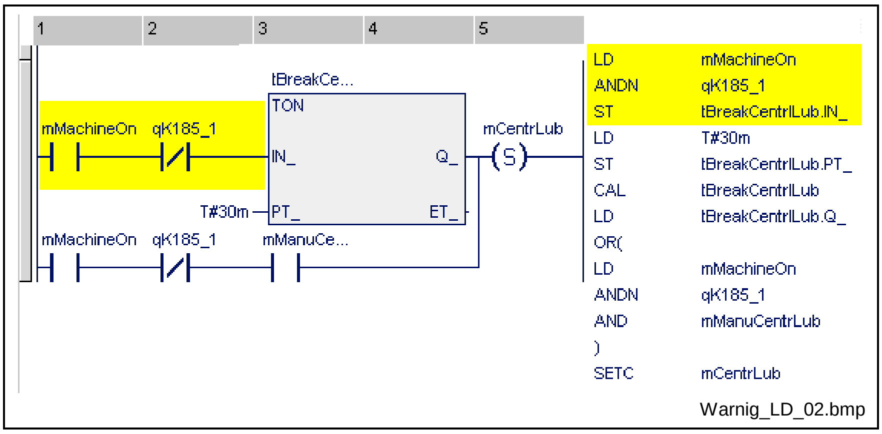
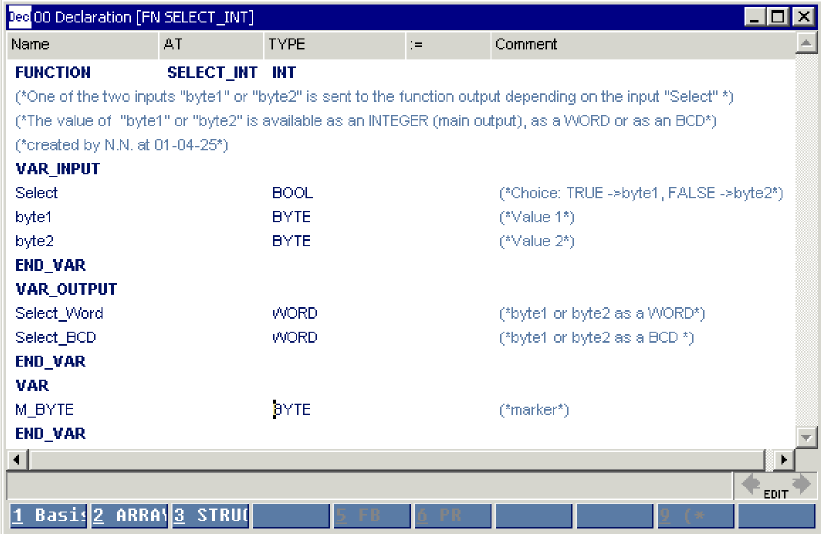


Fig. 7-24: Workaround when using FN/FB in OR branches

Entry of a Ladder Diagram with Additional Symbols

A repeated entry of the SELECT function in the ladder diagram allows display of the work with additional elements, such as jumps, non-Boolean assignments, type and code converters, and the like. A comment network is placed before. The declaration part of this ladder diagram shows the following structure:



select_int_dekl.bmp

Fig. 7-25: Declaration part of the function "SELECT_INT"

Input sequence	Comment
87***m_1***<Enter>1***Select***<Enter>	Cursor on jump to m_1
4<Enter>	Network with negated jump is complete
3***byte1***<Enter>***M_BYTE***<Enter><Enter>	Assignment of ‘byte1’ to ‘M_BYTE’
87***m_2***<Enter><CursorRight><Enter>	Unconditional jump in segment
0***m_1***<Enter>3***byte2***<Enter>***M_BYTE***<Enter><Enter>	Label ‘m_1’: Assignment of ‘byte2’ to ‘M_BYTE’
0***m_2***<Enter>6	Label ‘m_2’: Selection of the function via selection window

Position the cursor on the CONCAT_BYTE function in the selection window and confirm by pressing <Enter>.

Input sequence	Comment
16#0<Enter>***M_BYTE***<Enter> ***SELECT_BCD***<Enter>	Upper input 16#0, lower input ‘M_BYTE’; as output variable from end of network, in between filled-in!
96***INT_TO_BCD_WORD***<Enter>	Type converter from INT to BCD_WORD, ignore color variation! Cursor on output FN CONCAT_BYTE.
3***SELECT_INT***<Enter>	Temporary flag provides function value.
96***WORD_TO_INT***<Enter>	Type converter from WORD to INT (cursor is positioned on input ‚‘WORD_‘).
3***SELECT_WORD***<Enter>	Temporary flag provides WORD value.

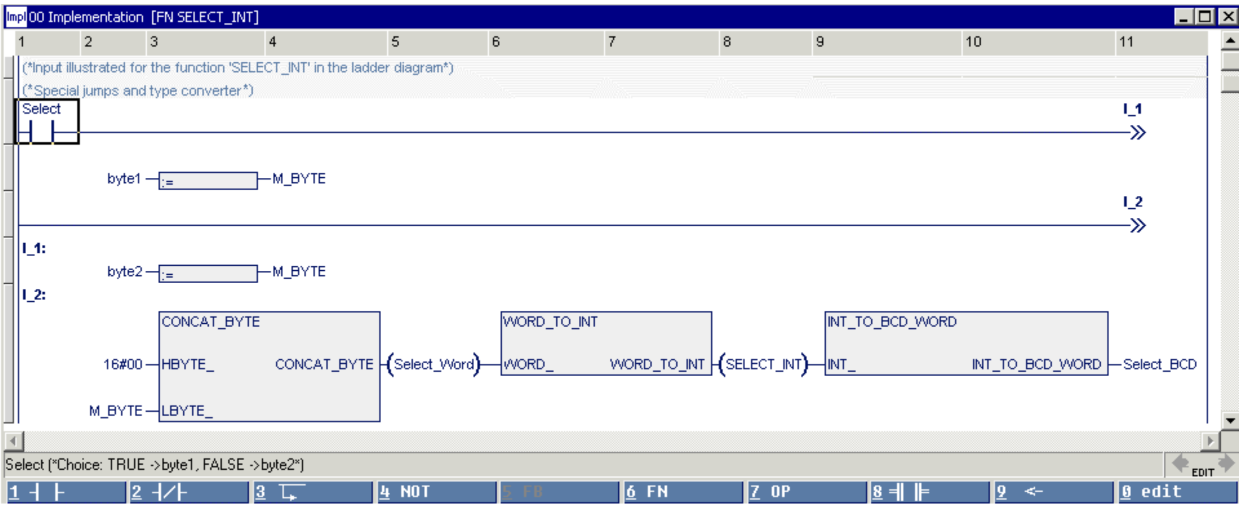
Finally a comment has to be entered before the first network.

Position the cursor on line 1, column 1, contact ‘Select’, by pressing the <Ctrl>+<Home> keys or dragging the cursor directly.

Input sequence	Comment
99***Input illustrated for the function ’SELECT_INT’ in the ladder diagram***<Enter>	Place before, comment, <Enter> for terminating the line, new comment line is opened.
Special jumps and type converter <CursorDown>	Comment line is terminated with exit.

If a network is to be inserted between the two comment lines, open the pop-up menu by pressing the right mouse button or <Shift>+<F10> and select the "Separate network" menu item.

Note: The name of the variable can be entered directly or a selection window can be opened in the empty field by pressing <Alt>+<Enter>.



select_int_kop.bmp

Fig. 7-26: Ladder diagram for function "SELECT_INT"

Edge Contacts and Edge Coils in the Ladder Diagram

It is often possible to simplify networks in a ladder diagram or at least to represent them more clearly, if edge contacts are used.

Here, the transition from a contact to its edge contact and vice versa is an online change (and, analogously, from a coil to its edge coil).

Contacts for Detection of Transitions (Edges)

At the particular place where it is used, the "P" contact replaces the combination of contact and instance of an R_TRIG function block. The state to the right of the edge contact is TRUE from one execution process to the next, if TRUE is applied to the left of the contact and the value of the variable of the edge contact changes from FALSE to TRUE.

At the particular place where it is used, the "N" contact replaces the combination of contact and instance of an F_TRIG function block. The state to the right of the edge contact is TRUE from one execution process to the next, if TRUE is applied to the left of the contact and the value of the variable of the edge contact changes from TRUE to FALSE.

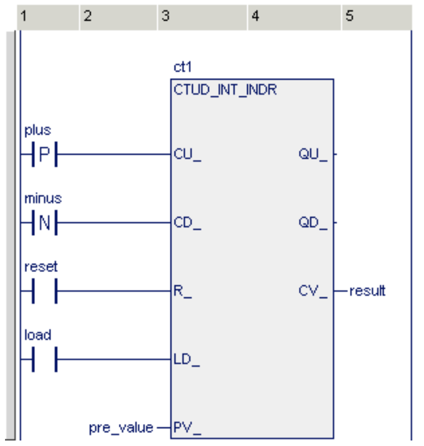
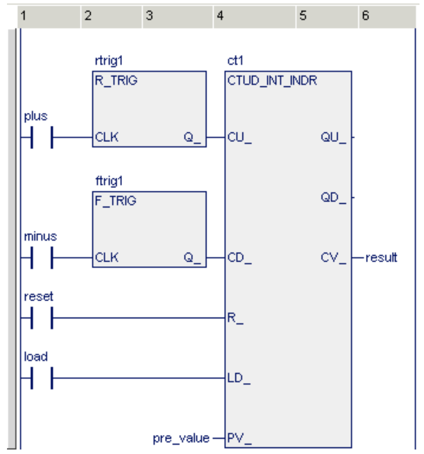
Edge contacts	**with R_TRIG / F_TRIG**
	

Fig. 7-27: Comparison of edge contacts and R_TRIG / F_TRIG

Note: A separate old value is assigned to each use of an edge contact!

The pulse is active for exactly one PCL cycle, i.e. the network must be executed at least twice.

Coils for Detection of Transitions (Edges)

The state of a variable pertaining to an edge coil is TRUE from one network execution process to the next if

- there is a FALSE-TRUE transition to the left of a P coil,
- there is a TRUE-FALSE transition to the left of an N coil.

Note: A separate old value is assigned to each use of an edge coil!

The pulse is active for exactly one PCL cycle, i.e. the network must be executed at least twice.

Online Changes and Edge Evaluation, LD

If contacts are converted into edge contacts or coils for detecting edge transitions, this results in the following transitions:

Rules for contacts:

- If a contact is converted into a contact with edge evaluation, it assumes an old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the contact.
- If a contact with "P" edge evaluation is converted into a contact with "N" edge evaluation, the old value is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the contact. (This also applies to conversion of N into P!)
- If a contact with "P" edge evaluation is converted into another contact with "P" edge evaluation (or N into N), ***the current old value remains unchanged!*** This value and the current value of the variable form the basis of the behavior of the contact.
- A new variable assumes a new old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the contact.
- An inserted new contact for edge evaluation assumes an old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the contact.

Changing the contact type and the value of the variable

New	Same variable				NEW variable			
Old	-|P|-	-|N|-	**-|P|-**	**-|N|-**	-|P|-	-|N|-	**-|P|-**	**-|N|-**
-| |-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-|/|-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-|P|-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-|N|-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-| |-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-|/|-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-|P|-	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	0-1-0	FALSE
-|N|-	FALSE	0-1-0	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE

Fig. 7-28: Online change if contacts are concerned

Explanation:

-| |- Contact: Normally open contact, state of the variable FALSE

-|P|- Contact: positive edge, state of the variable TRUE, etc. ..

Rules for coils:

- If a coil is converted into a coil with edge evaluation, it assumes an old value which is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the coil.
- If a coil with "P" edge evaluation is converted into a coil with "N" edge evaluation, the old value is initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the coil. (This also applies to conversion of N into P!)
- If a coil with "P" edge evaluation is converted into another coil with "P" edge evaluation (or N into N), ***the current old value remains unchanged!*** This value and the current value of the variable form the basis of the behavior of the coil.
- ***A new variable assumes the old value of its predecessor.*** This value and the current value of the variable form the basis of the behavior of the coil.
- An inserted new coil for edge evaluation assumes an old value initialized with FALSE. This value and the current value of the variable form the basis of the behavior of the coil.

Changing the coil type and the value of the variable

New	Same variable				NEW variable			
Old	-(P)-	-(N)-	**-(P)-**	**-(N)-**	-(P)-	-(N)-	**-(P)-**	**-(N)-**
-( )-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-(/)-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-(P)-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-(N)-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-( )-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-(/)-	FALSE	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE
-(P)-	FALSE	0-1-0	FALSE	FALSE	FALSE	0-1-0	FALSE	FALSE
-(N)-	FALSE	0-1-0	0-1-0	FALSE	FALSE	0-1-0	0-1-0	FALSE

Fig. 7-29: Online change if coils are concerned

Explanation:

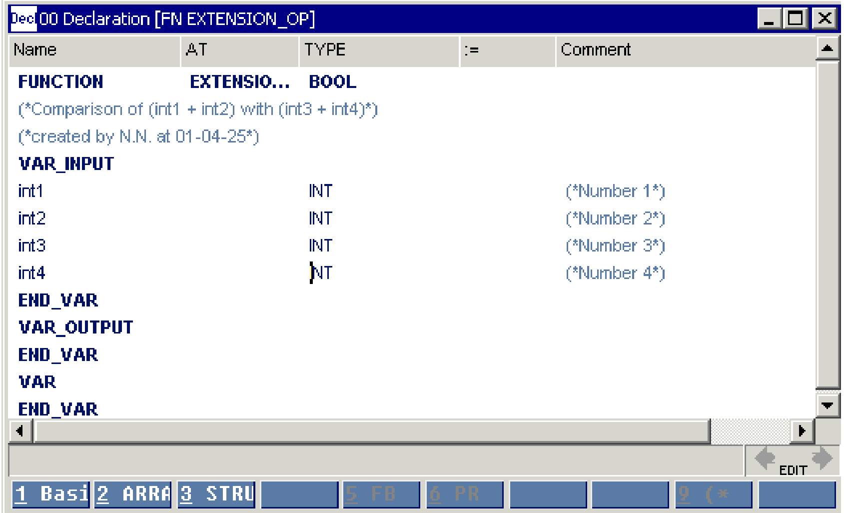
-( )- Coil: normal, state of the variable FALSE

-(P)- Coil: positive edge, state of the variable TRUE, etc. ..

Operators in the Ladder Diagram

Ladder diagram networks can be opened and supplemented with operators. An operator is selected with the footer command '7-OP'. To achieve this, position the cursor in the selection window on the required operator position and confirm by pressing the <Enter> key.

The example here is a function. The declaration part is the starting point.



Vergl_Sum_dekl.bmp

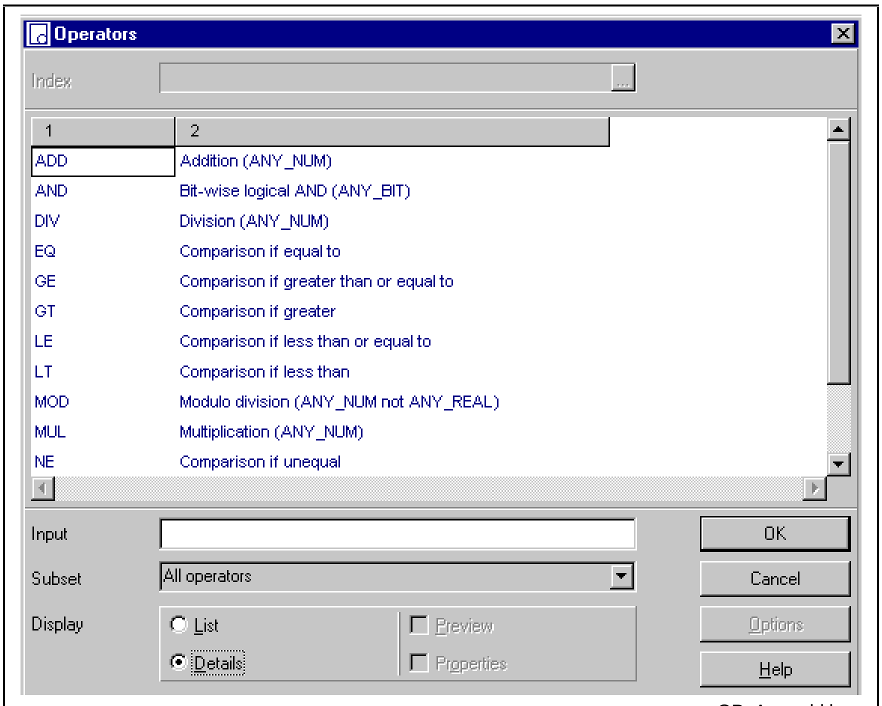
Fig. 7-30: Declaration part of the function "EXTENSION_OP"

After having completed the declaration part, change to the ladder diagram editor.

Input sequence

Input sequence	Comment
9***Comparison of two sums of INT figures***<CursorDown>	Comment input, completed by moving the cursor
7	Selection window for operators

Position the cursor in the selection window on the operator "EQ" and confirm with <Enter>.

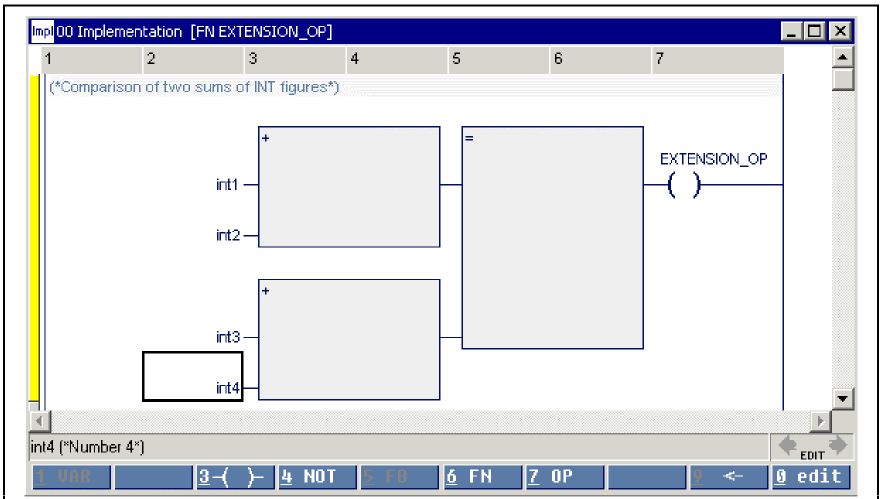


OP_Auswahl.bmp

Fig. 7-31: Selection window for operators

Input sequence	Comment
INT1<Enter>	First operand of the upper sum
INT3<Enter>	First operand of the lower sum
EXTENSION_OP<Enter>	Function output value, cursor to "INT1"
7	Selection of the adder, confirm with <Enter>
INT2<Enter>	Second operand of the upper sum, cursor to INT3
7	Selection of the adder, confirm with <Enter>
INT4<Enter>	Second operand of the lower sum

Note: The name of the variable can be entered directly or a selection window can be opened in the empty field by pressing <Enter>.



Vergl_Sum_kop.bmp

Fig. 7-32: Ladder diagram for "***EXTENSION_OP***", margin yellow, untested

Functions in the Ladder Diagram

As is the case with operators, several functions can be arranged in a network, with the special feature that a network can be connected only to the upper input and the lower output.

Temporary type incompatibilities can be ignored. The only fact to be observed is the requirement that the left marginal marking has to change from yellow or red to gray when the input is completed and the network exited. If this requirement is not met, press <Ctrl>+<F1> for online help.

Example

The WINDOW function FENSTER checks whether the VALUE is between MIN and MAX. If this is the case, the result is "1", if this is not the case, the result is "0".

The VALUE is composed of SUMMAND1 and SUMMAND2.

As SUMMAND1 is available as a four-digit BCD-coded number, it has to be converted into an integer number.

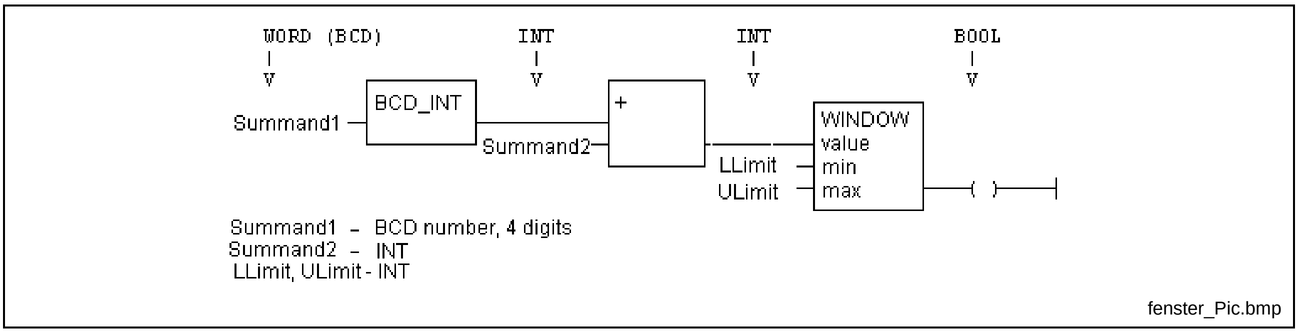


Fig. 7-33: Example for functions and operations

The user-specific "Window" function has to be made available before it can be used. At least the declaration part of this function, that means the interface, must exist, the implementation can be supplied later.

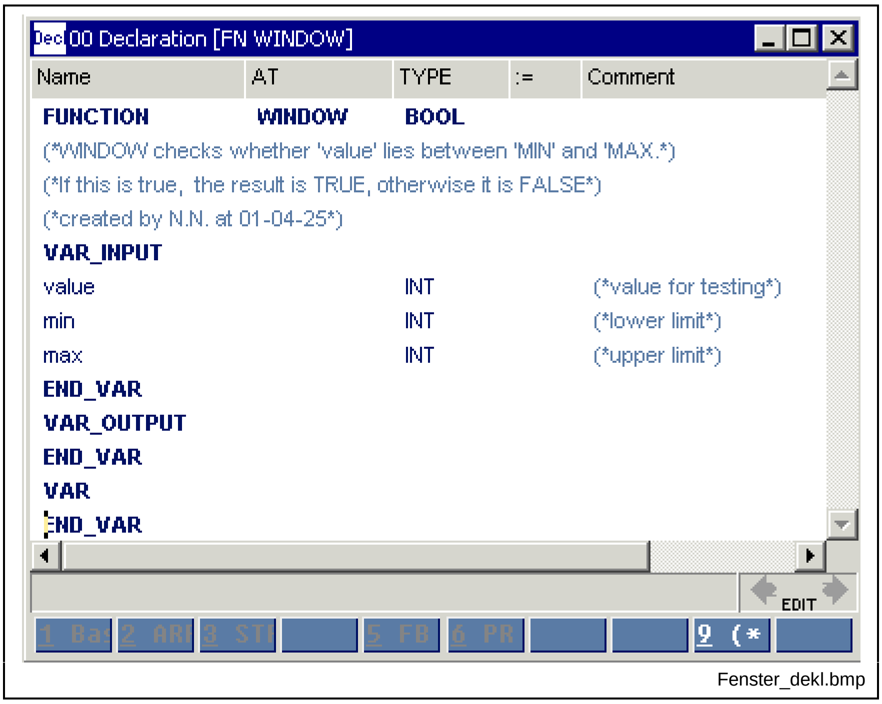
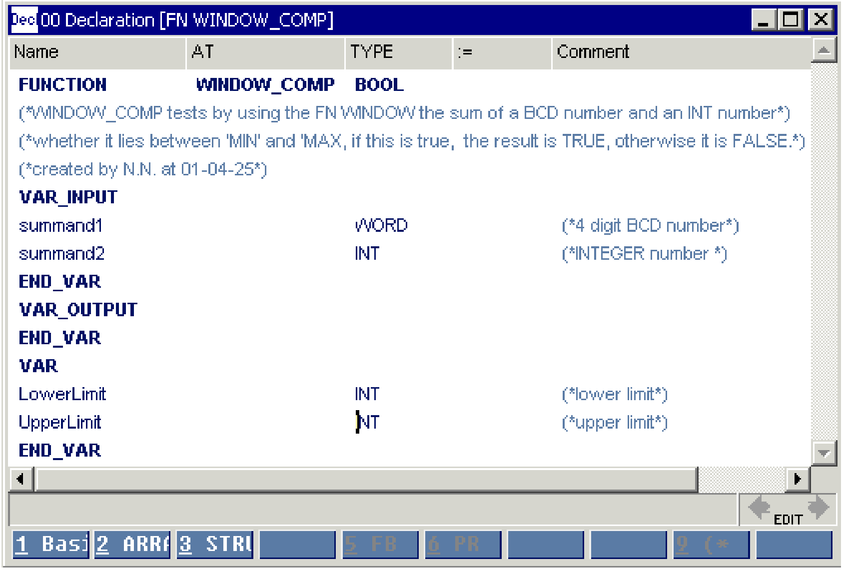


Fig. 7-34: Declaration part of the "Window" function

Rexroth Bosch Group

Declaration part of the example



fenster_vgl_dekl.bmp

Fig. 7-35: Declaration part of the "WINDOW_COMP" example

Enter the "Window" function, always starting with the most complicated point. Then do the extensions to the right or to the left.

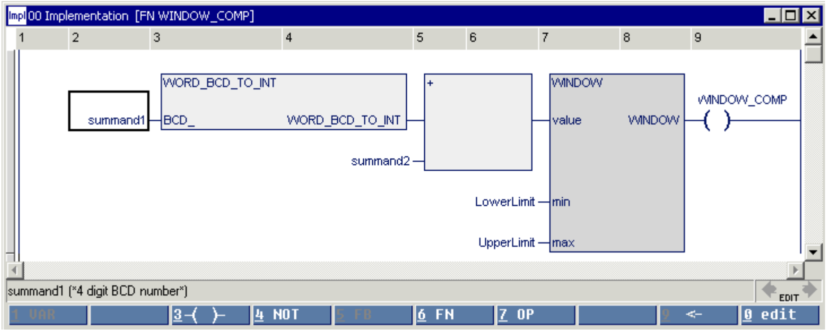
Input sequence	**Comment**
6	FN selection window, select "WINDOW" and confirm with <Enter>
Summand1<Enter>	Enter the highest summand, ignoring type errors.
LowerLimit<Enter>	Lower limit value
UpperLimit<Enter>	Upper limit value
WINDOW_COMP	Function value as result

If you exit the network with this status, the color of the marginal marking changes to red. A type error occurred between the first input and the "WINDOW" function. If you cannot identify the error, call up the pop-up menu by pressing the right mouse button or the <Shift>+F10 keys, or directly by pressing <Ctrl>+<F1> for online help.

Then you can continue the entry.

Input sequence	**Comment**
	Cursor on summand1
6	FN selection window, select "WORD_BCD_TO_INT", press <Enter>, Position cursor on output of FN "WORD_BCD_TO_INT"
7	OP selection window, select "ADD", <Enter>
Summand2<Enter>	Enter the second summand.

If you exit the network now, the color of the marginal marking changes from red to gray, i.e. the input is correct.



fenster_vgl_kop.bmp

Fig. 7-36: Ladder diagram of FN "WINDOW_COMP"

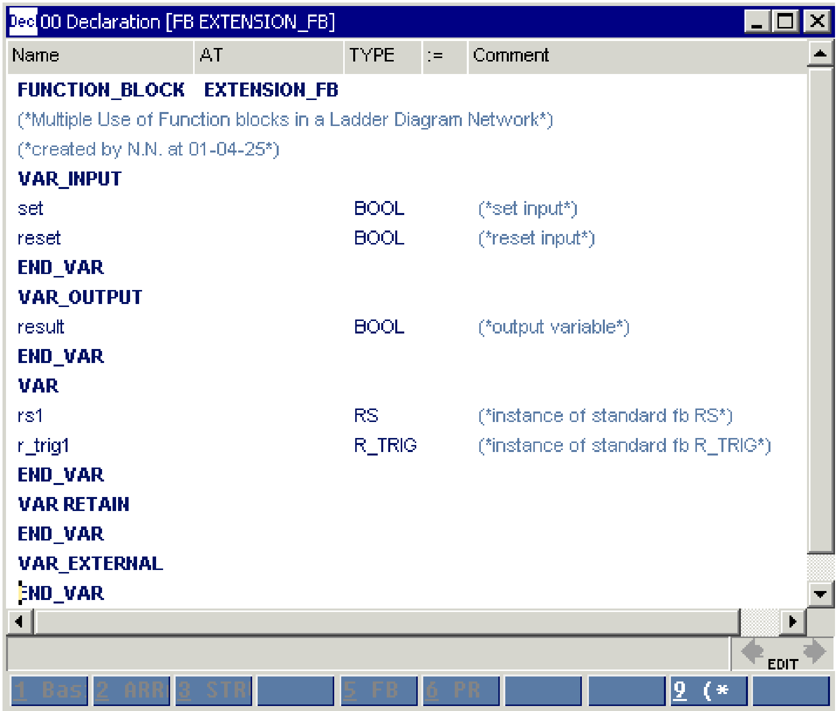
If the cursor is positioned on the name of a function which was written by the user, branching to this function is possible by pressing the <Ctrl>+<Enter> keys or double-clicking the mouse.

Function Blocks in the Ladder Diagram

The ladder diagram editor allows several function blocks per network. They can be inserted in the same way like operations and functions.

Note: Function blocks can have networks at each input and output. If they are inserted in existing networks, the user can choose the desired one from a selection window.

The basic principles of the call-up will be explained by the example of an RS flipflop, whose setting input should react to the edge of the input signal.

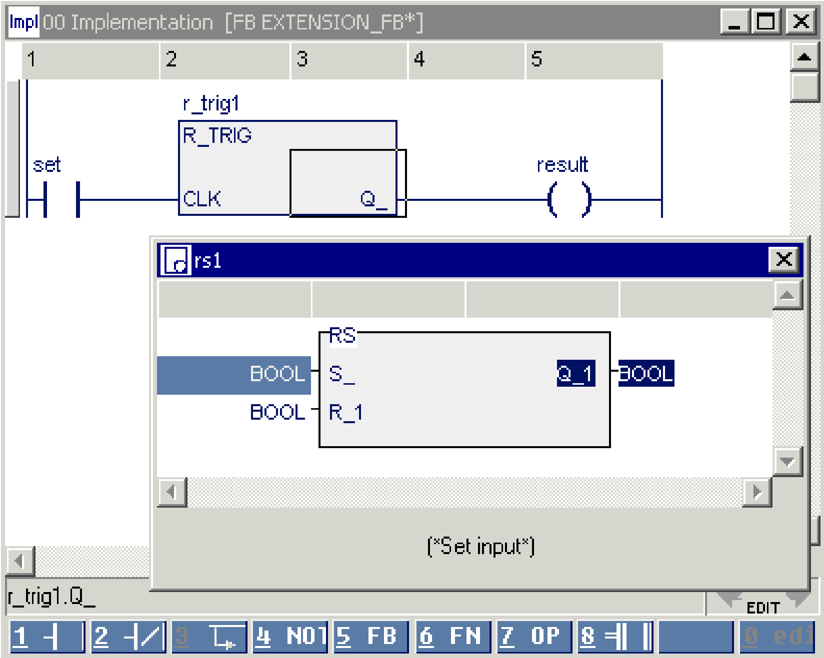


bsp_fb_dekl.bmp

Fig. 7-37: Declaration part for function block "EXTENSION_FB"

After declaration, the function block "r_trig1" is first unconditionally called up in the following figure. Block "rs1" is then inserted.

Input sequence	**Comment**
5	FB selection window, select r_trig1 , press <Enter>
1***set***<Enter>	Input of the edge evaluation, cursor on FB output
1***result***<Enter>	Output of the edge evaluation, cursor on FB output
5	FB selection window, select rs1 , press <Enter>



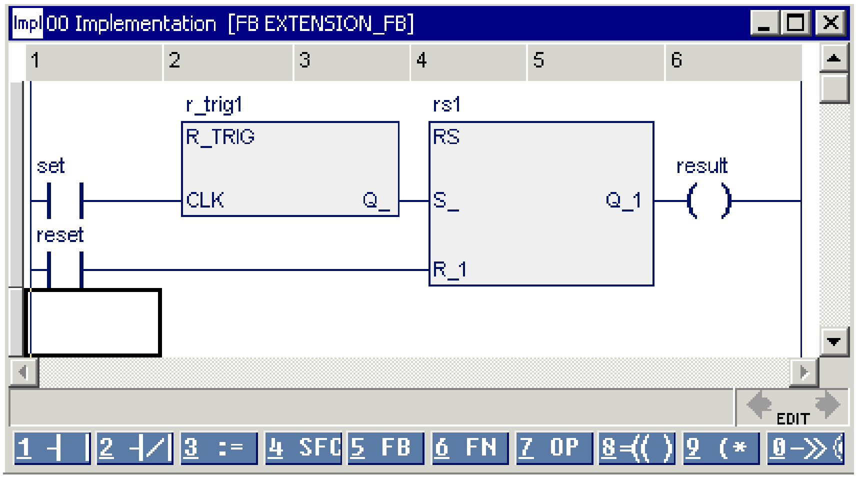
bsp_fb_auswahl.bmp

Fig. 7-38: Selection window for determination of the connection points

Either the set input or the reset input can be selected as the connection to the block to be inserted, by pressing the cursor key / <Enter>.

The output is already preselected.

Input sequence	**Comment**
Cursor on "S_" <Enter>	Block is inserted. Cursor on "R_1"
1***reset***<Enter>	Termination of the network



bsp_fb_kop.bmp

Fig. 7-39: Complete ladder diagram

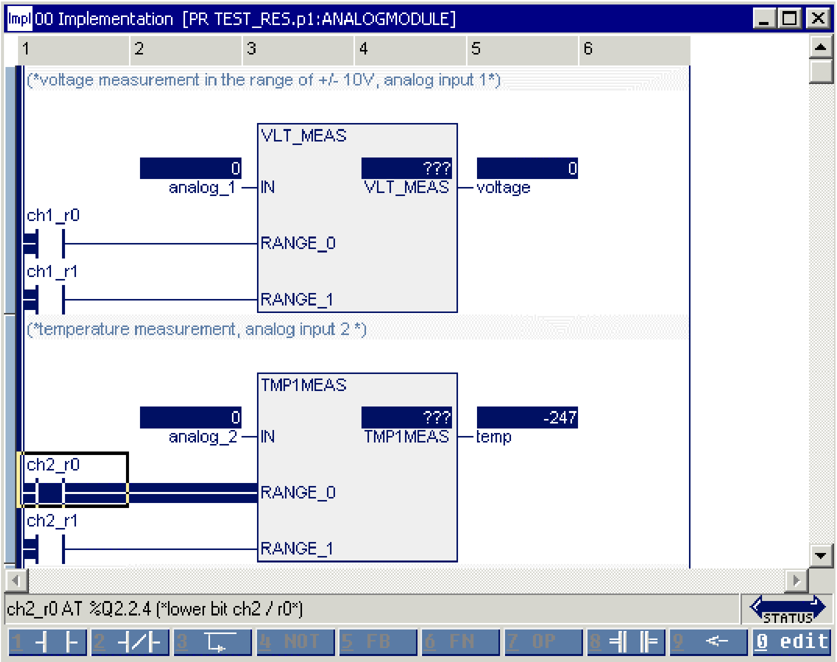
7.6 Options, Ladder Diagram Editor

The options relevant for the ladder diagram editor can be selected with the "Extras / Options" menu item:

Group	Option	Meaning
Desktop	Restore size and position during startup	The desktop is restored in the same size and position.
	Restore MDI window during startup	MDI windows are opened in the same order when restarting the system.
	Auto save	Allows the automatic saving of the current file in presettable time intervals without any prompt.
	Sound	Activation or deactivation of a beep sound.
View / All	Apply column width modifications automatically	Restoring of the column with same width.
	Apply declaration comment in implementation	Comments, that have been entered in the respective declaration line are displayed in the implementation. The implementation can be changed; the comment is then doubled, the declaration line remains unaffected.
	Variable display	With symbols (name) or absolute (address).
	Display of absolute variables	The user can select from I/Q, E/A and I/O for absolute addresses.
	Truncating very long texts Truncating very long numbers	Texts and numbers can be truncated to the right or left, and can be represented with or without "..." marking.
View	Columns	7
	Column width for the individual columns	72
	(with standard values)	
	Comments	Indication of the declaration comments above the variables.
	Absolute represented	Indication of the absolute addresses above the variables.

Fig. 7-40: LD editor options

7.7 Status Display in the Ladder Diagram Editor



kop_status.bmp

Fig. 7-41: Status display in the ladder diagram

Further ways to get status information are:

- Start / Force <Shift>+<F8> for elementary variables (ANY_ELEMENTARY)
- Start / Status ARRAYs / Structures <Shift>+<F3>

7.8 Online Editing in the Ladder Diagram

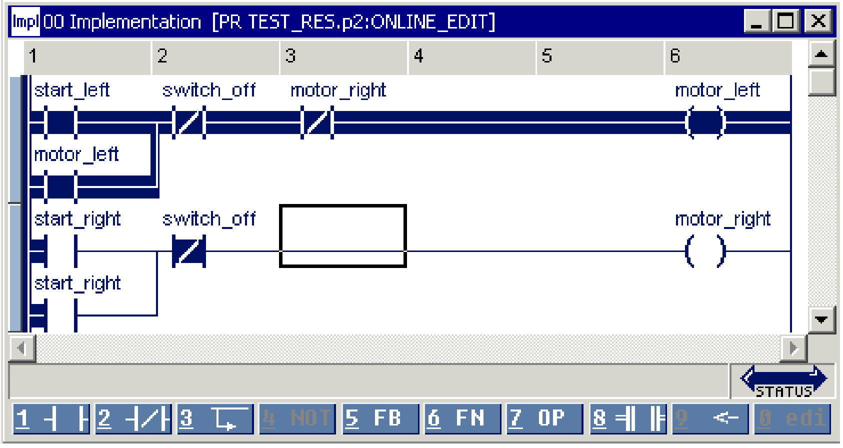
The online edit feature in the present version allows the exchange of codes for a program, a function block or a function without changing the data of the concerned POU. A machine or plant can continue its working cycle although the program code was changed (also see Online Changes and Edge Evaluation, LD).

Changes in the implementation of **one** program organization unit, which require neither declaration nor imports, are allowed at present. All other changes are not online capable.

Status of the POU >>Status<<

A running PLC program with activated status display is the initial condition for online editing. In the following example the "motor_left" locking contact is to be added online.

Identification: "Status" is indicated in the right bottom corner.



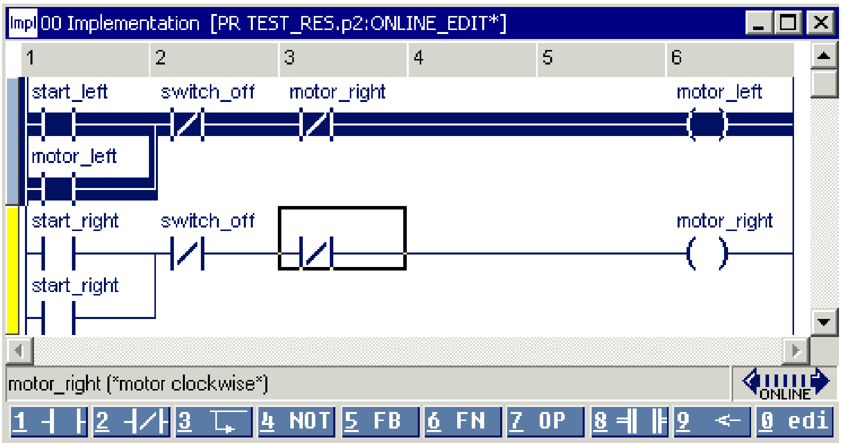
online_kop_01.bmp

Fig. 7-42: Before an online modification, still status = On

Status of the POU >>Online<<

The online editing mode is initiated by inserting the contact. The color of the marginal mark for the actual network changes from blue to yellow (for colors see Chapter "Editing Features, Varying Color in the Ladder Diagram Editor").

Identification: "Online" is indicated in the right bottom corner.



online_kop_02.bmp

Fig. 7-43: Inserting the locking contact online (1st step)

Rexroth
Bosch Group

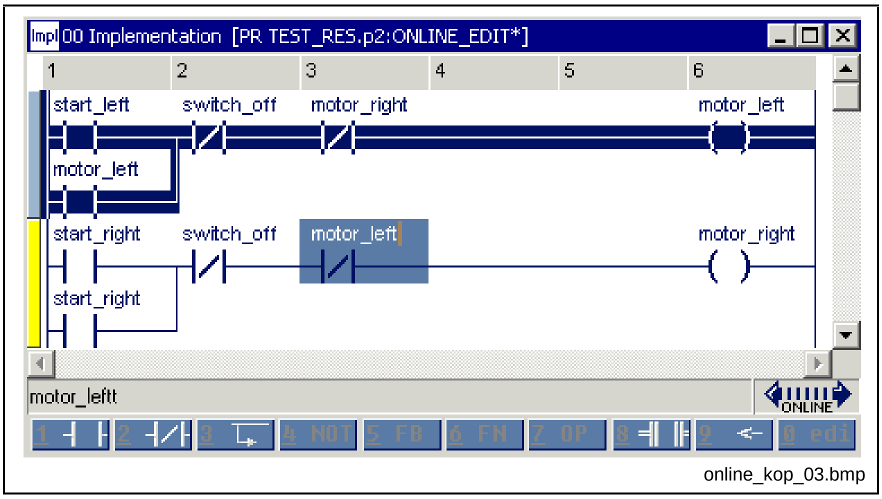


Fig. 7-44: Inserting the locking contact online (2nd step)

Download of the change

The online changing is completed with the download of the changed code into the control by means of the "Start / Download "xx" in control "xx"" menu item or by pressing <Ctrl>+<F9>.

Identification: "Status" is indicated in the right bottom corner.

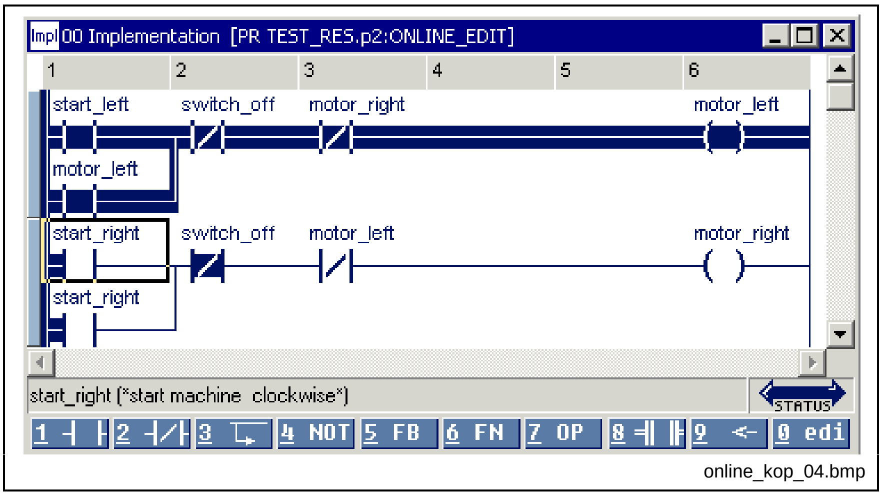


Fig. 7-45: Completion of the online changing with download

Changes which are not online capable

Online changes in the implementation of **one** program organization unit, which require neither declaration nor imports, are allowed at present.

Kind of change	**Operating mode**
Deletion of a network	Online
Deletion of a contact or an IL line	Online
Insertion of a new network or a new IL line	Online
Insertion or change of a contact, a coil, an instruction or the like	Online
Insertion of labels or jumps	Online
Insertion of a currently declared function block	Online
Insertion of a function which was already used in the POU (new imports are not online capable as yet!)	Online
Change of network properties of existing networks (at least one network of the POU has to provide activated network properties!)	Online
Change of network comments in IL / ladder diagram	Online
New declaration or change of declaration of variables or instances of programs / function blocks, as well as in all lists	Edit
Insertion or deletion of steps, transitions and actions	Edit
First application of network properties or deletion of the last network properties	Edit
Change of the action qualifier and/or the action time of an action block	Edit
Change of comments in all lists, in SFCs, action blocks and in all declarations	Edit
Changes in the IO editor	Edit
Changes in a second file, if there is already one file in the Online Edit mode	Edit

Fig. 7-46: Overview of online capable changes (selection)

7.9 Pop-up Menu, LD Editor <Shift>+<F10>

The pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	Explanation
Open	Branch, also <Ctrl>+<Enter>
Edit comment	Edit the comment of the current LD element
New network	- by **adding** an empty **IL line** before the current network - by **adding** an empty **LD network** before the current network - by **adding** an empty **IL line** behind the current network - by **adding** an empty **LD network** before the current network
Delete network	Deletion of the current network (see marginal marking).
Separate network	Command to disconnect networks e.g. for multi-line comments.
Connect network	Service command
Convert network to	- an **IL** network - convert complete editor contents into IL networks using **"IL (all)"**
ProVi messages	Display and modification of the diagnosis properties
Import implementation	The ASCII file selected from the "WinPCL text files" is attached to the current element.
Export implementation	The complete contents of the ladder diagram editor is exported as an ASCII file and stored in the folder "WinPCL text files".
Export network	The complete LD network is exported as an ASCII file and stored in the folder "WinPCL text files" (marginal marking).
Syntax text	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the interface of the data type or of the function block type of the current line, also use <Shift>+<F1>.
Cross reference help	List of all places where the variable is used. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force	Allows the entry of a variable name. The value of the variables is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variable and the start of the program code execution.
Status ARRAYs / Structures	Display of the status of array and structure elements, forcing by pressing the <Shift>+<F10> keys or the right mouse button.
Print current window	Print of the editor contents with <Ctrl>+<P>.
Display of the options	Display of the (symbolic) names of the variable or display of the absolute addresses of the variable, if provided
Internals	Search for faults in the programming system, to be used only if approved by the service.

Fig. 7-47: Pop-up menu of the ladder diagram editor

7.10 Block Commands, LD Editor

Select the networks by pressing and holding the SHIFT KEY while using the corresponding arrow key or by pressing and holding the left mouse key while dragging it across the text.

You can select LDs for every network by clicking the gray bar on the left outer side.

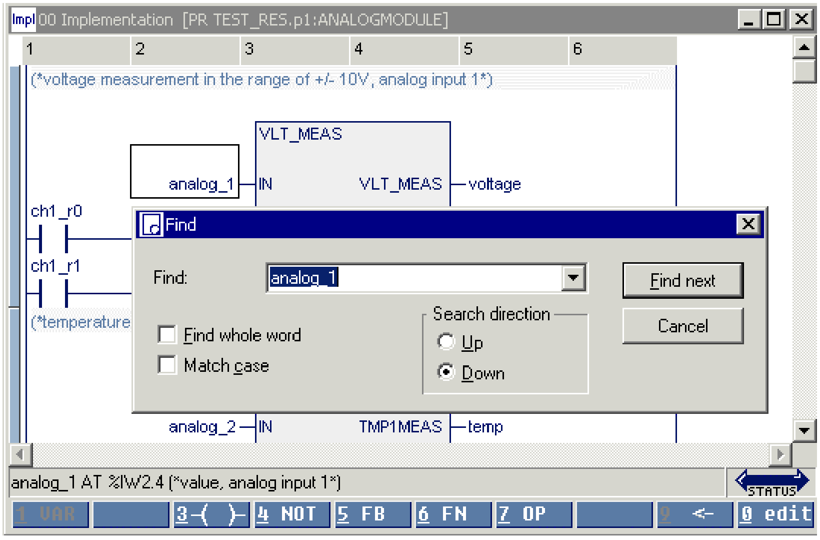
Extending the selection	**Key combination**
One character to the right	<Shift>+arrow key<to the right>
One character to the left	<Shift>+arrow key<to the left>
To the end of the line	<Shift>+<End>
To the beginning of the line	<Shift>+<Home>
Down by one line	<Shift>+<down> arrow key
Up by one line	<Shift>+<up> arrow key
Down by one page	<Shift>+<Page down>
Up by one page	<Shift>+<Page up>

Deletion of text	**Keys**
Deleting the character to the left of the cursor	BACKSPACE KEY
Deleting the character to the right of the cursor	<Del>

Copying and moving of text	**Key combination**
Copying the text selected to the clipboard	<Ctrl>+<C>
Moving the text selected to the clipboard	<Ctrl>+<X>
Pasting the contents from the clipboard	<Ctrl>+<V>

7.11 Search and Replace, Ladder Diagram Editor

is in the first version and provides the features of a text editor:

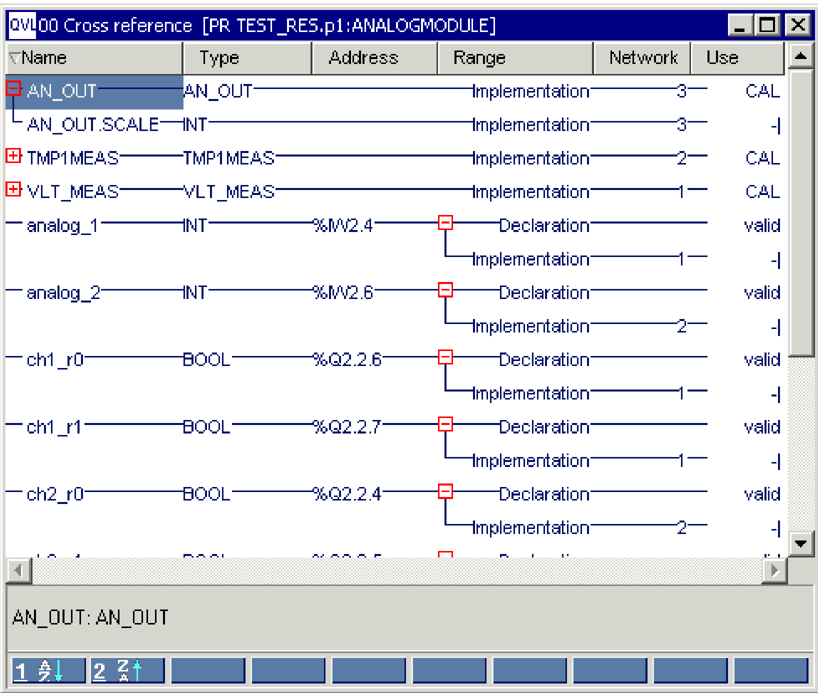


kop_suchen.bmp

Fig. 7-48: Search in the ladder diagram editor

7.12 Cross Reference List, LD Editor

In contrast to the cross references of the pop-up menu, the overview opened via "View \ cross reference list" shows all variables. Of course, only variables from lines with the correct syntax can be resolved by their place of use. All faulty names or names with double declaration are displayed and can, thus, be reached by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.



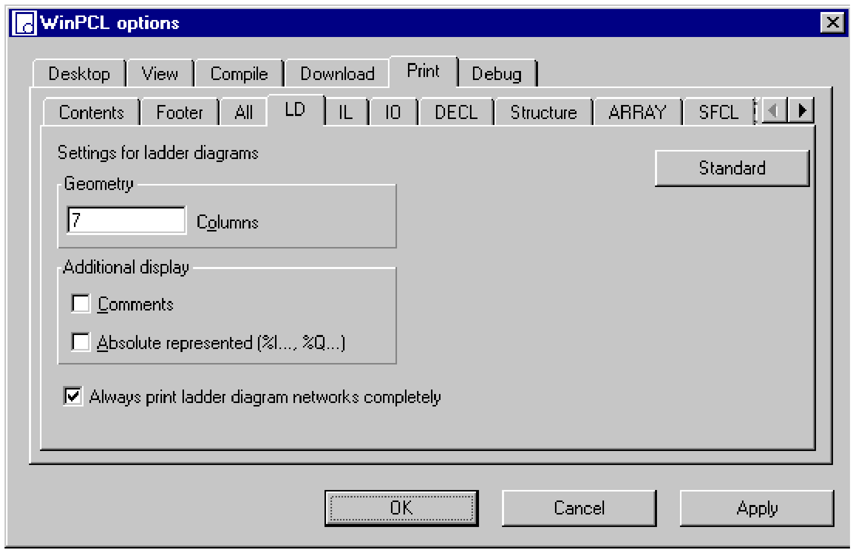
kop_qvl.bmp

Fig. 7-49: Cross reference list with LD applications

Pictogram	Meaning
-[ ]-, -[/]-	Normally open contact, Normally closed contact
-|, |-	Input, output of functions or function blocks
CAL	Block call
-( )-, -(/)-	Relay, negated relay
-(S)-, -(R)-	Set, reset relay

7.13 Documentation, Ladder Diagram Editor

Documentation must be implemented by using the column number and width specified under / Options, Ladder Diagram Editor / View / LD.



print_kop.bmp

Fig. 7-50: Print options in ladder diagrams

In addition, the declaration comment and / or the absolute represented, if provided, can be displayed above the variables.

The "Apply" button activates the column width set for the instruction list editor. The width of the column can either be entered in the window shown above or preset in the editor by dragging the headers.

The "Standard" button resets the default.

The "OK" button applies the setting and closes the dialog window.

The "Cancel" button closes the window; the previous values are kept.

Detailed information on the real print process and the features is to be found in the section on WinPCL.

8 SFC Editor

8.1 Basic Sequential Function Chart Elements (SFC Elements)

The SFC elements allow the establishment of SFCs within function blocks and programs, with SFC standing for SEQUENTIAL FUNCTION CHART.

SFCs are supported in two performance categories:

- simple SFCs with one or a few steps having been used for structuring programs and function blocks, as has been usual under DosPCL (BASIC type, should not be used any longer),
- one or more SFCs in a program or function block with system-supported mode control and/or diagnosis with criteria analysis (IndraStep type).

One of these two types must be selected when opening a Opening an SFC in the SFC List .

The two types are different from each other in their data types assigned to the SFCs, transitions and actions and supported by the system. The IndraStep type completely covers the BASIC type, i.e. it contains an extended set of system variables.

SFCs contain steps and transitions which are connected to each other through oriented lines.

Moreover, alternative (OR) branches and parallel (AND) branches can be realized in the SFCs.

Steps

The step is a basic element of the sequential function chart.

In every step

- zero times,
- one or
- several actions

can be released (see chapter "Action block editor").

With a program running on the control, a step is either active or inactive.

Initial step

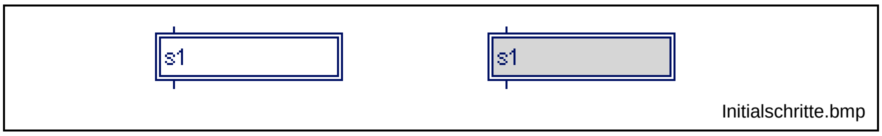
Within an SFC, it is always the first or starting step that is of particular importance. It is called initial step of the SFC. The SFC starts and terminates with this initial step.

If a step is entered before, this step becomes the initial step.

A name has to be entered for each step, also for the initial step.

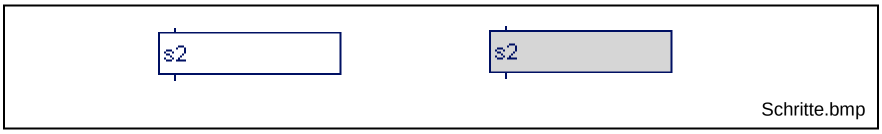
Further a comment on the step is required after entry of the step name.

Graphic representation of steps



Left: Color white -> initial step does not yet contain any actions
Right: Color gray -> initial step contains one or several actions

Fig. 8-1: Initial steps



Left: Color white -> step does not yet contain any actions
Right: Color gray-> step contains one or several actions

Fig. 8-2: Steps

Elements of a step

When assigning data types to steps, transitions, actions and to the SFC, the user defines to which elements per step the program has access.

If he selects the BASIC features stored in the library (firmware data type _tSTEP), he has access to the following elements:

_tSTEP	**STRUCT**	**Firmware data types step**
X	BOOL	Step flag TRUE, if step is active
F	BOOL	TRUE - forcing of the step, possible only in manual mode
SYNC	BOOL	TRUE - request to set this step for synchronization
T	TIME	Step active time - read only, time elapsed since activation of the step
END_STRUCT		

Fig. 8-3: _tSTEP structure

Example:

Time monitoring of step sA1:

Label	Operation	Operand	Comment
	LD	sA1.T	(If) step active time of step sA1
	LE	T#15s	Equal to or less than 15 sec
	JMPC	....	Jump to ...

Fig. 8-4: Time monitoring of step sA1

Transitions

The transition is the second basic element of the sequential function chart. A condition, under which the control of one or several previous steps of this transition to one or several subsequent steps is running along the oriented lines, is assigned to each transition. The transition takes place if the transition condition is fulfilled.

A condition can be written as

- instruction list or
- ladder diagram network.

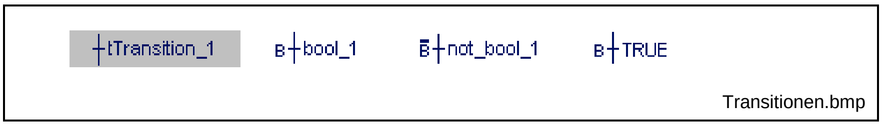
Functions can also be used within the condition.

A not fulfilled transition condition is defined as permanently fulfilled.

A name has to be entered for each transition. You are prompted to enter a comment after input of the transition name but this is not mandatory; you can continue with <Enter>. If the name is entered twice, the same content is assumed automatically, but a new comment on the application is requested.

Individual Boolean variables or the constants TRUE (permanently fulfilled) and FALSE (never fulfilled) can be entered directly as an alternative.

Graphic representation of transitions



Left: Transition, network as transition (condition)
Middle left: Boolean variable as transition (condition)
Middle right: Negated Boolean variable as transition (condition)
Right: TRUE (and FALSE)

Fig. 8-5: Transitions

Elements of a transition (not available for Boolean and negated Boolean transitions as well as TRUE and FALSE)

When assigning data types to steps, transitions, actions and the SFC itself, the user defines the elements his program can access per transition:

If he selects the BASIC features stored in the library (data type _tTRANSITION), he has access to the following elements:

_tTRANSITION	**STRUCT**	**Firmware data types transition**
JOG	BOOL	Write, only in AUTOMATIC JOG mode; TRUE if advancing of the transition after firing is disabled.
END_STRUCT		

Fig. 8-6: _tTRANSITION structure

Example:

Advancing of the transition tA1 in the automatic jog mode is disabled:

Label	Operation	Operand	Comment
	LD	TRUE	Load TRUE
	ST	tA1.JOG	Transition tA1 disabled

Fig. 8-7: Advancing of the transition tA1 in the automatic jog mode is disabled

Rexroth Bosch Group

Multiple Request of Conditions in Transitions

If two successive transitions are made conditional on the same variable in one and the same SFC, there are the following possibilities:

- The **state** of the same variable is evaluated in both transitions:
  - Both transitions have the same name (and, thus, the same content) - the actions of either step are processed in successive PLC cycles.
  - Both transitions have different names, but nevertheless the same content - the actions of either step are processed in successive PLC cycles.

- The (rising) **edge** of the same variable is evaluated in both transitions:
  - Both transitions have the same name (and, thus, the same content) - the edge is effective for the first transition only. To achieve the next transition, the variable must realize a 1-0-1 transition (Whenever the switch is actuated, the step is advanced).
  - Both transitions have different names, but nevertheless the same content - the edge is effective for both transitions only. If the switch is actuated only once (0-1 transition), the actions of either step are processed one after the other in successive PLC cycles.

Oriented Lines

The SFC direction within an SFC is defined by oriented lines. An SFC is processed from top to bottom and/or from left to the right. There may be jumps within an SFC.

For a clearer arrangement, however, the representation of an SFC does not contain any crossings.

Graphic representation of oriented lines

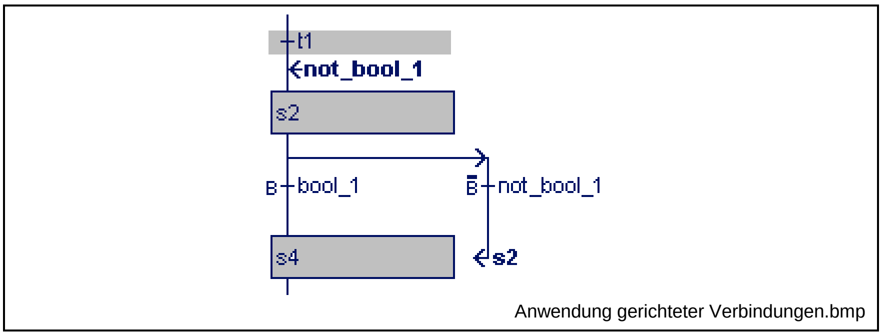


Fig. 8-8: Oriented lines (jumps with destinations)

Alternative SFCs

The alternative SFCs always start with a transition after the branching point and end with a transition. In exceptional cases, an alternative SFC can consist of one transition only.

Starting from one step, i.e. "Step sB1" in the example, the SFC branches depending on which transition condition is fulfilled:

For transition

- tB1 to step sC1,
- tB2 to step sC2 and
- tB3 to step sB1 (jump).

If the conditions of several transitions are fulfilled at the same time, the leftmost SFC is continued.

Example:

The conditions of the transitions tB2 and tB3 are fulfilled at the same time, the SFC is continued to step sC2.

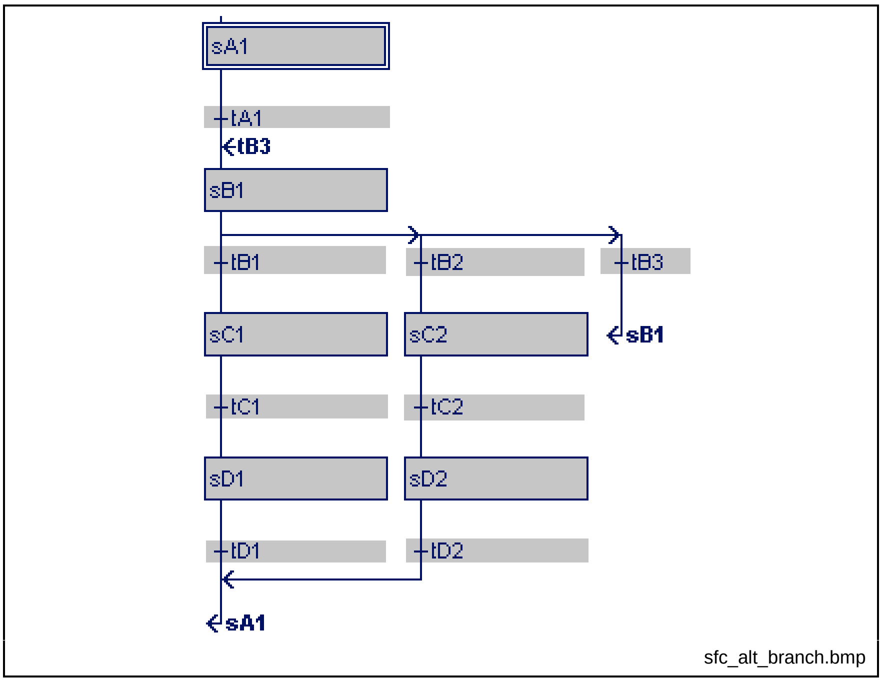


Fig. 8-9: Alternative SFCs

Normal SFCs end in a junction after the terminating transitions tD1 or tD2, see example. If the exceptional jump case is concerned, the entry point (in the example before step sB1) is identified by a connecting arrow and the name of the jump element (in the example transition tB3).

Parallel SFCs

After a common transition, there is a transition to two or several SFCs which have to be executed in parallel.

The SFCs start and end with one step before the junction.

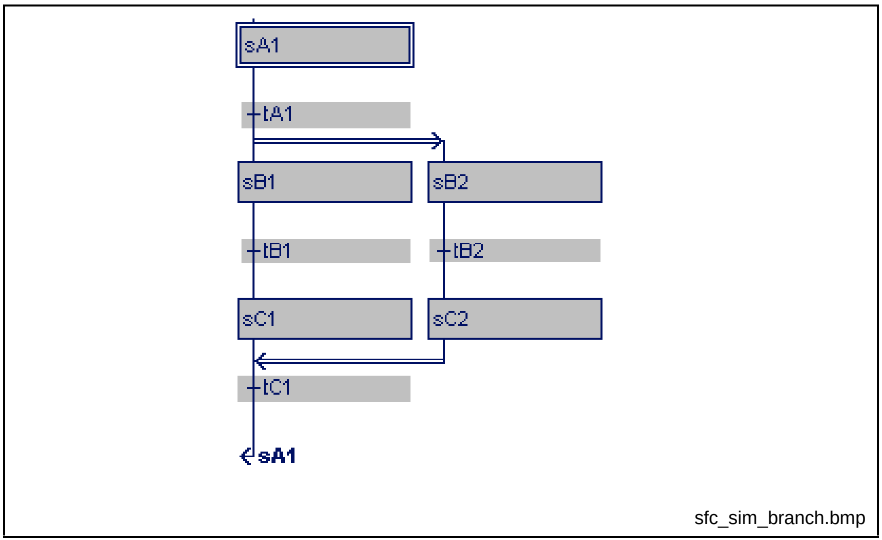


Fig. 8-10: Parallel SFCs

The steps sB1 and sB2 become active when transition condition tA1 is fulfilled and the previous step sA1 is active.

The SFCs are executed at the same time, but independently from each other.

The junction is controlled by the common transition tC1 and takes place after execution of the two SFCs, that means after sC1 and sC2 and the transition condition tC1 are fulfilled.

Execution Rules of the Sequential Function Chart

Sequence of step / transition / step

The first step , i.e. the initial step sA1, becomes active upon start of an SFC. The further execution sequence can be seen from the following illustration of a simple SFC without branches.

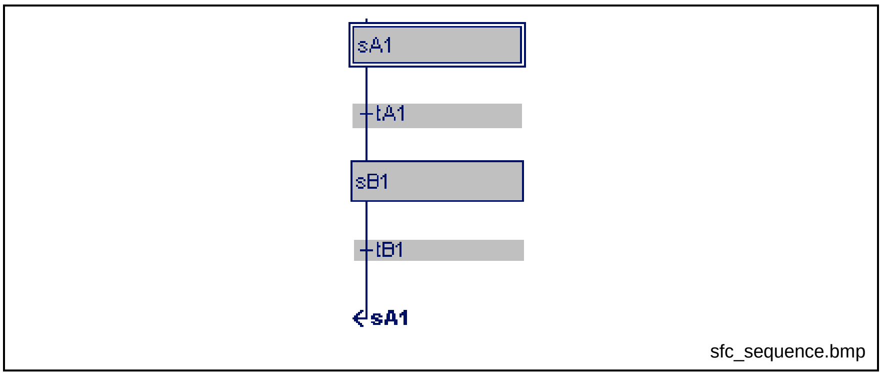


Fig. 8-11: Sequence of step / transition / step

Condition is that step sA1 is active. The condition which belongs to transition tA1 is calculated. Step sA1 remains active if this condition is not fulfilled, that means is logic 0.

If the condition is fulfilled, that means is logic 1, the transition switches and the subsequent step sB1 becomes active. Step sA1 is deactivated simultaneously.

Execution of alternative branches

In the alternative sequence shown below step sB1 is assumed as being active. The calculation of the transition starts at the left with the order tB1, tB2, tB3.

Step sB1 remains active if none of the conditions belonging to the transactions are fulfilled.

If a condition, tB1 or tB2, is recognized as being fulfilled the following step sC1 or sC2 is activated and step sB1 is deactivated.

Further transitions belonging to this branch are no longer executed as their previous step sB1 is not active any more.

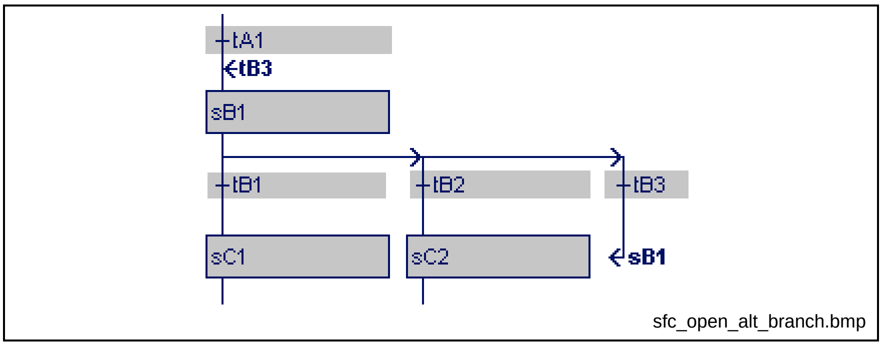


Fig. 8-12: Opening of an alternative branch

The junction of alternative SFCs or even a jump point are executed normally.

Execution of simultaneous branches

In the SFC with conjunctive branching shown below, step sA1 is assumed as being active. The condition which belongs to transition tA1 is calculated.

Step sA1 remains active if this condition is not fulfilled, that means is logic 0. If the condition is fulfilled, that means is logic 1, the transition switches and the subsequent steps sB1 and sB2 become active. Step sA1 is deactivated simultaneously.

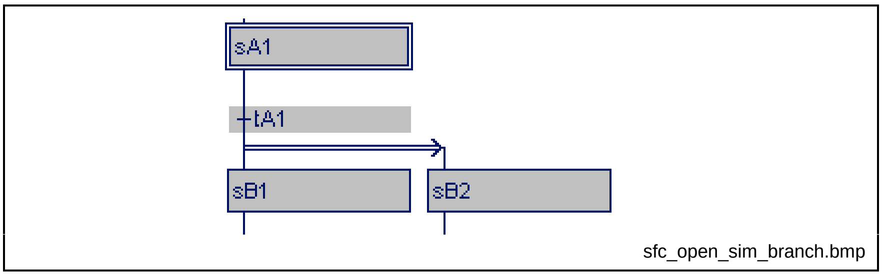


Fig. 8-13: Opening of a simultaneous branch

There is one active step in each of the parallel partial SFCs after passing of the simultaneous branch. All active steps are executed once within one PLC cycle, starting from left to right.

Execution of the junction of simultaneous SFCs

The combination of simultaneous SFCs is explained by means of the following figure.

Step sD1 can become active only if the condition of the previous transition tC1 is fulfilled and all previous steps (sC1 and sC2 in the example) are active. Step sD1 deactivates the previous steps sC1 and sC2 after it was activated.

sD1 cannot be activated if one of the previous steps is not active and/or the transition condition is not fulfilled.

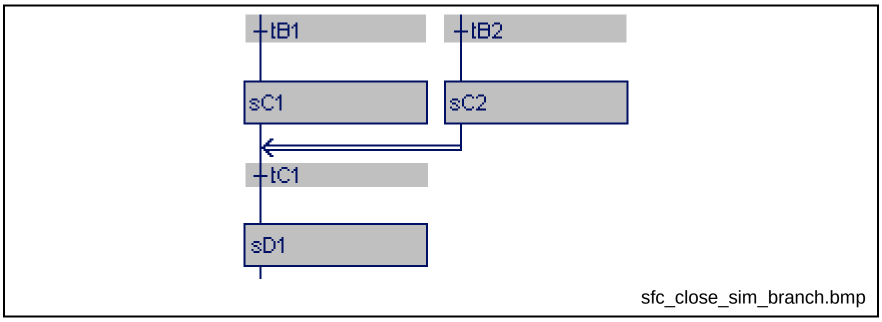


Fig. 8-14: Closing of a simultaneous branch

8.2 Entering SFCs in the SFC Editor

The work in the SFC editor will be described below by means of a simple example, the operating mode control for a SCARA robot.

At first, opening of a sequence (SFC) is explained. This is followed by a description of the insertion of steps, transitions, branches, and junctions. Next the deletion methods are shown. Work with block commands is explained in the concluding examples.

Opening an SFC in the SFC List

An SFC is opened in the "View / SFC" menu item.

This SFC list contains all SFCs contained in the POU with

- its steps,
- transitions, and
- actions;

and

- any steps,
- transitions, and
- actions,

not contained in the sequences mentioned above.

The user can enter any name in the "Name" column. The SFC is called up in the following with this name.

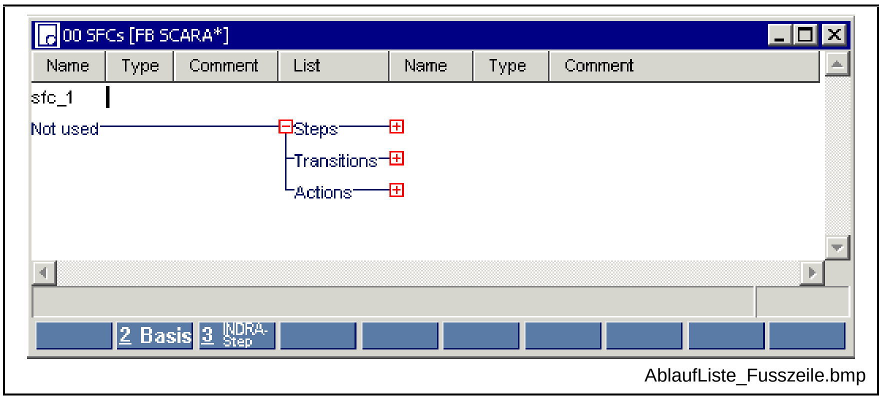
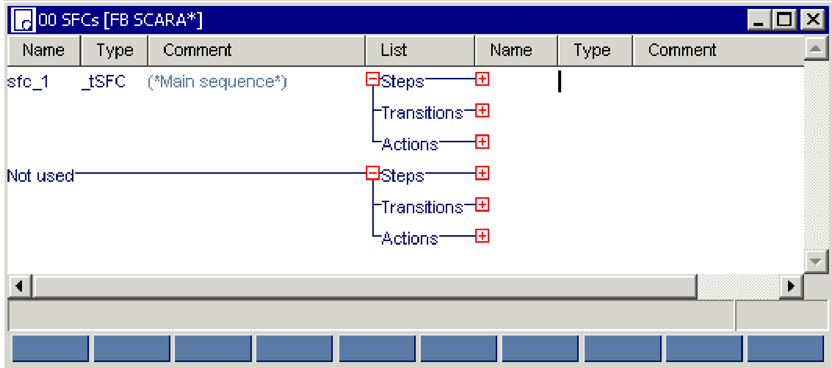


Fig. 8-15: Entering the SFC name; continue with footer to enter the type

Use the footer command "2-Basis" to complete the "Type" column. This command activates a complete set of data types for the SFC, steps, transitions and actions.

A comment can be attached.



AblaufListe_Fusszeile_01.bmp

Fig. 8-16: SFC table for the example "Scara"

The table comprises the lists of steps, transitions and actions pertaining to the particular SFC concerned.

Note: When the Basis version is selected, the data types _tSFC, _tSTEP, _tTRANSITION, _tACTION are assigned to the SFC elements.

The footer command "3-Indra-Step", however, opens a dialog window for selecting an SKD file. Further procedures are described in the IndraStep documentation.
Double-clicking or pressing the <Ctrl>+<Enter> keys permits branching into the SFC.

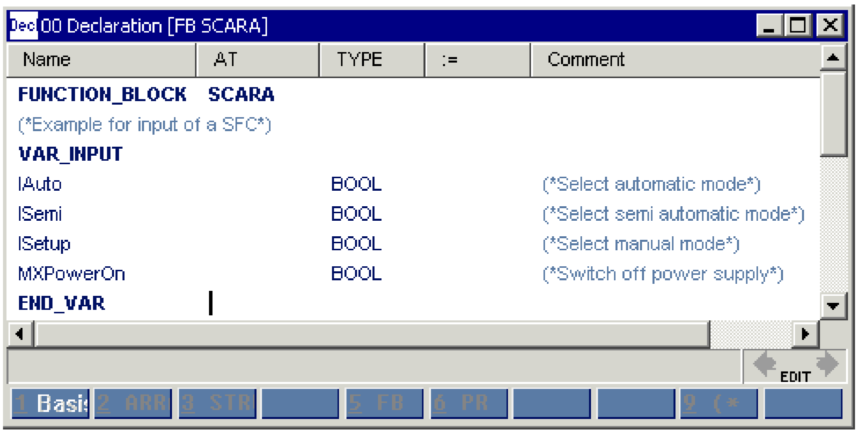
Pressing the <Enter> key inserts a blank line where a further SFC can be designed.

Note: If the cursor is positioned on the already existing SFC, the blank line is placed before it. With the cursor at any other position, the blank line is added behind.

Program Example of the "Scara" SFC

The following is assumed:

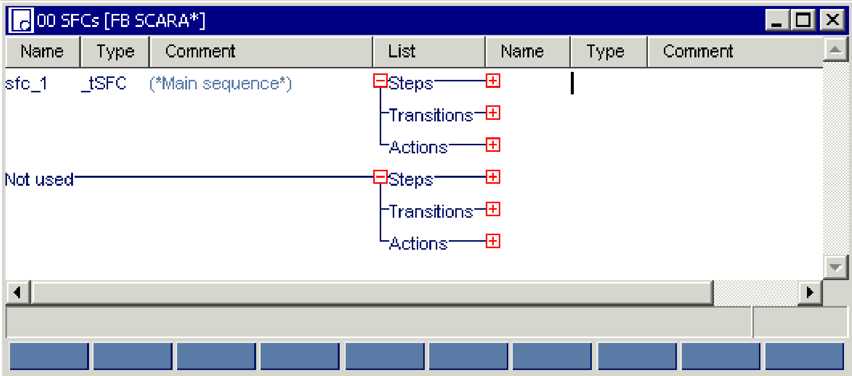
- the declaration of the Boolean variables IAuto, ISemi, ISetup and MXPowerOn in "View / declaration editor".



Deklaration Scara.bmp

Fig. 8-17: Declaration of the variables

- Opening an SFC in the SFC List in View / SFC

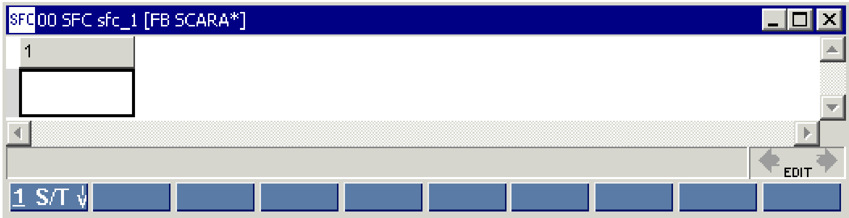


AblaufListe_Fusszeile_01.bmp

Fig. 8-18: SFC list for the "Scara" example

- Branching to the SFC editor

When you enter a new SFC structure within a program, first an empty screen appears.



sfc_scara_empty.bmp

Fig. 8-19: Empty SFC editor ready for entering the SFC

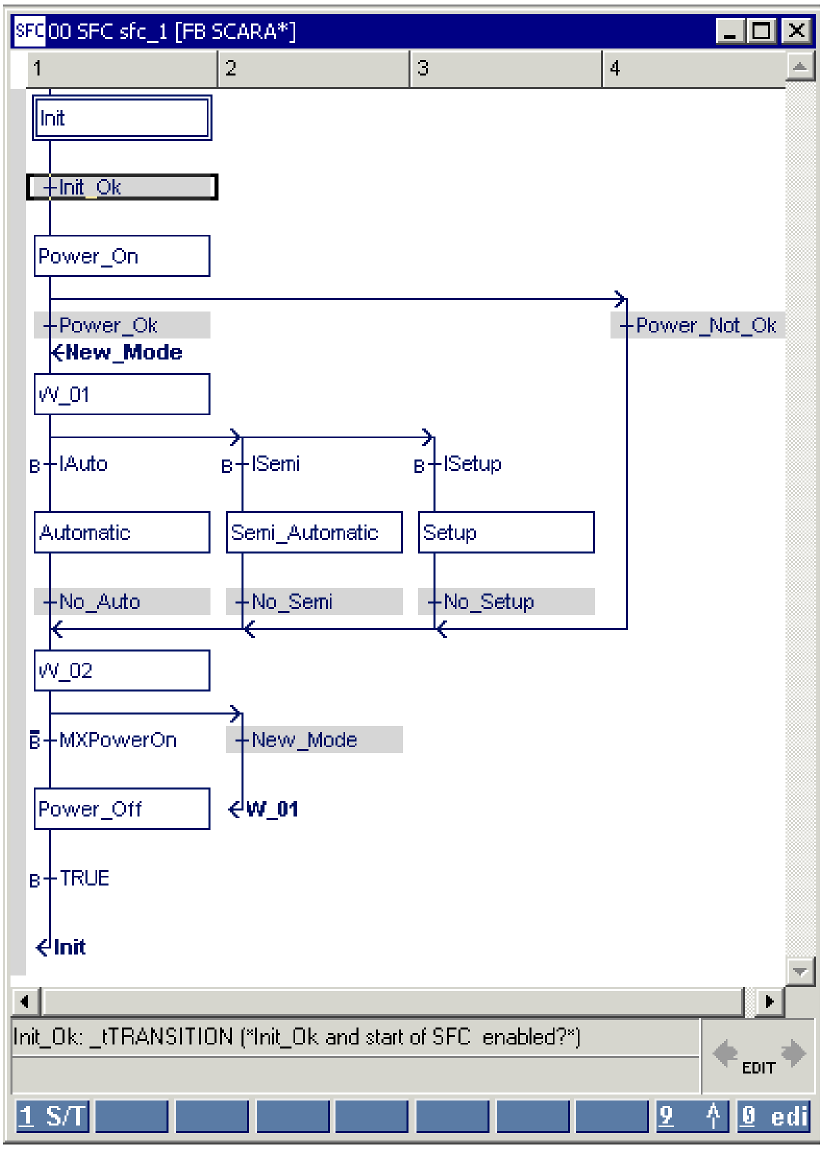
The applicable commands can be found in the footer.

Number	Contents
<1>	Insertion of a pair of **s**tep / **t**ransition or **t**ransition / **s**tep pair
<2>	Opening / Closing of a simultaneous branch (AND branch)
<3>	Opening / closing of an alternative branch (OR branch)
<9>	Toggling insertion before / insert behind
<0>	Editing of step and transition names and comments

Fig. 8-20: Provided footer commands and their functions

The previous figure shows the currently possible way, to enter the initial step and the following transition by pressing the <1> key.

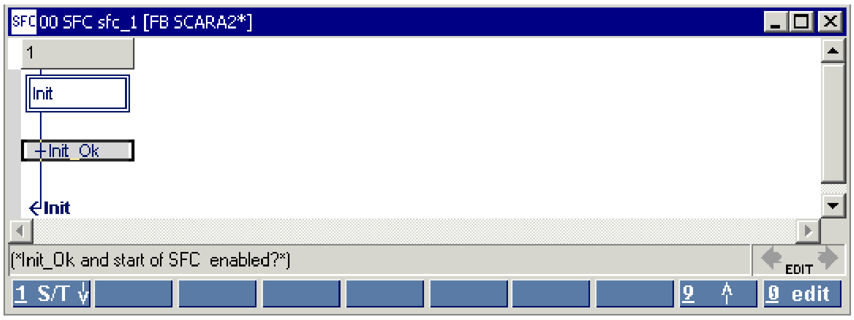
The names including pertaining step and transition comments have to be added for both step and transition. If no comment is to be entered, confirm the comment field by pressing the <ENTER> key. The above-mentioned "Scara" SFC is shown below. It should be self-explaining by the chosen name.



Ablauf scara.bmp

Fig. 8-21: "Scara" SFC

Input sequence	**Comment**
<1>***Init***<Enter> ***General initialization, clear error***<Enter> ***Init_Ok***<Enter> ***Init_Ok and start of SFC enabled?***<Enter>	Input of the first pair, step with comment and transition with comment



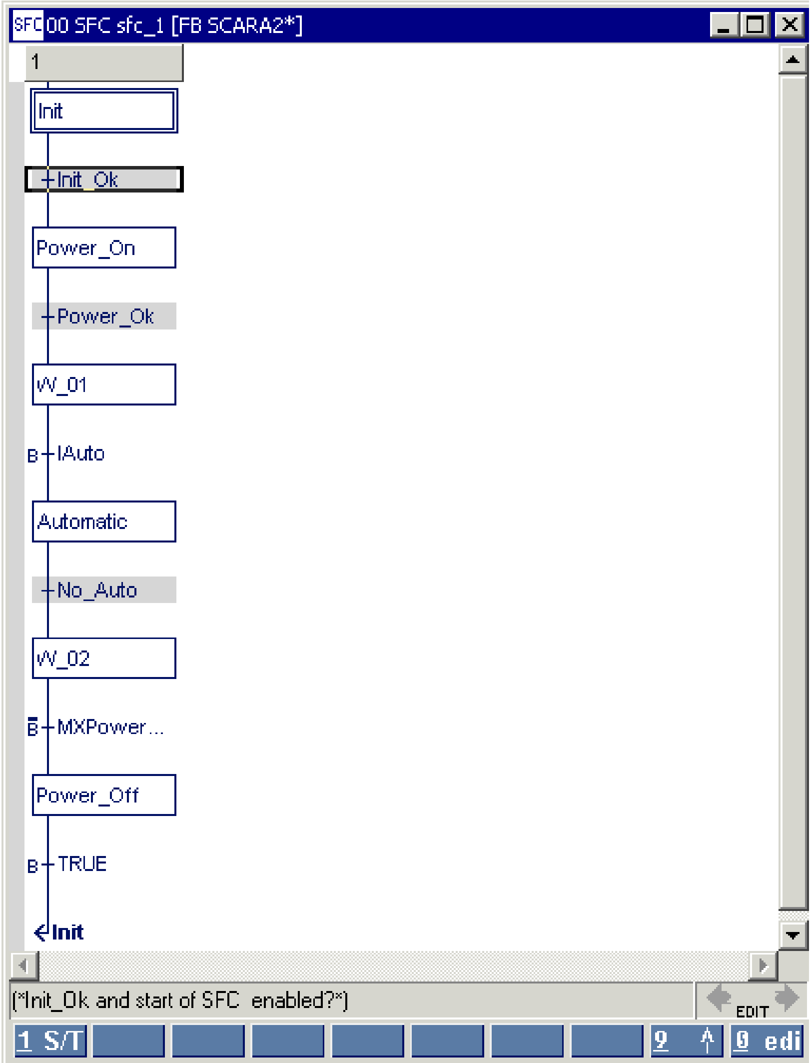
Ablauf scara 01.bmp

Fig. 8-22: Initialization step with transition and return jump

The white basic color of the step indicates that the element is not filled in yet.

Thereafter, the complete sequence of the main path must be entered without considering planned branches or junctions.

Input sequence	**Comment**
<1>***Power_On***<Enter> ***Turn-on power amplifier***<Enter> ***Power_Ok*** <Enter> ***LV activation runs as scheduled?***<Enter>	Next pair in the SFC
<1>***W_01***<Enter> ***Wait for mode selection or LV error***<Enter> ***IAuto***<Enter> ***Automatic mode enabled?***<Enter>	Boolean variable
<1>***Automatic***<Enter> ***Automatic single mode***<Enter> ***No_Auto***<Enter> ***Automatic deactivated?***<Enter>	
<1>***W_02***<Enter> ***Wait for mode selection or deactivation?***<Enter> ***MXPowerOn***<Enter> ***Turn off LV***Enter>	
<4>	Negation of a Boolean variable
<1>***Power_Off***<Enter> ***Turn off LV***<Enter> ***TRUE***<Enter> <Enter>	Renunciation of comment



Ablauf scara 02.bmp

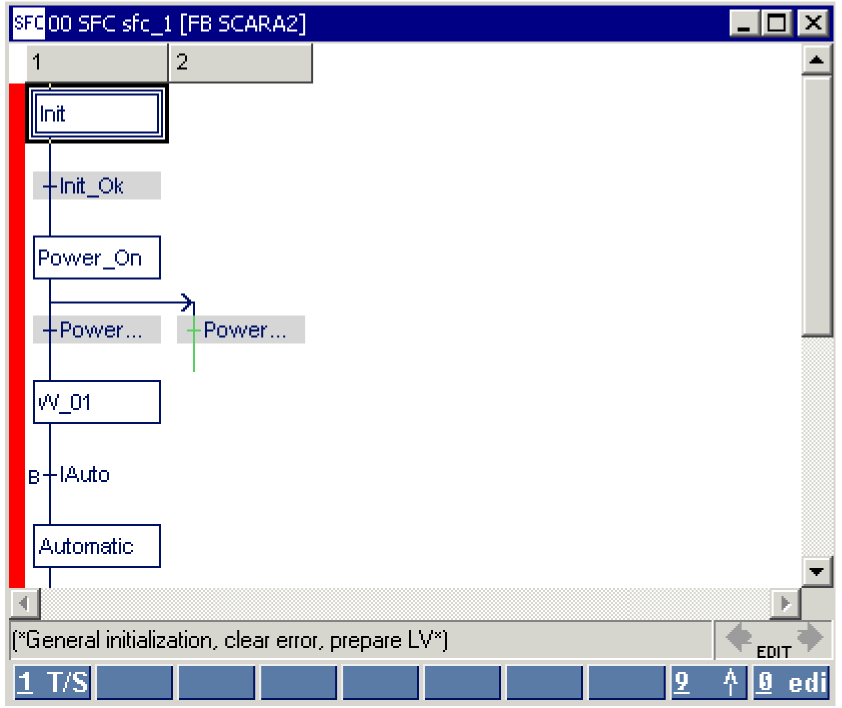
Fig. 8-23: Main sequence without branches

The input is continued as shown in the figure below:

Input sequence	Comment
	Cursor on transition Power_Ok
<3>**Power_Not_Ok**<Enter> **Error upon turn-on?**<Enter>	OR branch, transition with comment

The opened branch is to be closed next. If the cursor is on a transition and is moved across the steps and transitions already entered, you can see that closing of the alternative is offered below the transition in the line of the footer commands.

The branch is terminated below the transition No_Auto.



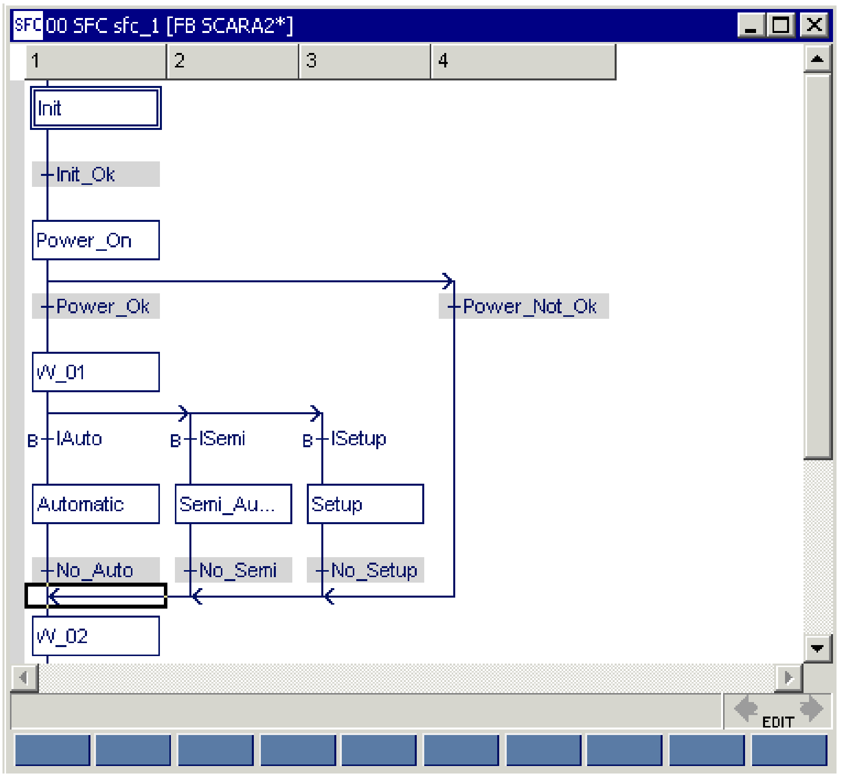
Ablauf scara 03.bmp

Fig. 8-24: Termination of the branch

Input sequence	Comment
	Cursor below transition No_Auto
<3>	Termination of the branch

According to the defined task, the structure is expanded by the alternative operating mode steps Semi_Automatic and Setup.

Input sequence	Comment
	Cursor above transition IAuto
<3>***ISemi***<Enter> ***Single mode enabled?***<Enter>	OR branch, Boolean transition
<1>***Semi_Automatik***<Enter> ***Single mode***<Enter> ***No_Semi***<Enter ***Single mode deactivated?***<Enter>	Step with comment + transition with comment, Cursor below transition No_Auto
<3>	Close branch Cursor above transition Isemi
<3>***ISetup***<Enter> ***Manual mode enabled?***<Enter>	Boolean variable
<1>***Setup***<Enter> ***Manual mode***<Enter> ***No_Setup***<Enter ***Manual mode deactivated?***<Enter>	Cursor below transition No_Semi
<3>	Terminate branch



Ablauf scara 04.bmp

Fig. 8-25: Alternative operating modes

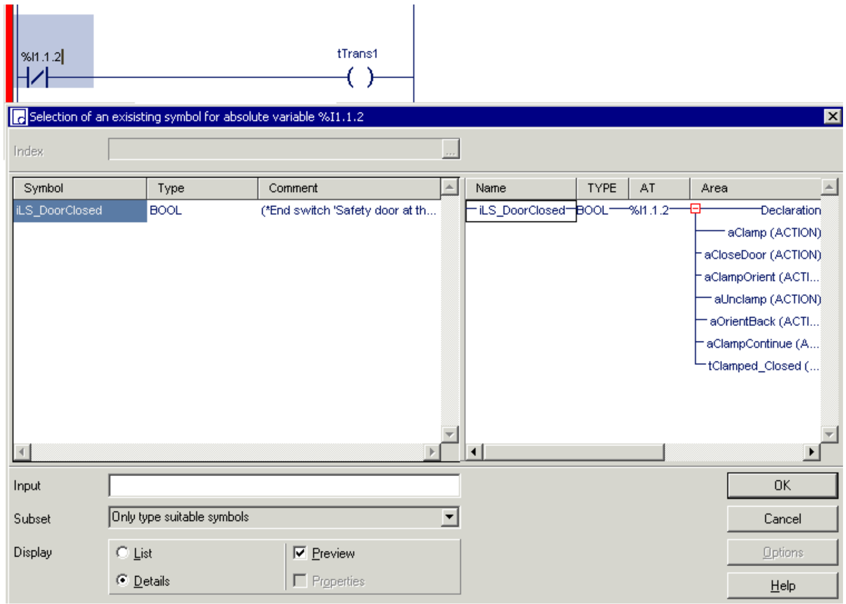
The return jump still missing for operating mode change follows:

Input sequence	Comment
	Cursor above transition MXPowerOn
<3> ***New_Mode***<Enter ***New mode selected?***<Enter>	Cursor below transition Power_Ok
<3>	The structure is now completely terminated.

Selection Window, Absolute Addressed Variables

When entering an absolute address – e.g. %I1.1.2 – WinPCL verifies, if a name, that is linked with the address via a declaration exists in the validity area of the file. If a name is found, it is indicated in the selection window.

When confirming the element obtains this name, when rejecting the absolute address remains at the element.



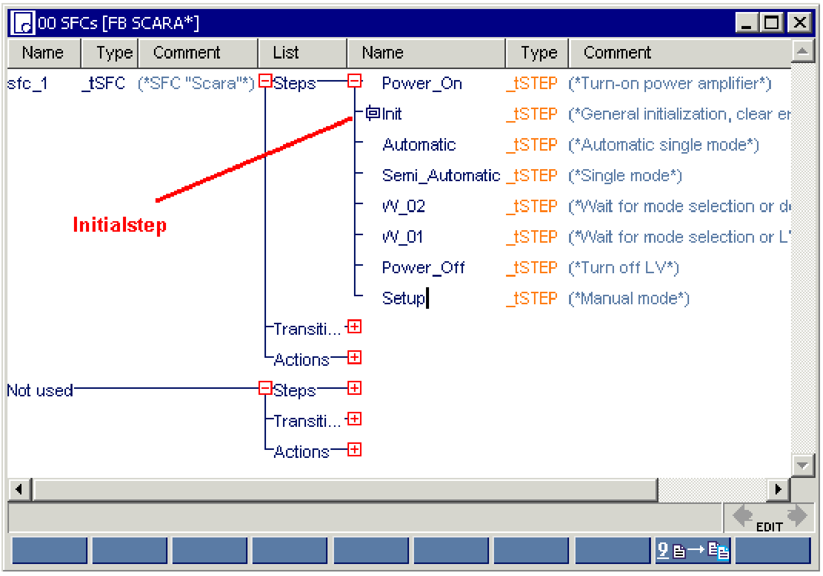
Auswahlfenster_Abs_Var_Trans.bmp

Fig. 8-26: Window to select the absolute address (here: in a detail of a transition)

Viewing the SFC in the SFC List

In the SFC list, the steps, transitions, and actions including their comments can be viewed in a drop-down menu (by clicking the mouse or pressing the <+> key).

The figure below shows the step list.



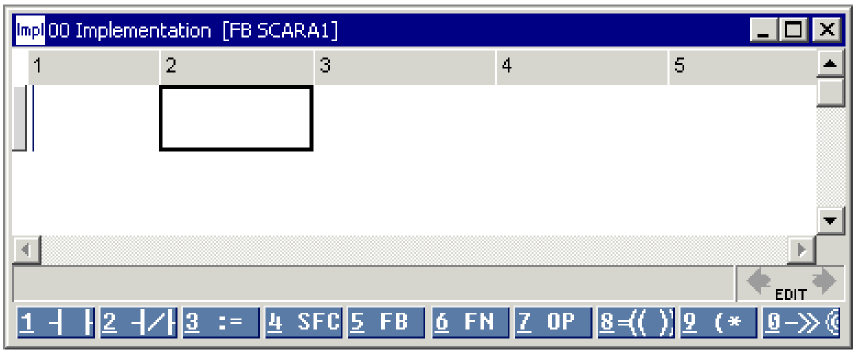
AblaufListe_01.bmp

Fig. 8-27: Steps of the Scara SFC in the SFC list

Double-clicking the mouse or pressing the <Ctrl>+<Enter> keys with the cursor positioned on the name of the step (or transition or action) permits "branching" into the step.

Entry of the Sequence for Execution in View / Implementation

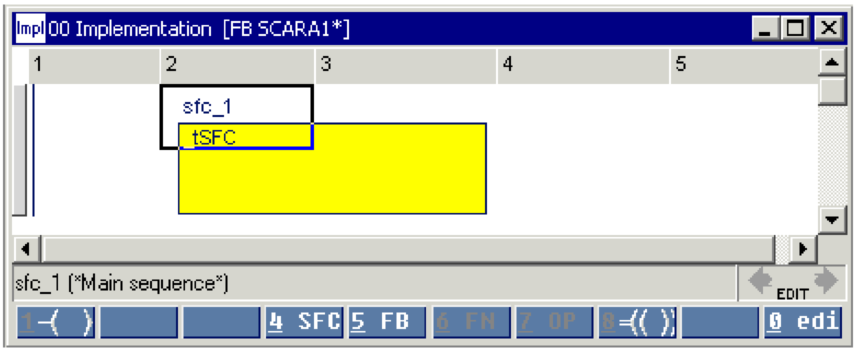
At that point, the SFC does exist in the POU, but has not yet been executed. Thereafter, the SFC in the menu item "View / Implementation" (LD or IL) must be used.



Impl_Ablauf_01.bmp

Fig. 8-28: Entering the SFC in a blank LD network. "4-SFC"

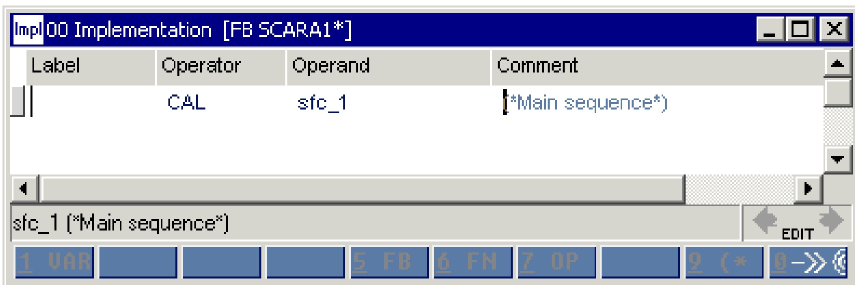
After clicking on "4-SFC", the desired SFC can be selected from the selection window. This SFC appears as a yellow "block" in the LD.



Impl_Ablauf_02.bmp

Fig. 8-29: Calling up the SFC in the implementation of the FB in the LD

After moving from the ladder diagram (LD) to the instruction list (IL), the following equivalent window opens:



Impl_Ablauf_03.bmp

Fig. 8-30: Calling up the SFC in the implementation of the FB in the IL

In this line, which is essential to execution, the name of the SFC is called up using the CAL instruction. The comment on the SFC is applied automatically.

Caution: The SFC is no function block instance!

Note: The SFC will not be executed if you forget this call.

Rexroth Bosch Group

Move to the sequence (SFC) by double-clicking the mouse or pressing <Ctrl>+<Enter> with the cursor positioned on the sequence name ("sfc_1").

The implementation can have any number of additional IL lines and LD networks. The SFC is executed at the corresponding place. Loading of the variables of the SFC for realizing operating modes is a typical application.

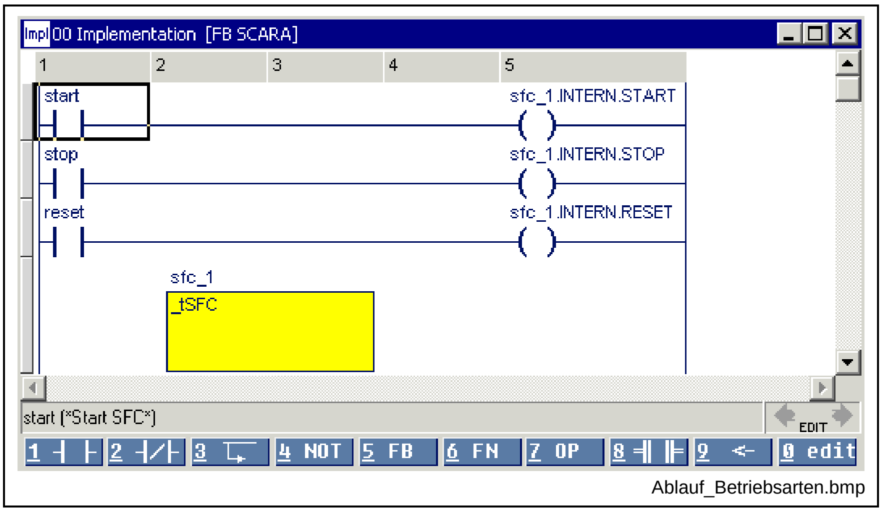


Fig. 8-31: SFC mode control in the ladder diagram

Note: Deleting an SFC from the implementation represents an online change.

The previous state of the SFC remains unchanged, but execution of the SFC is stopped.

Insertion of Steps, Transitions, Branches and Junctions

Additional elements are to be inserted in an existing SFC. To this end, the current footers of the editor and the original SFC will be shown, the necessary key combinations will be listed, and the new SFC will be specified.

Insertion of Steps and Transitions

A step from where a pair of transition and step is inserted, is the starting point. The SFC must not contain more than one opened branch.

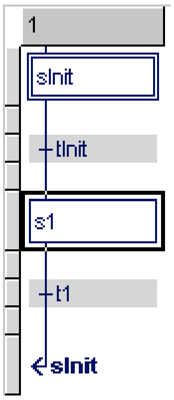
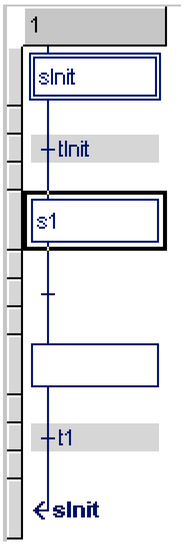
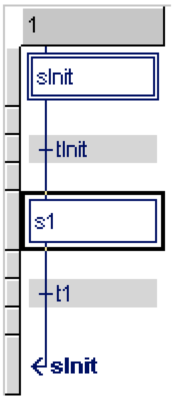
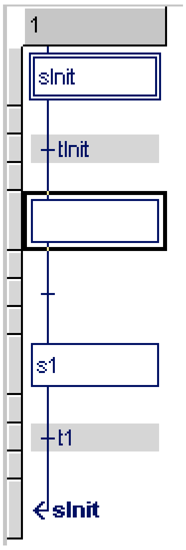
Before	Key sequence	After
	Place behind	
	Place before	

Fig. 8-32: Insertion of steps and transitions beginning with the step

A transition from where a pair of step and transition is inserted, is the starting point. The SFC must not contain more than one opened branch.

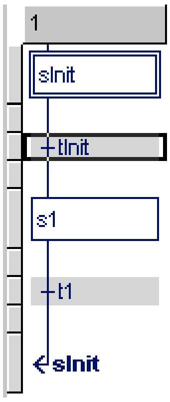
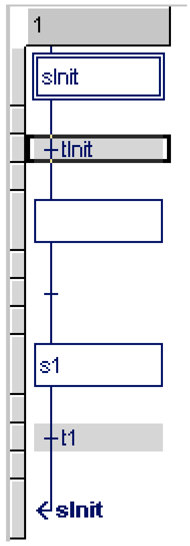
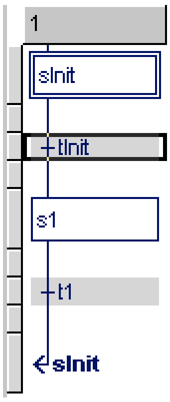
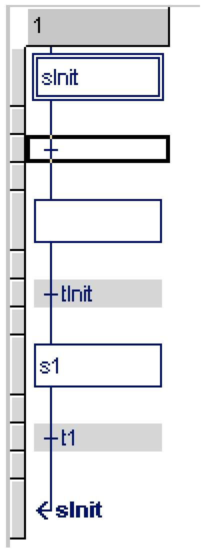
Before	**Key sequence**	**After**
	Place behind	
	Place before	

Fig. 8-33: Insertion of steps and transitions beginning with a transition

Opening and Closing OR Branches

OR branches always start after steps and end after transitions.

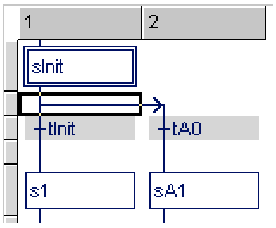
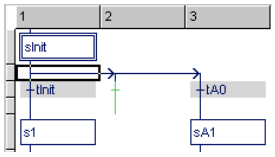
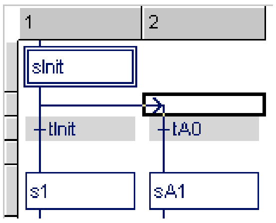
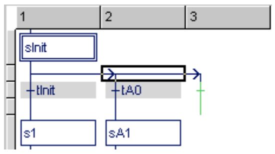
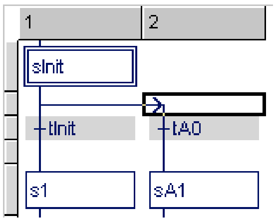
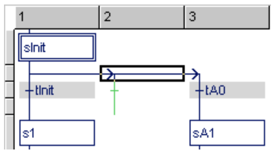
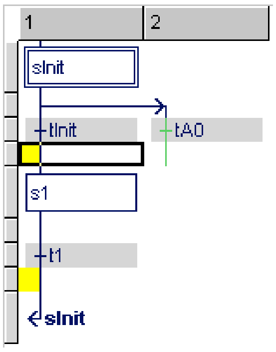
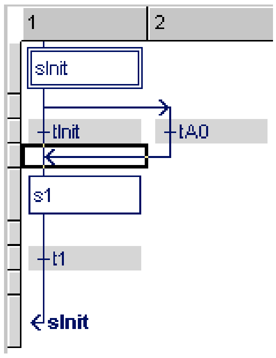
Before	Key sequence	After
		
		
	 	
	Closing must always be done to the left! Possible positions are shown in yellow in the figure.	
		

Fig. 8-34: Opening and closing OR branches

Opening and Closing AND Branches

AND branches always start after transitions and end after steps.

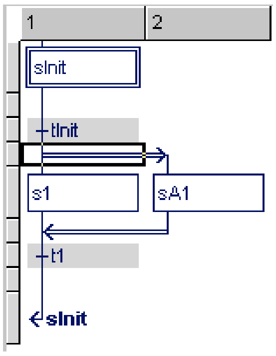
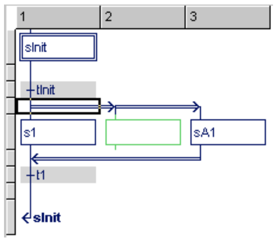
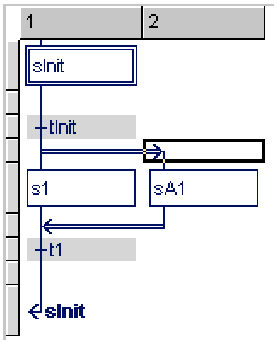
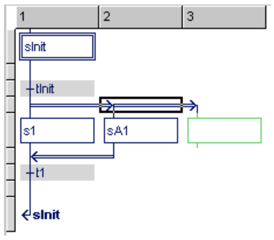
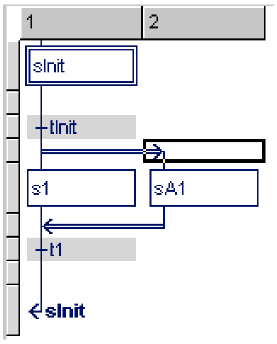
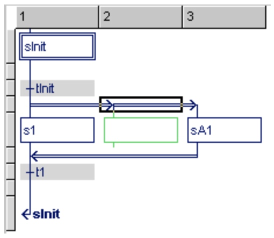
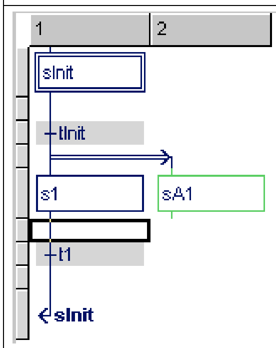
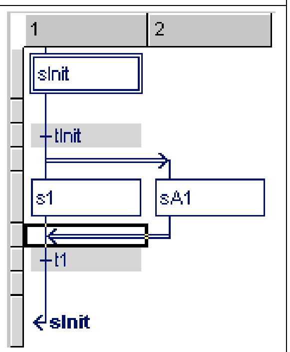
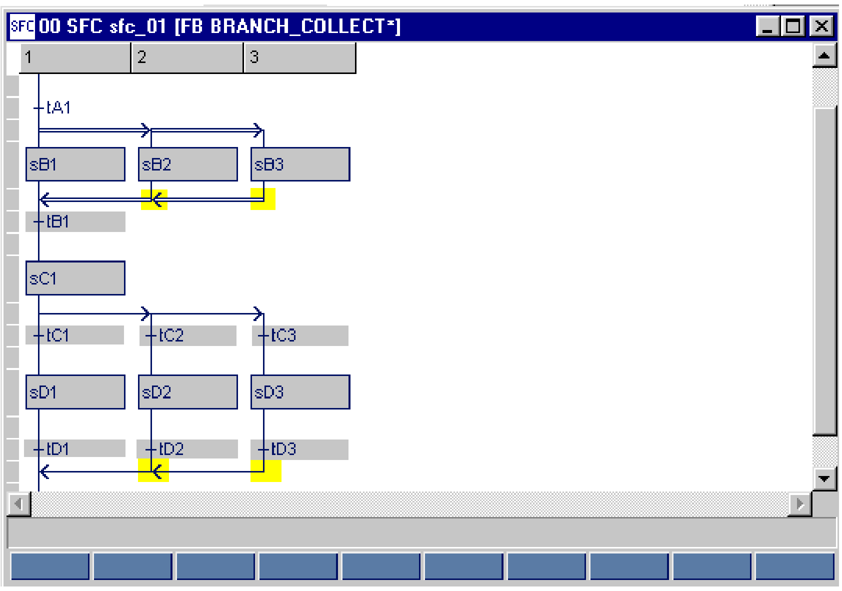
Before	**Key sequence**	**After**
		
		
		
	Closing must always be done to the left!	
		

Fig. 8-35: Opening and closing AND branches

Opening Branches

The opening of branches is triggered with the <Del> key.



branch_collect_02_open.bmp

Fig. 8-36: Cursor positions for opening branches with the <Del>- key

Note: Opening of branches requires that the SFC is completely closed at that point, since an SFC structure can only have one open branch.

Opening of the main branch is not possible.

Deletion of Steps, Transitions and Branches

The deletion of steps, transitions and branches is triggered with the <Del> key.

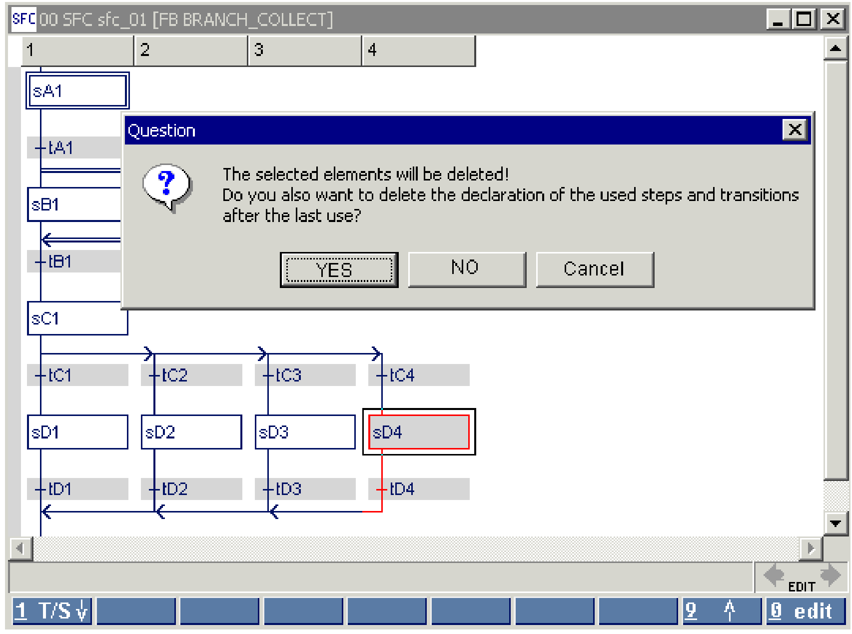
Note: As a standard, a step is always deleted including its action blocks and actions and a transition is always deleted including its advancing conditions, if used for the last time.

A warning is displayed before deletion takes place.

The <Del> key deletes

1. pairs, consisting of the step where the cursor is positioned and an immediately following transition, e.g. sD1 cursor position and tD1,
2. pairs, consisting of the transition where the cursor is positioned and an immediately following step, e.g. tC1 cursor position and sD1,

Rexroth Bosch Group



Löschen SFC_01.bmp

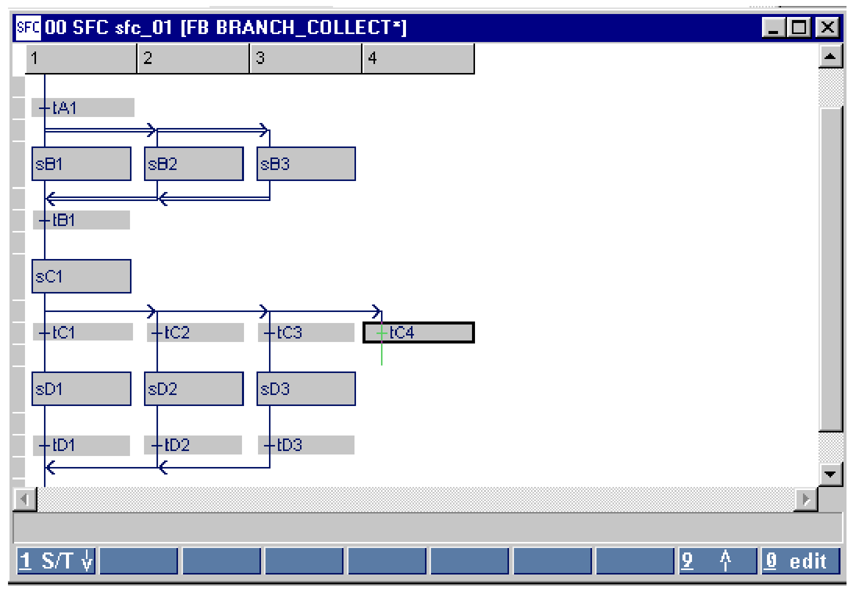
Yes Deletion of the red step + the red transition in the SFC and in the SFC list

No Deletion of the red step + the red transition in the SFC, but they are preserved in the SFC list.

Cancel The deletion process is stopped, steps and transitions remain preserved.

Fig. 8-37: Deletion of step and transition in pairs with the <Del> key

3. the respective single element, i.e. step or transition, of an open branch, e.g. transition tC4,



branch_collect_03_erase.bmp

Fig. 8-38: Deletion of the last element of an open branch

4. the complete branch (except the main branch at the junction), no matter whether this branch is open or closed.

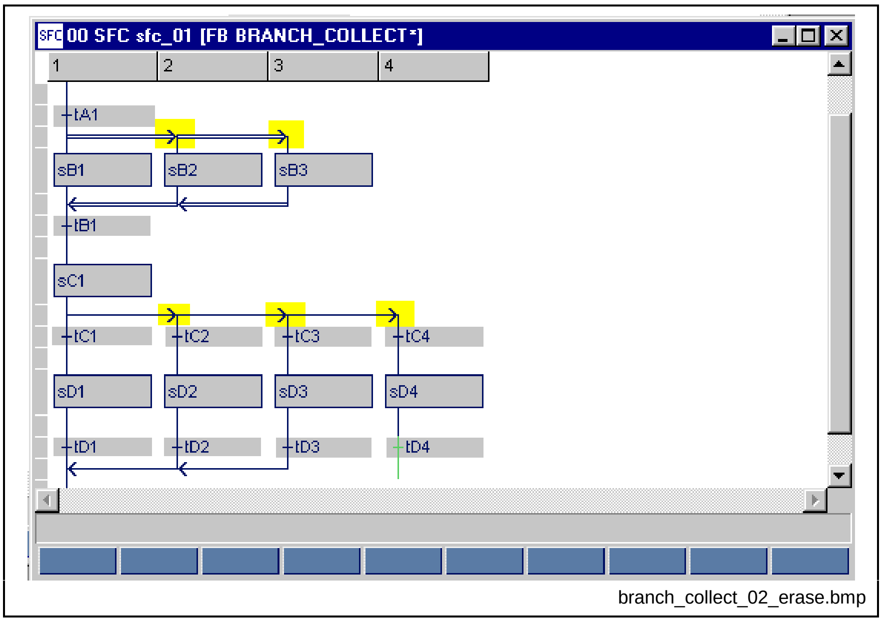
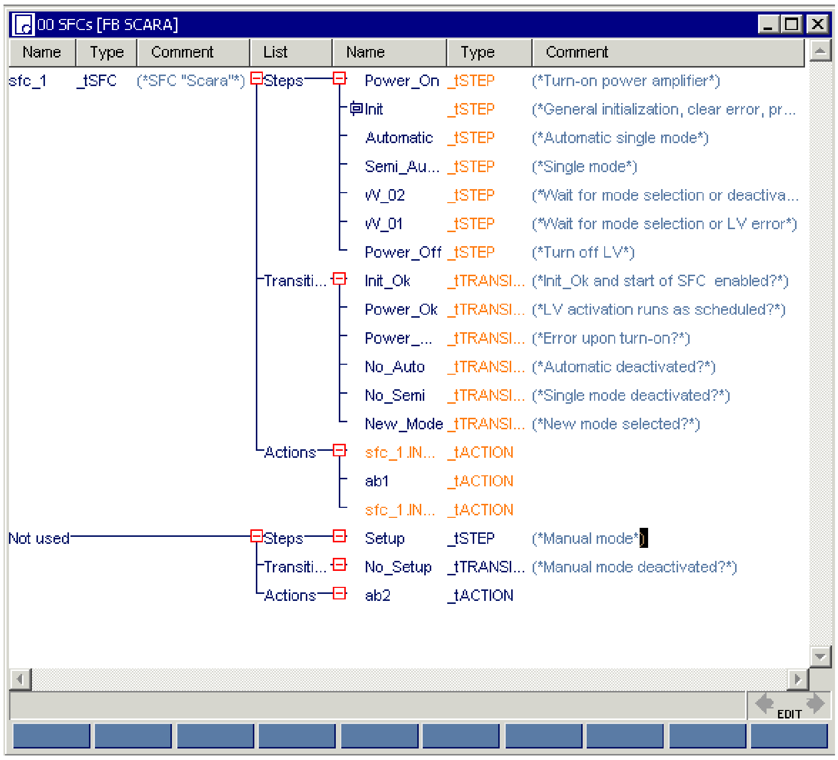


Fig. 8-39: Cursor positions for deleting the branches below

Note: If the opening and closing lines were superimposing each other after deletion, deletion would be prevented, e.g. tB1 cursor position and sC1 in the figure above !

Preserving Deleted Steps, Transitions and Actions; Re-use

If the prompt **"Also delete the declaration when using the step (transition / action block) for the last time"** is answered with NO, they remain preserved. They can be viewed in the lower part of the SFC list.



AblaufListe_02.bmp

Fig. 8-40: SFC list, "Setup" step and "No_Setup" transition deleted

The step "Setup" and the transition "No_Setup" are moved to the unused area, since they have been used for the last time before deletion.

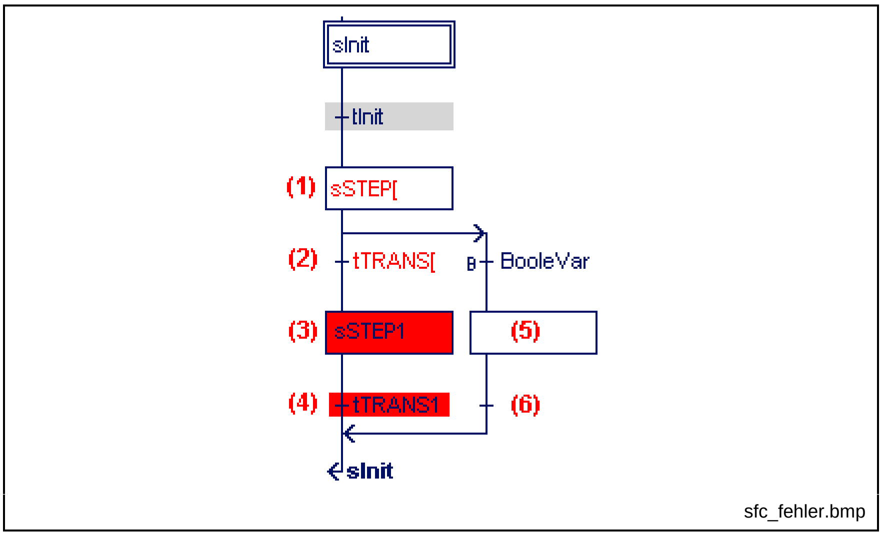
The step "Setup" is accompanied by the action "ab2" which it uses exclusively. This action has likewise been used only once.

Note: Blank steps or blank transitions as well as Boolean transitions are not moved to the lower area, because they can be reproduced without any problems.

If re-entered in one of the SFCs, the step or transition concerned is again moved to the appropriate position. The step is accompanied by its action.

8.3 Editing Features, Varying Color in the SFC Editor

The basic colors of a correct SFC are dark-blue font on a gray or white background. The color of the element changes to red if a step or transition is not filled in completely or is faulty. The following illustration shows some of the error situations:



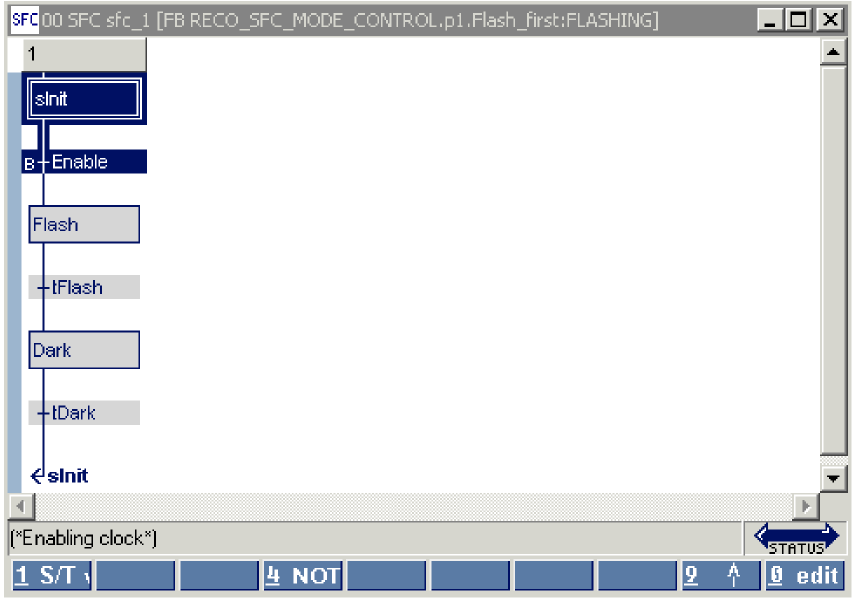
(1): Incorrect step name because of "["
(2): Incorrect transition name because of "["
(3): Error in an action block, is handed to the top
(4): Error in the transition, is handed to the top
(5): Step name is missing
(6): Transition name is missing

Fig. 8-41: Faulty SFC

8.4 Status Display in the Sequential Function Chart

If a program resides in the control system for execution, the following status information can be displayed in the SFC editor:

- Color-coded steps are active, and their action blocks can be executed.
- Color-coded transitions are tested for being fulfilled.



sfc_status.bmp

Fig. 8-42: Status display in the SFC editor

Further ways to obtain status information are:

- Start / Force <Shift>+<F8> for elementary variables (ANY_ELEMENTARY)
- Start / Status ARRAYs / Structures <Shift>+<F3>.

8.5 Options of the Sequential Function Chart

The options relevant for the SFC editor can be selected by means of the "Extras / Options" menu item:

Group	Option	Meaning
Desktop	Restore size and position during startup	The desktop is restored in the same size and position.
	Restore MDI window during startup	MDI windows are opened in the same order when restarting the system.
	Create backup copy	Automatic storage of the source condition which was loaded into the control in "downloaded files".
	Auto save	Allows the automatic saving of the current file in presettable time intervals without any prompt.
	Sound	Activation or deactivation of a beep sound.
View / All	Apply column width modifications automatically	Column width changes are automatically stored.
	Apply declaration comment in implementation	Comments, that have been entered in the respective declaration line are displayed in the implementation. The implementation can be changed; the comment is then doubled, the declaration line remains unaffected.
	Variable display	With symbols (name) or absolute (address).
	Display of absolute variables	The user can select from I/Q, E/A and I/O for absolute addresses.
	Truncating very long texts	Texts and numbers can be truncated to the right or left, and
	Truncating very long numbers	can be represented with or without "..." marking.
View / SFC	Column width for the individual columns	72
	(with standard values)	

Fig. 8-43: SFC editor options

8.6 Pop-up Menu, Sequential Function Chart <Shift>+<F10>

This pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	Explanation
Open	Branch to the step, transition, also <Ctrl>+<Enter>
Edit comment	Input or change of the step or transition comment
Delete	Deletion of the current element and its successor.
Step time	Input or change of the minimum and maximum step dwell time
Diagnosis properties	Display and modification of the diagnosis properties.
Import implementation	The ASCII file chosen from the "WINPCL text files" is attached to the current element.
Export step / transition	The current step / transition pair is exported as an ASCII file and stored in the folder "WINPCL text files".
Export SFC	The complete SFC is exported as an ASCII file and stored in the folder "WINPCL text files".
Syntax text	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Error help	The sequence, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the data type of the current element, where the cursor is positioned.
Cross reference help	List of all places where the current element is used. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force PLC in operating mode "STATUS"	Allows the entry of a variable name. The value of the variables is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of program code execution.
Status ARRAYs / Structures	Display of the status of array and structure elements, forcing by pressing the <Shift>+<F10> keys or the right mouse button.
Print current window	Print of the editor contents by pressing <Ctrl>+<P>.
Options	Optimization of the column width.
Internals	Search for faults in the programming system, to be used only if approved by the service.

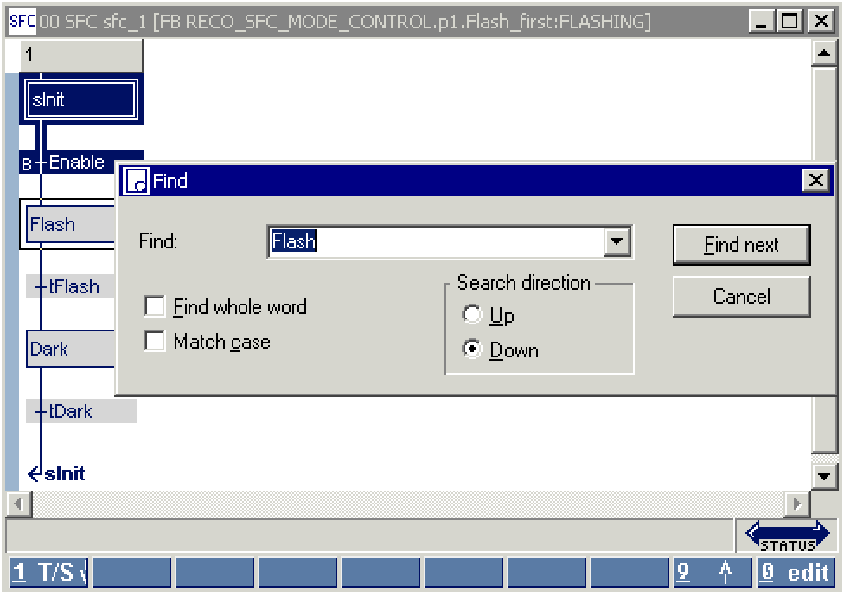
Fig. 8-44: Pop-up menu of the SFC editor

8.7 Block Commands, Sequential Function Chart

So far, block commands have not been realized in the sequential function chart.

8.8 Search and Replace, Sequential Function Chart

The search function is in the first version and provides the features of a text editor:



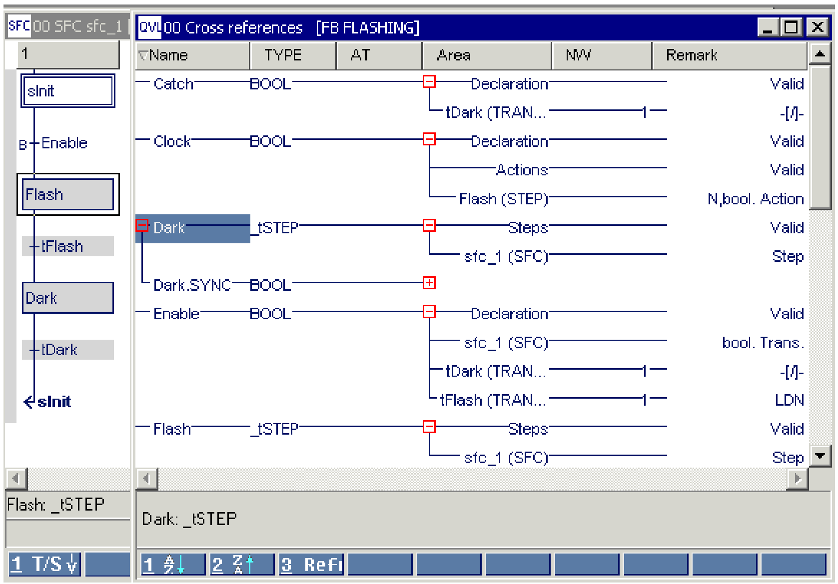
sfc_suchen.bmp

Fig. 8-45: Search in the SFC editor

The replace function is in preparation.

8.9 Cross Reference List, Sequential Function Chart

In contrast to the cross references of the pop-pup menu, the overview you obtained via "View / Cross reference list" shows all variables. Only variables from lines with a correct syntax can be resolved by their place of use. However, all faulty names or names with double declaration are displayed and can, thus, be reached with by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.



sfc_qvl.bmp

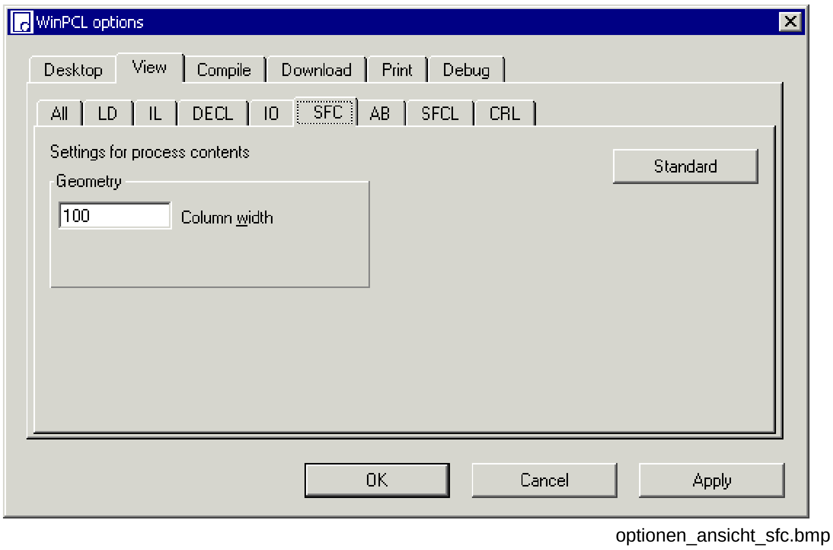
Fig. 8-46: Excerpt from the cross reference list of a function block with SFC elements

Name	Type	Area	Use	Comment
Flash	TSTEP	Steps	Valid	Step list
		SFC sfc_1	Step	Step in SFC sfc_1
Enable	BOOL	Declaration	Valid	Declaration of the Boolean variables
		SFC sfc_1		Boolean transition in SFC sfc_1
		TRANSITION continue	Normally open contact in network 1	Ladder diagram network
		TRANSITION continue	Negated reading of network 1	IL network
Clock	BOOL	Declaration	Valid	Declaration of the Boolean variables
		Actions	Valid	Action list
		STEP Flash	Boolean action, non-storing (N)	Action block in step Flash
Count	TACTION	Actions	Valid	Action list
		STEP Flash	Action, once with step activation (P1)	Action block in step Flash

Fig. 8-47: Comment on cross reference list (shortened)

8.10 Documentation, Sequential Function Chart

The SFCs are documented (print from the editor, <Ctrl>+<P>) using the column width defined under Extras / Options / View / SFC.

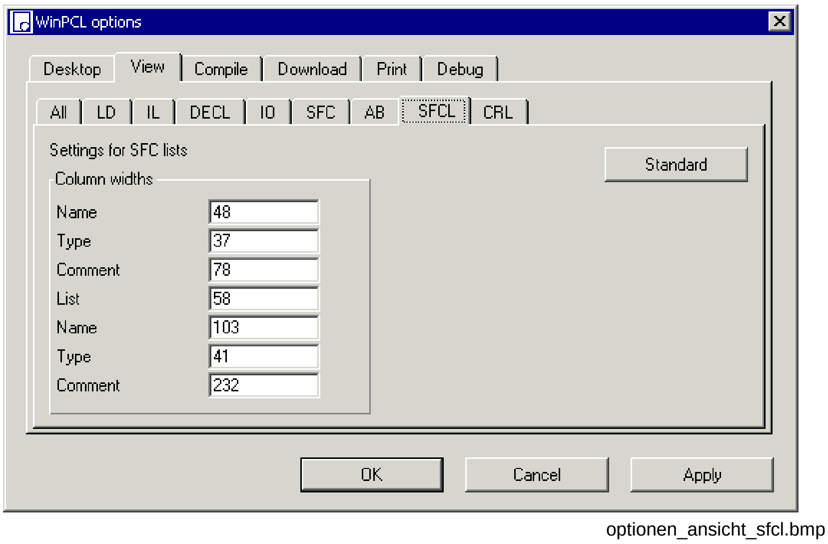


optionen_ansicht_sfc.bmp

Fig. 8-48: Options, sequential function chart (SFC)

The "Apply" button activates the column width set for the SFC editor. The width of the column can either be entered in the window shown above or preset in the editor by dragging the headers.

SFC lists are documented using the settings defined in Extras / Options / SFCL (SFC element lists).



optionen_ansicht_sfcl.bmp

Fig. 8-49: Options of the SFC element lists

The "Apply" button activates the column width set for the lists. The width of the column can either be entered in the window shown above or preset in the editor by dragging the headers.

The "Standard" button resets the default.

The "OK" button applies the setting and closes the dialog window.

The "Cancel" button closes the window; the previous values are kept.

Detailed information on the real print process and the features is to be found in the main chapter on WinPCL.

9 Action Block Editor

9.1 Action Blocks and Their Operating Principle

Zero, one or several actions (ACTION) arranged in action blocks (ACTION BLOCK) can be connected to each step of a sequential function chart (SFCs).

A step without actions results in waiting for fulfillment of the following transition condition.

An action may include

- a Boolean variable,
- a negated Boolean variable,
- a sequence of instructions in IL or
- a number of networks in LD.

It is possible to make single or multiple use of an action in only one SFC.

However, the action can also be unused and "in reserve".

A zero-use action is present if an action block in a step is deleted but its declaration is to be preserved or if a step is deleted and deletion of the action blocks pertaining thereto is cancelled.

This action can be integrated in use again by entering its name in an action block, without the details getting lost.

The SFC list gives an overview of the actions which are actually existing in an SFC in the current program organization unit (see Actions in the SFC List).

Function blocks with internal SFCs can be included in the IL or the LD networks of an action.

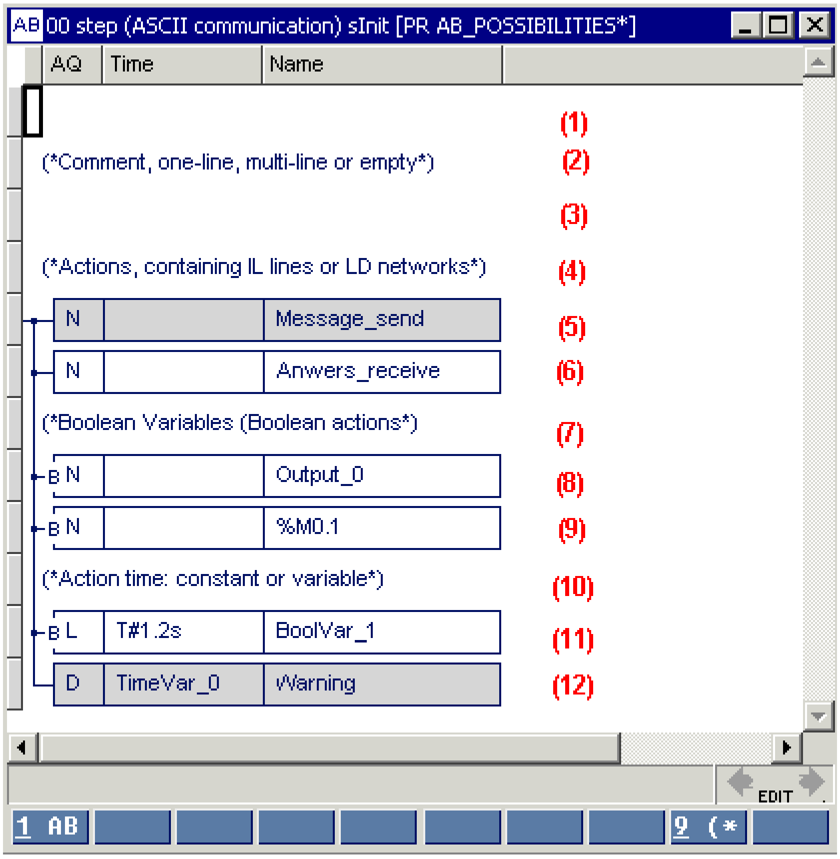
A comment can be assigned to each action. This comment, like the declaration comment of a variable, is bound to the name of the action and is available at the place where the action is used.

The time period required for executing of the action is defined by the action block.

Structure of an Action Block

The action block editor, i.e. the basic editor, with four columns (see figure), serves for entering or changing action blocks:

- Connection line: The connection line determines whether a comment or an action block uses the line.0
- Action qualifier "AQ": defines the execution type for the action after the step has become active.
- Action time "Time": Some action qualifiers have to be supplemented by this time. A constant or a variable of type TIME can be used as action time.
- Action name "Name": The ‘action name' field can contain the name of a complex action - instruction list or LD networks. Alternatively, the name of a true or negated Boolean variable can be entered as an absolute or a symbolic value. For a clear overview, these Boolean variables are marked with "B" or "/B". An empty action, which means that it is not filled in yet, shows a white background, a filled-in action a gray background.



AB_Möglichkeiten.bmp

(1): New empty line; decision by footer commands whether an action block 1-AB or a comment 9- (* is to be entered.
(2): One-line comment, can be used as often as required
(3): Empty line
(5): Action has already been filled in with IL lines or LD networks.
(6): Action is still empty but the name is not the name of a Boolean variable.
(8): Boolean variable is controlled, here symbolic name.
(9) Negated Boolean variable, here absolutely addressed flag.
(11): Action time is defined by the time constant "T#...".
(12): Action time is defined by a variable of type TIME.

Fig. 9-1: Possibilities in the action block editor

Time period	in	Time constants for the action time
ms	Milliseconds	T#14ms,
s	Seconds	T#14s,
m	Minutes	T#47m,
h	Hours	T#147h,
d	Days	T#14d,
Mixed		T#25h15s, T#1h3.3s

Fig. 9-2: Example for action times (time constants)

Note: If a variable of type "TIME" is used as action time, this variable can be updated only before the execution starts.

9.2 Action Block Editing

The action block editor will be described by means of a simple example illustrating the following steps: entry, placing action blocks before or behind, deletion, action blocks.

You can move to the action block editor by positioning the cursor on the required step in the SFC editor and

- pressing the <Ctrl>+<Enter> keys,
- double-clicking the step, or
- triggering the "Open" command in the pop-up menu which can be opened by pressing <Shift>+<F10> or the right mouse button.

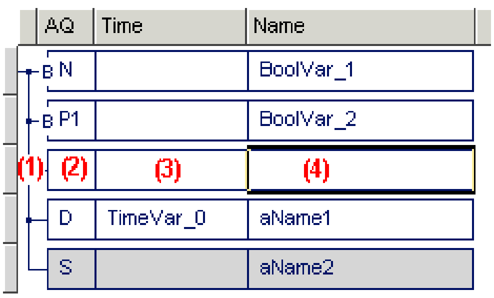
It is also possible to move to the action block editor by using the "View / SFCs" menu item or by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys with the cursor positioned on the desired step.

Entering an Action Block, Placing it Behind and Before

Entering an action block requires an empty line.

This empty line is either provided after branching or can be generated by pressing the <Enter> key.

- Pressing the <Enter> key in the extreme left column generates a preceding empty line.
- Pressing the <Enter> key at any other position, except as confirmation of the entry, generates a following empty line.



nach und voran.bmp

(1): Placing before
(2), (3), (4): Placing behind

Fig. 9-3: Positions for placing the line before and behind

The action block is entered in an empty line with the footer:

- 1 - AB: draws the basic body of an action block,
- 9 - (*: allows the entry of a single-line comment.

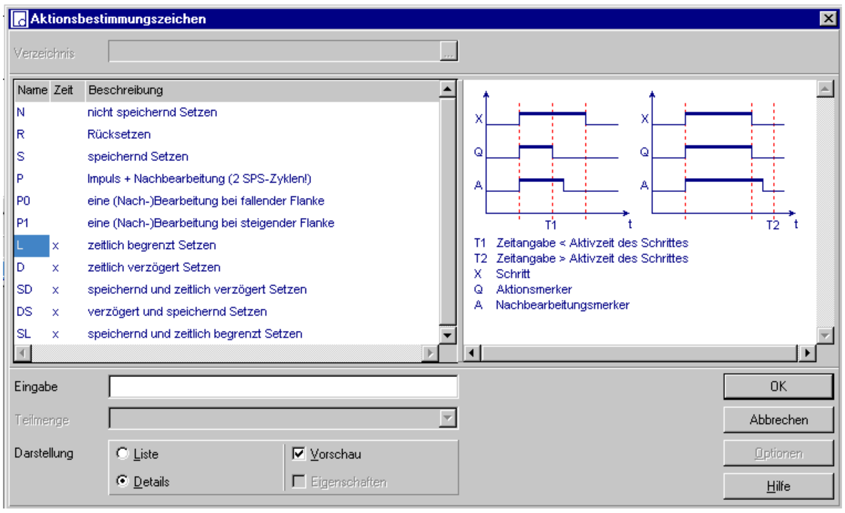
The input of an action block becomes through following list boxes, that about footer commands <ALT>+ are attainable number key, supports:

Key combinations	Name	Column	Remark
<ALT>+<5>	AQ	AQ	Selection Window, Action Qualifier
<ALT>+<6>	Time	Time	Selection Window, Time Variables
<ALT>+<7>	Name	Name	Selection Window, Actions / Boolean Variable

Selection Window, Action Qualifiers

The footer command "<Alt>+<5>-AQ" opens the selection window for action qualifiers. They can be represented by a list (space saving) or detail information.

If the option "Preview" is selected, a graphic representation of the step flag / action flag / postprocessing flag appears.



Auswahl_BZ.bmp

Fig. 9-4: Selection window, action qualifiers

The name of the desired action qualifier can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Time Variables

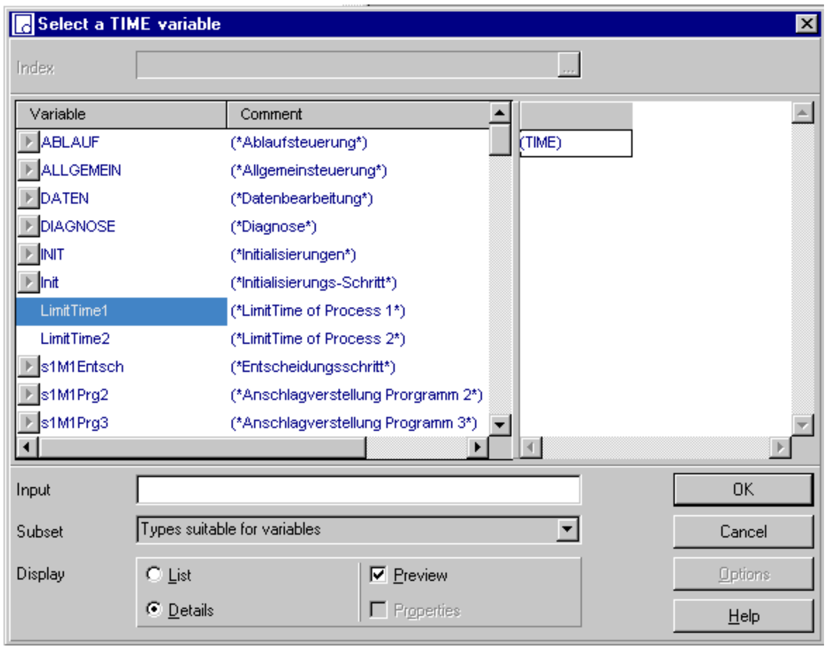
The footer command "<Alt>+<6>" opens the selection window for time variables. They can be represented by a list (space saving) or detail information.

If the option "Preview" is selected, a graphic representation of the time variable including its comment appears.

In addition to the name of the time variable, its declaration comment is displayed.

Multi-element variables (instances of structures, ARRAYs or FBs) are marked by a preceded gray arrow. You can open this variables by double-clicking or pressing <Enter> on the respective variable. The elements can be (recursively) selected.

In future, a switch for the return path upwards to the instance name is planned.



Auswahl_Time.bmp

Fig. 9-5: Selection window, time variables

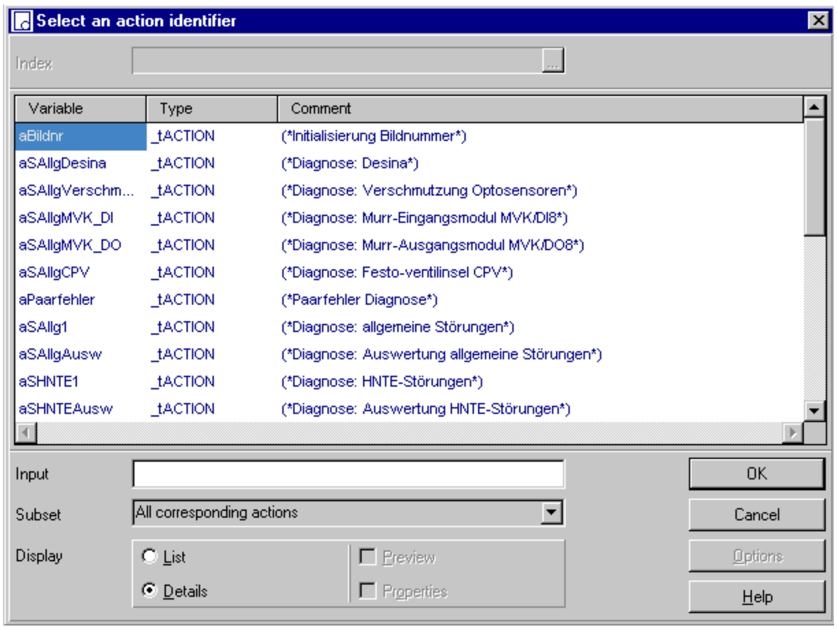
The name of the desired time variable can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Selection Window, Actions / Boolean Variables

The footer command "<Alt>+<7>-Name" opens the selection window for time variables. They can be represented by a list (space saving) or detail information.

If the option "Preview" is selected, a graphic representation of the action / Boolean variable including its comment appears.

In addition to the name of the actions / Boolean variables their declaration comment is displayed.



Auswahl_Aktion.bmp

Fig. 9-6: Selection window, actions / Boolean variables

The name of the desired time action / Boolean variable can be entered in the input field. While entering letter by letter of the name the cursor in the selection window jumps to the respective item with the corresponding initial letters.

Furthermore, the following subsets can be selected:

- All suitable actions (control, whether type fitting the step)
- All suizable actions and Boolean variables
- All Boolean variables

The items can be displayed as list or detailed representation.

Selection Window, Absolute Addressed Variables

When entering an absolute address – e.g. %I1.1.2 – WinPCL verifies, if a name, that is linked with the address via a declaration exists in the validity area of the file. If a name is found, it is indicated in the selection window.

When confirming the element obtains this name, when rejecting the absolute address remains at the element.

The window is used in the action block editor to enter an action name.

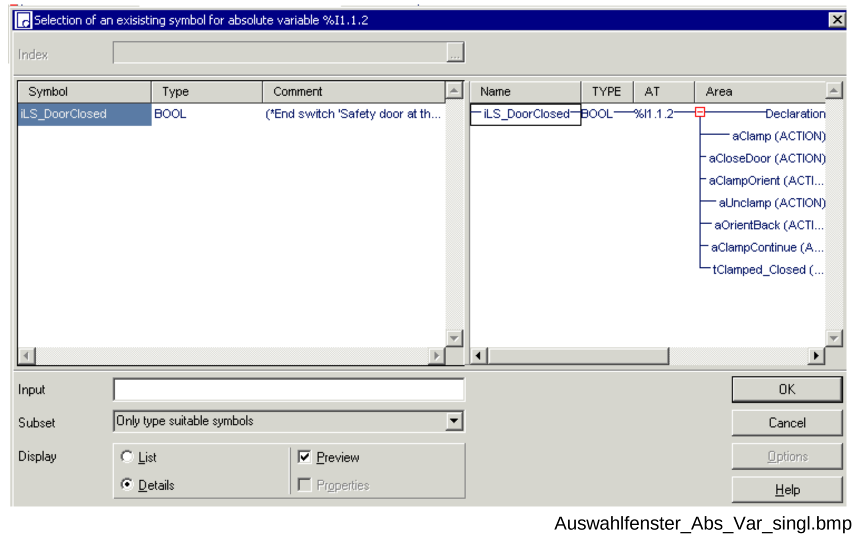
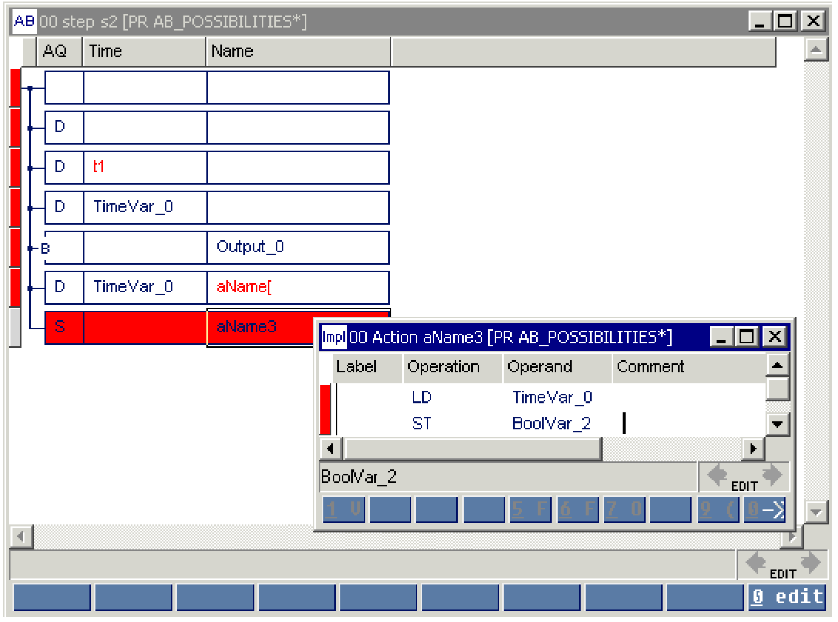


Fig. 9-7: Window to select the absolute address

Editing Features, Varying Color in the Action Block Editor

The basic colors of a correct action block are a dark-blue font on a gray or white background. If an action block is not filled in completely or correctly, the color of the marginal marking or of one of the fields changes to red. The following figure shows some of the error situations:



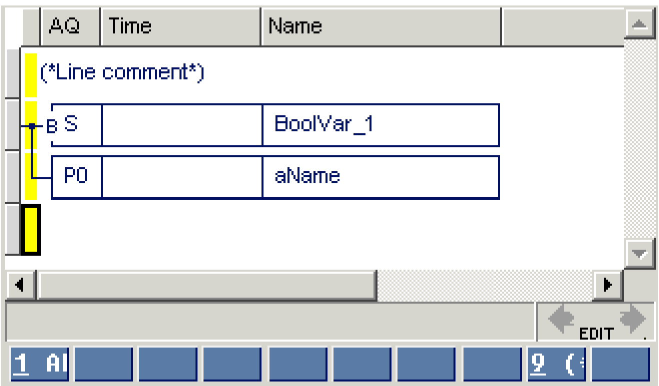
AB_Fehler.bmp

(1): Action block is still empty
(2): Correct action qualifier, time and action name are missing
(3): Correct action qualifier, time is faulty (wrong type)
(4): Action name or variable name is missing
(5): Action qualifier is missing
(6): Incorrect name because of "["
(7): Error inside the action, is handed to the top

Fig. 9-8: Incorrect action blocks

Deletion of an Action Block

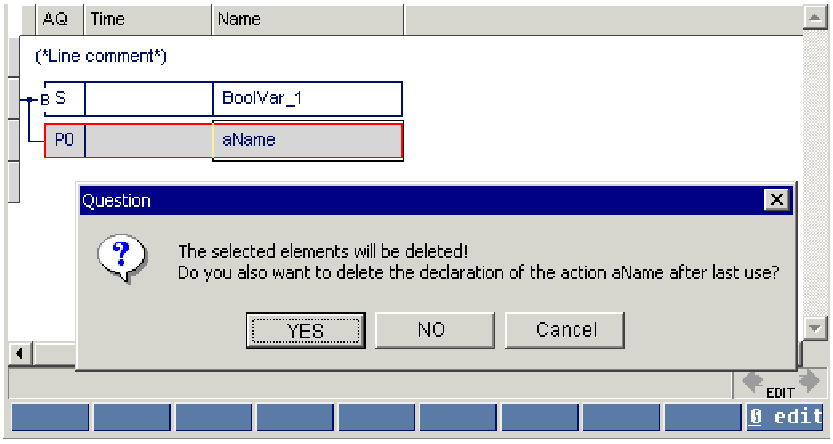
The respective action block, the comment included therein and the empty line can be deleted by pressing the <Del> key on the left connection line. The deletion positions were subsequently marked in yellow.



Löschen AB.bmp

Fig. 9-9: Positions for deleting action blocks and comments

Before an action block with ladder diagram networks or IL lines is definitively deleted, the following warning is displayed:



Löschen AB_01.bmp

Yes Deletion of the action block and deletion of the action "aName"
No Deletion of the action block, the action "aName" is still present in the SFC list under "not used".
Cancel: The deletion process is canceled, the action block is preserved

Fig. 9-10: Warning displayed before deletion of an action block

The element to be deleted is in a red box.

Text Modifications in an Action Block

If corrections are necessary, the cursor has to be positioned to the desired position in the respective line.

If you start writing immediately, the new text completely replaces the previous text.

The previous text can be edited if you press '0'-Edit before you start with the text entry.

Note: A new, empty action is created when you change the name of an action.

Multiple Use of Actions

An action can be used in several action blocks, if necessary with different action qualifiers, in several steps. However, these steps must also be used in the same SFC.

The following footer commands can be applied to actions with double or multiple use:

Footer commands

<0>-Edit

With entry of the new name, a new independent action is created. The system asks the user whether he also wants to delete the declaration of the action entered up to that point after the last use.

When the name of an already existing action is entered, this action becomes effective including its content.

(For further activities, see Actions in the SFC List)

Detail Level of the Action Block Editor

For being complete , the actions included in the action blocks have to be filled in. This does not apply to Boolean actions.

1. Position the cursor on the desired action block.
2. Press <Ctrl>+<Enter> or double-click the mouse.

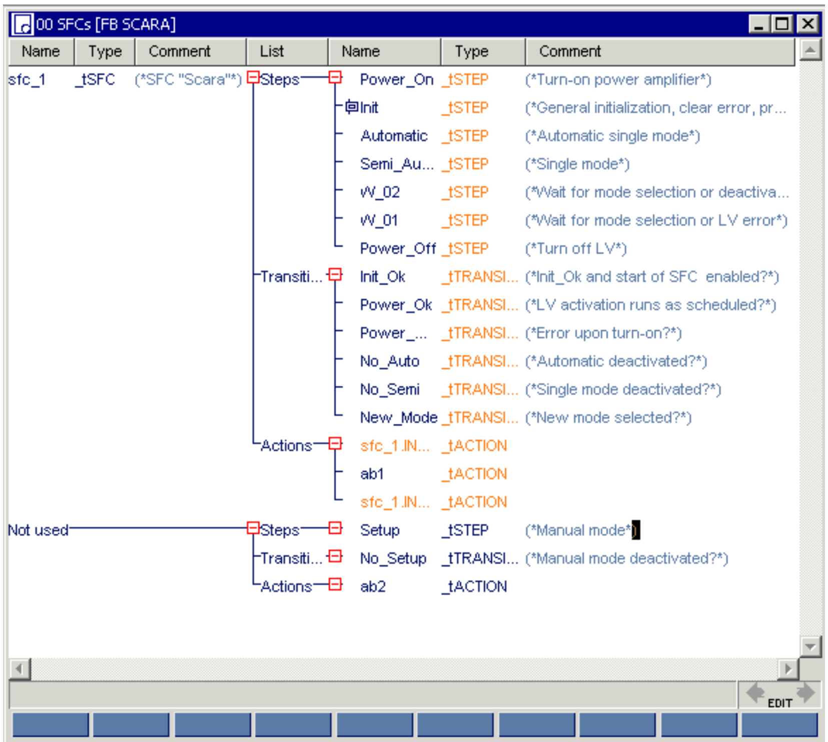
Actions in the SFC List

In the sequence (SFC) of a function block or a program, it is possible to make zero, single or multiple use of actions . The necessary overview is provided by a list showing how the actions are assigned to the steps of the SFCs. This list can be opened using the "View / SFCs" menu item.

Note: It is possible to make single or multiple use of an action in an SFC.

In a POU, it is possible to make zero use of an action.

The use of an action in several SFCs of the POU is not permitted.



AblaufListe_03.bmp

Action "ab1" used at least once in a step of sfc_1.
Action "ab" not used in sfc_1, either not assigned to a step or pertaining to step "Setup" which is not used either.

Fig. 9-11: Actions in the SFC list

The SFC list allows the following activities:

- Unused actions can be deleted by pressing the <Del> key.
- Actions can be provided with comments.
- Double-clicking the mouse or pressing the <Ctrl>+<Enter> keys with the cursor positioned on an action name permits branching into the action.
- Actions can be renamed, i.e. their name is updated at the places where they are used (editing the name).
- Actions can be  doubled; original and copy have the same contents; the copy is "not used".
- Actions are executed in the order they are listed in the SFC list. The starting order can be seen from the SFC graphic. The order can be modified by means of the  and  keys. The order can be rearranged according to the starting order by means of the  key.

System Data for Actions and Action Blocks

Similar to steps, transitions and the SFC itself, additional variables can be assigned to the actions:

Name	Type	Comment
action_name.Q	BOOL	Indicates whether the action is being executed.
action_name.A	BOOL	Indicates whether the action is being executed, postprocessed or forced (all execution methods).
action_name.F	BOOL	Is the variable by which the action can be forced and which indicates at the same time whether the action is being forced.
action_name.JOG	BOOL	Only important in automatic jog mode; indicates that the following transition is fulfilled.

Fig. 9-12: Variables which are assigned to an action with _tACTION

These structure elements are not available for Boolean actions and negated Boolean actions.

General Method of Action Execution

Chronological coordination of action execution

Execution of the actions is firmly bound to the graphic structure of the SFC. Each step of the SFC forms a complex functionality which is assigned to a real process. For that reason, the actions must be executed as specified by the steps. The structure interpreter realizes the following execution features.

- Run through the steps of the SFC takes place line by line from left to right and top to bottom.
- The action blocks are executed in the order defined by the order of the steps, i.e. from left to right and from top to bottom (see Actions in the SFC List).
- If an action is used several times, its classification is done as early as possible.
- Multiple use is not possible (see Execution by Action_Control).

Chronological coordination of the action postprocessing

Postprocessing of all actions takes place in alternation with the ActionControl outputs Q and A.

Since actions must be postprocessed only once, this is achieved independently of the graphic structure of the SFC. The actions are postprocessed according to their order in the action list.

Examples will be described in the following sections:

Definition of the instant ‘Step is active’

If the condition of the transition ‘tOn’ is fulfilled, the step is evaluated as being active in the same PLC cycle upon calculation of the action.

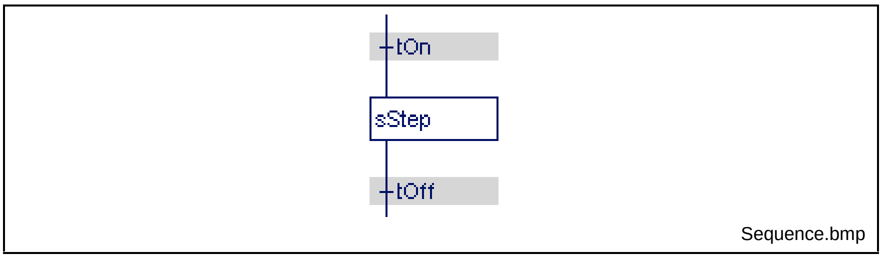


Fig. 9-13: Step becomes active

Definition of the instant ‘Step is inactive’

If the condition of the transition "tOff" is fulfilled, the step is evaluated as being inactive in the same PLC cycle upon calculation of the action.

A step becoming inactive causes the Boolean variables and the negated variables to be still calculated in the same PLC cycle, as indicated above.

Consequences from postprocessing - example

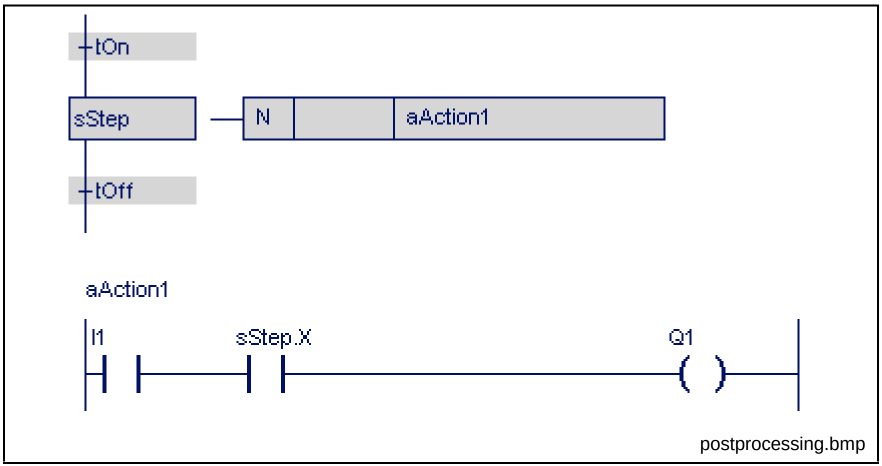


Fig. 9-14: Consequences from postprocessing

The assignment of the variables xxx.Q and xxx.A shows whether normal execution (xxx.Q / xxx.A: TRUE / TRUE) or already postprocessing (xxx.Q / xxx.A: FALSE / TRUE) is running.

In this example this means, that relay Q1 is switched by the normally open contact I1 while step sStep is active.

Postprocessing takes place after sStep has become inactive (sStep.X : = FALSE). Relay Q1 is deactivated independently of the normally open contact I1.

If postprocessing is to be prevented, the action can be skipped by means of the combination (xxx.Q / xxx.A: FALSE / TRUE).

Execution by Action_Control

The IEC-61131-3 uses the Action_Control model for defining the execution rules for actions of an SFC. This model assumes that for each action of a POU an instance of the function block Action_Control, defined by the standard, exists.

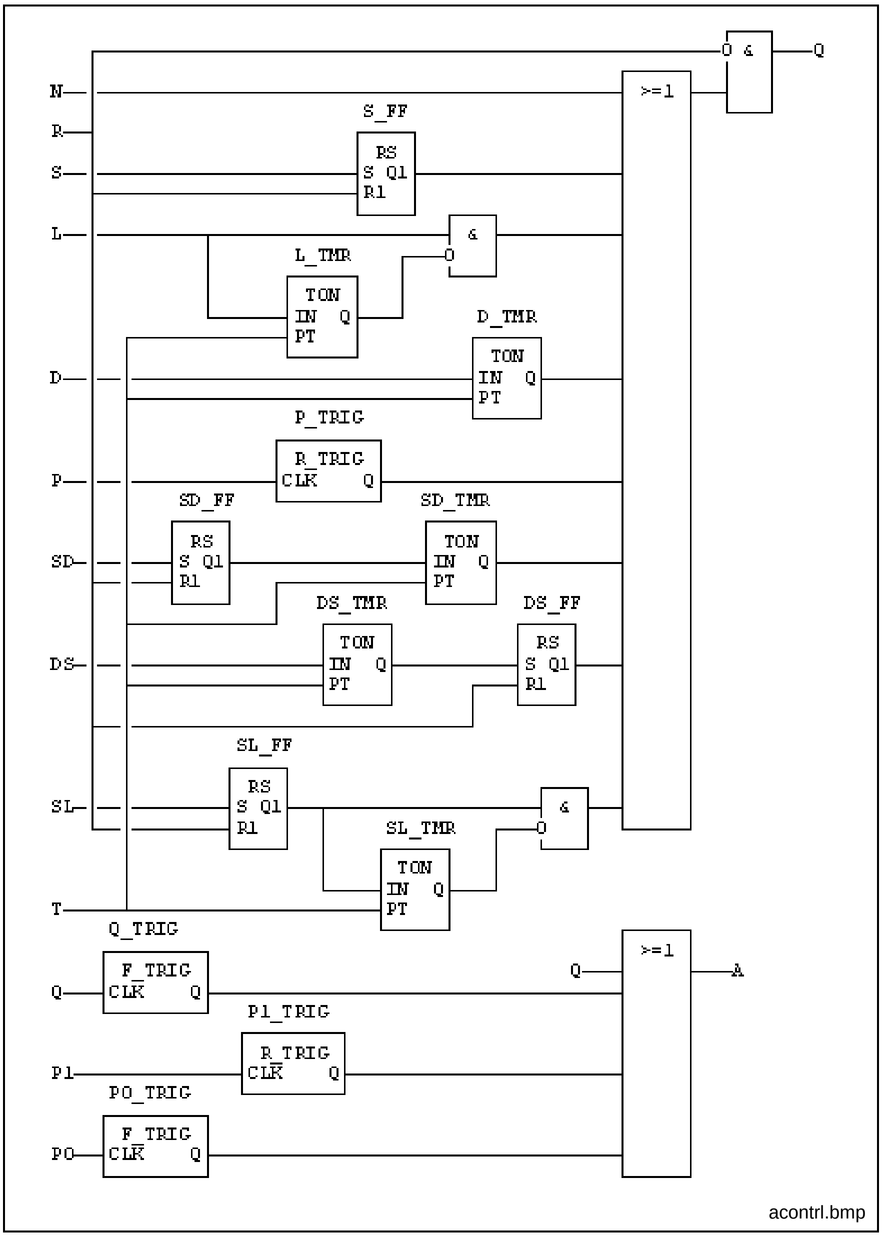


Fig. 9-15: Action control according to EN 61131-3

This function block shows the execution of the related action by its outputs Action_Control.Q or Action_Control.A as follows (forcing of actions not included).

- The input connection of the Action_Control function block is established by the action blocks which are connected to the steps. A connection between a step and an input of an instance of the function block Action_Control exists if an action block of this step references the same action, which is assigned to the function block instance Action_Control.
- Dependent on the step activity such a connection can be active or inactive.

- An input of a function block instance Action_Control is to be considered as connected to TRUE if there is at least one active connection to a step. Otherwise the input is considered to be connected to FALSE.

Action Qualifiers and their Execution

The following qualifiers and action times are supported:

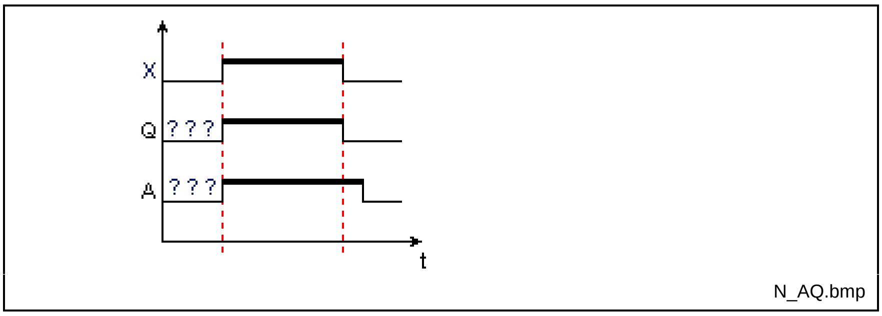
Action qualifier	Action time required	Comment
N	No	Non-storing
R	No	Dominating reset
S	No	Storing
L	Yes	Non-storing, limited in time
D	Yes	Non-storing, delayed in time
P	No	Pulse + postprocessing (2 PCL cycles)
P0	No	(Post) processing with falling edge 1x
P1	No	(Post) processing with rising edge 1x
SD	Yes	Storing, delayed in time
DS	Yes	Delayed in time, storing
SL	Yes	Storing, limited in time

This section deals with the specific reactions for all action qualifiers in relation to the step being active. The execution of a Boolean variable, a negated Boolean variable and a complex action will be studied.

N

Non-storing.

Execution is running as long as the ‘step’ is active.



X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-16: Time diagram for action qualifier "N"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive..

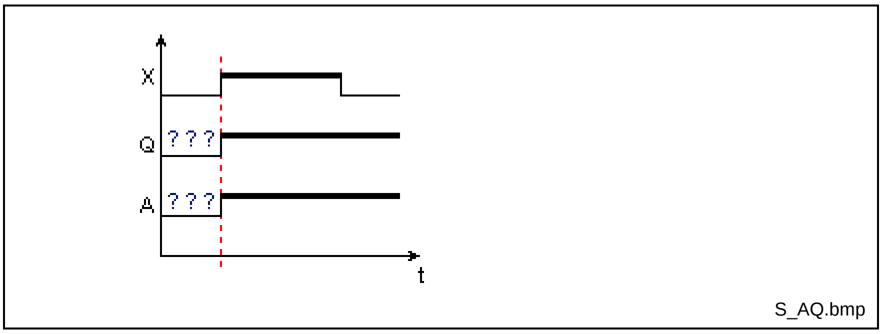
The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is executed and postprocessed according to the action flag.

S

Storing.

Execution starts when the step becomes active until the reset command is given.



X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-17: Time diagram for action qualifier "S"

	Boolean variable	**Negated Boolean variable**	**Complex action**
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

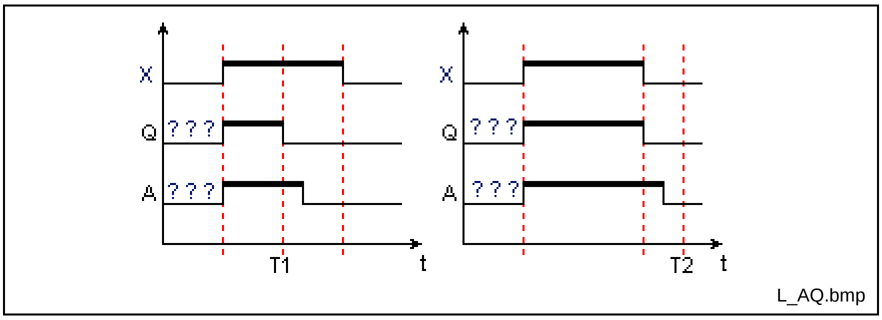
The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is executed and postprocessed according to the action flag.

L

Non-storing, limited in time.

Execution is restricted to the defined duration. It is running as long as ‘Step’ is active and not any longer.



T1: Time < active time of the step
T2: Time > active time of the step
X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-18: Time diagram for action qualifier "L"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

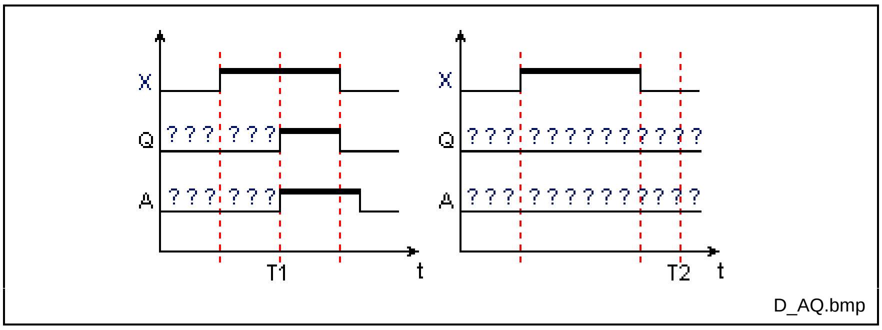
The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is executed and postprocessed according to the action flag.

D

Non-storing, delayed in time.

Execution starts with a time delay and runs as long as the step is active. No processing takes place if the step has become inactive before the time has elapsed.



T1: Time < active time of the step
T2: Time > active time of the step
X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-19: Time diagram for action qualifier "D"

	Boolean variable	**Negated Boolean variable**	**Complex action**
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

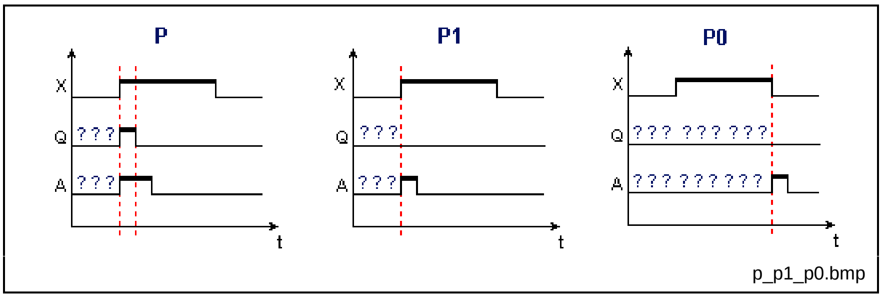
A complex action is executed and postprocessed according to the action flag.

P

Pulse, extended by P0 / P1.

Execution is running only as long as an PLC cycle takes.

- An action with P action qualifier is executed and postprocessed as shown in the following figure.
- An action with P0 or P1 action qualifier, however, is only postprocessed.



X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-20: Time diagrams for action qualifiers "P", "P1" and "P0"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

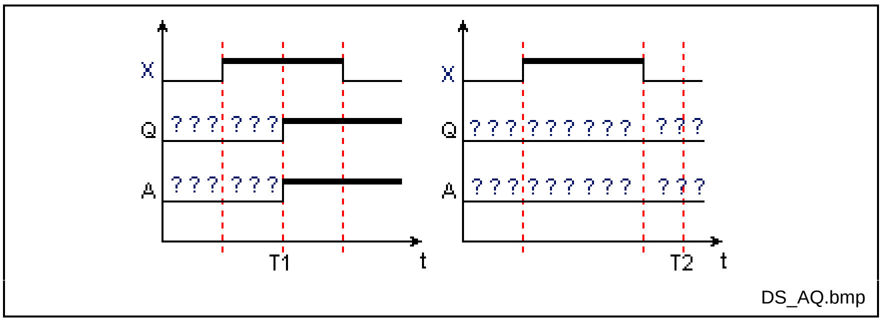
The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is executed and postprocessed according to the action flag.

DS

Delayed and storing.

Start of execution is delayed by the time specified. However, execution must start before the step becomes inactive. Execution must be completed with the reset command.



T1:	Time < active time of the step
T2:	Time > active time of the step
X:	Step is active
Q:	Action flag
A:	Postprocessing flag

Fig. 9-21: Time diagram for action qualifier "DS"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

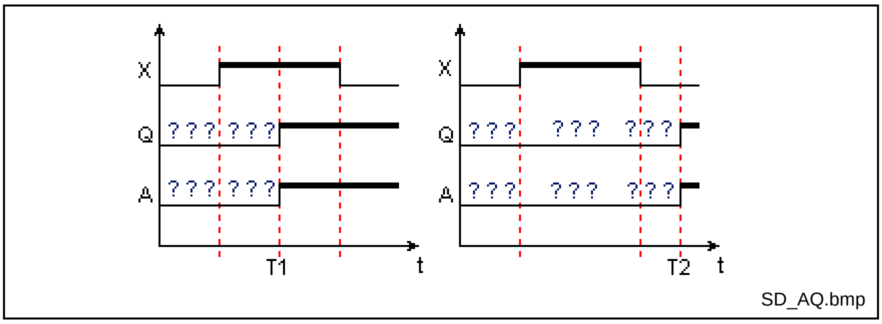
A complex action is executed and postprocessed according to the action flag.

SD

Stored and delayed.

The order for execution is stored. The actual start of execution, however, is delayed by the time specified. Execution must be completed with the reset command.

Note: Execution of the action may begin after the triggering step has become inactive.



T1: Time < active time of the step
T2: Time > active time of the step
X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-22: Time diagram for action qualifier "SD"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is processed and post-processed according to the action flag.

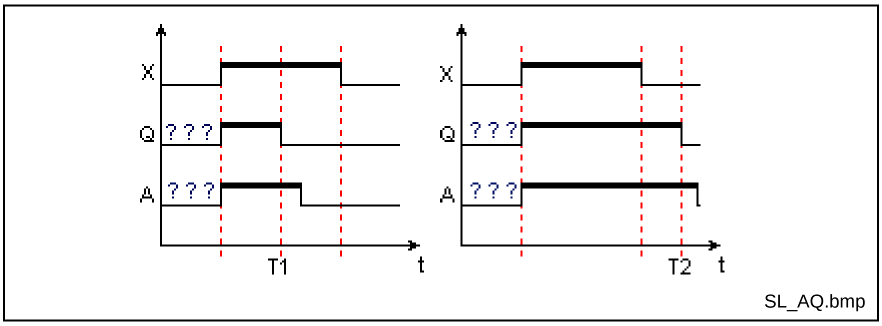
Rexroth Bosch Group

SL

Stored with time limit.

The order for execution is stored. Execution starts immediately and is completed after the time specified.

Note: Before being re-activated, an action, even if completed, must be reset by an R-command (also see Execution by Action_Control).



T1: Time < active time of the step
T2: Time > active time of the step
X: Step is active
Q: Action flag
A: Postprocessing flag

Fig. 9-23: Time diagram for action qualifier "SL"

	Boolean variable	Negated Boolean variable	Complex action
???	It is not possible to make a statement on execution.		
TRUE	TRUE	FALSE	Execution / postprocessing
FALSE	FALSE	TRUE	No execution

The action flag is followed by a Boolean variable. The variable becomes ‘0’ in the same PLC cycle after the step has become inactive.

The action flag is followed by an inverted negated Boolean variable. The variable becomes ‘1’ in the same PLC cycle after the step has become inactive.

A complex action is processed and post-processed according to the action flag.

R

Reset

The action qualifier ’R’ is special in that it becomes effective only if an action is being executed, i.e. included in the list of actions to be executed. The action qualifier ’R’ remains ineffective if it is used by mistake without an action or variable being entered in the list for execution.

The further description assumes that the same action or variable, triggered by an action qualifier "S", "SD", "DS" or "SL" has been transmitted for execution.

Boolean variable

The variable was logic ’1’ and will be logic ’0’.

Negated Boolean variable

The variable was logic ’0’ and will be logic ’1’.

Complex action

The action will be postprocessed.

Note: Action qualifier "SL".

Before being re-activated, an action, even if completed, must be reset by an R-command (also see Execution by Action_Control).

Actions with Function Blocks which Contain SFC Structures

Complex actions can contain instruction lists or LD networks in which function blocks with SFC structures are to be executed. With fulfilled transition, the execution of such a function block allows the start of the SFC within the function block.

The function block and, thus, its SFC is no longer processed if the action is deactivated. They retain their current values, the SFC retains its current state. Execution of the SFC is continued where it was interrupted, if the function block is executed again at a later point.

If, however, the action which includes the function block with SFC structure is called up with the action qualifier "R", execution is not continued. The SFC structure and the function block with SFC structures called up by this structure are reset.

9.3 Setup Support on Action Block Level

Forcing of Actions with System Support

Forcing of actions is only possible in the manual mode and only for actions which are called up by at least one step. Boolean and negated Boolean actions are not supported.

Actions which are not used by the SFC (see Actions in the SFC List) do not reside in the control after having been downloaded and can, thus, not be forced.

Forcing of actions means that execution of an action can be initialized independently of output Q of its Action_Control component and, thus, independently of the condition of the steps of the POU.

Interplay of the system variables for actions when forcing an action with xxx.F

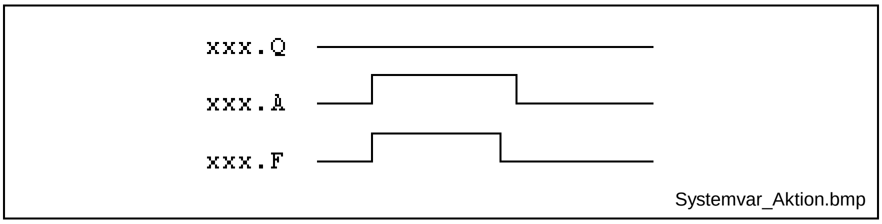


Fig. 9-24: System variables when action xxx is forced

The following table contains all permitted assignments of system variables of action "xxx":

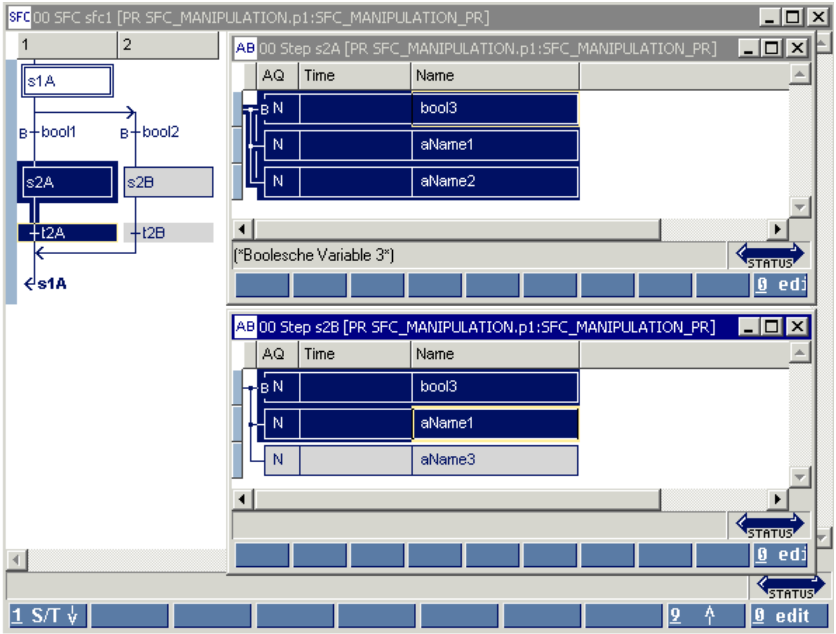
xxx.Q	xxx.A	xxx.F	Comment
FALSE	FALSE	FALSE	No action execution
FALSE	TRUE	FALSE	Action postprocessing
FALSE	TRUE	TRUE	Action execution
TRUE	TRUE	FALSE	Action execution
TRUE	TRUE	TRUE	Action execution

Fig. 9-25: Assignment of system variables of action xxx

By evaluating these flags, each named action can realize a postprocessing identification.

Steps for initializing the forcing of "sfc1"

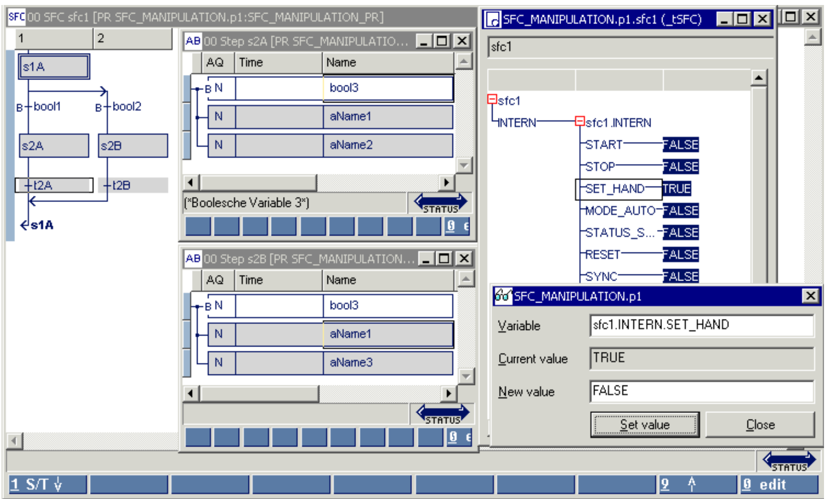
- Starting point is the status of "sfc1" shown in the figure below.



AB Betriebsart_00.bmp

Fig. 9-26: "**sfc1**" with active step "**s2A**" and action blocks

- Switch over from automatic mode to manual mode. Each SFC has a number of variables which are created by typifying with "1-Basis" or "3-Indra-Step" in the "View / SFCs" menu item. The variable "**sfc1.intern.SET_HAND**" must be set to TRUE. This is done using the "Start / Status ARRAYS / Structures" menu item and selecting the structure "**sfc1**", to the right in the figure below.

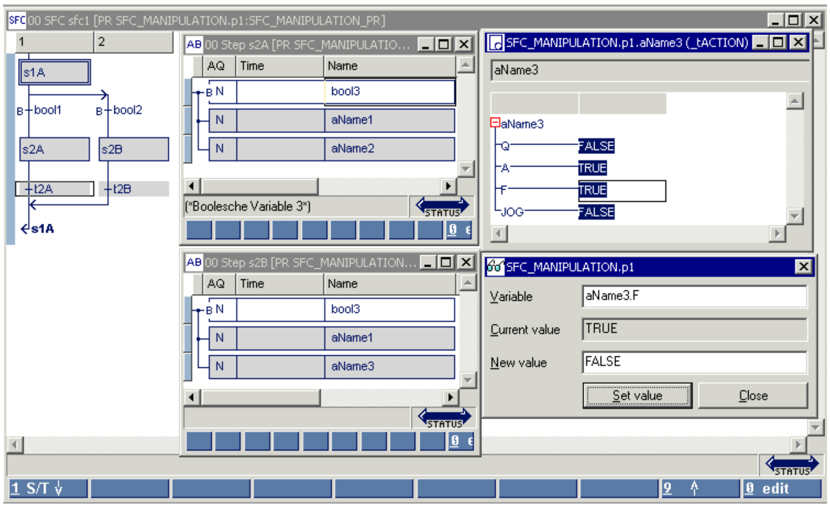


AB Betriebsart_01.bmp

Fig. 9-27: Change over to manual mode for "**sfc1**"

- Position the cursor on the variable "**SET_HAND**", click the right mouse button or press <Shift>+<F10> and replace the value FALSE (automatic mode) by TRUE (manual mode). It is characteristic of the manual mode that all steps are deactivated. Switchover from AUTO to MANUAL causes a reset of all actions referenced by the SFC.

- The action "**aName3**" is to be activated. To achieve this, call up the "Start / Status ARRAYS / Structures" menu item and select the structure pertaining to the action "**aName3**".



AB Betriebsart_02.bmp

Fig. 9-28: Action "**aName3**" is forced

Note: The value of the variable aName3.F (force flag) is preserved during the next phase of the automatic mode. The action is forced again if the control returns to the manual mode.

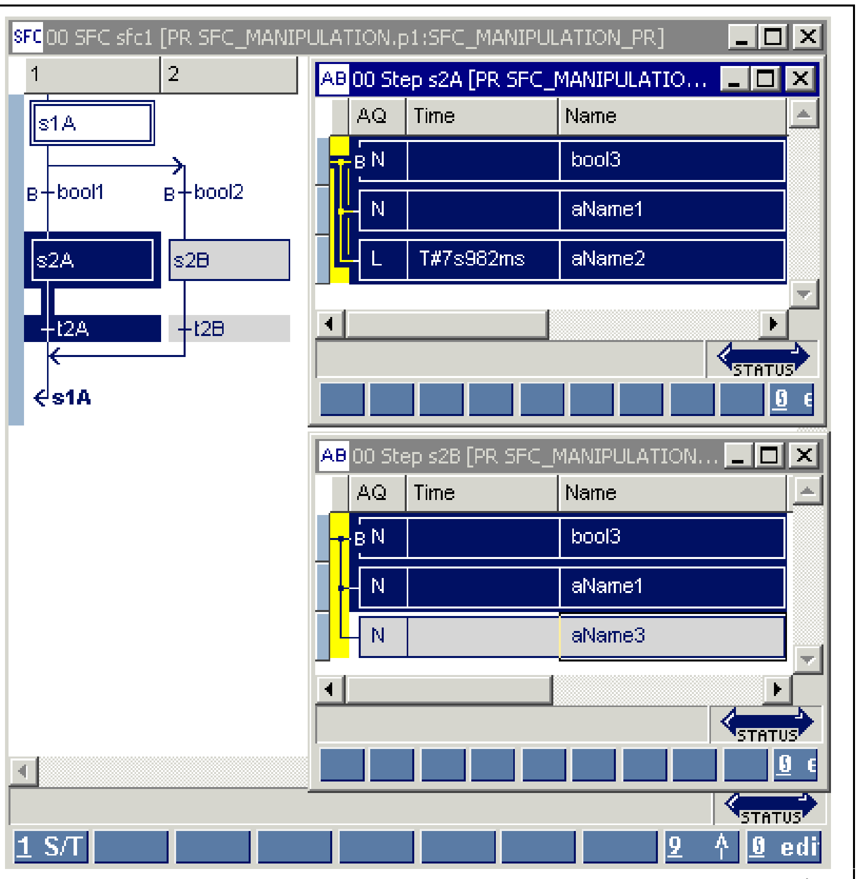
- To return to the automatic mode, the variable "**sfc1.intern.SET_HAND**" must be set to FALSE again. This is done using the "Start / ARRAYS / Structures" menu item and selecting the structure "**sfc1**".

Note: If the active steps were not changed those steps which were active before the manual mode become active again .

Status Display in the Action Block Editor

If a program resides in the control system for execution, the following status information can be displayed in the action block editor:

- Blue power rail in the left margin, position in yellow, step is active
- Colored bit variables are logic "1"
- Colored negated bit variables are logic "0"
- If colored, more complex actions are postprocessed
- For time-forced actions, the currently running time is displayed above the entered time



AB status.bmp

Top window: SFC level, step s2A active, step 2B inactive
Central window: Actions are activated, time is running
Bottom window: No activation, actions are activated by external steps

Fig. 9-29: Status display in the action block editor

Further ways to obtain status information are:

- Start / Force <Shift>+<F8> for elementary variables (ANY_ELEMENTARY)
- Start / Status ARRAYs / Structures <Shift>+<F3>

9.4 Options, Action Block Editor

The options relevant for the action block editor can be selected by means of the "Extras / Options" menu item:

<table>
<tr><th>Group</th><th>Option</th><th colspan="2">Meaning</th></tr>
<tr><td rowspan="5">Desktop</td><td>Restore size and position during startup</td><td colspan="2">The desktop is restored in the same size and position.</td></tr>
<tr><td>Restore MDI window during startup</td><td colspan="2">MDI windows are opened in the same order when restarting the system.</td></tr>
<tr><td>Create backup copy</td><td colspan="2">Automatic storage of the source condition which was loaded into the control in "downloaded files".</td></tr>
<tr><td>Auto save</td><td colspan="2">Allows the automatic saving of the current file in presettable time intervals without any prompt.</td></tr>
<tr><td>Sound</td><td colspan="2">Activation or deactivation of a beep sound.</td></tr>
<tr><td rowspan="5">View / All</td><td>Apply declaration comment in implementation</td><td colspan="2">Comments, that have been entered in the respective declaration line are displayed in the implementation. The implementation can be changed; the comment is then doubled, the declaration line remains unaffected.</td></tr>
<tr><td>Variable display</td><td colspan="2">With symbols (name) or absolute (address).</td></tr>
<tr><td>Display of absolute variables</td><td colspan="2">The user can select from I/Q, E/A and I/O for absolute addresses.</td></tr>
<tr><td>Truncating very long texts</td><td colspan="2" rowspan="2">Texts and numbers can be truncated to the right or left, and

can be represented with or without "..." marking.</td></tr>
<tr><td>Truncating very long numbers</td></tr>
<tr><td rowspan="5">View / AB</td><td rowspan="5">Column width for the individual columns

(with standard values)</td><td>First</td><td>10</td></tr>
<tr><td>AQ</td><td>30</td></tr>
<tr><td>Time</td><td>80</td></tr>
<tr><td>Name</td><td>120</td></tr>
<tr><td>Last</td><td>250</td></tr>
</table>

Fig. 9-30: Action block editor options

9.5 Pop-up Menu, Action Block Editor <Shift>+<F10>

This pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items	Explanation
Open	Branch to an action, also <Ctrl>+<Enter>
Edit comment	Input of a comment on an action.
Delete	Deletion of the current action block.
Import implementation	The ASCII file chosen from the "WinPCL text files" is loaded into the action block editor.
Export action	The current action block is exported as an ASCII file and stored in the folder "WinPCL text files".
Export AB of the step	The current action blocks of the step are exported as an ASCII file and stored in the folder "WinPCL text files".
Syntax text	List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Error help	The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Declaration help	Description of the properties of the current action block.
Cross reference help	List of all places of use. The place of use can be reached by double-clicking the mouse or pressing the <Ctrl>+<Enter> keys.
Force	Allows the entry of a variable name. The value of the variables is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of the program code execution.
Status ARRAYs / Structures	Status display for the elements of an array or a structure. Selection is done through a tree structure till the specific element is reached.
Print current window	Prints the editor contents.
Options	Toggles the absolute address and the name of the variable.
Internals	Search for faults in the programming system, to be used only if approved by the service.

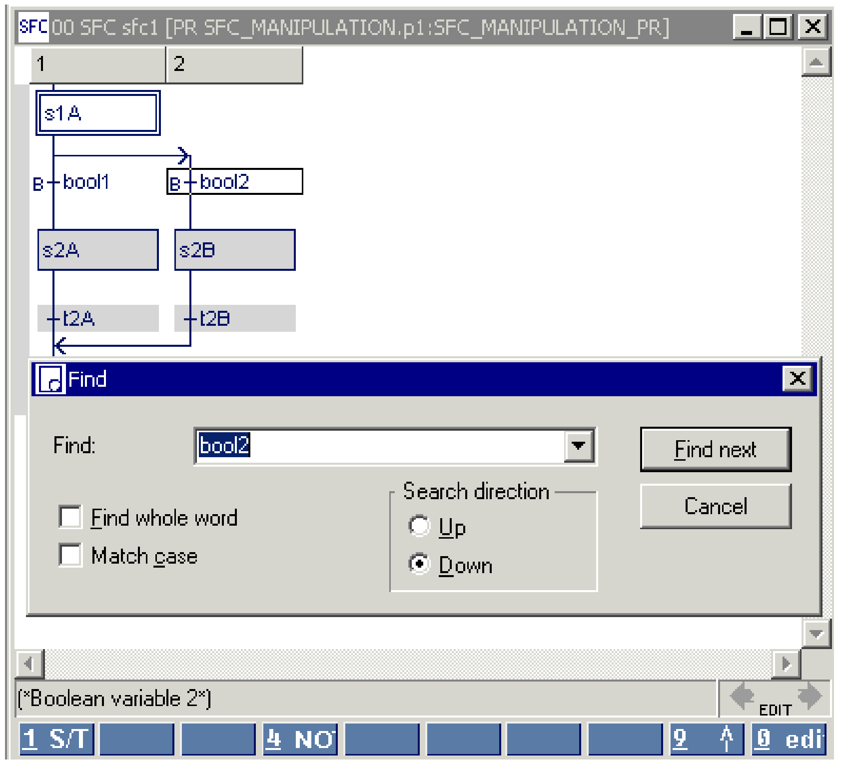
Fig. 9-31: Pop-up menu of the action block editor

9.6 Block Commands, Action Blocks

Not implemented yet.

9.7 Search and Replace, Action Block Editor

The search function is in the first version and provides the features of a text editor:



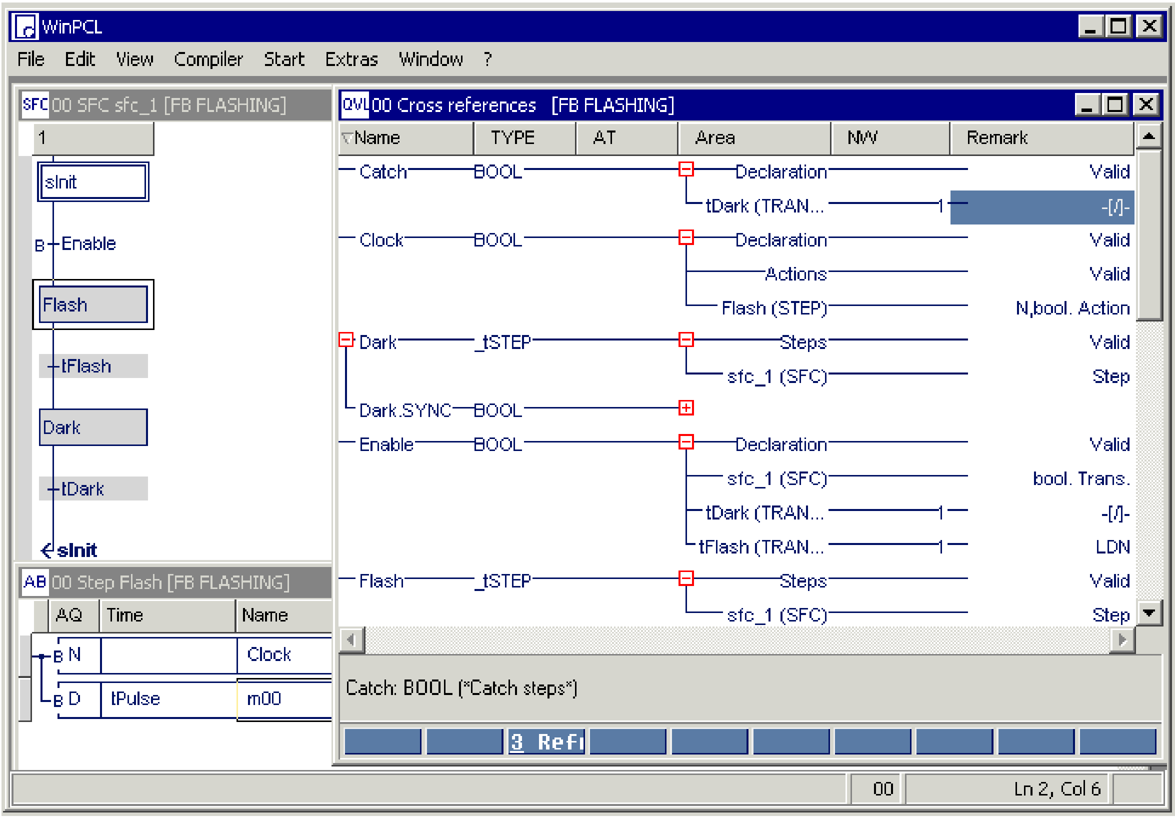
AB_Suchen.bmp

Fig. 9-32: Search function in the action block editor

The replace function is in preparation.

9.8 Cross Reference List, Action Block Editor

In contrast to the cross references of the pop-up menu, the overview obtained by means of "View / Cross reference list" shows all variables. Of course, only variables from lines with the correct syntax can be resolved by their place of use. All faulty names or names with double declaration are displayed and can, thus, be reached by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.



ab_qvl.bmp

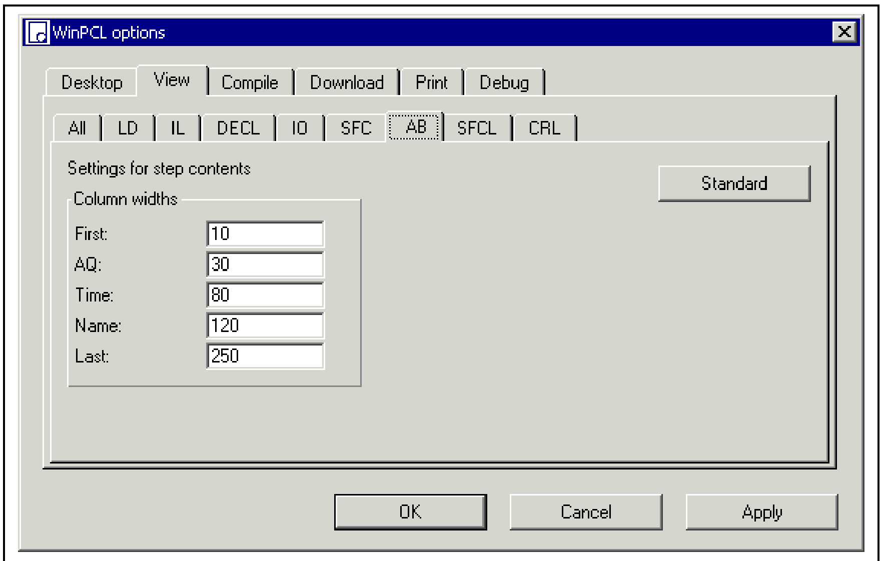
Fig. 9-33: Cross reference list of a function block with SFC element (excerpt)

Name	Type	Area	Use	Comment
Flash	tSTEP	Steps	Valid	Step list
		SFC sfc_1	Step	Step in SFC sfc_1
Enable	BOOL	Declaration	Valid	Declaration of the Boolean variables
		SFC sfc_1		Boolean transition in SFC sfc_1
		TRANSITION continue	Normally open contact in network 1	Ladder diagram network
		TRANSITION continue	Negated reading of network 1	IL network
Clock	BOOL	Declaration	Valid	Declaration of the Boolean variables
		Actions	Valid	Action list
		STEP Flash	Boolean action, non-storing (N)	Action block in step Flash
Count	tACTION	Actions	Valid	Action list
		STEP Flash	Action, once with step activation (P1)	Action block in step Flash

Fig. 9-34: Comment on cross reference list (shortened)

9.9 Documentation, Action Block Editor

The SFCs are documented (print from the editor, <Ctrl>+<P>) using the column width defined under Extras / Options / View / AB.



optionen_ansicht_ab.bmp

Fig. 9-35: Action block (AB) options

The "Apply" button activates the column width set for the SFC editor. The width of the column can either be entered in the window shown above or preset in the editor by dragging the headers.

The "Standard" button resets the default.

The "OK" button applies the setting and closes the dialog window.

The "Cancel" button closes the window; the previous values are kept.

Detailed information on the real print process and the features is to be found in the main chapter on WinPCL.

10 I/O Editor

10.1 General Notes on the I/O Editor

The task of the connections to programmable logical controllers is the transfer of information to the PLC for processing.

The connections do not only differ by their physical facts, but also by transmission protocols, smallest transferable data volume and other facts. By this way the whole efficiency of an interface connection can only be realized with the configurator that matches this connection, e.g.:

- INTERBUS: IBS CMD G4
- PROFIBUS: FIOCON

The programming system has solely the responsibility and possibility to allow an access to the information that comes from this interface connection via a page frame.

An example for the consequent work

IBS CMD G4 -> I/O editor -> Declaration resource

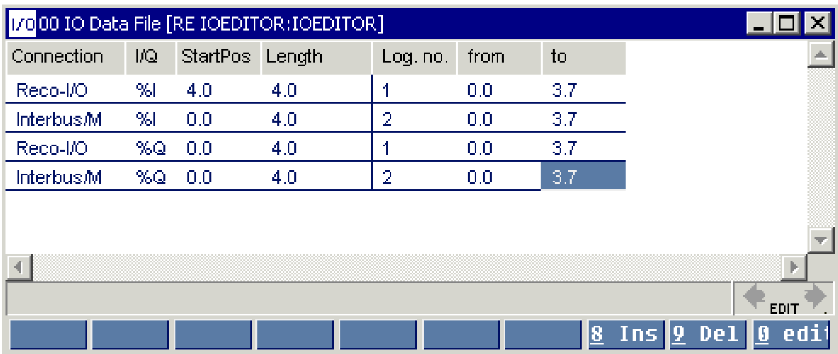
is to be found in the chapter "Function blocks / Firmware function blocks / Preparation for control of an INTERBUS".

Note: New in version 4VRS: generating and condensing gaps for inserting and removing bus units (to be opened using the Pop-up Menu, I/O Editor <Shift>+<F10>).

10.2 Structure of an I/O Editor

I/O Table

The basic element of the I/O editor is shown in the figure below:



IO_Edit_01.bmp

Interface connection 1: INTERBUS
Interface connection 2: RECO I/O

Fig. 10-1: Display table of the I/O editor

Interface connection

The information between interface connection and control is transmitted by means of a memory area which is divided in two, the area of the inputs and the outputs (see left side of the table in the figure above).

It is further necessary to indicate the start position of the information and its length.

1. 1st line:

Connection: INTERBUS, 4-byte input module

Area of inputs (%I),

Start position 0.0 (byte 0, bit 0)

Length 4.0 (4 bytes, 0 bits)

2. 2nd line:

Connection: INTERBUS, 4-byte output module

Area of inputs (%Q),

Start position 0.0 (byte 0, bit 0)

Length 4.0 (4 bytes, 0 bits)

3. 3rd line:

Connection: RECO-I/O, 4-byte input module

Area of inputs (%I),

Start position 4.0 (byte 4, bit 0)

Length 4.0 (4 bytes, 0 bits)

Note: Depending on the interface connection, the start position contains the input user information starting with the forth byte, since the bytes 0-3 contain diagnosis information.

4. 4th line:

Connection: RECO-I/O, 4-byte output module

Area of outputs (%Q),

Start position 0.0 (byte 0, bit 0)

Length 4.0 (4 bytes, 0 bits)

The subdivision into bytes and bit allows the usage of bit modules, in packed form as well as starting at the byte limit and leaving the remaining part free. If bit modules in packed form are used, the start position then is on any bit number, the next bit module follows in the memory immediately.

Note: The user is responsible for the correct entry on the left side. There is no check that the interface connection is really available!

Assignment of the logical addresses

Any number between 1 and 999 can be set for **logical numbers**. In other words, the user can establish the groups for his interface connections or the like by himself (see right side of the table in figure 8-1).

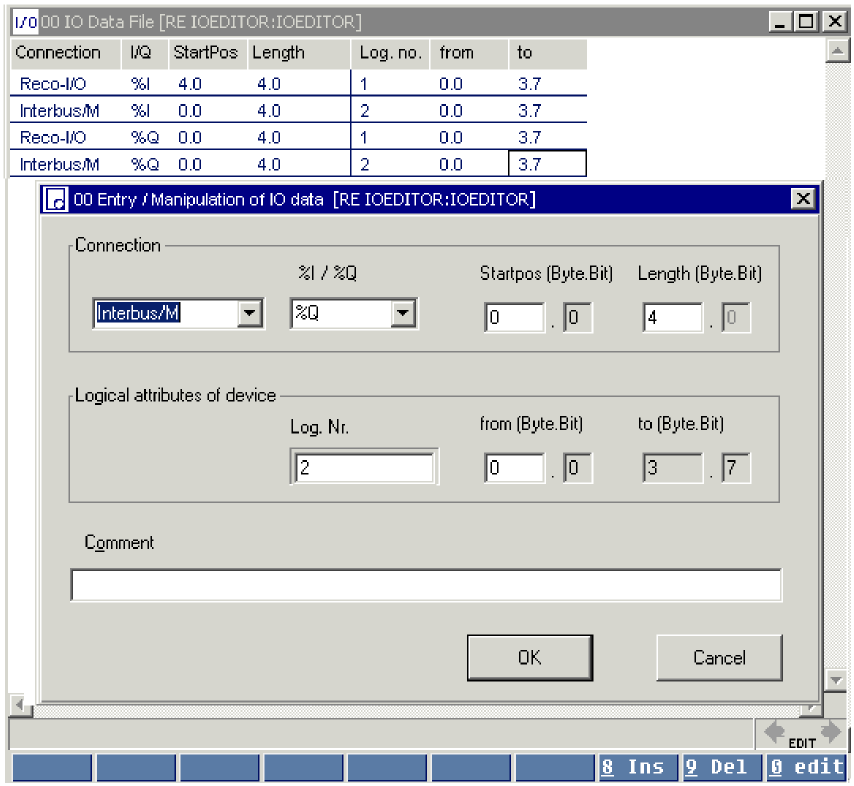
Furthermore the user can also define any start address "**from**" byte (.bit), normally "0.0". In this manner, the user can have the user inputs start with 0.0, as shown in line 3 of the figure above, although they start with byte 4 in the memory of the interface connection.

The end address "**to**" is calculated automatically.

Structure of the Input Mask in the I/O Editor

If a new line is to be entered in the I/O table or an existing line is to be changed, the input mask for changing the current line or for entering a new line is activated either with the footer command 8 Ins (<Ins> key) or with 0-edit.

When you fill in the mask, the input is checked for being complete. Overlapping numbers or coinciding access to storage locations of the interface connections are detected and denied.



IO_edit_Eingabemaske.bmp

Fig. 10-2: Input mask of the I/O editor

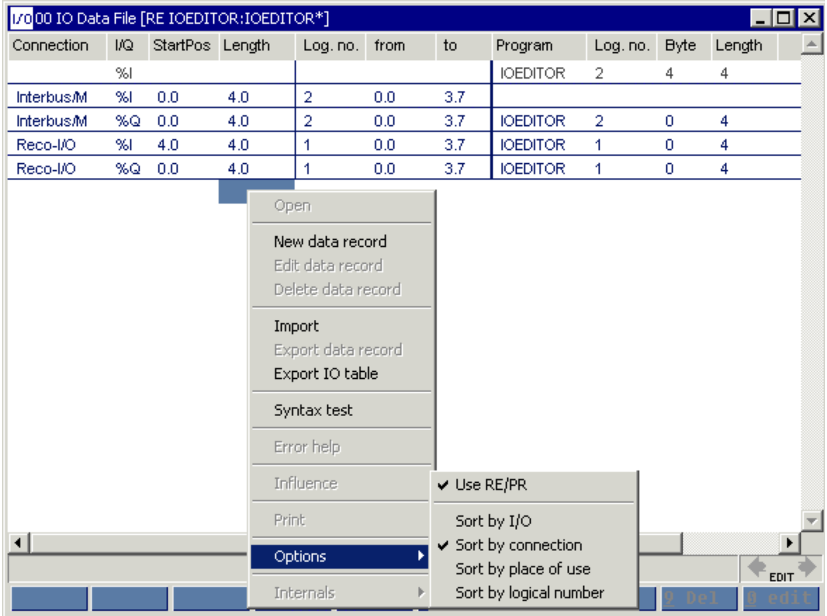
Rexroth
Bosch Group

Below follows an overview of possible interface connections (The control type set in the system configurator determines the connections respectively approved for selection).

Interface connection	%I / %Q	Comment
M keys	%I	Lateral keys on BTV20, BTV15 screens
PLC keys	% and %Q	Backlit keys on the TBV20, incl. key switch
BT bus or INTERBUS/M	%I and %Q	Connection of up to four operator terminals, BRA20, BTM15, etc.
MTC200	%I and %Q	Interface to processes and axes of the CNC control, as seen from the PLC
INTERBUS/M or PROFIBUS DP/M or DeviceNet/M	%I and %Q	• I/O modules in switch cabinet design, e.g. RECO Inline modules • I/Os in the peripheral equipment, e.g. SM modules in IP65 design
RECO-I/O	%I and %Q	Direct I/O modules on RECO controls (mostly not in connection with INTERBUS or PROFIBUS field bus)
INTERBUS/M or PROFIBUS DP/S	%I and %Q	• PLC interface to other MTC200, ISP200 units in the machine (transfer concept), e.g. via RMG gateway modules • PLC interface to third-party PLCs, e.g. via RMG gateway modules

Check for Use of the I/O Areas in Resource and Programs

Press the <Shift>+<F10> keys or click the right mouse button to activate the pop-up menu shown below. A third part of the table can be faded in with the "Options / Use RE / PR" menu item.



IO Verwendung RePr.bmp

Fig. 10-3: Check of I/O use

It is checked here whether the I/Os required in the resource or in the programs match the declared I/Os.

- Line 1: The offered inputs are not required.
- Line 2: Is highlighted in gray, the following is required for the resource RES_IO_TEST, 4-byte inputs, from %I2.4

This is obviously an incorrect declaration.

Either go to line 1, press the 0-edit button, keep the log. no. and change the start address "**from**" from 0.0 to 4.0.

- or -

go into the declaration of RES_IO_TEST and change the addresses of the corresponding variables.

Note: An I/O table should not contain gray-highlighted lines, as these variables are not supplied by the interface connection. A flag storage location is assigned instead!

Logic Address Assignment by Example of a BT Bus

General notes on the BT bus

The BT bus can be used to connect up to four operator terminals of type BTM15/16 or BTA20.

The address assignments required for programming can be found in the documents accompanying of the devices to be connected.

Device type	Storage assignment in the input / output core image
BTM15	Depending on the configuration: 2 bytes for digital I/Os (always assigned) 2 additional bytes for each module (except handwheel) 4 additional bytes for handwheel module
BTM16	14 bytes
BTM20	6 bytes

Fig. 10-4: Storage requirements of operating devices

Addressing

The BT bus is addressed by assigning a logic user number in the I/O editor of the PLC programming interface. A separate logic address can be assigned to each input core image storage as well as to each output core image storage, but it is also possible to use the same address.

Each of the two core image storages has a size of 128 bytes, which are available for the connected operating devices. The number of bytes assigned in the core image storage depends on the operating device.

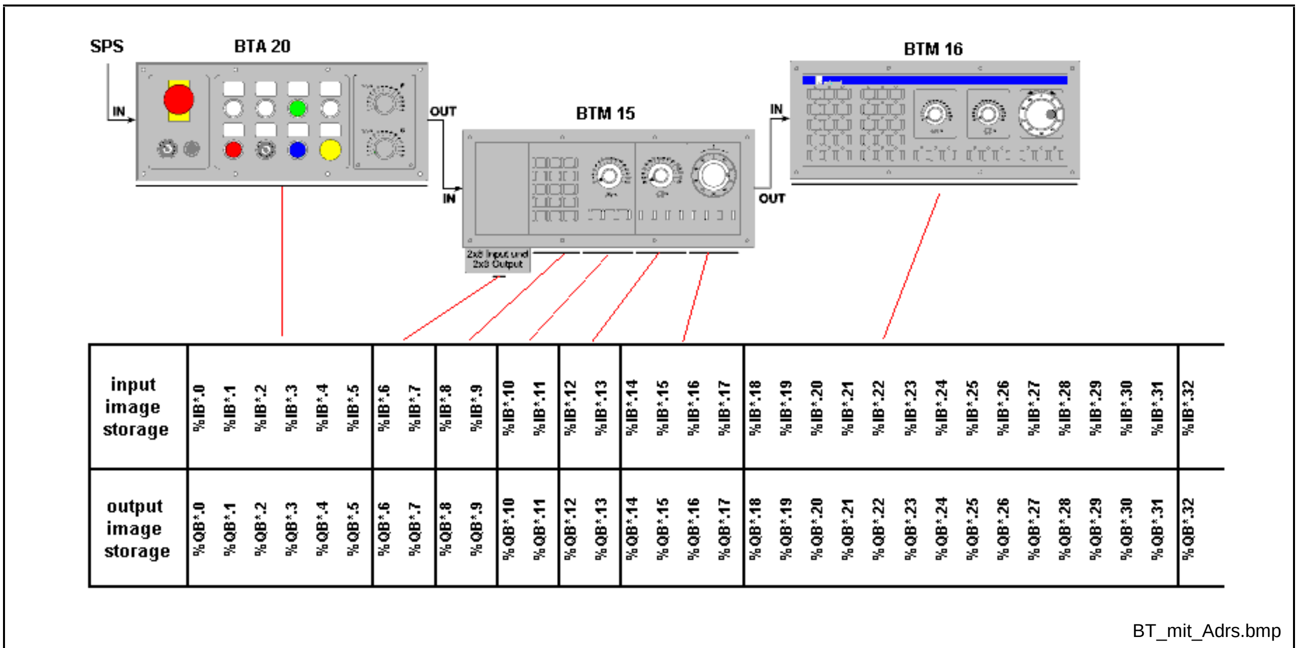
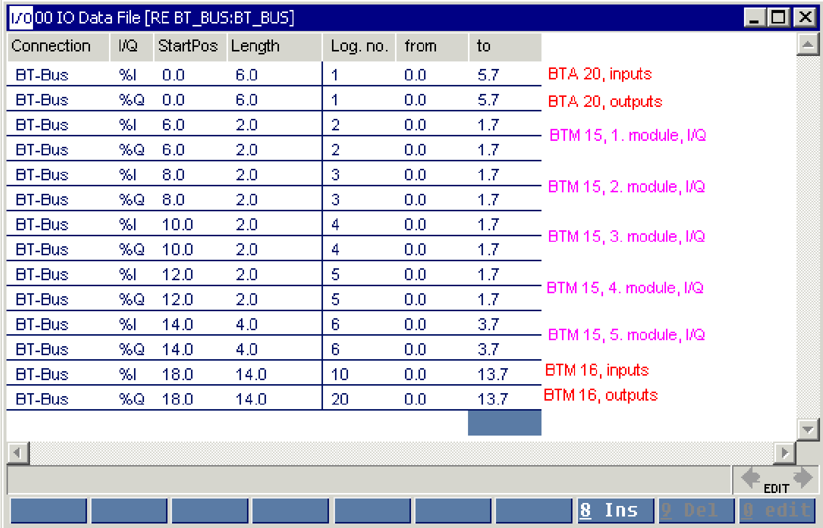


Fig. 10-5: Devices in BT bus and addresses

I/O table in the I/O editor

Addressing can be defined by the user himself, depending on the requirement.



Connection	I/Q	StartPos	Length	Log. no.	from	to
BT-Bus	%I	0.0	6.0	1	0.0	5.7
BT-Bus	%Q	0.0	6.0	1	0.0	5.7
BT-Bus	%I	6.0	2.0	2	0.0	1.7
BT-Bus	%Q	6.0	2.0	2	0.0	1.7
BT-Bus	%I	8.0	2.0	3	0.0	1.7
BT-Bus	%Q	8.0	2.0	3	0.0	1.7
BT-Bus	%I	10.0	2.0	4	0.0	1.7
BT-Bus	%Q	10.0	2.0	4	0.0	1.7
BT-Bus	%I	12.0	2.0	5	0.0	1.7
BT-Bus	%Q	12.0	2.0	5	0.0	1.7
BT-Bus	%I	14.0	4.0	6	0.0	3.7
BT-Bus	%Q	14.0	4.0	6	0.0	3.7
BT-Bus	%I	18.0	14.0	10	0.0	13.7
BT-Bus	%Q	18.0	14.0	20	0.0	13.7

BT_IO_Ed.bmp

Fig. 10-6: Addresses in the I/O editor

A common logic number was assigned for the BTA20. The inputs and outputs of this device were concentrated in one memory area.

For the BTM15, a logic address was assigned to each module, with this address being assigned jointly to the inputs and outputs of the module.

For the BTM16, the inputs and outputs of the complete device were grouped, but they were provided with different logical numbers.

10.3 Special Functions of the I/O Editor

Shifting I/O Addresses

Requirements for Shifting I/O Addresses

By defining the logic I/O devices, the directly represented variables (%I…, %Q…) used in the user programs (resource and included programs) are assigned to the addresses in the memory of the particular connection (DPR).

Checks are carried out in the input dialog, to avoid overlapping, especially of DPR addresses.

If, for example, physical addressing is used in the INTERBUS - i.e. the memory address of a device is automatically generated from the latter's position in the bus -, the addresses of all following devices are shifted when a device is inserted. Up to now, it was very difficult to adjust this shifting of addresses in the I/O editor. Starting from the end (because multiple assignment of addresses is not permitted by the check described above), the addresses of all devices must be changed manually.

Note: If logic addressing is used in the INTERBUS, this functionality is not required because the addresses of the inserted inputs and outputs are defined by the user.

The functionality required is intended to support insertion and removal of input and output areas in the DPR by the program, by shifting all following addresses (of the same data direction - %I… / %Q…) accordingly. This is achieved by the following operations:

- when a device is inserted or deleted,
- by a special call using the context-sensitive menu of the I/O editor.

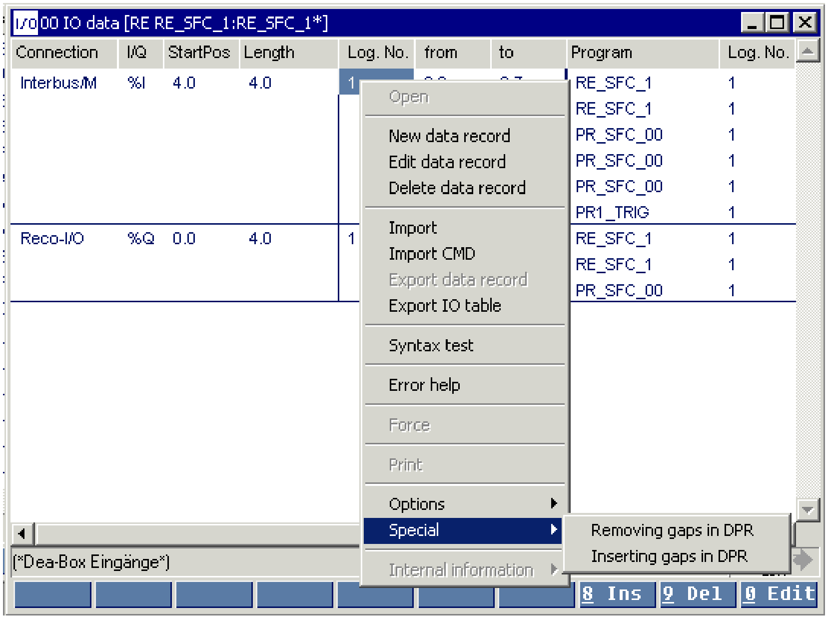
Inserting / Deleting a Device

If a device is inserted at an address which is already occupied by an already entered device, the user is requested whether he wishes to insert the area with the data capacity specified and to shift all following addresses of the same data direction. Analogously, the user when deleting a device (or when altering the device data) is requested whether he wishes to condense the now empty area again.

Since this functionality is required for specific I/O connections only, they can be used to a very limited degree only. To this end, all connection names (according to the names in the IO_Dev.ini file), for which this functionality is to be used (the INTERBUS connections are already entered as a default), must be entered in the options (Plc.ini).

At present, this setting cannot be altered within the WinPCL GUI.

When "Specials / Removing gaps in DPR" or "Specials / Inserting gaps in DPR" is selected in the pop-up menu of the I/O editor, a dialog opens which can be used to insert or remove separate areas within the already defined I/O addresses.



IO_Ed_spezial_0.bmp

Fig. 10-7: Pop-up menu call in the I/O editor

Removing gaps

The following input dialog permits removing of existing gaps in the DPR:

IO_Ed_spezial_1.bmp

Fig. 10-8: Removing gaps

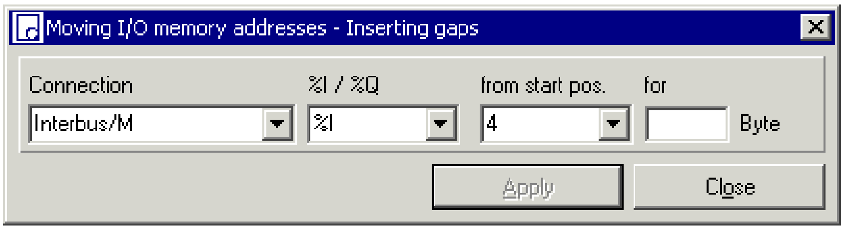
Position	Comment
Connection	Combobox which can be used to select a connection from the connections entered in the I/O editor (this selection is independent of the options setting described above).
%I / %Q	Data direction (input data / output data)
From start pos.	Address of the gap in the DPR to be condensed or removed (the starting addresses of all existing gaps are provided for selection)
For	Size of the area to be removed
Apply	The area selected is removed. Before the removal, the system checks whether the area concerned does not exceed the gap to be removed. The window is not closed; the dialog contents are updated.
Close	The dialog window is exited.

Fig. 10-9: Steps performed when gaps are removed

The "Apply" button is passive if no gap has been selected (or if there is no gap in the selected data direction of the particular connection) or if the value specified for the size of the area fails to be reasonable. The text field for the size of the area to be deleted displays the range permitted for this value.

Inserting gaps

The following input dialog is provided for inserting gaps in the DPR:



IO_Ed_spezial_2.bmp

Fig. 10-10: Inserting gaps

Position	**Comment**
Connection	Combobox which can be used to select a connection from the connections entered in the I/O editor (this selection is independent of the options setting described above).
%I / %Q	Data direction (input data / output data)
From start pos.	Address in the DPR where the gap is to be inserted (the starting addresses of all existing devices are provided for selection)
For	Size of the area to be inserted
Apply	The area selected is inserted. The window is not closed; the dialog contents are updated.
Close	The dialog window is exited.

Fig. 10-11: Steps performed when gaps are inserted

The "Apply" button is passive, if the value specified for the size of the area fails to be reasonable.

Applying Configuration Data from CMD to the I/O Editor

As seen from WinPCL, the I/O peripherals are configured in two stages:

- Parameterization and configuration of the bus by means of an appropriate configurator.
- Definition of the logic devices in the WinPCL I/O editor, for decoupling the physical I/O devices from the absolute identifiers (%I… / %Q…) used in the program.

The advantage of this decoupling is that, if the I/O peripherals are modified / extended, they must, in general, only be adjusted within the WinPCL I/O editor and not in all declaration parts of the programs. However, the disadvantage is that all logic I/O users must be entered manually and that, for the desired I/O connection, an appropriate address must be assigned to them.

This disadvantage is eliminated by the present tool, which can be used to read the data of the external configuration tool into WinPCL and to define the logic I/O devices. As a result, manual entries are reduced to a high degree.

Our specific case involves the configuration of the INTERBUS using the "CMD" configuration tool by Phoenix Contact.

CMD Export Requirements for IBS Configuration Data

1. Within CMD, one of the functions permits export of the configuration data in text format ➔ Parameterization memory - Pop-up menu \ Write ASCII File \ Project Data (*.csv). The projecting data are written as a text file in the form of a table, the data elements of which are separated by a semicolon.

2. The pop-up menus of the WinPCL I/O editor provide an import function which can read and interpret the CSV file of the CMD tool.

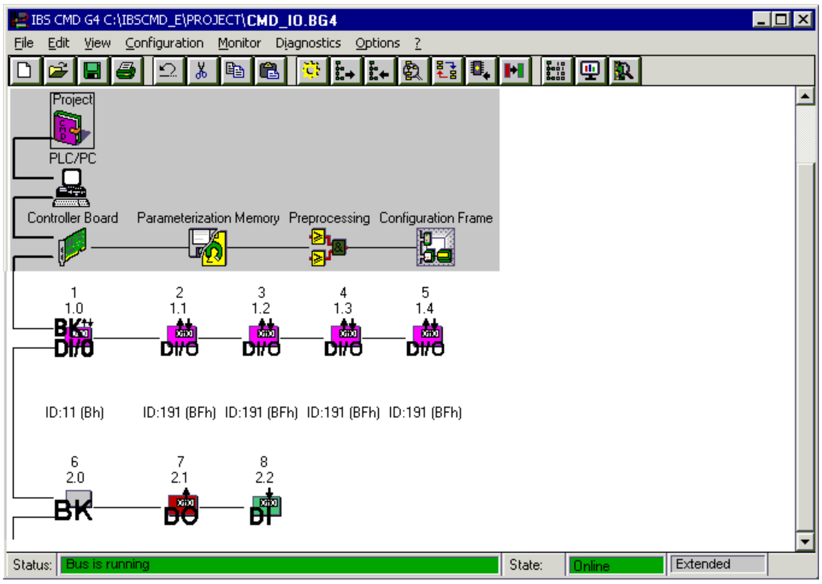
3. The following methods for assigning the physical device data to the logic devices are available:
   - **Segment-oriented assignment:** Each segment (remote bus terminal + local bus device) is comprised to a logic device. The logic device number corresponds to the segment number. Since each local bus device has the same segment number as the pertaining bus terminal, the assignment is unique.
   - **Device-oriented assignment:** Each device is accepted as a separate logic device in the I/O database. The logic device number consists of the segment number and the position within the segment (segment number * 100 + position).
     It is not reasonable to assign the logic device numbers completely according to the arrangement of the devices in the INTERBUS, because there is no fixed relation between the logic devices and the physical I/O devices. When devices are added subsequently, this would lead to an amount of modifications which is not justifiable.

4. The combination formed by the entries of "Station name" / "Device name" is used as comment in case of the **device-oriented assignment.**

5. If the **segment-oriented assignment** is concerned, only the device comment of the bus terminal can be taken into consideration.

Activities in CMD

After CMD has been started, CMD reads the bus structure from the control (cursor on the configuration frame, reading from the memory). Thereafter, the project should be saved.

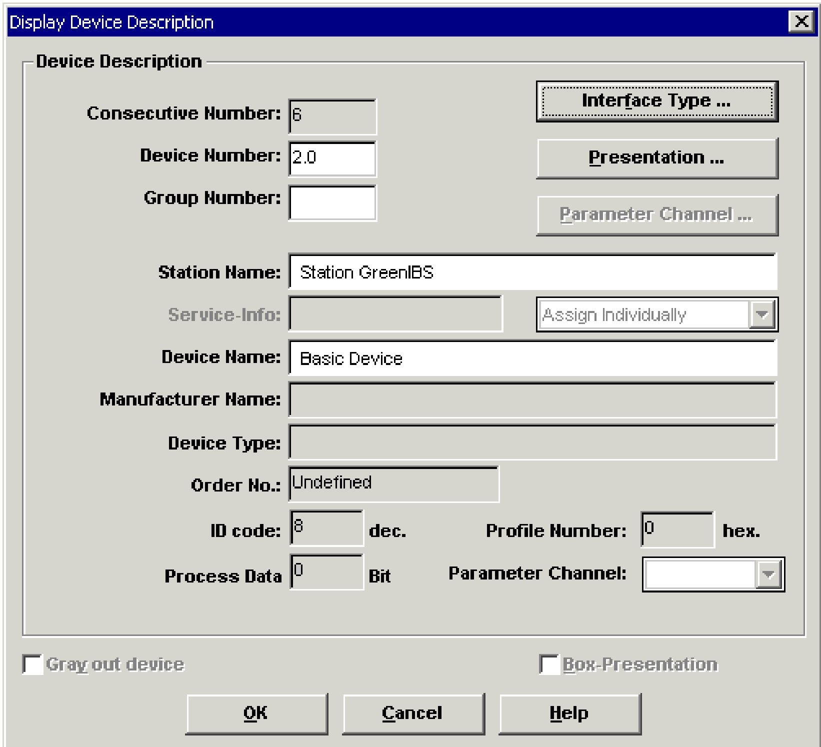


CMD_IO_ruecklesen_01.bmp

Fig. 10-12: Reading the bus structure from the control

It is now possible to write to the various devices.

To achieve this, the cursor must be positioned on the icon of the particular device and the write window must be opened (pop-up menu, Description <F9>).



Teilnehmerbeschr_02.bmp

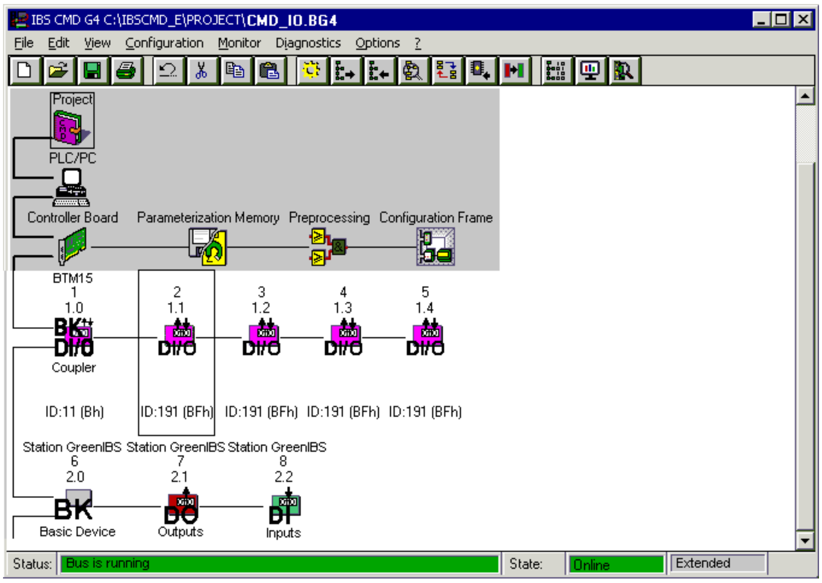
Fig. 10-13: Writing to the device (here lower branch of bus terminal)

The window offers 2.0 as the device number.

The station name is assigned for the entire line (bus branch).

The device name is assigned to the device itself.

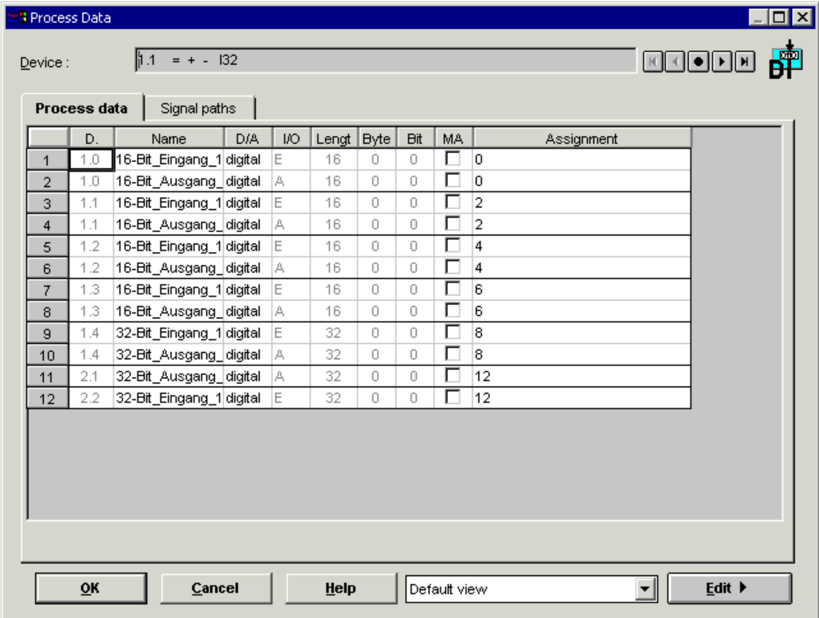
The following bus structure is achieved after updating.



Teilnehmer_mit_Beschr_03.bmp

Fig. 10-14: Bus structure after 4 devices have been written to.

The "Controller Board" icon can be used to open the process data display (via pop-up menu).

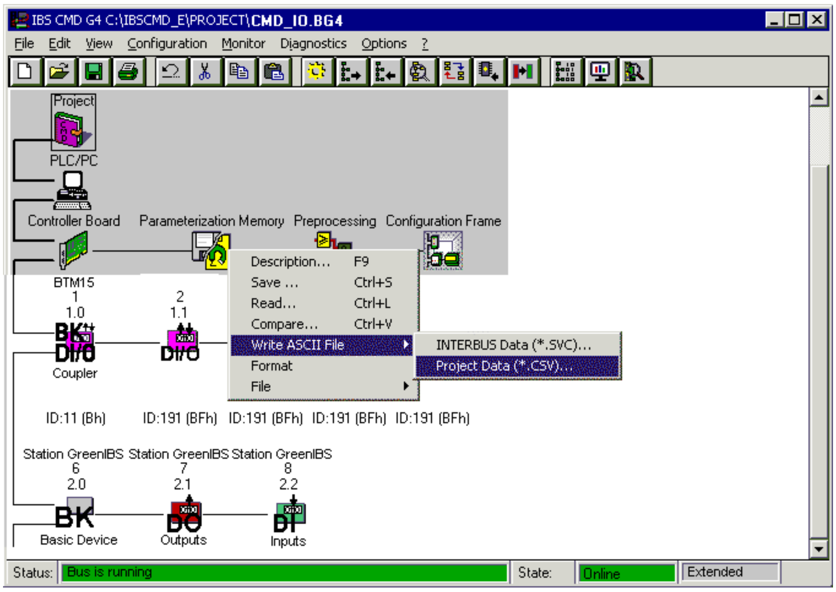


Prozessdaten_04.bmp

Fig. 10-15: Process data display

This display shows the device numbers, the names and the type of the connections (inputs = E / outputs = A).

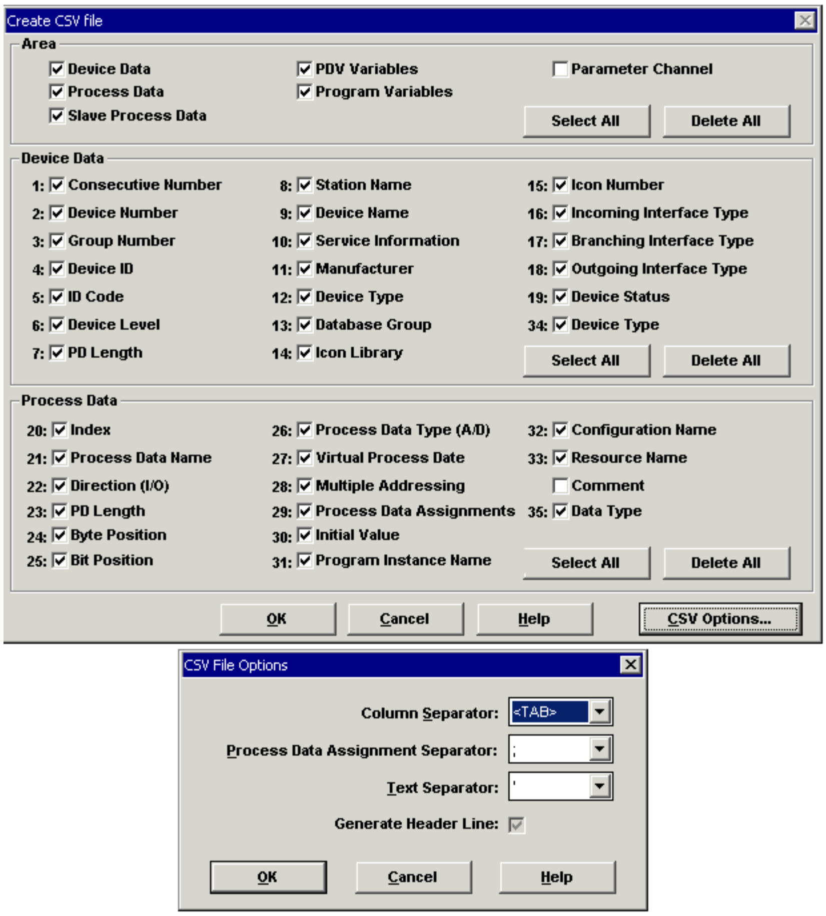
The project data are written as CSV file in the context-sensitive menu of the parameterization memory (pop-up menu).



Ausschreiben_csv_05.bmp

Fig. 10-16: Writing the CSV file

The settings of the data contents of the CSV file and, in particular, the options of the CSV file may not be changed. If some of the data are omitted, it might be that the data cannot be applied.



Einstellung der CSV-Datei.bmp

Fig. 10-17: Setting of CSV files

CSV File, Data Format

The format of the CSV file is represented as Excel import (reduced) in the figure below. Its evaluation is required only if there are difficulties in applying the data.

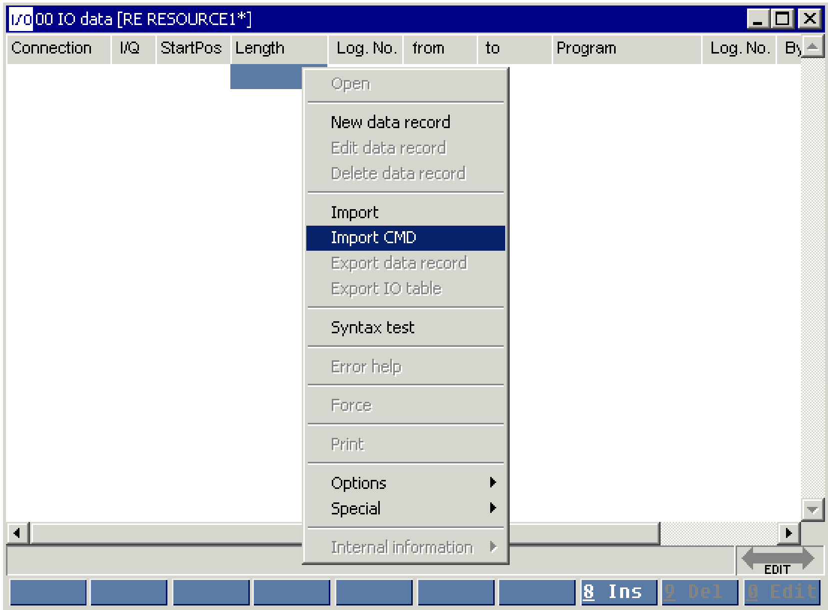
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	
1	No	LogDevNo	GrpNo	UniqueId	Ident	BusLevel	PDLen	StationName	DeviceName	IconNo	TypA1	TypW1	TypW2	DevState	PdIdx	PdName	PdDir	Pd
2	8	100	FFFF	7	11	0	16	'BTM15'	'Coupler'	11	1	1	1	1	1	'16-Bit_Eingang_1'	1	
3	8	100	FFFF	7	11	0	16	'BTM15'	'Coupler'	11	1	1	1	1	2	'16-Bit_Ausgang_1'	2	
4	9	101	FFFF	8	191	1	16	''	''	191	1	0	1	1	1	'16-Bit_Eingang_1'	1	
5	9	101	FFFF	8	191	1	16	''	''	191	1	0	1	1	2	'16-Bit_Ausgang_1'	2	
6	10	102	FFFF	9	191	1	16	''	''	191	1	0	1	1	1	'16-Bit_Eingang_1'	1	
7	10	102	FFFF	9	191	1	16	''	''	191	1	0	1	1	2	'16-Bit_Ausgang_1'	2	
8	11	103	FFFF	10	191	1	16	''	''	191	1	0	1	1	1	'16-Bit_Eingang_1'	1	
9	11	103	FFFF	10	191	1	16	''	''	191	1	0	1	1	2	'16-Bit_Ausgang_1'	2	
10	12	104	FFFF	11	191	1	32	''	''	191	1	0	1	1	1	'32-Bit_Eingang_1'	1	
11	12	104	FFFF	11	191	1	32	''	''	191	1	0	1	1	2	'32-Bit_Ausgang_1'	2	
12	13	200	FFFF	12	8	0	0	'Station GreenIBS'	'Basic Device	8	1	1	1	1	65535	'@'	0	
13	14	201	FFFF	13	189	1	32	'Station GreenIBS'	'Outputs'	189	1	0	1	1	2	'32-Bit_Ausgang_1'	2	
14	15	202	FFFF	14	190	1	32	'Station GreenIBS'	'Inputs'	190	1	0	1	1	1	'32-Bit_Eingang_1'	1	
15	16	300	FFFF	15	12	0	0	''	''	12	1	1	1	1	65535	'@'	0	

exel_csv.bmp

Fig. 10-18: CSV file in Excel format (reduced)

Importing the CSV File in the I/O Editor

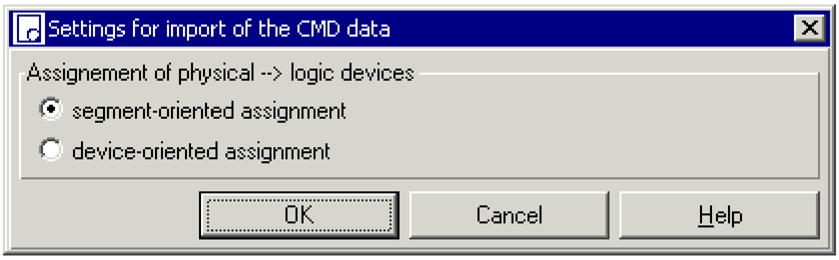
In the context-sensitive menu of the I/O editor, the "Import CMD" menu item data must be called up.



cmd_import.bmp

Fig. 10-19: Calling up the CMD import in the pop-up menu of the I/O editor

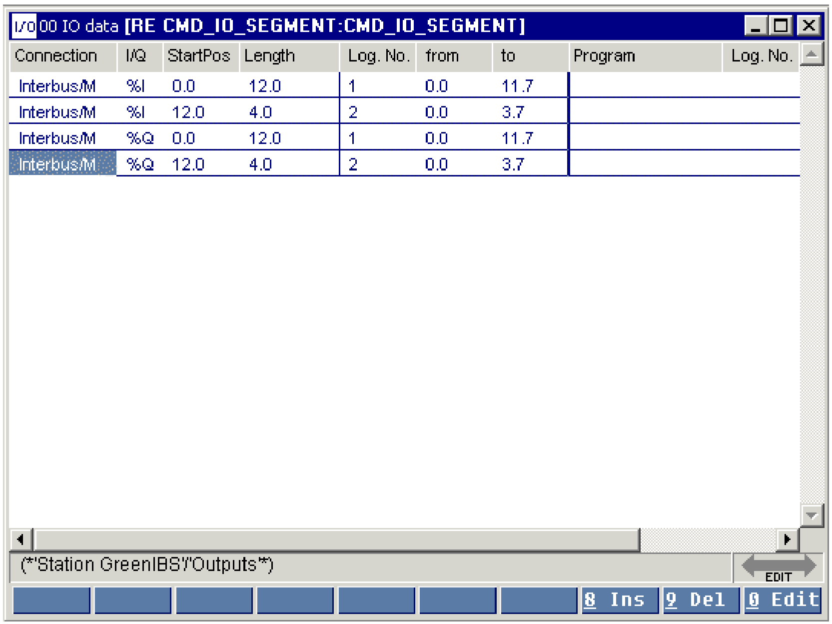
After the CSV file exported in the CMD beforehand has been selected, the selection of the assignment method is confirmed in a separate dialog.



Zuordnungsmode.bmp

Fig. 10-20: Assignment mode: physical devices, logic devices

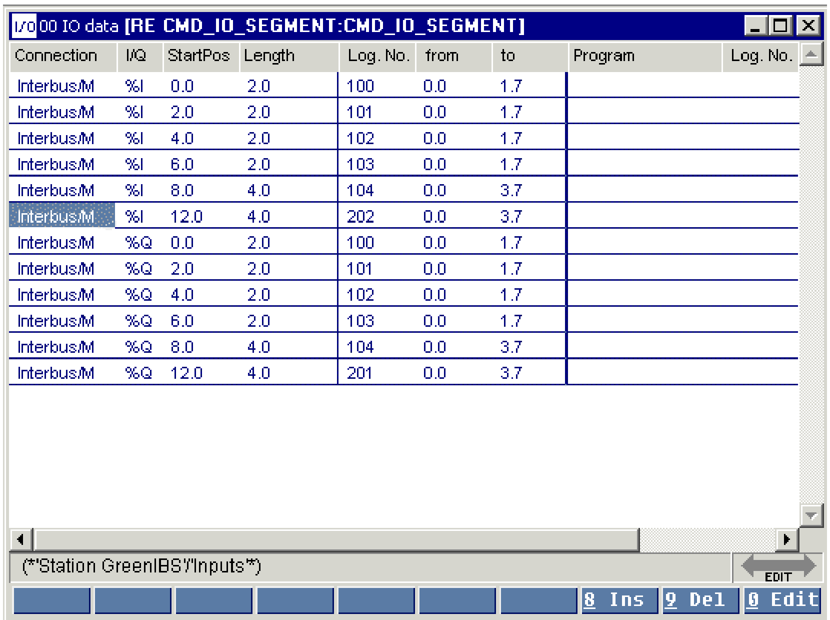
Depending on the setting selected, the I/O tables are displayed in different resources.



Res_segment_06.bmp

Fig. 10-21: Resource with I/O table with segment-oriented assignment

Each segment (remote bus terminal + local bus device) is comprised to a logic device. The logic device number corresponds to the segment number. Since each local bus device has the same segment number as the pertaining bus terminal, the assignment is unique.



Connection	I/Q	StartPos	Length	Log. No.	from	to	Program	Log. No.
Interbus/M	%I	0.0	2.0	100	0.0	1.7		
Interbus/M	%I	2.0	2.0	101	0.0	1.7		
Interbus/M	%I	4.0	2.0	102	0.0	1.7		
Interbus/M	%I	6.0	2.0	103	0.0	1.7		
Interbus/M	%I	8.0	4.0	104	0.0	3.7		
Interbus/M	%I	12.0	4.0	202	0.0	3.7		
Interbus/M	%Q	0.0	2.0	100	0.0	1.7		
Interbus/M	%Q	2.0	2.0	101	0.0	1.7		
Interbus/M	%Q	4.0	2.0	102	0.0	1.7		
Interbus/M	%Q	6.0	2.0	103	0.0	1.7		
Interbus/M	%Q	8.0	4.0	104	0.0	3.7		
Interbus/M	%Q	12.0	4.0	201	0.0	3.7		

Res_teilnehmer_07.bmp

Fig. 10-22: Resource with I/O table with device-oriented assignment

Each device is accepted as a separate logic device in the I/O database. The logic device number consists of the segment number and the position within the segment (segment number * 100 + position).

It is not reasonable to assign the logic device numbers completely according to the arrangement of the devices in the INTERBUS, because there is no fixed relation between the logic devices and the physical I/O devices. When devices are added subsequently, this would lead to an amount of modifications which is not justifiable.

Note: If the I/O peripherals are modified or extended, e.g. by adding additional I/O devices, the CSV file can be imported once again. In this case, the old logic devices of the "INTERBUS/M" connection contained in the I/O database of WinPCL are deleted before the new data are read and after an appropriate safety prompt has been answered.

Restrictions to the Import of CMD Files

- In the settings defining the data to be written to the CSV file, the standard settings may not be changed; it is mandatory to enter the parameters mentioned above (device number, direction (I/O), PD length, process data assignment). The options of the CSV file defining the separator between the particular elements remain unchanged. The heading line is mandatory.
- If **inserted subsequently,** additional modules must be inserted in CMD **manually,** because the segments are re-numbered when the configuration frame is automatically read from the connection, thus causing the assignment to the logic device numbers getting lost in WinPCL.
- Using the settings of the connection group in CMD, it is possible to assign an address to the registers of the bus master (diagnosis and standard function register) in the DPR. However, these data are not contained in the CSV file. As a result, it is not possible to assign logic devices to these data within WinPCL when the CMD data are imported. For that reason, these "devices" must be entered manually. That is the reason why it is not permitted to automatically delete all INTERBUS devices from the WinPCL I/O editor during import. Only existing devices are overwritten.

Restrictions to Segment-Oriented Assignment

- If two successive bit modules (data capacity < 8 bits) residing in different segments are comprised in one data byte, it is not possible to read these modules..
- The process data within one segment must be arranged successively.
- A multiple assignment of process data is not supported at first.
- If a change is made within a segment (e.g. if an I/O module is replaced by a module with a higher data capacity), the data assignment of the following local bus device may change; this cannot be eliminated by an alteration in the I/O editor. In such cases, the absolute identifiers in the declaration parts of the programs must be adjusted.

Example: User 1.1 (16-bit input) is replaced by a higher-capacity module (32-bit input).

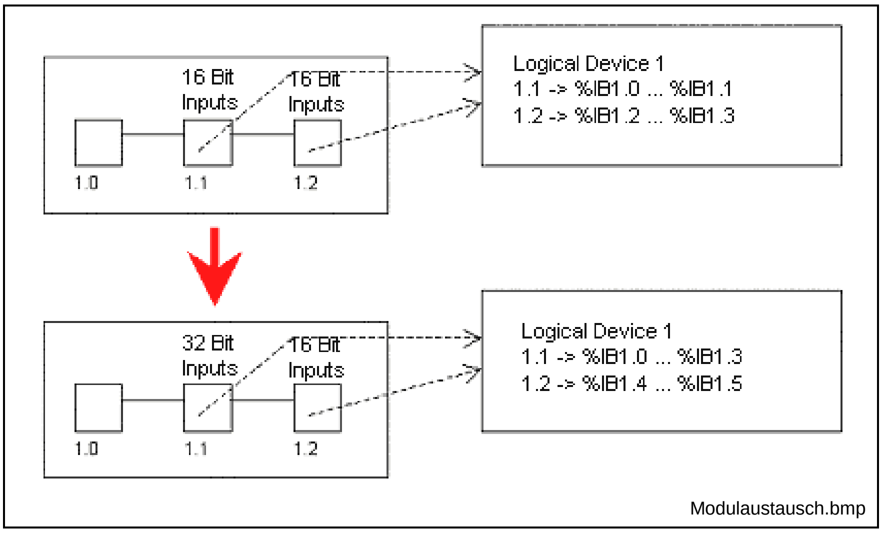


Fig. 10-23: Module replacement

Restrictions to Device-Oriented Assignment

- Successive bit modules (data capacity < 8 bits) whose process data are residing in one data byte are comprised to a logic device.
- According to the above proposal for assigning the segment number / position (CMD) to the logic device numbers (WinPCL), the assigned position numbers may not be higher than 99 (normal case: max. 8 local bus devices per bus terminal).

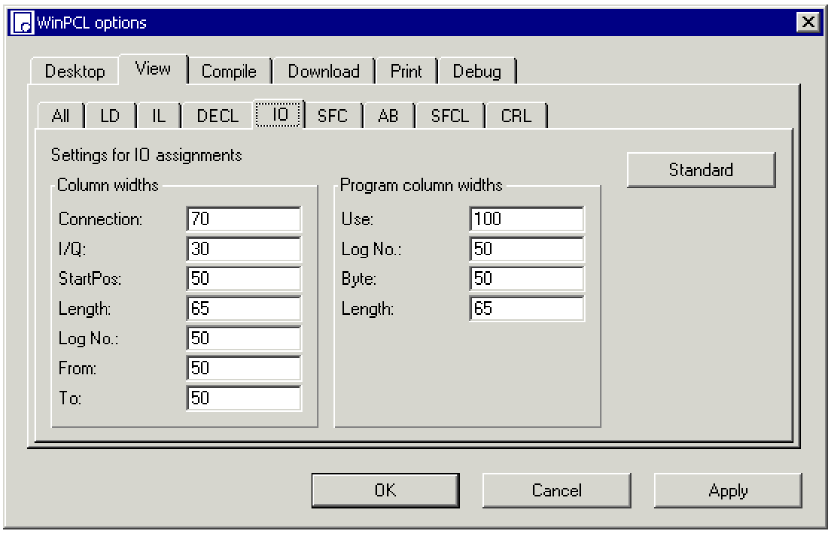
10.4 Status Display, I/O Editor

There is no active status display for the I/O editor.

The option of displaying the Bit/BYTE/WORD/DWORD variable values (%IDx.x or %QDx.x) by means of the "Start / Force, <Shift>+<F8>" menu item is in preparation.

10.5 Options, I/O Editor

The settings for the column width in the I/O editor can be selected using the Extras / Options menu item.



optionen_ansicht_io.bmp

Fig. 10-24: "Extras / Options" for column width settings

10.6 Pop-up Menu, I/O Editor <Shift>+<F10>

This pop-up menu contains the essential commands for this editor. It can be opened by pressing the right mouse button or the <Shift>+<F10> keys.

Menu items		Explanation
Open		
New data record		Opens the mask for input of a new data record.
Edit data record		Opens the mask for changing the current data record.
Delete data record		Deletes the current data record.
Import		The ASCII file chosen from the "WinPCL text files" is loaded into the I/O editor.
Import CMD		The CSV file written from the CMD is read and made visible in the I/O editor.
Export data record		The current data record is exported as ASCII file and stored in the folder "WinPCL text files".
Export I/O table		The I/O table is exported as ASCII file and stored in the folder "WinPCL text files".
Syntax text		List of all errors in the current editor. You can move to the place where the error occurred by double-clicking the mouse or by pressing the <Ctrl>+<Enter> keys.
Error help		The line, where the cursor is positioned, is tested for correct syntax. If an error is detected, this error is explained, also possible with <Ctrl>+<F1>.
Force (STATUS mode)		Allows the input of an absolute address. The value is indicated and can be forced once. The window remains open and the process can be activated again. Forcing takes place between the update of the input variables and the start of program code execution.
Print current window		Prints the editor contents.
Options		Toggles the absolute address and the name of the variable.
	Use RE/PR	Opens the third part of the I/O table for comparing the required I/Os with the existing I/Os.
	Sort by I/O	Table is sorted by inputs and outputs.
	Sort by interface connection	Table is sorted by interface connections.
	Sort by place of use	Table is sorted by place of use of RE/PR.
	Sort by logical no.	Table is sorted by logical numbers.
Special		Inserting and removing gaps when devices are deleted and added
	Removing gaps	Input dialog for removing gaps in the DPR
	Inserting gaps	Input dialog for inserting gaps in the DPR
Internals		Search for faults in the programming system, to be used only if approved by the service.

11 Data Types in WinPCL

11.1 General Agreements

This chapter defines textual and graphic elements which are common to all programming languages of the system.

Character Set

Within the programming system the ASCII character set, ISO-646 IRV, given as Table 1 – Row 00 of IDO/IEC 10646 is used.

Large and small letters lead to different names, i.e. 'abcd' distinguishes from 'ABCD' and 'AbcD'. Umlauts can be used only in the comments.

Identifiers (Names)

An *identifier* is a string of letters, digits, and underline characters which shall begin with a letter or underline character.

The case of letters is significant in identifiers, e.g., the identifiers `abcd`, `ABCD`, and `aBCd` shall be interpreted different.

Furthermore, underlines shall be significant in identifiers, e.g., `A_BCD` and `AB_CD` shall be interpreted as different identifiers. Multiple leading or multiple embedded underlines are not allowed.

Spaces are not allowed in identifiers.

An identifier (Name) of variables, steps, transitions, actions, programs, function blocks, functions and data types must not contain more than 32 signs. The limitation is necessary, as external programs cooperating with WinPCL, can not handle more than 32 signs.

Keywords

Keywords are unique combinations of characters utilized as individual syntactic elements as defined in ‘EN 61131-3’. Keywords shall not contain imbedded spaces. They must not be employed as names. Otherwise, the programming system rejects them.

Note: Keywords contain only upper case letters and '_'.

Use of White Space

The user shall be allowed to insert one or more characters of "white space" anywhere in the text of programmable controller programs except within keywords, literals, enumerated values, identifiers, directly represented variables, or delimiter combinations . "White space" is defined as the SPACE character with encoded value 32 decimal, as well as non-printing characters such as tab, newline, etc. for which no encoding is given in IEC/ISO 10646-1.

Comments

User comments shall be delimited at the beginning and end by the special character combinations "`(*`" and "`*)`".

Comments do not have any syntactic or semantic importance in all programming languages. They can be entered in the corresponding columns of the editors (for example Declaration-, IL editor ...) or by means of a separate comment window.

Note: The use of nested comments, e.g., `(* (* NESTED *) *)`, shall be treated as an **error.**

Pragmas

Pragmas shall be delimited at the beginning and end by curly brackets "{" and "}", respectively. The syntax and semantics of particular pragma constructions are **implementation dependent**. Directives shall be permitted anywhere in the program where spaces are allowed, except within character strings.

Note: The use of pragmas is not allowed in user applications.

External Representation of Data

External representations of data in the various programmable controller programming languages shall consist of

- numeric literals,
- character strings,
- and time literals.

Numeric Literals

There are two classes of numeric literals: integer literals and real literals. A numeric literal is defined as a decimal number or a based number. The maximum number of digits for each kind of numeric literal shall be sufficient to express the entire range and precision of values of all the data types which are represented by the literal in a given implementation.

Single underline characters (_) inserted between the digits of a numeric literal are not allowed.

Decimal literals shall be represented in conventional decimal notation. Real literals shall be distinguished by the presence of a decimal point. An exponent indicates the integer power of ten by which the preceding number is to be multiplied to obtain the value represented. Decimal literals and their exponents can contain a preceding sign (+ or -).

Integer literals can also be represented in base 2, 8, or 16. The base shall be in decimal notation. For base 16, an extended set of digits consisting of the letters A through F shall be used, with the conventional significance of decimal 10 through 15, respectively. Based numbers shall not contain a leading sign (+ or -).

Boolean data shall be represented by integer literals with the value zero (0) or one (1), or the keywords FALSE or TRUE, respectively.

Feature description	Examples
Base 10 literals (decimal)	240
Base 2 literals (dual)	2#11100000
Base 8 literals (octal)	8#340
Base 16 literals (hexadecimal)	16#E0

Fig. 11-1: Numeric literals

Floating point numbers can contain up to 7 Digits (REAL), and/or 13 Digits (LREAL): 1.234567, -123.4567 ...

Floating point number with exponent can contain besides an exponent: 1.234567E+7, -1. 234567 E-15

Floating point numbers (REAL) can be entered in the ranges: -3.402823E38...-1.175495E-38 and 1.175495E-38...3.402823E38.

Signs are allowed before the exponent and the base number.

The internal format of REAL (1 bit sign, 8 bits exponent and 23 bits mantissa) can not depict certain values. The next representable value is taken, i.e. the larger a number becomes, the more inaccurate is its resolution.

Therefore, there is no conventional zero. It is rounded as -1.175495E-38 or 1.175495E-38 depending on wether it is calculated in the negative or positive range (see also Standard Data Types).

Note: Zero is defined as 0.0

Time Literals

Duration data shall be delimited on the left by the keyword `T#` . The representation of duration data in terms of days, hours, minutes, seconds, and milliseconds, or any combination thereof, shall be supported. The least significant time unit can be written in real notation without exponent.

"Overflow" of the most significant unit of a duration literal is permitted, e.g., the notation `T#25h15m` is permitted.

The following time units are permitted:

Feature description	Examples
day (d)	T#14d
hour (h)	T#147h
minute (m)	T#47m
second (s)	T#14s
Millisecond (ms)	T#12ms
combination	T#25h3s, T#12m13s127ms, T#1m33.3s

Fig. 11-2: Duration literal features

A decimal point is possible.

The boundary for the TIME data type is with the time base of 2 ms: 99d10h5m34s590ms. Presently, this value is also indicated, if the variable of the TIME data type contains the bit pattern 16#FFFFFFFF. Currently, the input is on 23d23h59m59s999ms limited.

Note: The minimum time pattern is 2ms.

Character String Literals

A single-byte character string literal is a sequence of zero or more characters from Row 00 of the ISO/IEC 10646 character set prefixed and terminated by the single quote character ('). An empty character string is allowed. In single-byte character strings, the three-character combination of the dollar sign ($) followed by two hexadecimal digits shall be interpreted as the hexadecimal representation of the eight-bit character code.

Two-character combinations beginning with the dollar sign shall be interpreted as shown in table below when they occur in character strings.

No.	Example	Explanation
1	‘’	Empty string (length zero)
	‘A’	String of length one containing the single character A
	‘ ‘	String of length one containing the "space" character
	‘$05’	String of length one containing the '♣'
	‘$’’	String of length one containing the "single quote" character
	‘RL’	String of length two containing CR and LF characters
	‘$$1.00’	String of length five which would print as "$1.00"

Fig. 11-3: Character string literal features

No.	Combin.	Interpretation when printed
2	$$	Dollar sign
3	$’	Singl quote
4	$L	Line feed
6	$P	Form feed (page)
7	$R	Carriage return
8		Tab

Fig. 11-4: Combination with '$'

Note: $N, Newline, is not supported.

11.2 Data Types and Initial Values

The programming system allows the application of

- Standard Data Types, elementary, predefined,
- Firmware Data Types, data types, which are available in the library in addition to elementary data types,
- User Data Types, data types additionally declared by the user.
- This chapter also contains instructions on how to use Structures (STRUCT) and ARRAYs and
- Pointer and Address of.

11.3 Standard Data Types

Elementary Data Types, Value Ranges and Initial Values

The EN 61131-3 standard specifies the elementary data types and the value ranges and initial values permissible for these data types, all listed in the table below.

Data types with a data capacity of 64 bits are released for only a part of the controls.

Keyword	Data type	Bits	Initial	Range of value
BOOL	Boolean	1	0	TRUE/FALSE, 1/0
BYTE	Bit string of length 8	8	0	Bit string only, not a number
WORD	Bit string of length 16	16	0	Bit string only, not a number
DWORD	Bit string of length 32	32	0	Bit string only, not a number
LWORD	Bit string of length 64	64	0	Bit string only, not a number
CHAR	single-byte Character	8	' ' (empty)	ANSI Character
WCHAR	double-byte Character	16	" " (empty)	UNICODE Character
STRING	Variable length single-byte charakter string	...	' ' (empty)	256 byte memory requirement
STRING[n]	with n CHARs	...	' ' (empty)	(n+1) byte memory requirement
WSTRING	Variable length double-byte charakter string	...	" " (empty)	512 byte memory requirement
WSTRING[n]	with n WCHARs	...	" " (empty)	2*(n+1) byte memory requirement
SINT	Short integer	8	0	-128 .... +127
INT	Integer	16	0	-32768 .... +32767
DINT	Double integer	32	0	-2147483648 .... +2147483647
LINT	Long integer	64	0	$-(2^{n-1})$ .... $+(2^{n-1})-1$
USINT	Unsigned short integer	8	0	0 .... +255
UINT	Unsigned integer	16	0	0 .... +65535
UDINT	Unsigned double integer	32	0	0 .... +4294967295
ULINT	Unsigned long integer	64	0	0 .... $+(2^{n})-1$
REAL	Real number	32	0.0	-3.402823E38 .... -1.175495E-38 1.175495E-38 .... 3.402823E38
LREAL	Long real number -	64	0.0	-1.7976931348623E308 .... -2.22507385850721E-308 2.22507385850721E-308 .... 1.7976931348623E308
TIME	Duration	32	T#0s	0ms ..23d23h59m59s999ms
DATE	Date only			in preparation
TOD	Time of day only			in preparation
DT	Date and Time of day			in preparation

Datentypen.bmp

Fig. 11-5: Elementary data types, value ranges and initial values

Rexroth Bosch Group

Note: At position 0 (string[0]), the STRING data type specifies the current length of the character string. The initial value of this position is zero!

If the value of a CHAR is loaded into a character string, the length of the character string must also be updated!

Note: The minimum resolution for the TIME data type is 2 ms. Values are rounded automatically.

The 32-bit version allows the display of a time interval of more than 24 days.

Note: Where the REAL and LREAL data types are concerned, the initial value of 0.0 is filed as one of the two values which is nearest to it. Take caution when comparing for "equality / inequality".

Location and size prefix features for directly represented variables

The address of a variable contains:

- percent sign "%" at the beginning,
- a *location prefix* (Input, Output, Memory, Memory RETAIN),
- a *size prefix* (BOOL, BYTE, ...),
- one or more unsigned integers, separated by periods (.).

Parts of the address of a variable

Location / Size		**Explanation**	
Location:	I or E O, Q or A M R	Input Output Memory Memory, RETAIN	
Size	none or X B W D L	BOOL BYTE WORD DWORD LWORD	(1 bit) (8 bit) (16 bit) (32 bit) (64 bit)

Fig. 11-6: Parts of the address of a variable

Example

%I1.2.3	logical device number 1, input byte 2, bit 3 or
%IX1.2.3	(another representation)
%Q127.5.2	logical device number 127, output byte 5, bit 2 or
%QX127.5.2	(another representation)
%IB1.2	logical device number 1, input byte 2,
%QW127.5	logical device number 127, output byte 5

Depending on the size, the bit strings BYTE, WORD, DWORD and LWORD may be redefined to numbers.

The following rules apply to data higher than one byte:

WORD: Specified slot and next one

e.g.: %IW1.0 HighByte: %IB1.0, LowByte: %IB1.1

DWORD: Specified slot and next three slots

e.g.: %ID1.0 HighByte: %IB1.0,....., LowByte: %IB1.3

LWORD: Specified slot and next seven slots

e.g.: %IL1.0 HighByte: %IB1.0,....., LowByte: %IB1.7

2048 flag bits are available for flags. These may be accessed through byte, word, double word, or long word address (standard in Tools\Options\Compile...).

Falg bits:	%M0.0 ... %M2047.7or %MX0.0... %MX2047.7
Flag bytes:	%MB0 ... %MB2047
Flag words:	%MW0 ... %MW2046
Flag double word:	%MD0 ... %MD2044
Flag long word:	%ML0 ... %ML2040

Example for the location of memory storage:

%MD1 contains the value 16#3F2E1D0C.

```
        |<-----%ML1---------------...------->|
        |<-----%MD1-------->|
        |<-%MW1-->|<-%MW3-->|
        |    |<-%MW2-->|    |
    00  | 3F | 2E | 1D | 0C | 00
  %MB0|%MB1|%MB2|%MB3|%MB4|%MB5
 ┌──────────┘    └──────────┐
 %M2.7  6  5  4  3  2  1  0
     0  0  1  0  1  1  1  0
```

At the same time, 2048 flag bits are available as retain flags (standard in Tools\Options\Compile...).

The retain flags may be accessed in the same manner through bit-, byte-, word-, double word, or long word addresses:

Flag bits:	%R0.0 ... %R2047.7 or %RX0.0...%RX2047.7
Flag bytes:	%RB0 ... %RB2047
Flag words:	%RW0 ... %RW2046
Flag double word:	%RD0 ... %RD2044
Flag long word:	%RL0 ... %RL2040

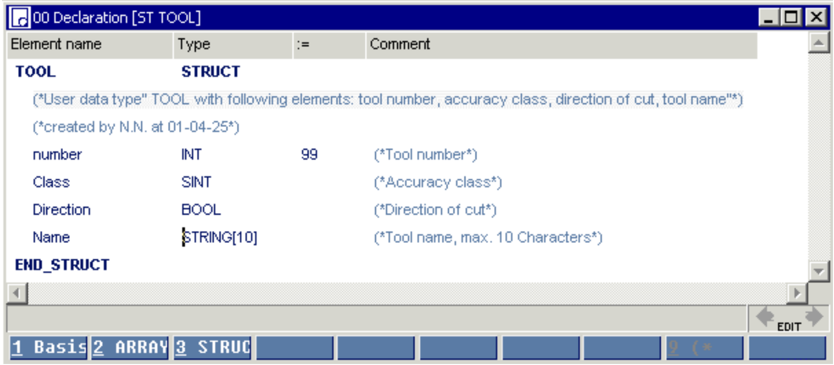
Note: The use of absolute addressed variables is allowed in a program, but not in a function block or in a function

Directly represented external variables are permitted in programs or function blocks.

Extensions to Elementary Data Types

Structures (STRUCT)

A structure consists of one or several elements, which can be of the elementary type or can be a structure or an array. Each element has its own name and, if it is of the elementary type, can have a user-defined initial value. Structures and arrays have their own initial values. In addition to the declaration comment of the structure, each element can have its own comment.



ST Werkzeug.bmp

Fig. 11-7: Structure of a declaration illustrated by the "TOOL" structure

The declaration comment is added to the line specifying the name. Of the four elements of the structure, "number" is defined with "99" by the user; the standard value "0" or "FALSE" is assigned to the other elements. The initial value of name is ' ' (empty) .

Accessing a structure element:

The following variable is agreed in the declaration part of a file:

Name	AT	TYPE	:=	Comment
Tool_1		TOOL		(*Definition of the variable "Tool_1"*)

Fig. 11-8: Declaration line (VAR…END_VAR range)

Based on the declaration, there are the following access possibilities:

OP	Operand	Comment
LD	Tool_1	(*Load complete structure*)
LD	Tool_1.Name	(*Load "Name" from "Tool_1"*)
LD	Tool_1.Name[5]	(*Load 5th letter from "Name" from Tool_1*)

Fig. 11-9: IL lines for access to structures and structure elements

Assigning an absolute address:

Irrespective of the dataset it contains, each structure starts with a word address. The size of the address range is based on the dataset specified by the data type.

Name	AT	TYPE	:=	Comment
y_axis	%IW100.8	iAXIS		(*Bus type "MTC200*)

Fig. 11-10: Absolutely addressed structure (axis of type "iAXIS")

ARRAYs

The elements of an array have a unique data type, which can be of the elementary type or can be a structure or even an array itself. The user can assign a unique initial value to all elements, if they are elementary. Structures and arrays have their own initial values.

The elements of an array are arranged dimensionally

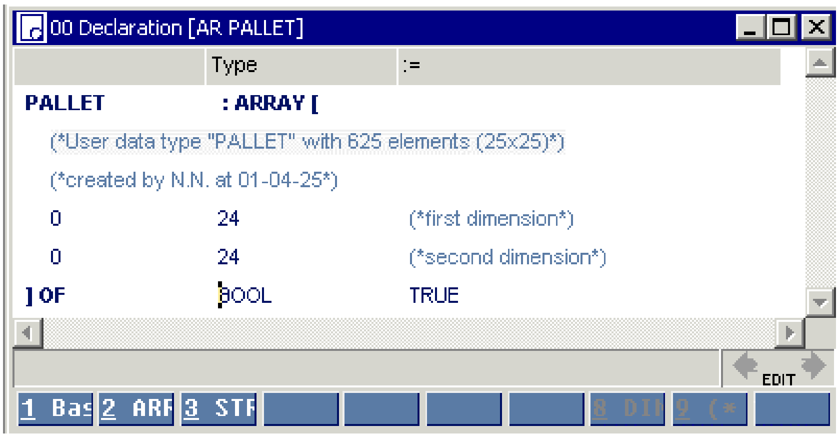
(1 to 4 dimensions).

In addition to the declaration comment of the array, a comment can be given for each dimension.

The declaration comment is added to the line specifying the name.

All dimensions start with the zero element. The unique data type is BOOL

The user sets the value for each element to TRUE.



AR palette.bmp

Fig. 11-11: Structure of a declaration illustrated by example of the "PALLET" elementary array

Accessing the array or an array element:

The following variable is agreed in the declaration part of a file:

Name	AT	TYPE	:=	Comment
Pallet_1		PALLET		(*Definition of the variable "Pallet_1"*)

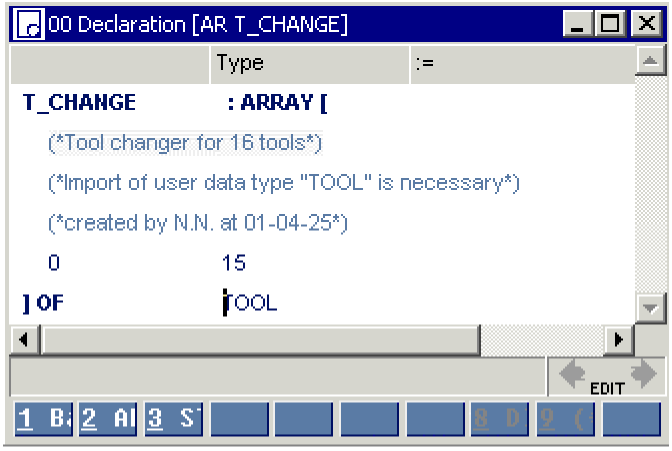
Fig. 11-12: Declaration line (VAR…END_VAR range)

Based on the declaration, there are the following access possibilities:

OP	Operand	Comment
LD	Pallet_1	(*Load complete array*)
LD	Pallet_1[1,3]	(*Load element 1, 3*)
LD	Pallet_1[length,width]	(*Load "length", "width" element*)

Fig. 11-13: IL lines for accessing the array or an array element

Instead of an elementary variable, the element of an array can also be a structure or even an array itself. The figure below shows an ARRAY of STRUCT.



AR Wechsler.bmp

Fig. 11-14: Declaration illustrated by example of the "T_Changer" structured array

The declaration comment is added to the line specifying the name. All of the elements are Structures (STRUCT), comprising several elements themselves.

Accessing the array or an array element:

The following variable is agreed in the declaration part of a file:

Name	AT	TYPE	:=	Comment
T_Changer_1		T_CHANGER		(*Def. of the variable T_Changer_1*)

Fig. 11-15: Declaration line (VAR…END_VAR range)

Based on the declaration, there are the following access possibilities:

OP	Operand	Comment
LD	T_Changer_1	(*Load complete array*)
LD	T-Changer_1 [10]	(*Load eleventh element, i.e. a complete structure*)
LD	T_Changer_1 [10].Name	(*Load "Name" of eleventh array element*)
LD	T_Changer_1 [10].Name[5]	(*Load fifth letter of "Name" of eleventh array element*)

Fig. 11-16: IL lines for accessing the array or an array element

Assigning an absolute address:

Irrespective of the data set it contains, each array starts with a word address. The size of the address range is based on the data set specified by the data type.

Name	AT	TYPE	:=	Comment
Pallet_1	%RW100	PALETTE		(*RETAIN ARRAY*)

Fig. 11-17: Absolutely addressed array (retain flag)

Pointer and Address of

WinPCL allows the data access with typed POINTER / ADRESSE OF.

Pointer

A pointer is declared in the declaration part of a program or function block.

'^' in front of the data type identifies the pointer.

The data type behind '^' defines length of the area and type and number of elements which are in the area.

Write and read access is monitored.

The initial value of a pointer is NIL (NIL pointer

Note: A pointer that does not show onto a variable (NIL pointer) yet is used during the program processing, so the rung is not executed (skipped over as it were).

The customer receives about S#ErrorFlg/S#ErrorNr (kind of error)/S#ErrorTyp (cause of error) the information and it should evaluate.

Name	AT	TYPE	Comment
VAR			
bitptr		^A_BOOL16	A_BOOL16: ARRAY [0..16] of BOOL
END_VAR			

Fig. 11-18: Declaration of a pointer, in the example "bitptr"

Address of

The point the pointer is to be directed at is transferred using a 'P#' operator :

Label	Operand	Operator	Comment
	LD	P#bytefeld[9]	Load address of the 9th element of the bytefeld variable
	ST	Bitptr	=> Start address of the section of type A_BOOL16

Fig. 11-19: "Address of" in the instruction list

Rexroth Bosch Group

Pointer points to

A pointer can point either for data access, in read or write form, to a complete area in accordance with the data type connected to it or to any element of its data type.

Label	Operand	Operator	Comment
	LD	bitptr^	Means: Fetch the complete section.
	LD	bitptr^[1]	Means: Fetch the first element (A_BOOL16 is a 1-d ARRAY).

Fig. 11-20: Example of access via pointer

Note: **The memory area of the "source" must always be greater than or at least equal to the area of the "destination".**

If the limit of the source area for read or write access is exceeded, an error is generated ***during runtime.*** The operation will not be carried out! (Example of an incorrect access: It is intended to copy a structure of 7 bits to one byte.)

Example

The bytefeld array is defined: A_BYTE40 ARRAY [0..39] of BYTE.

It is intended to copy 16 bits from this array, starting with byte 9.

The 16 bits are to be organized as bitfeld array "A_BOOL16 ARRAY [0..15] of BOOL".

BYTE 0	0	...	7			
	...		...			
BYTE 9	72	...	79	-->	Bit	0 ... 15
BYTE10	80	...	87			
	...		...			
BYTE 39	312	...	319			

Fig. 11-21: Structure of the "bytefeld" array and bits to be copied

Name	AT	TYPE	Comment
VAR			
bytefeld		A_BYTE40	Source
bitptr		^A_BOOL16	Pointer of type A_BOOL16
bitfeld		A_BOOL_16	Destination
END_VAR			

Fig. 11-22: Declaration for the example

‘bitptr’ is a pointer with address and type information.

The type information is transmitted into the declaration.

The pointer is a NIL pointer until an address is transmitted to the pointer.

The current address is transmitted at least once in the implementation with P#. It can be changed as often as required.

	BYTE 0	0	...	7
		...		...
P#bytefeld[9]	BYTE 9	72	...	79
	BYTE10	80	...	87
		...		...
	BYTE 39	312	...	319

Label	Operand	Operator	Comment
	LD	P#bytefeld[9]	Load address of BYTE 9.
	ST	bitptr	=> Start address of the section of type A_BOOL16
		....	
	LD	bitptr^	Copy the section from the bytefeld
	ST	bitfeld	into the bitfeld, type limits are checked
		....	
	LD	bitptr^	Means: Fetch the complete section (all 16 bits).
		....	
	LD	bitptr[1]	Means: Fetch bit 73.

Fig. 11-23: Implementation for the example

11.4 Firmware Data Types

Firmware data types have been developed for an effective support of firmware function blocks e.g. for operating serial interfaces or for diagnosis support. They are stored in the library of the programming system and can be used by the user, but cannot be changed.

- Serial Interfaces, Data Types
- PROFIBUS DP, Data Types
- ASIM, Data Types
- Sequential Function Chart, Data Types

Serial Interfaces, Data Types

Before being operated, a serial interface must first be parameterized. The parameters of an interface are grouped using the COM data type. The individual elements of the data type are designed as integer values, which correspond with the ident-numbers of the Rexroth-IONET protocol.

COM	**STRUCT**	**(*Firmware data types*)**
DEVICE	INT	Device number
SERNR	INT	Number of the serial interface
BAUD	INT	Baud rate
DATA	INT	Number of data bits
PARITY	INT	Parity
STOP	INT	Number of stop bits
PROTOKOL	INT	Protocol
HANDSH	INT	Handshake
END_STRUCT		

Explanation of the parameter values which can be set in the COM data type

Parameter	**Value**	**Explanation**
DEVICE	0...999	Device number
	0	Onboard or SIO4 interface
	1...999	Not defined under WinPCL

Parameter	**Value**	**Explanation**
SERNR	0...4	Number of the interface. 0 is the onboard interface of the PLC card.

Parameter	**Value**	**Explanation**
BAUD	1...21	Transmission rate

Value	Baud rate
1	50
2	75
3	110
4	134.5
5	150
6	200
7	300
8	600
9	1050
10	1200
11	1800
12	2000
13	2400
14	3600
15	4800
16	7200
17	9600
18	19200
19	38400
20	57600
21	115200

Parameter	Value	Explanation
DATA	1...4	Number of useful data bits

Value	Data bit
1	8
2	7
3	6
4	5

Parameter	Value	Explanation
PARITY	1...5	Type of parity check

Value	Parity
1	NONE
2	ODD
3	EVEN
4	MARK
5	SPACE

Parameter	Value	Explanation
STOP	1...3	Number of stop bits

Valu e	Stop bit
1	1
2	1.5
3	2

Parameter	Value	Explanation
PROTOKOL	1...7	Type of protocol

Value	Protocol
1	Not defined
2	Not defined
3	ASCII
4	SIS protocol
5	ASCII-RS232
6	ASCII-RS422
7	ASCII-RS485

Parameter	Value	Explanation
HANDSH	1...3	Type of handshake

Value	Handshake
1	None
2	Software
3	Hardware

PROFIBUS DP, Data Types

The following Firmware Data Types are available:

- PROFIBUS status information: DPGLOBAL
- Status bits of a PROFIBUS slave: DPSLDIAG

DPGLOBAL

Status information on the PROFIBUS DP, Data Types

The firmware data type DPGLOBAL is an "Array of BOOL", which indicates the status bits of the PROFIBUS. The array consists of the following elements:

Signal	Name	Meaning
CTRL	Control Error	Error in parameter setting.
ACLR	Autoclear Error	The master has stopped the communication with all slaves.
NEXC	Non Exchange Error	At least one slave did not obtain the data exchange status. No exchange of process data.
FAT	Fatal Error	No bus communication possible after fatal bus error, e.g. bus short-circuit.
EVE	Event Error	Bus short-circuits were detected by the master. The number of short-circuits is stored in the variable "bus_error_cnt". This bit is not deleted automatically.
NRDY	Host Not Ready Notification	The user program signals that it is not ready.
TOUT	Timeout Error	Timeout due to denied telegrams detected by the master. This bit is not deleted automatically.

Fig. 11-24: PROFIBUS status information:

DPSLDIAG

Slave status bits PROFIBUS DP, Data Types

The firmware data type DPSLDIAG is an array, which indicates the status bits of a PROFIBUS slave. The array consists of the following elements:

Signal	Meaning
StaNonEx	DP slave does not answer
StaNotRd	DP slave not ready
CfgFault	Error in parameter setting for DP slave
ExtDiag	DP slave reports extended diagnosis
NotSupp	DP slave reports invalid command
InvSlRes	Invalid DP slave answer
PrmFault	Last parameter telegram faulty
MastLock	DP slave parameterized by another master
PrmReq	DP slave not parameterized yet
StatDiag	DP slave diagnosis provided
S2_D2	Reserved
WDOn	Watchdog of the DP slave is active
FreezeMd	Freeze command active
SyncMd	Sync command active
S2_D6	Reserved
Deaktiv	DP slave not projected
S3_D0	Reserved
S3_D1	Reserved
S3_D2	Reserved
S3_D3	Reserved
S3_D4	Reserved
S3_D5	Reserved
S3_D6	Reserved
ExtDiag0	Data area overflow extended diagnosis
MastAdd	Address of the parameterizing DP master
IdentNr	Ident number of the DP slave

Fig. 11-25: Slave status signals

ASIM, Data Types

The following Firmware Data Types are available:

ASISLDIAG

The diagnostic information of an ASI slave are kept as instance of the data type described below:

Signal	Meaning
NO_RESPONSE	The device does not answer or is not available.
BUFFER_OVERFLOW	The number of the entries in the error buffer exceeds the maximum possible number.
RESERVED_2	At the moment not used.
CONFIGURATION_FAULT	The detected IO or ID code differs from the configured code.
RESERVED_4/5/6	At the moment not used.
NOT_ACTIVE	The slave is not active in the current configuration.
CONFIGURATION_DATA	Retrieved IO/OD code.
DEV_NOT_INITIALIZED	Slave was not initialized.
DEV_NOT_ACTIVE	Slave not active.
NO_FAULT	Slave indicates no error.
DEV_MISSING	Slave not available.
DEV_FOUND	Is not supported yet.
DIAG_0/1/2	Slave-specific diagnostic information. See Manual of the manufacturer.

Fig. 11-26: Slave status signal

Sequential Function Chart, Data Types

To the structuring elements of the sequential function chart, i.e. step, transition, action, and to the sequential function chart itself, upgradable Firmware Data Types are assigned, permitting the user to control the capacity of the sequential function chart to a sufficient extent. The minimum capacity is as follows:

- Actions: _tACTION
- Steps: _tSTEP
- Transition: _tTRANSITION
- SFC - internal: _tSFCINTERN
- SFC - external: _tSFC

Instead of this data type, the user can develop a data type which, in addition to the above mentioned elements, contains further elements which are attached.

_tACTION

Data type of actions (Sequential Function Chart, Data Types)

Name	Type	Comment
action_name.Q	BOOL	Indicates whether the action is being executed.
action_name.A	BOOL	Indicates whether the action is being executed, reprocessed or forced (all processing methods).
action_name.F	BOOL	Is the variable by which the action can be forced and which indicates at the same time whether the action is being forced.
action_name.JOG	BOOL	Only important in automatic jog mode; indicates that the following transition is fulfilled.

Fig. 11-27: Variables which are assigned to an action with _tACTION

The time diagrams for the variables action_name.Q and action_name.A run according to the action qualifier selected. The forcing sequence is to be found in the following figure.

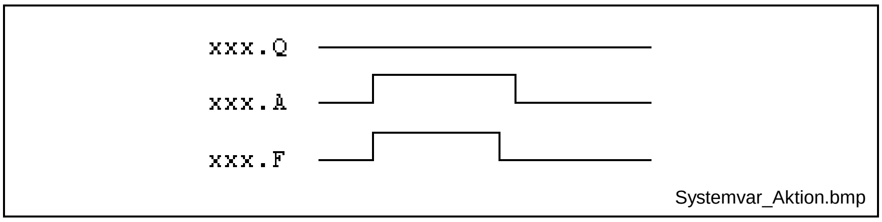


Fig. 11-28: Variables for forcing action xxx

Note: It is not possible to assign structure elements to Boolean actions and negated Boolean actions.

A forcing is not therefore either possible in the manual mode.

_tTRANSITION

Data type of transitions (Sequential Function Chart, Data Types)

Name	Type	Comment
trans_name.JOG	BOOL	Write, only in AUTOMATIC JOG mode, TRUE if the transition should not advance after firing

Fig. 11-29: Variables assigned to a transition with tTRANSITION

Instead of this data type, the user can develop a data type which, in addition to the above mentioned element, contains further elements which are attached.

Note: Boolean variables, negated Boolean variables, as well as the constants TRUE and FALSE, no JOG variable can be assigned.

_tSTEP

Data type of steps (Sequential Function Chart, Data Types)

Name	Type	Comment
step_name.X	BOOL	Step flag: TRUE, if step is active
step_name.F	BOOL	TRUE - forcing of the step, possible only the in manual mode
step_name.SYNC	BOOL	TRUE - request to set this step for synchronization
step_name.T	TIME	Step active time - read only, time elapsed since activation of the step

Fig. 11-30: Variables which are assigned to a step with _tSTEP

The step flag step_name.X indicates whether a step is active or not.

Using the step_name.F flag, the step can be activated and deactivated by a program or by forcing if MODE_AUTO = FALSE (no automatic mode).

The step_name.SYNC flag allows a step to the preselected in the manual mode. With the next synchronization, an attempt is made to activate this step as part of the new step set.

The step active time indicates for how long the step is already been active. It retains the last value after deactivation until the step is reactivated or RESET becomes active.

Instead of this data type, the user can develop a data type which, in addition to the above mentioned elements, contains further elements which are attached.

_tSFCINTERN

Internal data type of SFCs (Sequential Function Chart, Data Types)

- SFC - external: _tSFC

_tSFCINTERN	**STRUCT**	**Comment**
START	BOOL	Further processing of the sequential function chart, write: releasing one blocked with JOG, however filled transition condition
STOP	BOOL	Stop of the sequential function chart, write
SET_HAND	BOOL	Forcing of the operating mode TRUE, change sequential function chart from automatic mode to manual mode, Write
MODE_AUTO	BOOL	Indication of the current operating mode: TRUE, if sequential function chart in automatic mode - read only
STATUS_STOP	BOOL	Indication of the current operating mode: TRUE, if sequential function chart was stopped - read only
RESET	BOOL	Reset and initialization of the sequential function chart in manual and automatic mode, write
SYNC	BOOL	Attempt to activate the sequential function chart with preset steps (step_name.SYNC=TRUE) in automatic mode, write
JOG	BOOL	Transfer sequential function chart from automatic mode into automatic JOG mode, write
ERRORFLG	BOOL	Analog to S#ErrorFlg with POEs
ERRORNR	USINT	Analog to S#ErrorNr with POEs
ERRORTYP	INT	Analog to S#ErrorTyp with POEs
aIN_USER	_tACTION	Permanently executed system action, to be filled in by the user
aIN_SYSTEM	_tACTION	Permanently executed system action, used by the system
aOUT_SYSTEM	_tACTION	Permanently executed system action, used by the system
aOUT_USER	_tACTION	Permanently executed system action, to be filled in by the user
END_STRUCT		

Fig. 11-31: _tSFC structure

Note: Forcing of the action execution when the mode changes: Only steps are forced. There will be changes in the actions.

Instead of this data type, the user can develop a data type which, in addition to the above mentioned elements, contains further elements which are attached.

_tSFC

External data type of SFCs (Sequential Function Chart, Data Types)

_tSFC	**STRUCT**	**Comment**
INTERN	_tSFCINTERN;	(*Structure for controlling the sequential function chart*)
END_STRUCT		

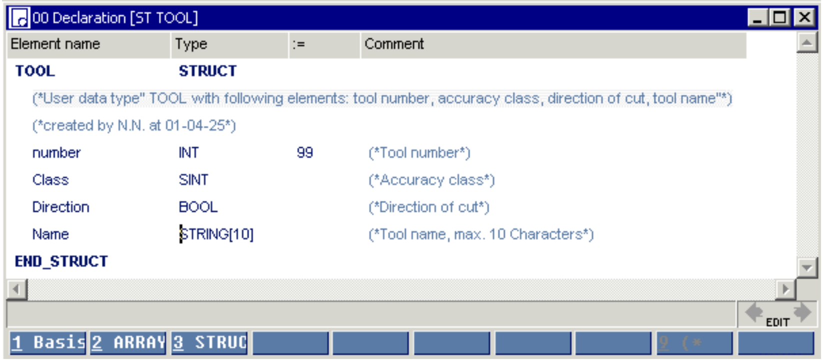
Fig. 11-32: _tSFC structure

11.5 User Data Types

User data types can be designed as structures or arrays.

Structures (STRUCT)

Structures consist of one or several elements, which can be of the elementary type or can be a structure or an array. Each element has its own name and, if it is of the elementary type, can have a user-defined initial value. Structures and arrays have their own initial values. In addition to the declaration comment of the structure, each element can have its own comment.



ST Werkzeug.bmp

Fig. 11-33: Structure of a declaration illustrated by the "TOOL" structure

The declaration comment is added to the line specifying the name.

Of the four elements of the structure, "number" is defined with "99" by the user; the standard value "0" or "FALSE" is assigned to the other elements. The name is ' ' (empty) .

ARRAYs

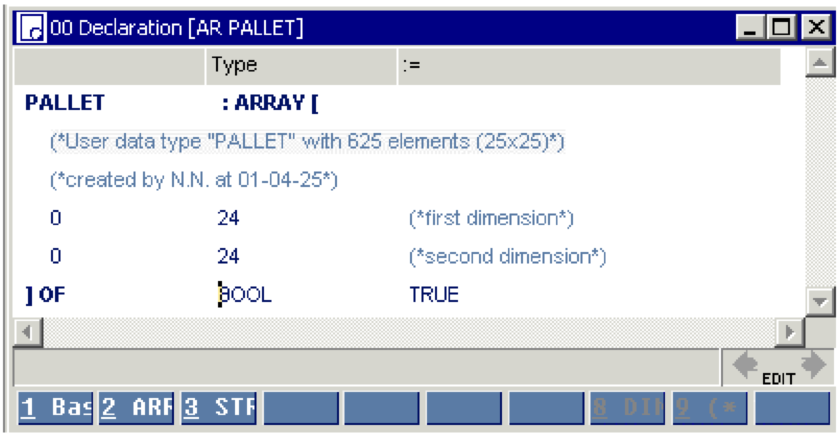
The elements of an array have a unique data type, which can be of the elementary type or can be a structure or even an array itself. The user can assign a unique initial value to all elements, if they are elementary. Structures and arrays have their own initial values.

The elements of an array are arranged dimensionally

(1 to 4 dimensions).

Each dimension starts with the element "0".

In addition to the declaration comment of the array, a comment can be given for each dimension.



AR palette.bmp

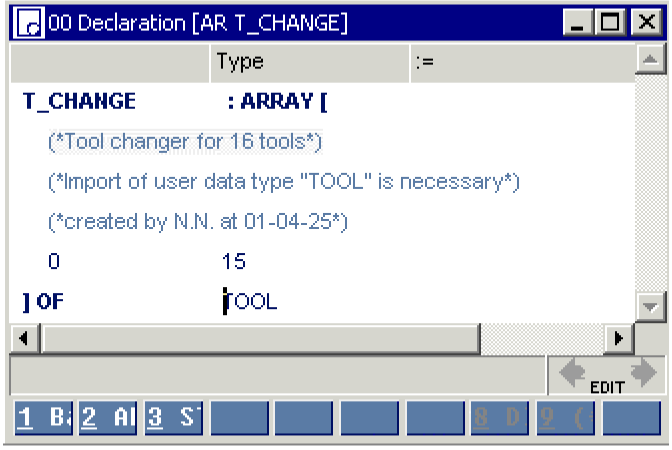
Fig. 11-34: Structure of a declaration illustrated by example of the "PALLET" elementary array

The declaration comment is added to the line specifying the name.

All dimensions start with the zero element.

The unique data type is BOOL

The user sets the value for each element to TRUE.



AR Wechsler.bmp

Fig. 11-35: Declaration illustrated by example of the "T_Changer" structured array

The declaration comment is added to the line specifying the name. All of the elements are Structures (STRUCT), comprising several elements themselves.

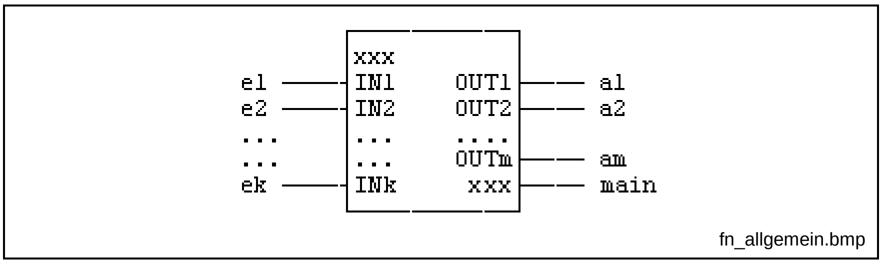
12 Functions in WinPCL

12.1 Functions, General Information

A function (FN) is a program organization unit which may have

- 1...k inputs,
- 1...m outputs,
- a main output and
- internal variables.

The topmost input of a function can be connected to a network, the following ones only to a variable or a constant.



xxx - Type name of the function (at top), main output
IN_i - Inputs of the function 1...k (IN_1: main input)
OUTj - Outputs of the function 1...m

Fig. 12-1: Function - general interface

The main output has the type name of the function, its type is identical to the type of the function. A network can be connected to the main output.

If necessary, a variable can be assigned to the 1...m outputs.

The call of a function with the same assignment at the inputs always supplies the same result at the output.

Computed intermediate results are rejected after the function value of the output has been determined.

A pre-initialization of variables is <u>not</u> possible.

A function can be used in any other program organization unit.

A distinction is made between standard and firmware functions as well as user-defined functions. Their interface is constant, even for further developed standard libraries and operating systems.

- Standard Functions: in accordance with EN 61131-3 (+ supplements)
- Firmware Functions: measured-value acquisition, communication of the PLC with the CNC, bus communication
- User Functions: written by the user himself.

Standard and firmware functions can be used, but <u>not modified</u>.

Their interface is constant, even for further developed standard libraries and operating systems.

12.2 Standard Functions

The standard functions and standard operations for the Rexroth control system comply with EN 61131-3.

Standard functions are available in all of the programming languages of the system and can be used directly, but not modified.

Functions for type and code conversion

- BYTE_BCD_TO_INT, BYTE_TO_CHAR, BYTE_TO_GRAY, BYTE_TO_INT, BYTE_TO_SINT, BYTE_TO_USINT
- CHAR_TO_BYTE
- DINT_TO_DWORD, DINT_TO_INT, DINT_TO_UDINT, DINT_TO_REAL, DINT_TO_TIME
- DWORD_TO_DINT, DWORD_TO_REAL
- GRAY_TO_BYTE
- INT_TO_BCD_WORD, INT_TO_BYTE, INT_TO_DINT, INT_TO_SINT, INT_TO_STRING, INT_TO_UINT, INT_TO_USINT, INT_TO_WORD
- REAL_TO_DINT, REAL_TO_STRING, REAL_TO_DWORD
- SINT_TO_BYTE, SINT_TO_INT
- STRING_TO_INT, STRING_TO_REAL
- UDINT_TO_DINT
- USINT_TO_BYTE, USINT_TO_INT
- UINT_TO_INT, UINT_TO_WORD
- TIME_TO_DINT
- WORD_BCD_TO_INT, WORD_TO_INT, WORD_TO_UINT

Numeric functions

- ABS_INT, SIGN_INT, as a supplement to the operations ADD, SUB, MUL, DIV, MOD
- SQRT_REAL, LN_REAL, LOG_REAL, EXP_REAL
- SIN_REAL, COS_REAL, TAN_REAL
- ASIN_REAL, ACOS_REAL, ATAN_REAL

Functions for time-to-integer conversion

- TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_MS

Functions for integer-to-time conversion

- MAKETIME

Bit string functions as a supplement to ':=', AND, OR, XOR

- SHL_BYTE, SHL_WORD, SHL_DWORD, SHL_LWORD
- SHR_BYTE, SHR_WORD, SHR_DWORD, SHR_LWORD
- ROL_BYTE, ROL_WORD, ROL_DWORD, ROL_LWORD
- ROR_BYTE, ROR_WORD, ROR_DWORD, ROR_LWORD
- CONCAT_BYTE, CONCAT_WORD
- HIGH_BYTE, LOW_BYTE, HIGH_WORD, LOW_WORD,

Character string functions

LEN, LEFT, RIGHT, MID, CONCAT_S, INSERT, DELETE, REPLACE, FIND, as a supplement to the STRING operations

Functions for Type and Code Conversion

For a part of the control technology the data type width was extended to 64 Bit. The corresponding functions are developed and introduced analogous to DINT, UDINT or REAL.

BYTE_TO_CHAR

The function BYTE_TO_CHAR converts any byte into a character of the extended ASCII character set (Standard Functions).

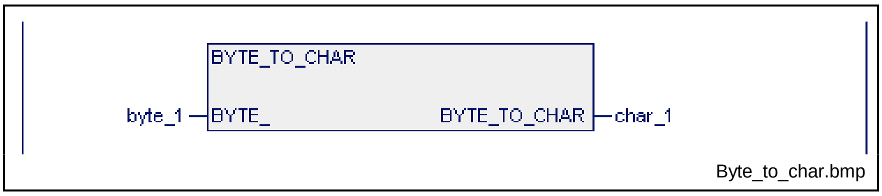


Fig. 12-2: Standard function BYTE_TO_CHAR

byte_1	char_1
...	...
0011 0001	'1'
...	...
0111 1010	'z'

Fig. 12-3: Value assignment BYTE_TO_CHAR

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

BYTE_TO_GRAY

The function BYTE_TO_GRAY deletes the high-order half-byte of the input variable and converts the low-order half-byte according to the table below (Standard Functions).

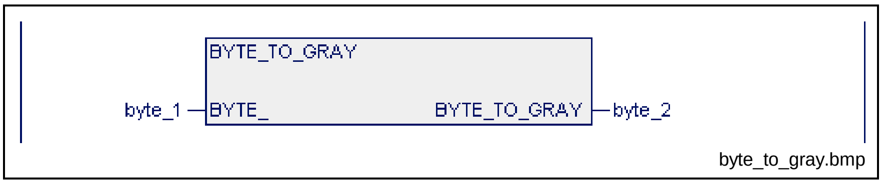


Fig. 12-4: Standard function BYTE_TO_GRAY

byte_1	byte_2
**** 0000	0000 0000
**** 0001	0000 0001
**** 0010	0000 0011
**** 0011	0000 0010
**** 0100	0000 0110
**** 0101	0000 0111
**** 0110	0000 0101
**** 0111	0000 0100
**** 1000	0000 1100
**** 1001	0000 1101
**** 1010	0000 1111
**** 1011	0000 1110
**** 1100	0000 1010
**** 1101	0000 1011
**** 1110	0000 1001
**** 1111	0000 1000

Fig. 12-5: Value assignment BYTE_TO_GRAY

****: An assignment unequal to 0000 results in an error message:

S#ErrorFlg: 1, S#ErrorNr: 4, S#ErrorTyp: -50

BYTE_TO_INT

The function BYTE_TO_INT generates an INT number from a byte (Standard Functions).

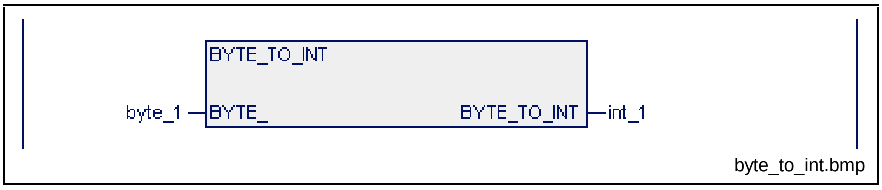


Fig. 12-6: Standard function BYTE_TO_INT

byte_1	int_1
0000 0000	0
0000 0001	1
....	....
1111 1111	255

Fig. 12-7: Value assignment BYTE_TO_INT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

BYTE_TO_SINT

The function BYTE_TO_SINT generates a SINT number from a byte (Standard Functions).

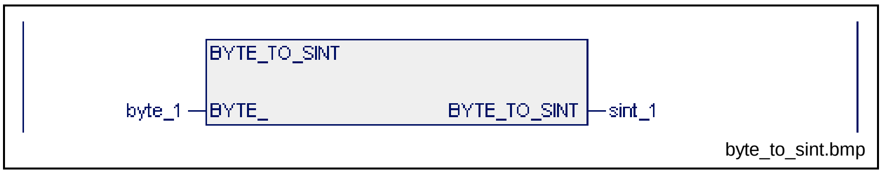


Fig. 12-8: Standard function BYTE_TO_SINT

byte_1	sint_1
0000 0000	0
0000 0000	1
...	...
0111 1111	127
1000 0000	-128
...	...
1111 1111	-1

Fig. 12-9: Value assignment BYTE_TO_SINT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

BYTE_TO_USINT

The function BYTE_TO_USINT generates a USINT number from a byte (Standard Functions).

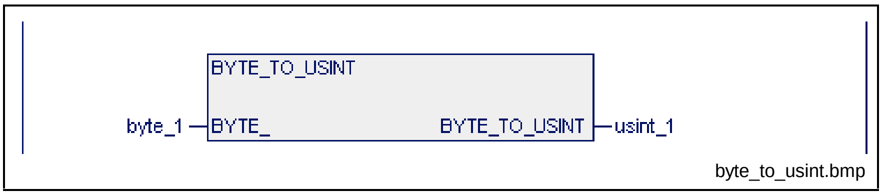


Fig. 12-10: Standard function BYTE_TO_USINT

byte_1	usint_1
0000 0000	0
0000 0001	1
...	...
1111 1111	255

Fig. 12-11: Value assignment BYTE_TO_USINT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

BYTE_BCD_TO_INT

The function BYTE_BCD_TO_INT converts a BCD-coded word into an INT number (Standard Functions).

In this conversion the half-bytes are converted separately and the result is overlaid.

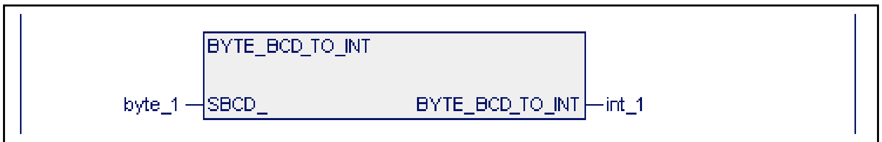


Fig. 12-12: Standard function BYTE_BCD_TO_INT

byte_1	int_1
0000 0000	0
0000 0001	1
...	...
0000 1001	9
0000 1010	Invalid
...	...
0000 1111	Invalid
0001 0000	10
...	...
1001 1001	99
and further	Invalid

Fig. 12-13: Value assignment BYTE_BCD_TO_INT

The result is invalid if one of the half-bytes has one of the following assignments:

1010, 1011, 1100, 1101, 1110, 1111.

Error message

S#ErrorFlg: 1, S#ErrorNr: 4, S#ErrorTyp: -51

CHAR_TO_BYTE

The function CHAR_TO_BYTE converts any character of the extended ASCII character set into a byte (Standard Functions).

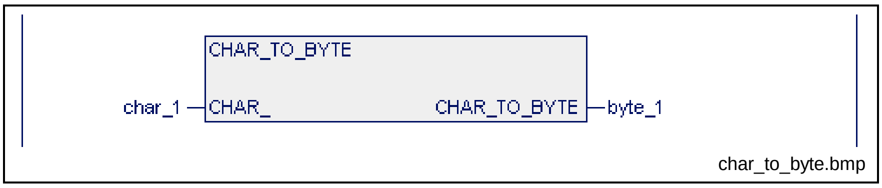


Fig. 12-14: Standard function CHAR_TO_BYTE

char_1	byte_1
...	...
’1’	0011 0001
...	...
’z’	0111 1010

Fig. 12-15: Value assignment CHAR_TO_BYTE

Note: Any character can be provided in the manner ’$00’ to ’$FF’. How to represent characters see also section >>Character String Literals<< in chapter "Data Types in WinPCL".

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

DINT_TO_DWORD

The function DINT_TO_DWORD generates a DWORD from a DINT number (Standard Functions).

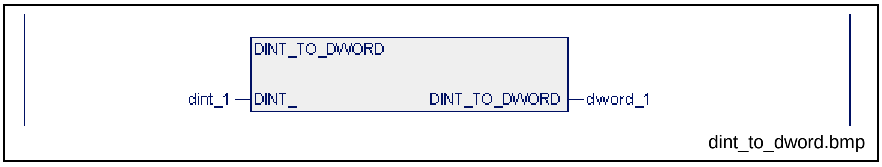


Fig. 12-16: Standard function DINT_TO_DWORD

dint_1	dword_1
-2147483648	16#8000 0000
...	...
-1	16#FFFF FFFF
0	16#0000 0000
1	16#0000 0001
...	...
2147483647	16#7FFF FFFF

Fig. 12-17: Value assignment DINT_TO_DWORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

DINT_TO_INT

The function DINT_TO_INT generates an INT number from a DINT number (Standard Functions).

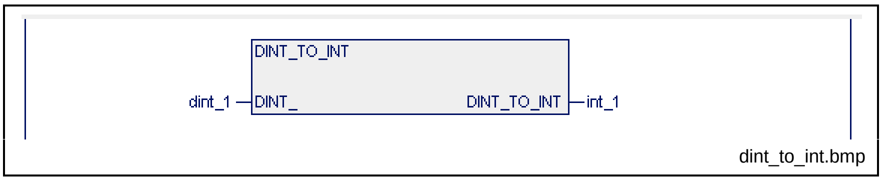


Fig. 12-18: Standard function DINT_TO_INT

dint_1	int_1	Error message
-2147483648	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-32769	Invalid	S#ErrorTyp: -154
-32768	-32768	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
32767	32767	S#ErrorTyp: 0
32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
2147483647	Invalid	S#ErrorTyp: -154

Fig. 12-19: Value assignment DINT_TO_INT

DINT_TO_UDINT

The function DINT_TO_UDINT converts a DINT number into a UDINT number. Negative input numbers will cause an error (Standard Functions).

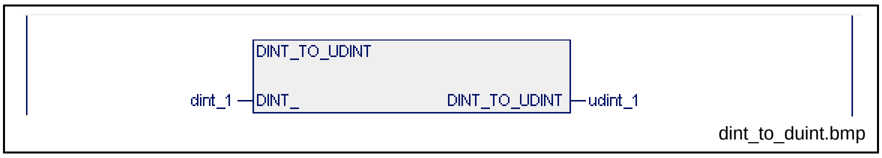


Fig. 12-20: Standard function DINT_TO_UDINT

dint_1	udint_1	Error message
-2147483648	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-1	Invalid	S#ErrorTyp: -171
0	0	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
2147483647	2147483647	S#ErrorTyp: 0

Fig. 12-21: Value assignment DINT_TO_UDINT

DINT_TO_REAL

The function DINT_TO_REAL converts the data type DINT into REAL (Standard Functions).

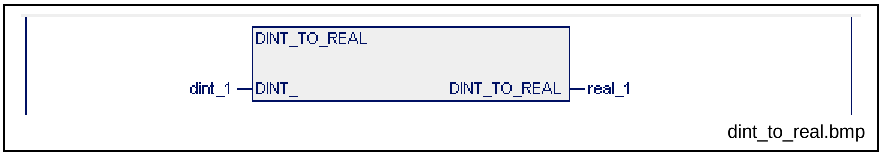


Fig. 12-22: Standard function DINT_TO_REAL

dint_1	real_1
-2147483648	-2147483648.0
...	...
0	0.0
...	...
2147483647	2147483647.0

Fig. 12-23: Value assignment DINT_TO_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Note: The sign ”+” is generally not indicated.
The numerical notation is optimized after maximum resolution.

DINT_TO_TIME

The function DINT_TO_TIME converts a double INTEGER value with the millisecond unit into a time value (Standard Functions).

Odd values for the indication of milliseconds are rounded up to the next even value.

(INTEGER for milliseconds 1, time value TIME T#2ms)

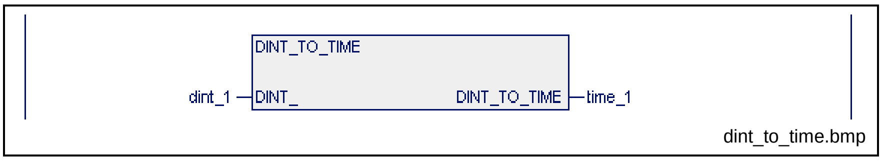


Fig. 12-24: Standard function DINT_TO_TIME

dint_1	time_1	Error message
-2147483648	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-1	Invalid	S#ErrorTyp: -156
0	0 ms	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
2073599998	23d23h59m59s998ms	S#ErrorTyp: 0
2073599999	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
2147483647	Invalid	S#ErrorTyp: -156

Fig. 12-25: Value assignment DINT_TO_TIME

DWORD_TO_DINT

The function DWORD_TO_DINT generates a DINT number from a DWORD (Standard Functions).

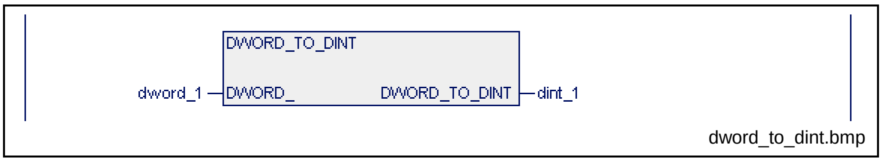


Fig. 12-26: Standard function DWORD_TO_REAL

dword_1	dint_1
16# 8000 0000	-2147483648
...	...
16# FFFF FFFF	-1
16# 0000 0000	0
16# 7FFF FFFF	2147483647

Fig. 12-27: Value assignment DWORD_TO_DINT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

DWORD_TO_REAL

The function DWORD_TO_REAL interprets the bit pattern of a DWORD as REAL number (Standard Functions).

dword_to_real.bmp

Fig. 12-28: Standard function DWORD_TO_REAL

dword_1	real_1	Error message
16# FFFF FFFF...16#FF7F FFFE	Invalid	S#ErrorFlg: 1 S#ErrorNr : 2 S#ErrorTyp: -237
16#FF7F FFFD...16#80800005	Valid number	
16#80800004 ... 16#80000001	Invalid	S#ErrorFlg: 1 S#ErrorNr : 3 S#ErrorTyp: -237
16#80000000, 16#00000000	0.000	
16#00000001...16#00800004	Invalid	S#ErrorFlg: 1 S#ErrorNr : 3 S#ErrorTyp: -237
16#00800005...16#7F7FFFFD	Valid number	
16#7F7FFFFE...16#7FFFFFFF	Invalid	S#ErrorFlg: 1 S#ErrorNr : 2 S#ErrorTyp: -237

Fig. 12-29: Value assignment DWORD_TO_REAL

Note: The sign "+" is generally not indicated.
The numerical notation is optimized after maximum resolution.

GRAY_TO_BYTE

The function GRAY_TO_BYTE deletes the high-order half-byte of the input variable and converts the low-order half-bate according to the table below (Standard Functions).

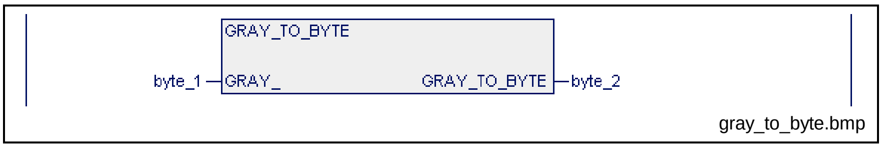


Fig. 12-30: Standard function GRAY_TO_BYTE

byte_1	byte_2
**** 0000	0000 0000
**** 0001	0000 0001
**** 0010	0000 0011
**** 0011	0000 0010
**** 0100	0000 0111
**** 0101	0000 0110
**** 0110	0000 0100
**** 0111	0000 0101
**** 1000	0000 1111
**** 1001	0000 1110
**** 1010	0000 1100
**** 1011	0000 1101
**** 1100	0000 1000
**** 1101	0000 1001
**** 1110	0000 1011
**** 1111	0000 1010

Fig. 12-31: Value assignment GRAY_TO_BYTE

****: An assignment unequal to 0000 results in an error message:

S#ErrorFlg: 1, S#ErrorNr: 4, S#ErrorTyp: -49

INT_TO_BCD_WORD

The function INT_TO_BCD_WORD generates a BCD-coded word from an INT number (Standard Functions).

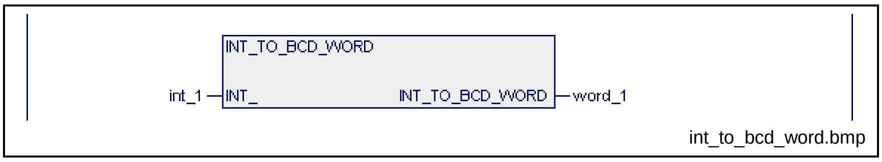


Fig. 12-32: Standard function INT_TO_BCD_WORD

Int_1	word_1	Error message
-32768	Invalid	S#ErrorFlg: 1
...	....	S#ErrorNr: 3
-1	Invalid	S#ErrorTyp: -57
0	0000 0000 0000 0000	S#ErrorFlg: 0
1	0000 0000 0000 0001	S#ErrorNr: 0
...	....	S#ErrorTyp: 0
9999	1001 1001 1001 1001	
10000	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
32767	Invalid	S#ErrorTyp: -57

Fig. 12-33: Value assignment INT_TO_BCD_WORD

INT_TO_BYTE

The function INT_TO_BYTE generates a byte from an INT number (Standard Functions).

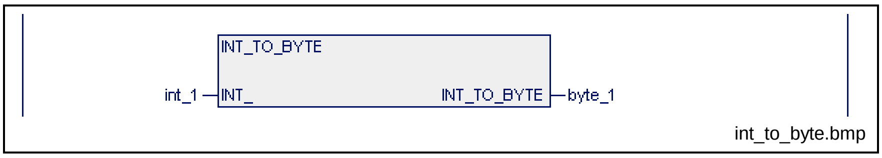


Fig. 12-34: Standard function INT_TO_BYTE

int_1	byte_1	Error message
-32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 4
-1	Invalid	S#ErrorTyp: -55
0	0000 0000	S#ErrorFlg: 0
1	0000 0001	S#ErrorNr: 0
...	....	S#ErrorTyp: 0
255	1111 1111	
256	Invalid	S#ErrorFlg: 1
....	...	S#ErrorNr: 4
32767	Invalid	S#ErrorTyp: -55

Fig. 12-35: Value assignment INT_BYTE

INT_TO_DINT

The function INT_TO_DINT generates a DINT number from an INT number (Standard Functions).

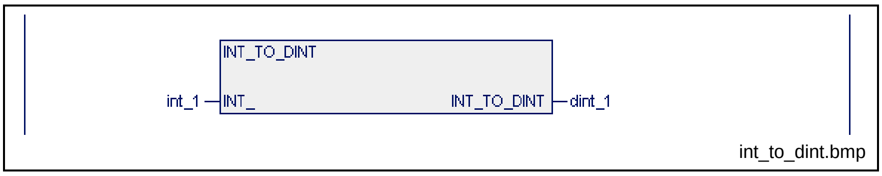


Fig. 12-36: Standard function INT_TO_DINT

int_1	dint_1
-32768	-32768
...	...
32767	32767

Fig. 12-37: Value assignment INT_TO_DINT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

INT_TO_SINT

The function INT_TO_SINT reduces an INT number to a SINT number (Standard Functions).

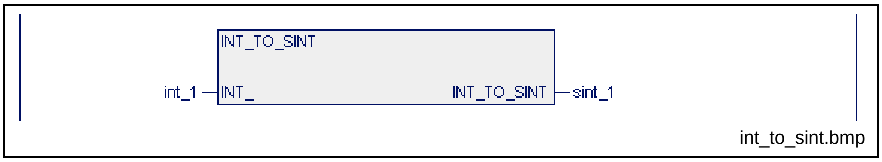


Fig. 12-38: Standard function INT_TO_SINT

int_1	sint_1	Error message
-32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-129	Invalid	S#ErrorTyp: -229
-128	-128	S#ErrorFlg: 0
0	0	S#ErrorNr: 0
127	127	S#ErrorTyp: 0
128	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
32767	Invalid	S#ErrorTyp: -229

Fig. 12-39: Value assignment INT_TO_SINT

INT_TO_STRING

The function INT_TO_STRING converts an INT number into a character string (STRING) (Standard Functions).

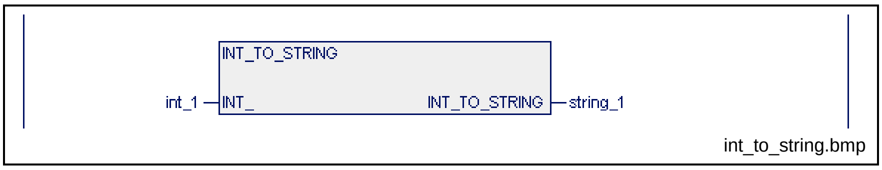


Fig. 12-40: Standard function INT_TO_STRING

int_1	string_1
-32768	'-32768'
0	'0'
32767	'32767'

Fig. 12-41: Value assignment INT_TO_STRING

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

INT_TO_UINT

The function INT_TO_UINT converts an INT number into an unsigned UINT number (Standard Functions).

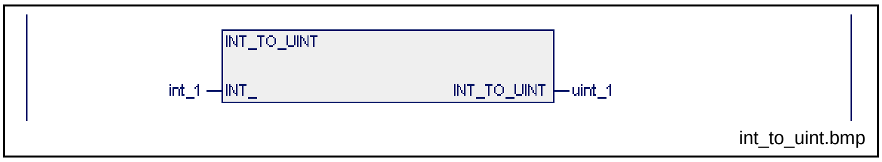


Fig. 12-42: Standard function INT_TO_UINT

int_1	uint_1	Error message
-32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-1	Invalid	S#ErrorTyp: -233
0	0	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
32767	32767	S#ErrorTyp: 0

Fig. 12-43: Value assignment INT_TO_UINT

INT_TO_USINT

The function INT_TO_USINT generates a USINT number from an INT number (Standard Functions).

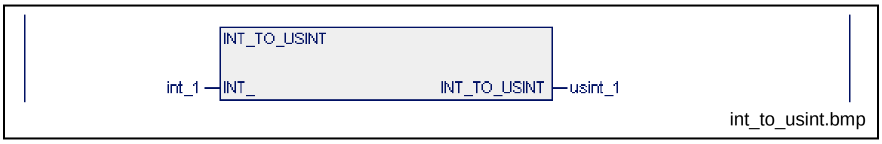


Fig. 12-44: Standard function INT_TO_USINT

int_1	usint_1	Error message
-32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
-1	Invalid	S#ErrorTyp: -59
0	0	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
255	255	S#ErrorTyp: 0
256	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
32767	Invalid	S#ErrorTyp: -59

Fig. 12-45: Value assignment INT_TO_USINT

INT_TO_WORD

The function INT_TO_WORD generates a word from an INT number (Standard Functions).

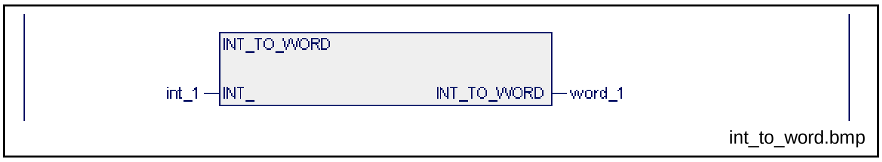


Fig. 12-46: Standard function INT_TO_WORD

int_1	word_1
-32768	1000 0000 0000 0000
-32767	1000 0000 0000 0001
....	....
-1	1111 1111 1111 1111
0	0000 0000 0000 0000
1	0000 0000 0000 0001
....	....
32767	0111 1111 1111 1111

Fig. 12-47: Value assignment INT_TO_WORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

REAL_TO_DINT

If possible, the function REAL_TO_DINT converts a REAL number into a DINT number (Standard Functions).

Also see the description under DWORD_TO_REAL.

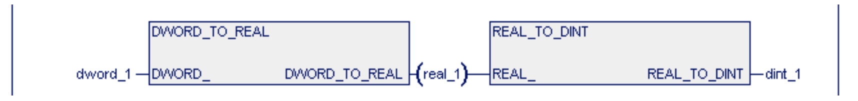


Fig. 12-48: Standard function REAL_TO_DINT

dword_1	real_1	dint_1
16#FFFFFFFF..16#FF7FFFFE	invalid	invalid
16#FF7FFFFD..16#CF000001	<-2147483648	invalid, (TRUE/3/-166)
16#CF000001..16#80800005	-2147483648..0	-2147483648..0
16#80800004..16#80000001	invalid	invalid
16#80000000, 16#00000000	0.000	0
16#00000001..16#00800004	invalid	invalid
16#00800005..16#4EFFFFFF	0..2147483520	0..2147483520 (1
16#4F000000..16#7F7FFFFD	>2147483647	invalid (TRUE/2/-166)
16#7F7FFFFE..16#7FFFFFFF	invalid	invalid

Fig. 12-49: Value assignment REAL_TO_DINT

(TRUE/3/-166): S#ErrorFlg/S#ErrorNr/S#ErrorTyp

(TRUE/2/-166): S#ErrorFlg/S#ErrorNr/S#ErrorTyp

(1: due to rounding less than 2147483647

Note: Only seven-digit values (e.g. 1234567.00, 1234.567, 123.45670 E+4) can be entered as input values for the FN REAL_TO_DINT. However, the above-mentioned multi-digit values can be indicated as output of the function DWORD_TO_REAL.

If the '.' to indicate a REAL number is forgotten, '0' is automatically added.

REAL_TO_STRING

The function REAL_TO_STRING converts a REAL number into a text (Standard Functions).

Also see the description under DWORD_TO_REAL.

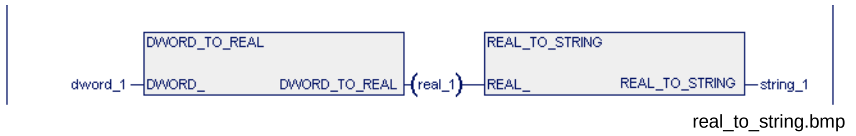


Fig. 12-50: Standard function REAL_TO_STRING

dword_1	real_1	string_1
16#FFFFFFFF..16#FF7FFFFE	invalid	invalid
16#FF7FFFFD..16#80800005	valid number	'-340282306073709660000000000000000000000.0...' ... '-0.0000000000000000'
16#80800004..16#80000001	invalid	invalid
16#80000000, 16#00000000	0.000	'0.0000000000000000'
16#00000001..16#00800004	invalid	invalid
16#00800005..16#7F7FFFFD	valid number	'0.0000000000000000' ... '340282306073709660000000000000000000000.0000000000000000'
16#7F7FFFFE..16#7FFFFFFF	invalid	invalid

Fig. 12-51: Value assignment REAL_TO_STRING

Errors can not occur in case of input of a valid REAL number.

S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

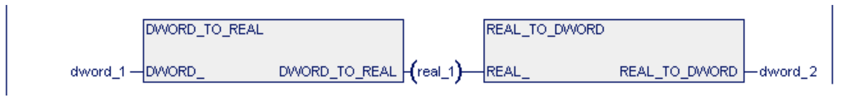
Note: Only seven-digit values (e.g. 1234567.00, 1234.567, 123.45670 E+4) can be entered as input values for the FN REAL_TO_STRING. However, the above-mentioned multi-digit values can be indicated as output of the function DWORD_TO_REAL.

If the '.' to indicate a REAL number is forgotten, '0' is automatically added.

REAL_TO_DWORD

The function REAL_TO_DWORD expresses the bit pattern of a REAL number as DWORD (Standard Functions).

Also see the description under DWORD_TO_REAL.



real_to_dword.bmp

Fig. 12-52: Standard function REAL_TO_DWORD

dword_1	**real_1**	**dword_2**
16#FFFFFFFF..16#FF7FFFFE	invalid	invalid
16#FF7FFFFD..16#80800005	valid number	16#FF7FFFFD..16#80800005
16#80800004..16#80000001	invalid	invalid
16#80000000, 16#00000000	0.000	16#00000000
16#00000001..16#00800004	invalid	invalid
16#00800005..16#7F7FFFFD	valid number	16#00800005..16#7F7FFFFD
16#7F7FFFFE..16#7FFFFFFF	invalid	invalid

Fig. 12-53: Value assignment REAL_TO_DWORD

Errors are not possible, if valid REAL numbers are applied to the input of the function:

S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Note: Only seven-digit values (e.g. 1234567.00, 1234.567, 123.45670 E+4) can be entered as input values for the FN REAL_TO_DWORD. However, the above-mentioned multi-digit values can be indicated as output of the function DWORD_TO_REAL.

If the '.' to indicate a REAL number is forgotten, '0' is automatically added.

SINT_TO_BYTE

The function SINT_TO_BYTE expresses a SINT number as a byte (Standard Functions).

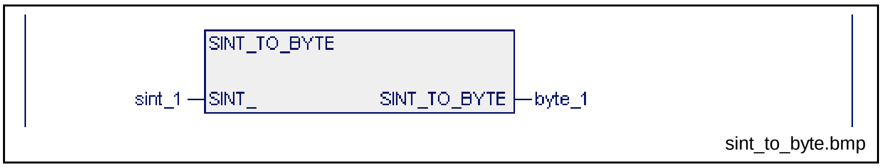


Fig. 12-54: Standard function SINT_TO_BYTE

sint_1	byte_1
0	0000 0000
1	0000 0001
...	....
127	0111 1111
-128	1000 0000
...	...
-1	1111 1111

Fig. 12-55: Value assignment SINT_TO_BYTE

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

SINT_TO_INT

The function SINT_TO_INT expresses a SINT number as an INT number (Standard Functions).

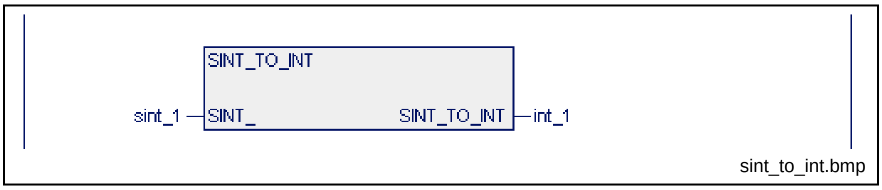


Fig. 12-56: Standard function SINT_TO_INT

sint_1	int_1
-128	-128
..	...
-1	-1
0	0
1	1
...	...
127	127

Fig. 12-57: Value assignment SINT_TO_INT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

STRING_TO_INT

The function STRING_TO_INT converts a character string (STRING) into an INT number (Standard Functions).

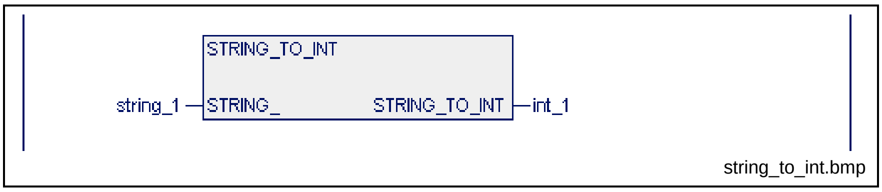


Fig. 12-58: Standard function STRING_TO_INT

string_1	int_1	Error message
'-999999'	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 3
'-32769'	Invalid	S#ErrorTyp: -141
'-32768'	-32768	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
'32767'	32767	S#ErrorTyp: 0
'32768'	Invalid	S#ErrorFlg: 1
....	.....	S#ErrorNr: 2
'9999999'	Invalid	S#ErrorTyp: -141
'paul'	Invalid	S#ErrorFlg: 1
		S#ErrorNr: 4
		S#ErrorTyp: -141

Fig. 12-59: Value assignment STRING_TO_INT

An empty STRING is converted into INT 0 without error message.

STRING_TO_REAL

The function STRING_TO_REAL converts a character string (STRING) into a REAL number (Standard Functions).

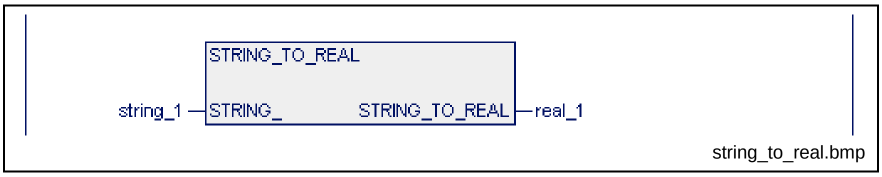


Fig. 12-60: Standard function STRING_TO_REAL

Rexroth Bosch Group

string_1	real_1	Error
'' (empty)	0.000	
'paul'	Invalid	(TRUE/1/-167)
'-3.40282E+48'	Invalid, expression out of range	(TRUE/2/-167)
'-3.40282E+38'	-34028210324961362000000000000000000000.000	
'-1.07374E+08'	-107374000.000	
'-1.40130E-45'	0.000	
0.00000E+00'	0.000	
'0.0'	0.0	
'1.40130E-45'	0.000	
'1.07374E+08'	107374000.000	
'3.40282E+38'	340282103249613620000000000000000000000.000	
'3.40282E+48'	Invalid, expression out of range	(TRUE/2/-167)

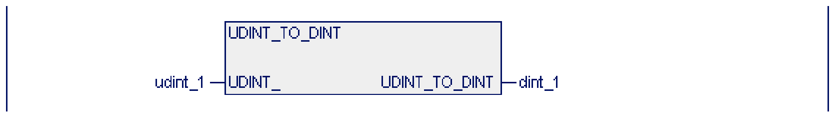
Fig. 12-61: Value assignment STRING_TO_REAL

(TRUE/1/-167): S#ErrorFlg/S#ErrorNr/S#ErrorTyp

Note: The sign "+" is generally not indicated.

UDINT_TO_DINT

The function UDINT_TO_DINT converts a UDINT number into a DINT number (Standard Functions).



udint_to_dint.bmp

Fig. 12-62: Standard function UDINT_TO_DINT

udint_1	dint_1	Error message
0	0	S#ErrorFlg: 0
...	....	S#ErrorNr: 0
2147483647	2147483647	S#ErrorTyp: 0
Error		
2147483648	Invalid	S#ErrorFlg: 1
...	....	S#ErrorNr: 2
4294967295	Invalid	S#ErrorTyp: -172

Fig. 12-63: Value assignment UDINT_TO_DINT

USINT_TO_BYTE

The function USINT_TO_BYTE generates a byte from a USINT number (Standard Functions).

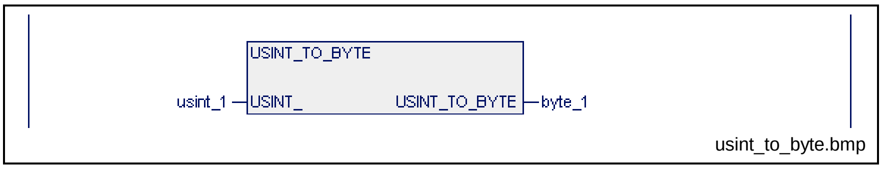


Fig. 12-64: Standard function USINT_TO_BYTE

usint_1	byte_1
0	0000 0000
1	0000 0001
...	...
255	1111 1111

Fig. 12-65: Value assignment USINT_TO_BYTE

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

USINT_TO_INT

The function USINT_TO_INT generates an INT number from a USINT number (Standard Functions).

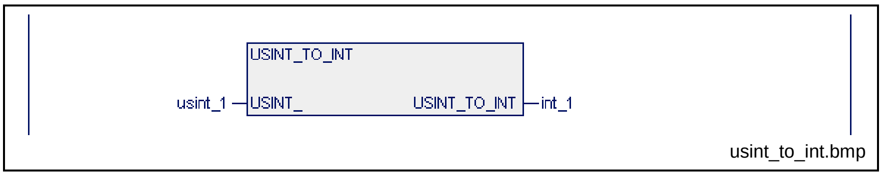


Fig. 12-66: Standard function USINT_TO_INT

usint_1	int_1
0	0
1	1
...	...
255	255

Fig. 12-67: Value assignment USINT_TO_INT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

UINT_TO_INT

The function UINT_TO_INT attempts to express a UINT number as an INT number (Standard Functions).

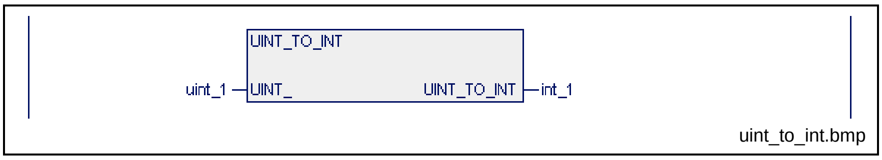


Fig. 12-68: Standard function UINT_TO_INT

uint_1	int_1	Error message
0	0	S#ErrorFlg: 0
1	1	S#ErrorNr: 0
...	...	S#ErrorTyp: 0
32767	32767	
Error		
32768	Invalid	S#ErrorFlg: 1
...	...	S#ErrorNr: 2
65535	Invalid	S#ErrorTyp: -232

Fig. 12-69: Value assignment UINT_TO_INT

UINT_TO_WORD

The function UINT_TO_WORD expresses a UINT number as a WORD (Standard Functions).

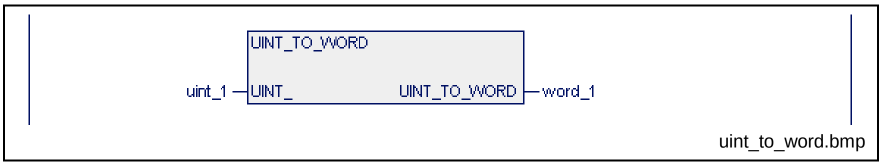


Fig. 12-70: Standard function UINT_TO_WORD

uint_1	word_1
0	0000 0000 0000 0000
1	0000 0000 0000 0001
.....	....
255	0000 0000 1111 1111
....	....
65535	1111 1111 1111 1111

Fig. 12-71: Value assignment UINT_TO_WORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

TIME_TO_DINT

The function TIME_TO_DINT converts a time value into a double INTEGER value of the milliseconds unit (Standard Functions).

time_to_dint.bmp

Fig. 12-72: Standard function TIME_TO_DINT

time_1	dint_1	Error message
0ms	0	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
23d23h59m59s999ms	2073599998	S#ErrorTyp: 0
Error		
> 23d23h59m59s999ms	Invalid	S#ErrorFlg: 1
		S#ErrorNr: 2
		S#ErrorTyp: -157

Fig. 12-73: Value assignment TIME_TO_DINT

WORD_BCD_TO_INT

The function WORD_BCD_TO_INT converts a BCD-coded word into an INT number (Standard Functions).

In this conversion the half-bytes are converted separately and the result is overlaid.

The result is invalid if one of the half-bytes has one of the following assignments:

1010, 1011, 1100, 1101, 1110, 1111.

Error message

S#ErrorFlg: 1, S#ErrorNr: 4, S#ErrorTyp: -52

word_bcd_to_int.bmp

Fig. 12-74: Standard function WORD_BCD_TO_INT

Rexroth Bosch Group

word_1	int_1
0000 0000 0000 0000	0
0000 0000 0000 0001	1
...	....
0000 0000 0000 1001	9
0000 0000 0000 1010	Invalid
...	...
0000 0000 0000 1111	Invalid
0000 0000 0001 0000	10
...	...
1001 1001 1001 1001	9999
And further	Invalid

Fig. 12-75: Value assignments WORD_BCD_TO_INT

WORD_TO_INT

The function WORD_TO_INT generates an INT number from a word (Standard Functions).

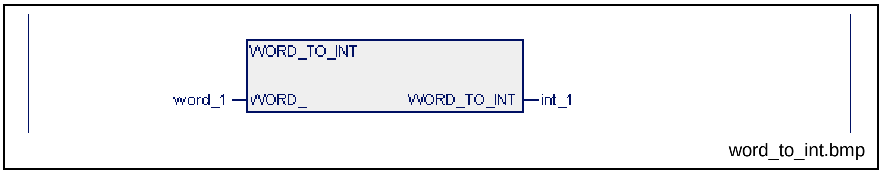


Fig. 12-76: Standard function WORD_TO_INT

word_1	int_1
0000 0000 0000 0000	0
0000 0000 0000 0001	1
...	....
0111 1111 1111 1111	32767
1000 0000 0000 0000	-32768
1000 0000 0000 0001	-32767
...	...
1111 1111 1111 1111	-1

Fig. 12-77: Value assignments WORD_TO_INT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.

WORD_TO_UINT

The function WORD_TO_UINT generates a UINT number from a word (Standard Functions).

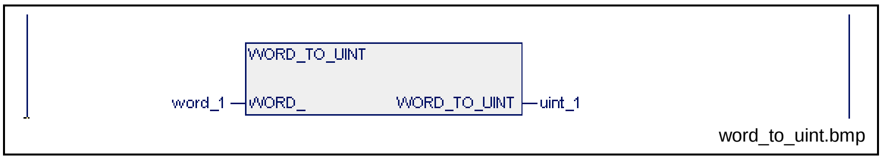


Fig. 12-78: Standard function WORD_TO_UINT

word_1	uint_1
0000 0000 0000 0000	0
0000 0000 0000 0001	1
...	...
0111 1111 1111 1111	32767
1000 0000 0000 0000	32768
...	...
1111 1111 1111 1111	65535

Fig. 12-79: Value assignments WORD_TO_UINT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Numeric Functions

Numeric functions are implemented as a supplement to the numerical operations ADD, SUB, MUL, DIV, MOD (Standard Functions).

ABS_INT

As result, the numerical function ABS_INT returns the value of the integer number applied to the input (Standard Functions).

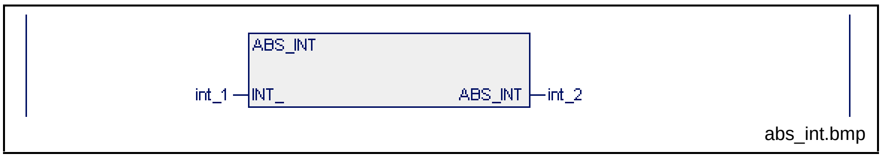


Fig. 12-80: Standard function ABS_INT

int_1	int_2	Error message
32767	32767	S#ErrorFlg: 0
...	...	S#ErrorNr: 0
-32767	32767	S#ErrorTyp: 0
Error		
-32768	Invalid	S#ErrorFlg: 1
		S#ErrorNr: 2
		S#ErrorTyp: -69

Fig. 12-81: Value assignments ABS_INT

SIGN_INT

As result, the numerical function SIGN_INT returns the sign of the integer number applied to the input (Standard Functions).

Note: Only available for INT!

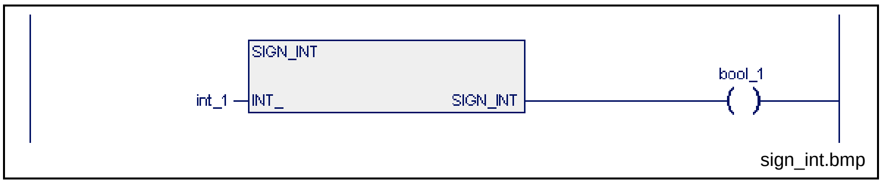


Fig. 12-82: Standard function SIGN_INT

int_1	bool_1
-35	0
0	1
+35	1

Fig. 12-83: Value assignments SIGN_INT

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

SQRT_REAL

The numerical function SQRT_REAL determines the square root of the REAL number applied to the input (Standard Functions).

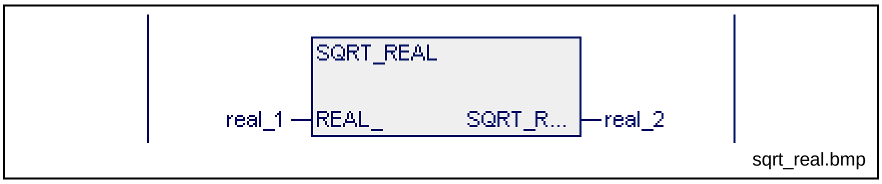


Fig. 12-84: Standard function SQRT_REAL

real_1	real_2	S#ErrorTyp	S#ErrorNr	S#ErrorFlg
-1.0	Invalid	-270	1	1
0.0	0.0	0	0	0
1.0	1.0	0	0	0
2.0	1.414	0	0	0
12.0	3.464	0	0	0
81.0	9.0	0	0	0

Fig. 12-85: Value assignments SQRT_REAL

LN_REAL

The numeric function LN_REAL determines the natural logarithm to the REAL number applied to the input (Standard Functions).

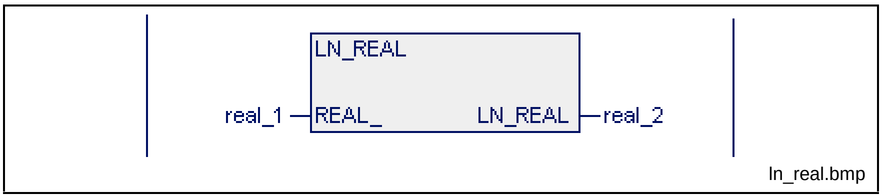


Fig. 12-86: Standard function LN_REAL

real_1	real_2	S#ErrorTyp	S#ErrorNr	S#ErrorFlg
-1.0	Invalid	-271	1	1
0.0	Invalid	-271	1	1
1.0	0.0	0	0	0
2.0	0.693	0	0	0
3.0	1.099	0	0	0
100.0	4.605	0	0	0

Fig. 12-87: Value assign-ments LN_RAL

LOG_REAL

The numeric function LOG_REAL determines the common logarithm to the REAL number applied to the input (Standard Functions).

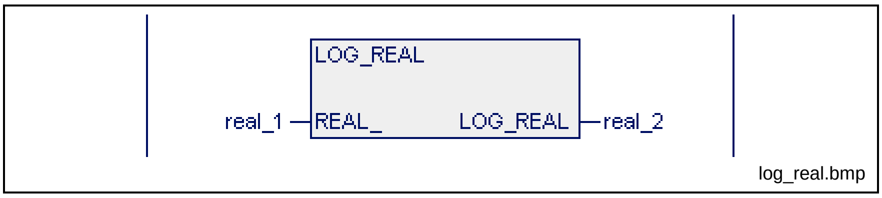


Fig. 12-88: Standard function LOG_REAL

real_1	real_2	S#ErrorTyp	S#ErrorNr	S#ErrorFlg
-1.0	0.0	-272	1	1
0.0	0.0	-272	1	1
1.0	0.0	0	0	0
2.0	0.301	0	0	0
3.0	0.477	0	0	0
100.0	2.0	0	0	0

Fig. 12-89: Value assignments LOG_REAL

EXP_REAL

The numeric function EXP_REAL determines the exponential value of the REAL number (Base "e") applied to the input (Standard Functions).

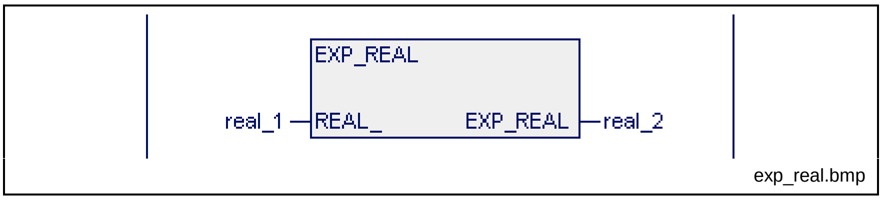


Fig. 12-90: Standard function EXP_REAL

real_1	real_2	S#ErrorTyp	S#ErrorNr	S#ErrorFlg
-1.0	0.368	0	0	0
0.0	1.0	0	0	0
1.0	2.718	0	0	0
2.0	7.389	0	0	0
3.0	20.086	0	0	0
100.0	Invalid	Invalid	Invalid	Invalid

Fig. 12-91: Value assignments EXP_REAL

Note: The range of real numbers of the result is exceeded for input values higher than 88. Check required!

SIN_REAL

The numeric function SIN_REAL determines the SIN to the REAL number applied to the input (input value in radian measure (Standard Functions).

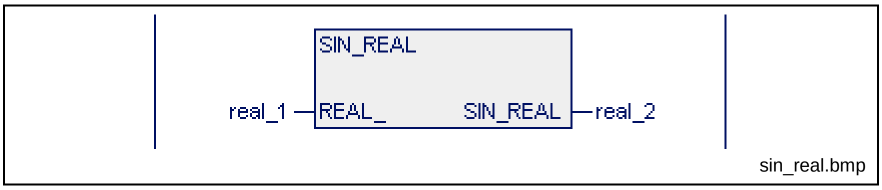


Fig. 12-92: Standard function SIN-REAL

	real_1	**real_2**
-30°	-0.524	-0.5
0°	0.0	0.0
30°	0.524	0.5
45°	0.785	0.707
60°	1.047	0.866
90°	1.571	1.0
120°	2.094	0.866

Fig. 12-93: Value assignments SIN_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

COS_REAL

The numeric function COS_REAL determines the COS to the REAL number applied to the input (input value in radian measure) (Standard Functions).

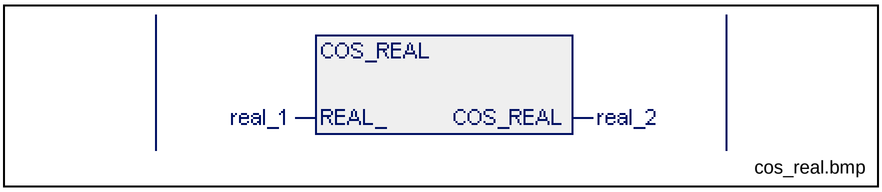


Fig. 12-94: Standard function COS_REAL

	real_1	**real_2**
-30°	-0.524	0.868
0°	0.0	1.0
30°	0.524	0.868
45°	0.785	0.707
60°	1.047	0.5
90°	1.571	0.0
120°	2.094	-0.5

Fig. 12-95: Value assignments COS_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

TAN_REAL

The numeric function TAN_REAL determines the TAN to the REAL number applied to the input (input value in radian measure) (Standard Functions).

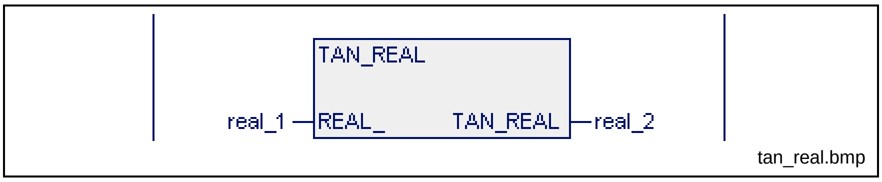


Fig. 12-96: Standard function TAN_REAL

	real_1	real_2
-30°	-0.524	-0.577
0°	0.0	0.0
30°	0.524	0.577
45°	0.785	1.0
60°	1.047	1.732
90°	1.571	very high
120°	2.094	-1.732

Fig. 12-97: Value assignments TAN_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

ASIN_REAL

The numeric function ASIN_REAL determines the main value to the REAL number applied to the input (Standard Functions).

Fig. 12-98: Standard function ASIN_REAL

real_1	real_2	
-0.5	-0.524	-30°
0.0	0.0	0°
0.5	0.524	30°
0.707	0.785	45°
0.866	1.047	60°
1.0	1.571	90°

Fig. 12-99: Value assignments ASIN_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

ACOS_REAL

The numeric function ACOS_REAL determines the main value to the REAL number applied to the input (Standard Functions).

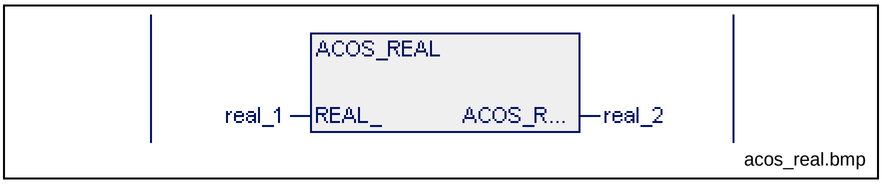


Fig. 12-100: Standard function ACOS_REAL

real_1	real_2	
1.0	0.0	0°
0.868	0.524	30°
0.707	0.785	45°
0.5	1.047	60°
0.0	1.571	90°
-0.5	2.094	120°

Fig. 12-101: Value assignments ACOS_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

ATAN_REAL

The numeric function ATAN_REAL determines the main value to the REAL number applied to the input (Standard Functions).

Note: Only available for REAL numbers!

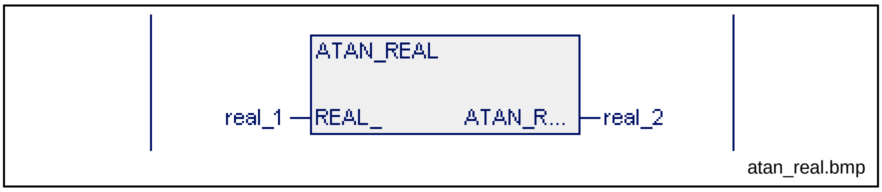


Fig. 12-102: Standard function ATAN_REAL

real_1	real_2	
-0.577	-0.524	-30°
0.0	0.0	0°
0.577	0.524	30°
1.0	0.785	45°
1.732	1.047	60°
Very great value	1.571	90°

Fig. 12-103: Value assignments ATAN_REAL

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Functions for Time-to-Integer Conversion

TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_MS

By means of the functions:

TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC, TIME_MS

a variable of the TIME data type is split into integer values (Standard Functions).

The function MAKETIME takes five integer values for day, hour, minute, second, and millisecond to generate a time value.

Conversion of TIME unit day to INTEGER

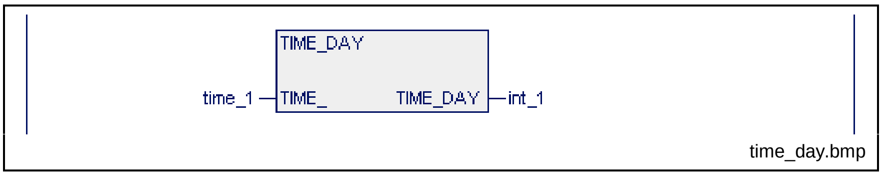


Fig. 12-104: Conversion of TIME unit day to INTEGER

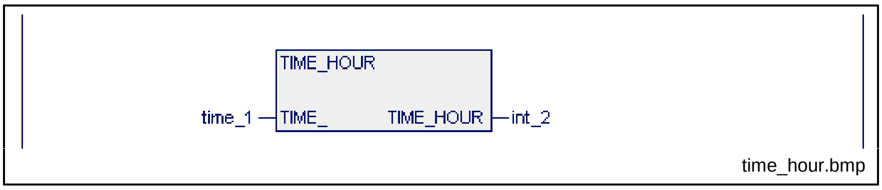


Fig. 12-105: Conversion of TIME unit hour to INTEGER

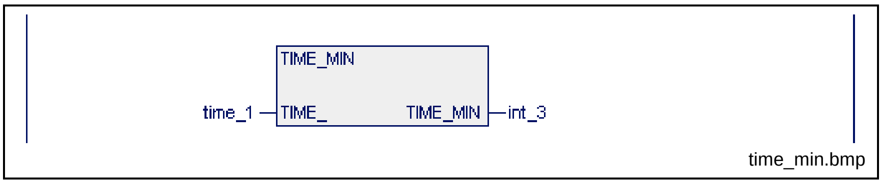


Fig. 12-106: Conversion of TIME unit minute to INTEGER

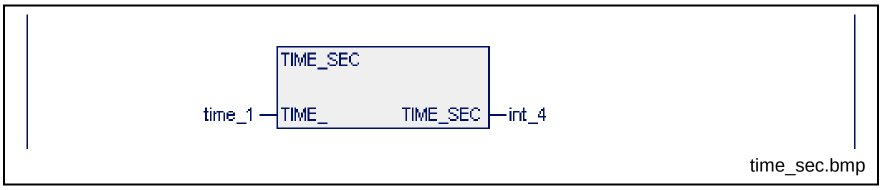


Fig. 12-107: Conversion of TIME unit second to INTEGER

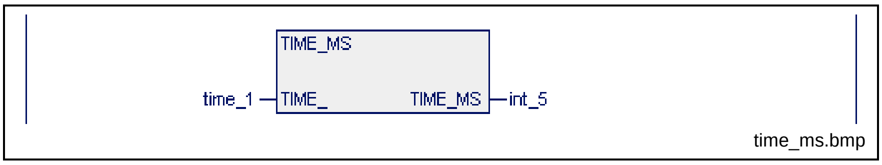


Fig. 12-108: Conversion of TIME unit millisecond to INTEGER

Name	Type	Comment
TIME_	TIME;	Time value to be converted
Function value	INT;	According to day, hour, minute, second, millisecond function

Operation of time-to-integer conversion

A time value is delivered to the functions TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC and TIME_MS. Depending on the function, the corresponding day, hour, minute, second or millisecond content is taken from the time value and provided as an integer value at the function output.

Error handling for time-to-integer conversion

The functions TIME_DAY, TIME_HOUR, TIME_MIN, TIME_SEC and TIME_MS do not generate any errors.

S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Examples of time-to-integer conversions

The execution time of a cycle is stored in the variable CYCLTIME. This time value is to be reduced to minutes and seconds and stored in the MCYCLMIN and MCYCLSEC variables.

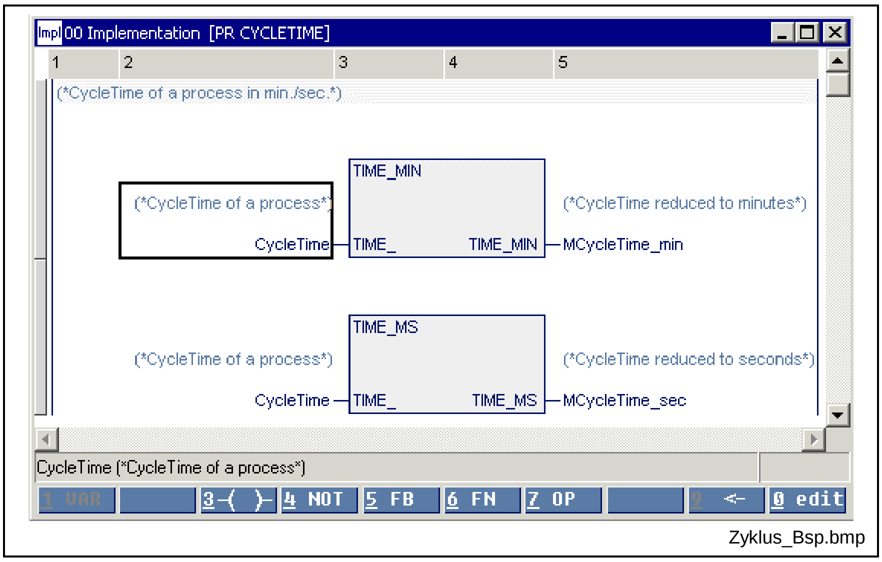


Fig. 12-109: Examples of time-to-integer conversions

CycleTime	McycleMin	McycleSec
T#2m15s150ms	2	15
T#1m59s820ms	1	59

INTEGER-to-TIME Conversion

MAKETIME

The function MAKETIME converts the integer values for day, hour, minute, second, and millisecond into a time value. The input values are summed up according to their unit. If input values are negative or if the maximum time value is exceeded, the function generates an error (Standard Functions).

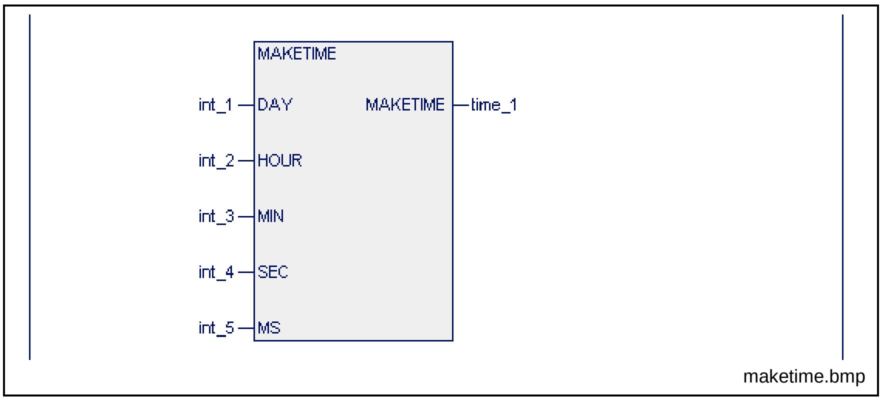


Fig. 12-110:Compose time value

Name	Type	Comment
DAY:	INT;	Day
HOUR:	INT;	Hour
MIN:	INT;	Minute
SEC:	INT;	Second
MS:	INT;	Millisecond
Function result	TIME	Converted time value

Function result

= MS + 1000 * (SEC + 60 * (MIN + 60 * (HOUR + 24 * DAY)

Error handling

As a result of programming errors, the MAKETIME function may be performed with integer values which are above the time range capable of being represented. In such a case, error handling reports the cause of the error.

Error type of the function blocks

MAKETIME conversion

MAKETIME: 210

Error numbers

Error No.	Meaning
1	Invalid input parameters The DAY, HOUR, MIN, SEC or MS inputs have negative values.
2	Range is exceeded The sum of the DAY, HOUR, MIN, SEC and MS inputs exceeds the maximum time value T#23d23h59m59s999ms.

Examples of integer-to-time conversions

The monitoring time for the processing duration of the individual stations runs in a data carrier. The information is stored as integer values for minutes and seconds. The data carrier is read and a time value is generated from the data for minutes and seconds. The time value serves as a preset value for the monitoring timer.

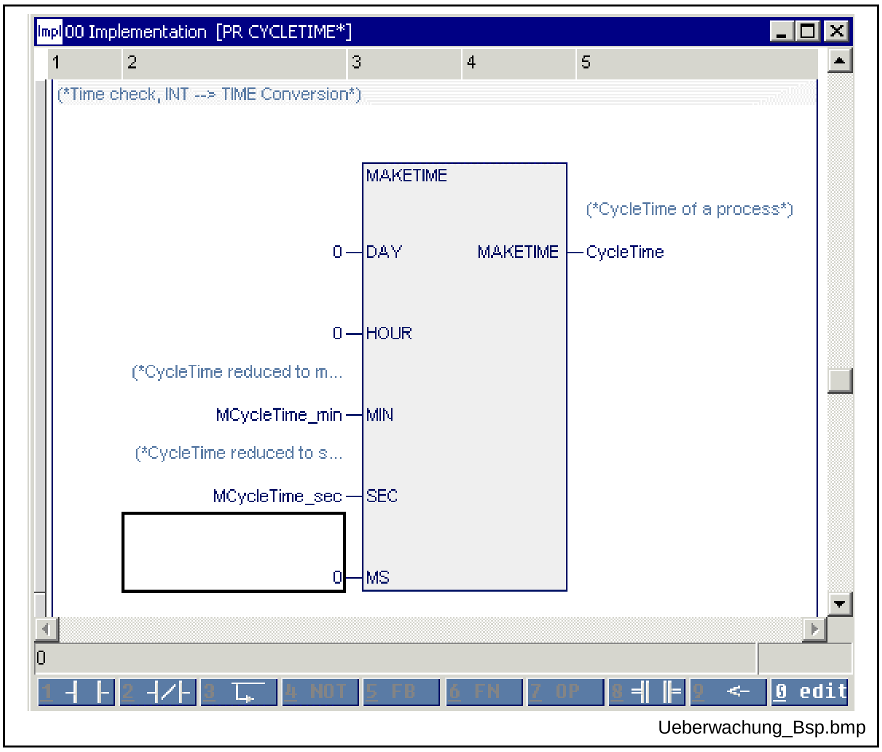


Fig. 12-111: Examples of integer-to-time conversions

MDATMIN	MDATSEC	MCYCLSEC
15	30	T#15m30s
0	125	T#2m5s
-1	0	T#0s -> S#ErrorFlg = TRUE S#ErrorNr = 1 S#ErrorTyp = -210

Bit String Functions

Bit string functions SHL_BYTE, SHL_WORD, SHL_DWORD serve as a supplement to the following operations:

- :=
- AND
- OR, XOR

SHL_BYTE, SHL_WORD, SHL_DWORD, SHL_LWORD

The bit string functions

- SHL_BYTE,
- SHR_WORD and
- SHL_DWORD
- SHL_LWORD, in preparation

permit that the bit string applied to the upper function input be shifted to the left bit by bit (Standard Functions).

The bit at the outer left is lost. The free bits are filled with 0.

The number of shift register clock pulses is defined by the second input (type INT).

No less than 0 and no more than (k-1) shift register clock pulses are permitted for a variable of k-bit width.

A negative number of shift register clock pulses or a number greater than (k-1) result in the command not being performed and in the error variables S#ErrorFlg, S#ErrorNr and S#ErrorTyp being set.

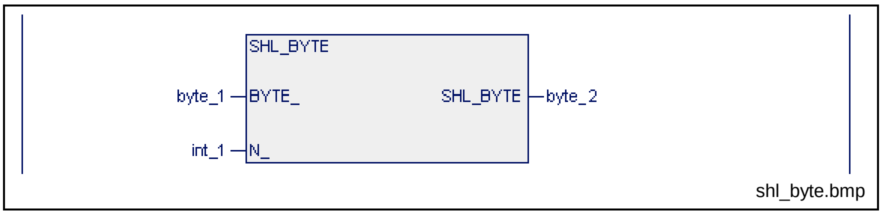


Fig. 12-112: Shifting a byte to the left bit by bit

Permitted are: 0 = int_1 = 7

byte_1	int_1	byte_2	Error message
00000000	0 ... 7	00000000	S#ErrorFlg: 0
...	0 ... 7	...	S#ErrorNr: 0
11001001	3	01001000	S#ErrorTyp: 0
11001001	4	10010000	
11111111	0 ... 7	...	
Error			
Any	Negative	Invalid	S#ErrorFlg: 1
Any	> 7	Invalid	S#ErrorNr: 1
			S#ErrorTyp: -70

Fig. 12-113: Value assignment SHL_BYTE

Further functions for longer bit strings are as follows:

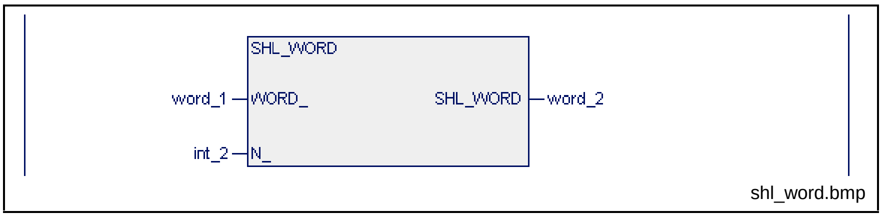


Fig. 12-114: Shifting a word to the left bit by bit

Permitted are: 0 = int_2 = 15

In case of error S#ErrorTyp: -71, S#ErrorNr:1, S#ErrorFlg: 1

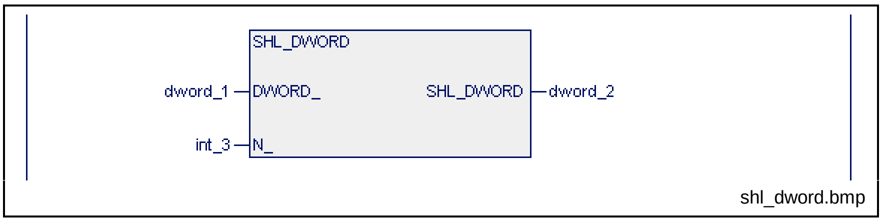


Fig. 12-115: Shifting a double word to the left bit by bit

Permitted are: 0 = int_2 = 31

In case of error S#ErrorTyp: -159, S#ErrorNr:1, S#ErrorFlg: 1

SHR_BYTE, SHR_WORD, SHR_DWORD, SHR_LWORD

The bit string functions:

- SHR_BYTE,
- SHR_WORD and
- SHR_DWORD
- SHR_LWORD, in preparation

permit that the bit string applied to the upper function input be shifted to the right bit by bit (Standard Functions).

The bit at the outer right is lost. The free bits are filled with 0.

The number of shift register clock pulses is defined by the second input, type INT.

No less than 0 and no more than (k-1) shift register clock pulses are permitted for a variable of k-bit width.

A negative number of shift register clock pulses or a number greater than (k-1) result in the command not being performed and in the error variables S#ErrorFlg, S#ErrorNr and S#ErrorTyp being set.

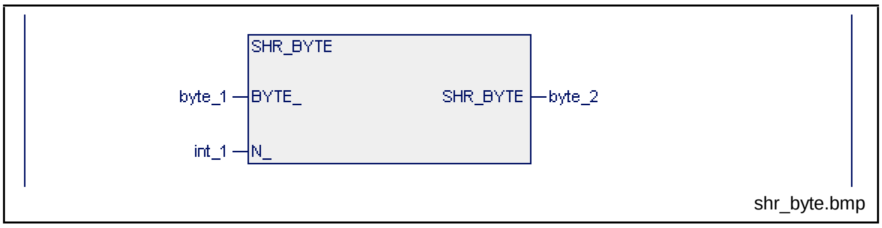


Fig. 12-116: Shifting a byte to the right bit by bit

Permitted are: 0 = int_1 = 7

byte_1	int_1	byte_2	Error message
00000000	0 ... 7	00000000	S#ErrorFlg: 0
...	0 ... 7	...	S#ErrorNr: 0
11001001	3	00011001	S#ErrorTyp: 0
11001001	4	00001100	
11111111	0 ... 7	...	
Error			
Any	Negative	Invalid	S#ErrorFlg: 1
Any	> 7	Invalid	S#ErrorNr: 1
			S#ErrorTyp: -72

Fig. 12-117: Value assignment SHL_BYTE

Further functions for longer bit strings are:

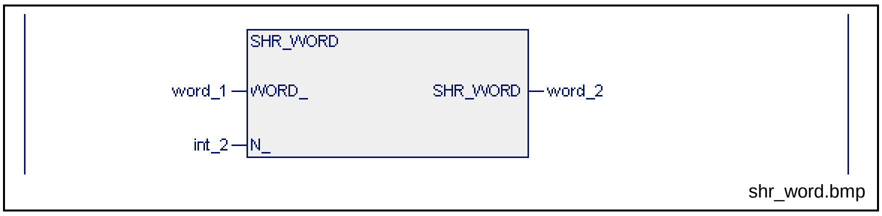


Fig. 12-118: Shifting a word to the right bit by bit

Permitted are: 0 = int_2 = 15

In case of error S#ErrorTyp: -73, S#ErrorNr:1, S#ErrorFlg: 1

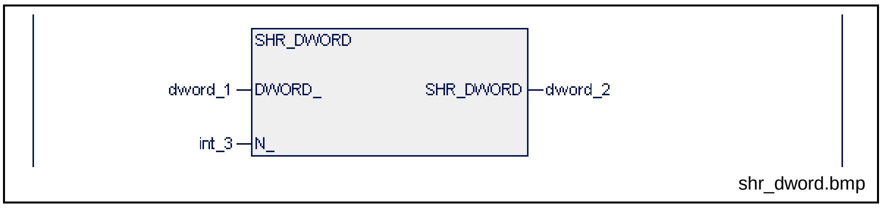


Fig. 12-119: Shifting a double word to the right bit by bit

Permitted are: 0 = int_2 = 31

In case of error S#ErrorTyp: -160, S#ErrorNr:1, S#ErrorFlg: 1

ROL_BYTE, ROL_WORD, ROL_DWORD, ROL_LWORD

The bit string functions:

- ROL_BYTE
- ROL_WORD and
- ROL_DWORD
- ROL_LWORD, in preparation

permit that the bit string applied to the upper function input be rotated to the left bit by bit (Standard Functions).

The number of shift register clock pulses is defined by the second input, type INT.

No less than 0 and no more than (k-1) shift register clock pulses are permitted for a variable of k-bit width.

A negative number of shift register clock pulses or a number greater than (k-1) result in the command not being performed and in the error variables S#ErrorFlg, S#ErrorNr and S#ErrorTyp being set.

The bit at the outer left rotates to the outer right.

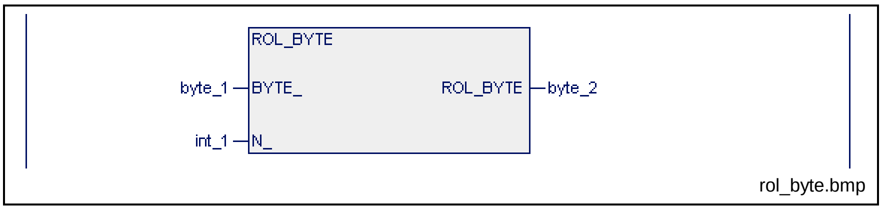


Fig. 12-120: Rotating a byte to the left bit by bit

Permitted are: 0 = int_1 = 7

byte_1	int_1	byte_2	Error message
00000000	0 ... 7	00000000	S#ErrorFlg: 0
...	0 ... 7	...	S#ErrorNr: 0
11001001	3	01001110	S#ErrorTyp: 0
11001001	4	10011100	
11111111	0 ... 7	...	
Error			
Any	Negative	Invalid	S#ErrorFlg: 1
Any	> 7	Invalid	S#ErrorNr: 1
			S#ErrorTyp: -74

Fig. 12-121: Value assignment ROL_BYTE

Further functions for longer bit strings are:

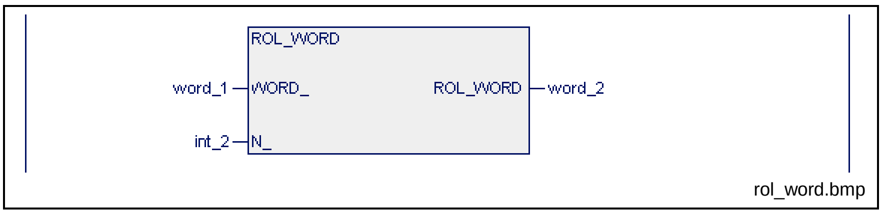


Fig. 12-122: Rotating a word to the left bit by bit

Permitted are: 0 = int_2 = 15

In case of error S#ErrorTyp: -75, S#ErrorNr:1, S#ErrorFlg: 1

Rexroth Bosch Group

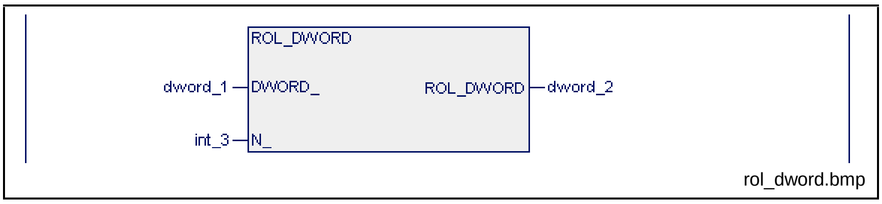


Fig. 12-123: Rotating a double word to the left bit by bit

Permitted are: 0 = int_2 = 31

In case of error S#ErrorTyp: -161, S#ErrorNr:1, S#ErrorFlg: 1

ROR_BYTE, ROR_WORD, ROR_DWORD, ROR_LWORD

The bit string functions

- ROR_BYTE
- ROR_WORD and
- ROR_DWORD
- ROR_LWORD, in preparation

permit that the bit string applied to the upper function input be rotated to the right bit by bit (Standard Functions).

The bit at the outer right rotates to the outer left.

The number of shift register clock pulses is defined by the second input, type INT.

No less than 0 and no more than (k-1) shift register clock pulses are permitted for a variable of k-bit width.

A negative number of shift register clock pulses or a number greater than (k-1) result in the command not being performed and in the error variables S#ErrorFlg, S#ErrorNr and S#ErrorTyp being set.

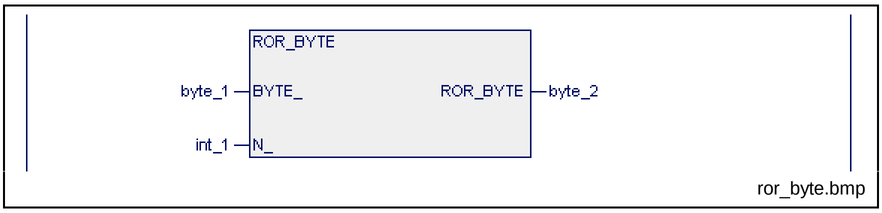


Fig. 12-124: Rotating a byte to the right bit by bit

Permitted are: 0 = int_1 = 7

byte_1	int_1	byte_2	Error message
00000000	0 ... 7	00000000	S#ErrorFlg: 0
...	0 ... 7	...	S#ErrorNr: 0
11001001	3	00111001	S#ErrorTyp: 0
11001001	4	10011100	
11111111	0 ... 7	...	
Error			
Any	Negative	Invalid	S#ErrorFlg: 1
Any	> 7	Invalid	S#ErrorNr: 1
			S#ErrorTyp: -76

Fig. 12-125: Value assignment ROR_BYTE

Further functions for longer bit strings are:

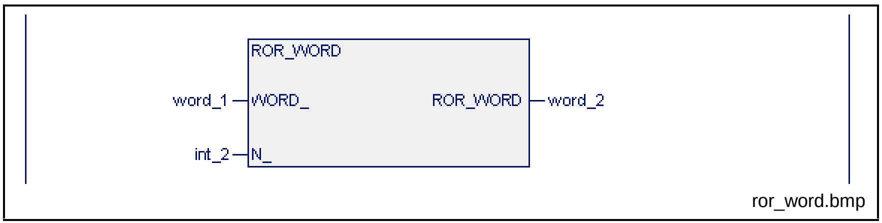


Fig. 12-126: Rotating a word to the right bit by bit

Permitted are: 0 = int_2 = 15

In case of error S#ErrorTyp: -77, S#ErrorNr:1, S#ErrorFlg: 1

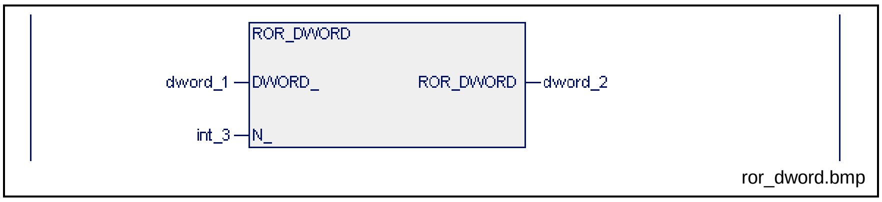


Fig. 12-127: Rotating a double word to the right bit by bit

Permitted are: 0 = int_2 = 31

In case of error S#ErrorTyp: -162, S#ErrorNr:1, S#ErrorFlg: 1

CONCAT_BYTE

The bit string function CONCAT_BYTE concatenates the two applied bytes to form a word.
The byte at the upper input becomes the high byte, the one at the lower input the low byte of the word (Standard Functions).

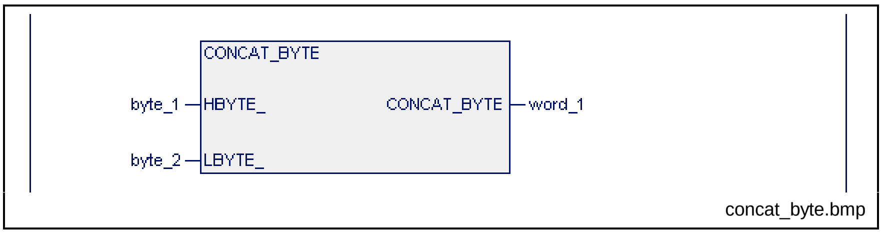


Fig. 12-128: Standard function CONCAT_BYTE

byte_1	**byte_2**	**word_1**
11001001	00110110	1100100100110110
16#C9	16#36	16#C936

Fig. 12-129: Value assignment CONCAT_BYTE

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

CONCAT_WORD

The bit string function CONCAT_WORD concatenates the two applied words to form a double word.
The word at the upper input becomes the high word, the one at the lower input the low word of the DWORD (Standard Functions).

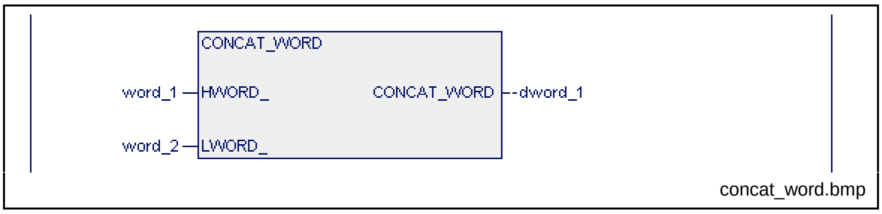


Fig. 12-130: Standard function CONCAT_WORD

word_1	word_2	dword_1
16#1122	16#3344	16#11223344

Fig. 12-131: Value assignment CONCAT_WORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

HIGH_BYTE

The bit string function HIGH_BYTE takes the high-order byte from the word applied to the input (Standard Functions).

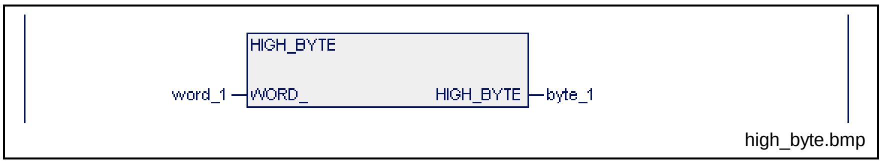


Fig. 12-132: Standard function HIGH_BYTE

word_1	byte_1
11001001 00110110	11001001
16#C936	16#C9

Fig. 12-133: Value assignment HIGH_BYTE

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

LOW_BYTE

The bit string function LOW_BYTE takes the low-order byte from the word applied to the input (Standard Functions).

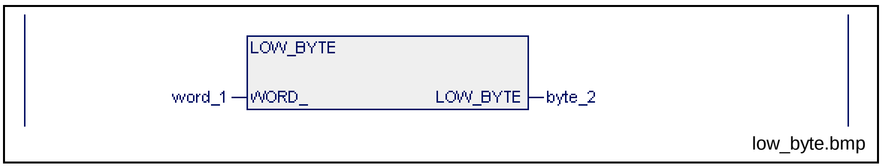


Fig. 12-134: Standard function LOW_BYTE

word_1	byte_2
11001001 00110110	00110110
16#C936	16#36

Fig. 12-135: Value assignment LOW_BYTE

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

HIGH_WORD

The bit string function HIGH_WORD takes the high-order word from the double word applied to the input (Standard Functions).

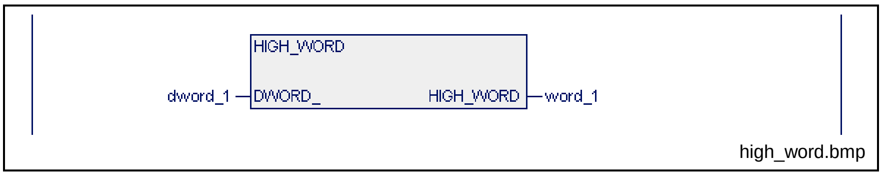


Fig. 12-136: Standard function HIGH_WORD

dword_1	word_1
16#12345678	16#1234

Fig. 12-137: Value assignment HIGH_WORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

LOW_WORD

The bit string function LOW_WORD takes the low-order word from the double word applied to the input (Standard Functions).

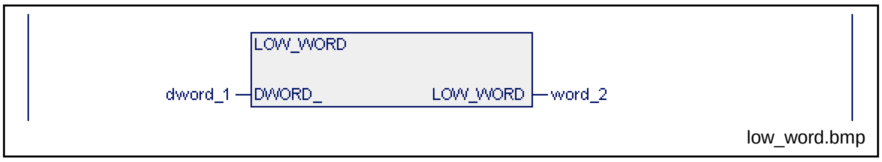


Fig. 12-138: Standard function LOW_WORD

dword_1	word_2
16#12345678	16#5678

Fig. 12-139: Value assignment LOW_WORD

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Character String Functions

For editing texts, the standard functions for character strings set forth below are implemented. They are provided as a supplement to the operations of the STRING data type (Standard Functions).

LEN	Determines the length of a character string
LEFT	Determines the leftmost L-characters.
RIGHT	Determines the rightmost L-characters.
MID	Expresses L-characters from TEXT as from position P.
INSERT	Inserts TEXT2 in TEXT1 as from position P.
DELETE	Deletes L-characters from TEXT as from position P.
REPLACE	Replaces L-characters from TEXT1 as from position P.
FIND	Seeks TEXT2 in TEXT1, indicates number.
CONCAT_S	Adds TEXT2 to TEXT1interruption-free.

The length of the character string can be between 0, empty character string, and 255.

If the size of the character string variables was limited, the processing starts from the left. Excess characters are rejected.

Indications of position or length at the inputs of the functions, type INT, result in an error message (S#ErrorFlg, S#ErrorNr, S#ErrorTyp) if the possible value is exceeded / fallen short or is in the negative number range.

LEN

The character string function LEN determines the length of a character string. An error message cannot be emitted (Standard Functions).

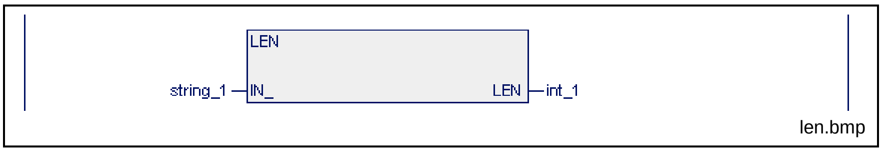


Fig. 12-140: Standard function LEN

string_1	**int_1**
’aBC	3
”	0
BC	2

Fig. 12-141: Value assignment LEN

Errors are not possible: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

LEFT

The character string function LEFT expresses the leftmost L_characters (Standard Functions).

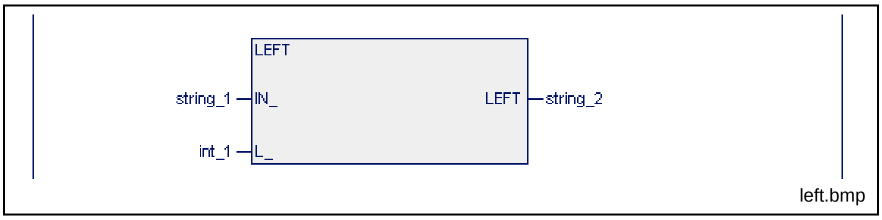


Fig. 12-142: Standard function LEFT

string_1	int_1	string_2	Error message
”	0	”	S#ErrorFlg: 0
’bcdef’	0	”	S#ErrorNr: 0
’bcdef’	2	’bc’	S#ErrorTyp: 0
’bcdef’	5	’bcdef’	
Error			
’bcdef’	6	Invalid, length exceeded	S#ErrorFlg: 1
”	1	Invalid, length exceeded	S#ErrorNr: 1
Any	< 0	Invalid, negative length	S#ErrorTyp: -143

Fig. 12-143: Value assignment LEFT

Limitation of the length of ‘string_2’ upon declaration:

Name	AT	TYPE	:=	Comment
string_2		STRING[2]		Results character string

Fig. 12-144: Declaration of string_2

string_1	int_1	string_2	Error message
’bcdef’	0	”	S#ErrorFlg: 0
’bcdef’	2	’bc’	S#ErrorNr: 0
’bcdef’	5	’bc’	S#ErrorTyp: 0

Fig. 12-145: Value assignment for results character string limited in length

RIGHT

The character string function RIGHT expresses the rightmost L_characters (Standard Functions).

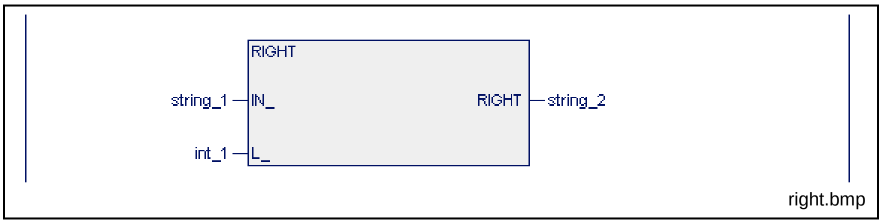


Fig. 12-146: Standard function RIGHT

string_1	**int_1**	**string_2**	**Error message**
”	0	”	S#ErrorFlg: 0
’bcdef’	0	”	S#ErrorNr: 0
’bcdef’	2	’ef’	S#ErrorTyp: 0
’bcdef’	5	’bcdef’	
Error			
’bcdef’	6	Invalid, length exceeded	S#ErrorFlg: 1
”	1	Invalid, length exceeded	S#ErrorNr: 1
Any	< 0	Invalid, negative length	S#ErrorTyp: -144

Fig. 12-147: Value assignment RIGHT

Limitation of the length of ‘string_2’ upon declaration:

Name	**AT**	**TYPE**	**:=**	**Comment**
string_2		STRING[2]		Results character string

Fig. 12-148: Declaration of string_2

string_1	**int_1**	**string_2**	**Error message**
’bcdef’	0	”	S#ErrorFlg: 0
’bcdef’	2	’ef’	S#ErrorNr: 0
’bcdef’	5	’bc’	S#ErrorTyp: 0

Fig. 12-149: Value assignment for results character string limited in length

Note: Length is always limited from the left!!!

MID

The character string function MID determines L_characters from position P_ to the right (Standard Functions).

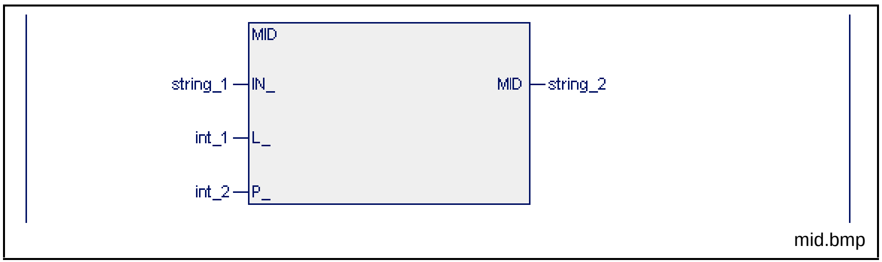


Fig. 12-150: Standard function MID

string_1	int_1	int_2	string_2	Error message
’bcdef’	0	1	”	S#ErrorFlg: 0
’bcdef’	2	1	’bc’	S#ErrorNr: 0
’bcdef’	5	1	’bcdef’	S#ErrorTyp: 0
’bcdef’	0	2	”	
’bcdef’	2	2	’cd’	
Error				
’bcdef’	6	1	Invalid, length exceeded	S#ErrorFlg: 1
’bcdef’	5	2	Invalid, length exceeded	S#ErrorNr: 1
’bcdef’	-1	*	Invalid, negative length	S#ErrorTyp: -145
’bcdef’	*	< 1	Invalid, position error	

Fig. 12-151: Value assignment MID

Note: Assigning the standard initialized variable int_2:=0 to the function results in an error!

Limitation of the length of ‘string_2’ upon declaration:

Name	AT	TYPE	:=	Comment
string_2		STRING[2]		Results character string

Fig. 12-152: Declaration of string_2

string_1	int_1	int_2	string_2	Error message
’bcdef’	0	1	”	S#ErrorFlg: 0
’bcdef’	2	1	’bc’	S#ErrorNr: 0
’bcdef’	5	1	’bc’	S#ErrorTyp: 0
’bcdef’	3	2	’cd’	
’bcdef’	4	2	’cd’	

Fig. 12-153: Value assignment for results character string limited in length

Note: Length is always limited from the left!!!

CONCAT_S

The character string function CONCAT_STRING permits the lower character string to be added to the upper one (Standard Functions).

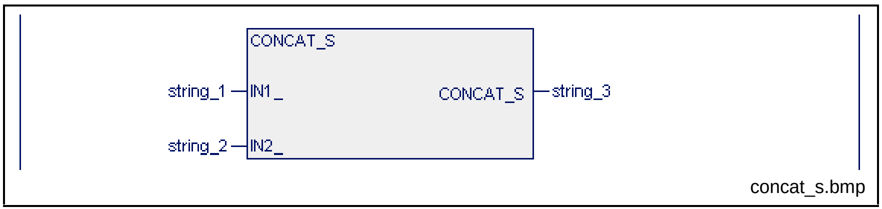


Fig. 12-154: Standard function CONCAT_S

string_1	string_2	string_3
"	"	"
'bcd'	"	'bcd'
'bcd'	'de'	'bcdde'

Fig. 12-155: Value assignment CONCAT_S

Limitation of the length of 'string_3' upon declaration:

Name	AT	TYPE	:=	Comment
string_3		STRING[4]		Results character string

Fig. 12-156: Declaration of string_3

string_1	string_2	string_3
"	"	"
'bcd'	"	'bcd'
'bcd'	'de'	'bcdd'

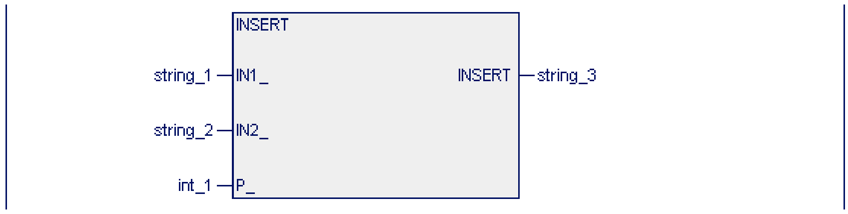
Fig. 12-157: Value assignment for results character string limited in length

Note: Length is always limited from the left!!!

Note: LEN(string_3) > 255: results in
S#ErrorFlg 1, S#ErrorNr 239, S#ErrorTyp -146
Other errors are not possible!

INSERT

The character string function INSERT permits the lower character string to be inserted after position P_ in the upper character string (Standard Functions).



insert.bmp

Fig. 12-158: Standard function INSERT

string_1	string_2	int_1	string_3	Error message
”	”	0	”	S#ErrorFlg: 0
”	’ef’	0	’ef’	S#ErrorNr: 0
’bcd’	’ef’	0	’efbcd’	S#ErrorTyp: 0
’bcd’	’ef’	1	’befcd’	
’bcd’	’ef’	2	’bcefd’	
’bcd’	’ef’	3	bcdef’	
Error				
*	*	-1	Invalid, position error	S#ErrorFlg: 1
”	’ef’	1	Invalid, length error	S#ErrorNr: 1
’bcd’	’ef’	4	Invalid, length error	S#ErrorTyp: -147

Fig. 12-159: Value assignment INSERT

Note: LEN(string_3) > 255 results in:
S#ErrorFlg 1, S#ErrorNr 239, S#ErrorTyp -147

Limitation of the length of ‘string_3’ upon declaration:

Name	AT	TYPE	:=	Comment
string_3		STRING[4]		Results character string

Fig. 12-160: Declaration of string_3

string_1	string_2	int_1	string_3	Error message
’bcd’	’de’	0	’debc’	S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Fig. 12-161: Value assignment for results character string limited in length

Note: Length is always limited from the left!!!

Note: LEN(string_3) > 255: results in
S#ErrorFlg 1, S#ErrorNr 239, S#ErrorTyp -147

DELETE

The character string function DELETE deletes L_characters to the right from and including position P_ (Standard Functions).

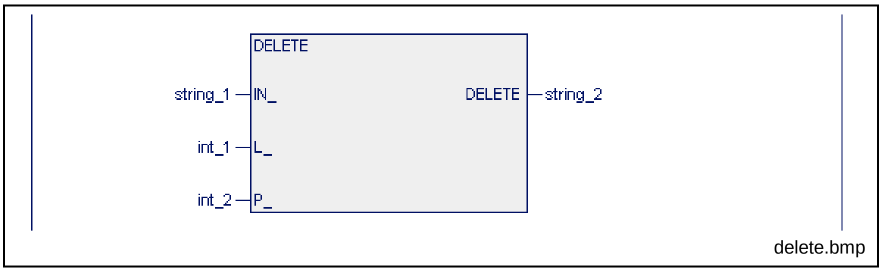


Fig. 12-162: Standard function DELETE

string_1	int_1	int_2	string_2	Error message
'bcdef'	0	1	'bcdef'	S#ErrorFlg: 0
'bcdef'	1	1	'cdef'	S#ErrorNr: 0
'bcdef'	2	1	'def'	S#ErrorTyp: 0
'bcdef'	4	1	'f'	
'bcdef'	5	1	''	
'bcdef'	0	2	'bcdef'	
'bcdef'	1	2	'bdef'	
'bcdef'	2	2	'bef'	
'bcdef'	4	2	'b'	
Error				
''	*	*	Invalid, nothing to delete	S#ErrorFlg: 1
*	-1	*	Invalid, length error	S#ErrorNr: 1
*	*	0	Invalid, position error	S#ErrorTyp: -148
'bcdef'	6	1	Invalid, length error	
'bcdef'	5	2	Invalid, length error	

Fig. 12-163: Value assignment DELETE

Note: Assigning the standard initialized variable int_2:=0 to the function results in an error!

Limitation of the length of 'string_3' upon declaration:

Name	AT	TYPE	:=	Comment
string_2		STRING[4]		Results character string

Fig. 12-164: Declaration of string_2

string_1	int_1	int_2	string_3	Error message
'bcdef'	2	1	'de'	S#ErrorFlg: 0
'bcdef'	4	1	'f'	S#ErrorNr: 0
'bcdef'	2	2	'be'	S#ErrorTyp: 0

Fig. 12-165: Value assignment for results character string limited in length

Note: Length is always limited from the left!!!

REPLACE

The character string function REPLACE causes the L_characters in the upper character string to be replaced by the lower character string, from position P (Standard Functions).

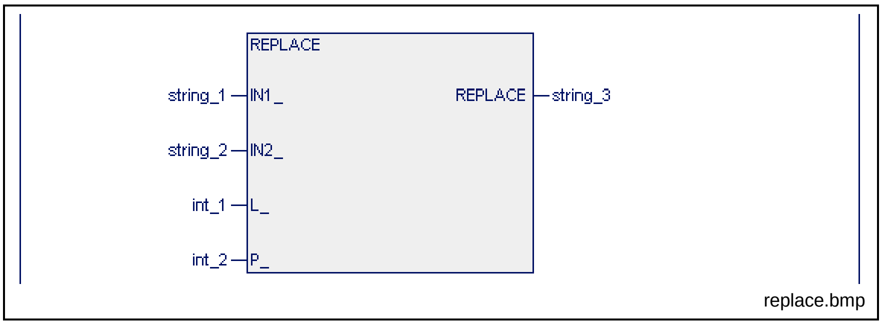


Fig. 12-166: Standard function REPLACE

string_1	string_2	int_1	int_2	string_3	Error message
’bcdef’	’xyz’	0	1	’xyzbcdef’	S#ErrorFlg: 0
’bcdef’	’xyz’	2	1	’xyzdef’	S#ErrorNr: 0
’bcdef’	’xyz’	4	1	’xyzf’	S#ErrorTyp: 0
’bcdef’	’xyz’	5	1	’xyz’	
’bcdef’	’xyz’	2	2	’bxyzef’	
’bcdef’	’xyz’	3	2	’bxyzf’	
’bcdef’	’xyz’	4	2	’bxyz’	
’bcdef’	’xyz’	0	5	’bcdefxyz’	
Error					
’’	*	*	*	Invalid, nothing to delete	S#ErrorFlg: 1
*	*	<0	*	Invalid, length error	S#ErrorNr: 1
*	*	*	<1	Invalid, position error	S#ErrorTyp: -149
’bcdef’	’xyz’	6	1	Invalid, length error	
’bcdef’	’xyz’	5	2	Invalid, length error	

Fig. 12-167: Value assignment REPLACE

Note: LEN(string_3) > 255 results in:
S#ErrorFlg 1, S#ErrorNr 239, S#ErrorTyp -149

FIND

The character string function FIND determines the position where the lower character string first begins in the upper one (Standard Functions).

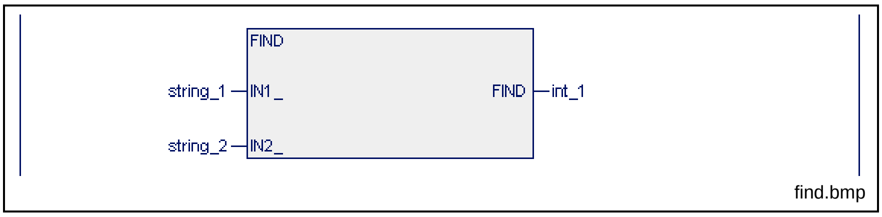


Fig. 12-168: Standard function FIND

string_1	string_2	int_1
”	”	1
’bcd’	”	0
”	’xy’	0
’bcdbcde’	’cd’	2

Fig. 12-169: Value assignment:

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

12.3 Firmware Functions

Firmware functions are intended to support the user. He can use them, but not alter them, as they are stored in the library of the programming system.

- Analog Module RMC12.2.-2E-1A, Functions
- PROFIBUS DP, Functions
- BT-Bus - Functions
- ASI-Bus, Functions, Function blocks and Data types
- INTERBUS, Function.

Analog Module RMC12.2.-2E-1A, Functions

This chapter provides an overview of the applications of the INTERBUS analog module RMC 12.2-2E-1A and the requisite firmware functions:

- Address assignment of the registers
- Setting the measuring ranges
- Voltage measurement VLT_MEAS up to ±10 V
- Current measurement AMP_MEAS up to ±20 mA
- Resistance measurement RES_MEAS up to 2000 Ω
- Temperature measurement TMP1MEAS from -100 °C up to +850 °C by means of Pt100 element
- Voltage and current output AN_OUT from ±10 V and +20 mA respectively
- Program example of analog module RMC12.2-E-1A

Address Assignment of the Registers - Analog Module

The analog values of the two channels, digitized by the Analog Module RMC12.2.-2E-1A, Functions are each provided as 16-bit input word. The assignment of the channels to the absolute addresses can be seen from the following figure.

The variables of the two input words and the variable of the output word OUT 0 are to be declared as INTEGER type in the declaration editor.

Word M	Word M+1	ISB Word
Analog value of **channel 1** (%IW*.4)	Analog value of **channel 2** (%IW*.4)	IN register
Output word **OUT 0** (QW*.0)	Parameter word **OUT 1** (%QW*.2)	OUT register

Fig. 12-170: Address assignment of the registers

Setting the Measuring Ranges - Analog Module

The measuring range of each of the two analog input channels of the Analog Module RMC12.2.-2E-1A, Functions is set by means of two bits, called RANGE_0 and RANGE_1. The assignment to the absolute addresses of the output word can be seen from the figure below.

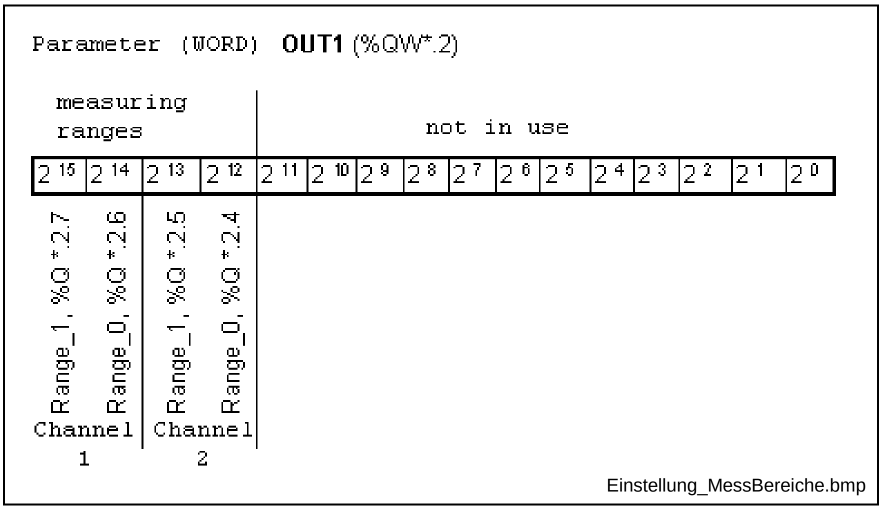


Fig. 12-171: Setting the measuring ranges

The following measuring ranges can be set with the bits RANGE_0 and RANGE_1.

Measuring range	Channel 1 RANGE_1 %Q*.2.7	Channel 1 RANGE_0 %Q*.2.6	Channel 2 RANGE_1 %Q*.2.5	Channel 2 RANGE_0 %Q*.2.4	Voltage	Current	Resistance	Tempe-rature
I	0	0	0	0	±0.5 V	---	200 Ω	-100 °C to +266 °C
II	0	1	0	1	±1.0 V	---	400 Ω	-100 °C to +850 °C
III.	1	0	1	0	±5.0 V	±20 mA	2000 Ω	---
IV	1	1	1	1	±10.0 V	---	---	---

Fig. 12-172: Overview of the settable measuring ranges

VLT_MEAS

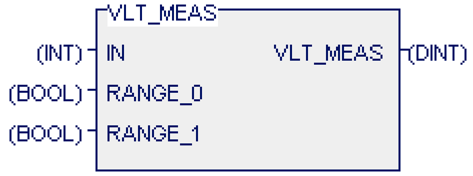
In connection with the Analog Module RMC12.2.-2E-1A, Functions, this function can be used to measure voltages of up to ±10 V. The resolution is indicated in the table below. The measuring range is selected by way of the two Boolean inputs RANGE_0 and RANGE_1. The analog value of the type INT is applied to the input IN. The output variable of the function contains the measured voltage value of the type DINT, whose unit is dependent on the set measuring range.

Error variables

If an inadmissible value is applied to the input IN, then the error variables are set as follows:

S#ErrorFlg: TRUE, S#ErrorNr: 1, S#ErrorTyp: -240

The measured value is in this case set to 0.



vlt_meas.bmp

IN: (INT): Analog value channel 1 or 2
RANGE_0: (BOOL) Lower-value bit of the measuring range
RANGE_1: (BOOL) Higher-value bit of the measuring range

Fig. 12-173: Firmware function voltage measurement VLT_MEAS

Measuring range	Voltage range	Resolution	Measured value unit
I	±0.5 V	250 µV	[10 µV]
II	±1.0 V	500 µV	[100 µV]
III.	±5.0 V	2.5 mV	[100 µV]
IV	±10.0 V	5 mV	[1 mV]

Fig. 12-174: Resolution and measured value unit in the measuring ranges

AMP_MEAS

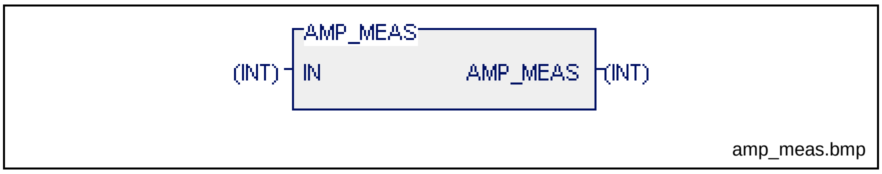
In connection with the Analog Module RMC12.2.-2E-1A, Functions, this function can be used to measure currents of up to ±20 mA. The resolution is 10 µA. The type INT analog value is applied to the input IN. The output variable of the function contains the measured type INT current value. The measured-value unit is 1 µA, with the measuring range of the channel having to be set to ±20 mA (measuring range III).

Error variables

If an inadmissible value is applied to the input IN, then the error variables are set as follows:

S#ErrorFlg: TRUE, S#ErrorNr: 1, S#ErrorTyp: -246

The measured value is in this case set to 0.



IN: (INT): Analog value channel 1 or 2

Fig. 12-175: Firmware function current measurement AMP_MEAS

Measuring range	Current range	Resolution	Measured value unit
III.	±20 mA	10 µA	1 µA

Fig. 12-176: Resolution and measured value unit

RES_MEAS

In connection with the Analog Module RMC12.2.-2E-1A, Functions, this function can be used to measure resistances of up to 2000 Ω. The resolution is indicated in the table below. The measuring range is selected by way of the two Boolean inputs RANGE_0 and RANGE_1. The analog value of the type INT is applied to the input IN. The output variable of the function includes the measured resistance value of the type INT, whose unit is dependent on the set measuring range.

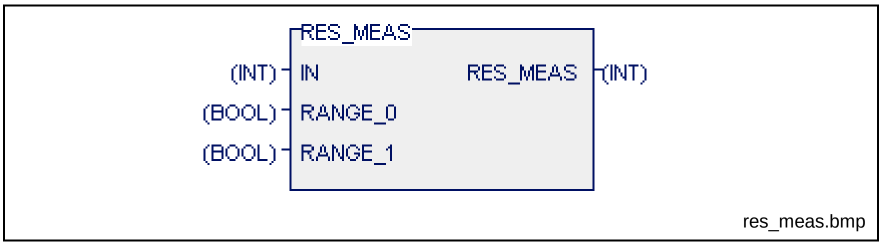
Error variables

If an inadmissible value is applied to the input IN, then the error variables are set as follows:

S#ErrorFlg: TRUE, S#ErrorNr: 1, S#ErrorTyp: -247

The resistance value is in this case set to -1.

If the measuring range IV (RANGE_0 and RANGE_1 are TRUE) is set, the resistance value is set to -2; the error variables are set as indicated above.



IN: (INT): Analog value channel 1 or 2
RANGE_0: (BOOL) Lower-value bit of the measuring range
RANGE_1: (BOOL) Higher-value bit of the measuring range

Fig. 12-177: Firmware function resistance measurement RES_MEAS

Measuring range	Resistance	Resolution	Measured value unit
I	200 Ω	100 mΩ	100 mΩ
II	400 Ω	200 mΩ	100 mΩ
III.	2000 Ω	1 Ω	1 Ω

Fig. 12-178: Resolution and measured value unit in the measuring ranges

TMP1MEAS

In connection with a PT100 element at the Analog Module RMC12.2.-2E-1A, Functions, this function can be used to measure temperatures ranging from -100 °C to +850 °C.

The implemented characteristic line describes that of an industrial platinum resistance thermometer which according to EN 60751 describes the interrelationship between temperature T and electrical resistance R_T as follows:

for the range -100 °C to 0 °C:

- $R_T = 100\Omega\ [1 + \mathrm{A}T + \mathrm{B}T^2 + \mathrm{C}(T - 100°\mathrm{C})T^3]$

for the range 0°C to 850°C:

- $R_T = 100\Omega\ (1 + \mathrm{A}T + \mathrm{B}T^2)$

The applicable constants are:

- A = 3.9083E-3 $°C^{-1}$, B = -5.775E-7 $°C^{-2}$, C = -4.183E-12 $°C^{-4}$

In the table below the resolution is indicated as a function of the set measuring range.

The measuring range is selected by way of the two Boolean inputs RANGE_0 and RANGE_1. The analog value of the type INT is applied to the input IN.

The output variable of the function includes the measured temperature value of the type INT, whose unit is dependent on the set measuring range.

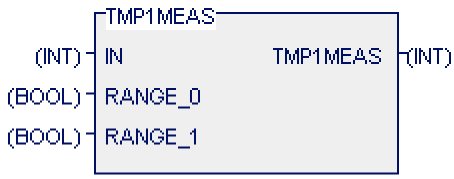
Error variables

If an inadmissible (negative) value is applied to the input IN, then the error variables are set as follows:

S#ErrorFlg: TRUE, S#ErrorNr: 1, S#ErrorTyp: -248.

In this case the temperature value adopts the value -10000.

If an invalid measuring range (III or IV) is set, the temperature value is set to -20000; the error variables adopt the above-indicated values.



tmp1meas.bmp

IN:	(INT):	Analog value channel 1 or 2
RANGE_0:	(BOOL)	Lower-value bit of the measuring range
RANGE_1:	(BOOL)	Higher-value bit of the measuring range

Fig. 12-179: Firmware function temperature measurement TMP1MEAS

Measuring range	Temperature	Resolution	Measured value unit
I	-100 ... +266 °C	0.5 °C	0.1 °C
II	-100 ... +850 °C	1 °C	1 °C

Fig. 12-180: Resolution and measured value unit in the measuring ranges

AN_OUT

This function can be used to provide voltages of up to ±10 V and currents of up to +20 mA at the analog output of the Analog Module RMC12.2.-2E-1A, Functions, with the minimum incremental width being 4.88 mV and 9.77 µA respectively.

The variable value of this function must be copied to the output word OUT 0. The input SCALE is used as scaling factor. It is advisable to set this factor to a value of 10, 100, 1000, or 10.000 for voltage output and to a value of 20, 200, 2000, or 20.000 for current output, so that the value applied to the input OUT represents the analog output value. The analog output value is calculated as follows:

$$\text{output voltage} = \frac{OUT}{SCALE} \cdot 10V$$

$$\text{output current} = \frac{OUT}{SCALE} \cdot 20mA$$

an_out_form.bmp

Fig. 12-181: Output voltage and output current - rule for calculation

Example:

For an output voltage of 1.6 V, the input variables can be assigned as follows:

- OUT: 160
- SCALE: 1000

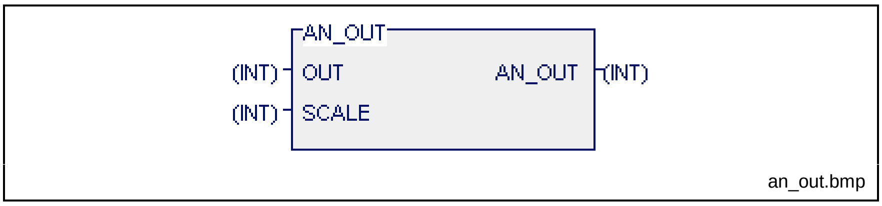
A current of 0 ... +20 mA is provided at the current output proportional to the output voltage of 0 …+10 V. For this example a current of 3.2 mA results.

Error variables

If ‘0’ is assigned to the SCALE input or the OUT value is greater in terms of amount than the SCALE value, then the error variables are set as follows:

S#ErrorFlg: TRUE, S#ErrorNr: 1, S#ErrorTyp: -249

In this case the output variable of the function adopts the value 0 which corresponds to an output value of 0 V and 0 mA.



OUT: (INT): Output value
SCALE: (INT): Scaling factor, +10 V and +20 mA

Fig. 12-182: Firmware function voltage and current output AN_OUT

Rexroth Bosch Group

Program Example of Analog Module RMC12.2-E-1A

In connection with the Analog Module RMC12.2.-2E-1A, Functions, the following measurement must be taken:

- channel 1: voltage measurement in the range of ±10 V
- channel 2: temperature measurement in the range of +300...+400 °C

In addition a current of 10 mA is output at the analog output.

The logical address 2 for the analog module was assigned by way of the IO editor.

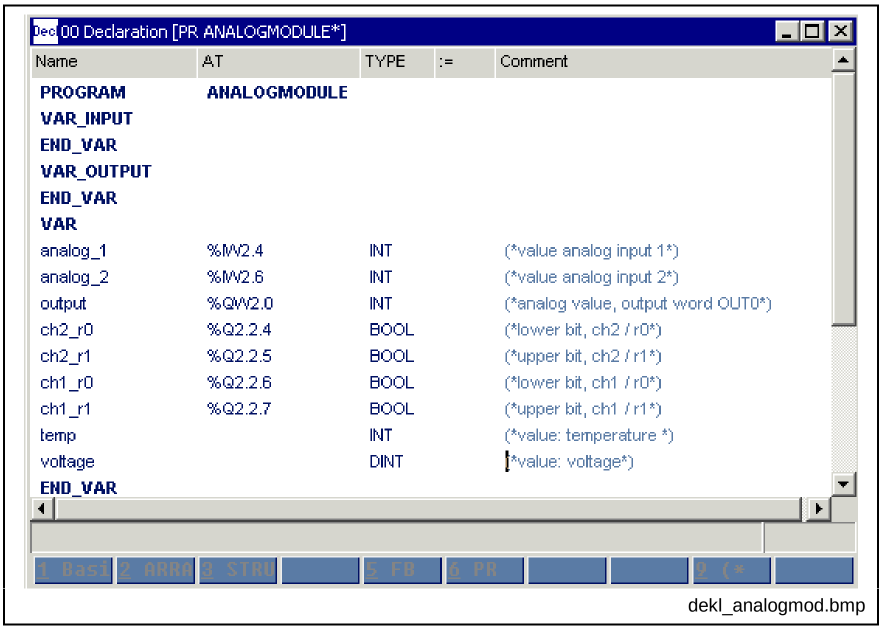


Fig. 12-183: Declaration part for the analog module example

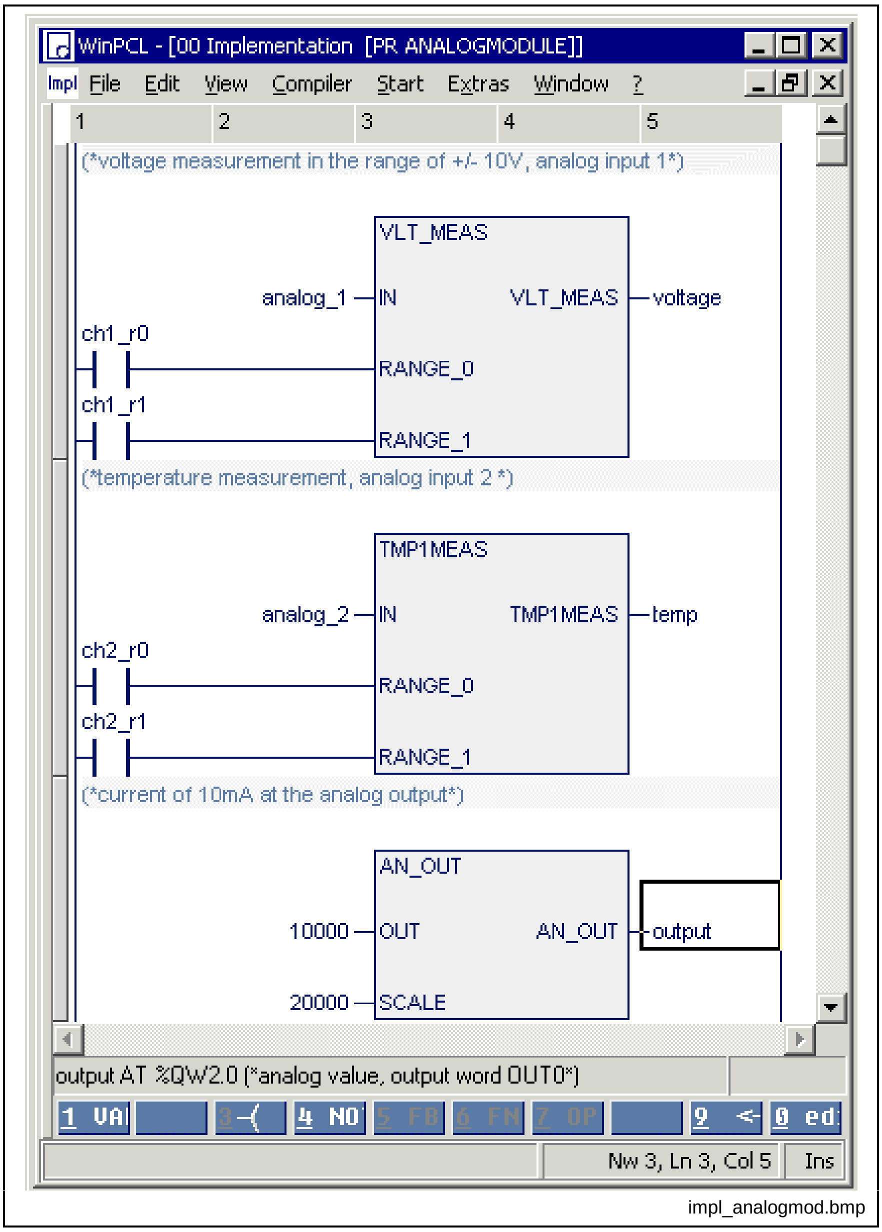


Fig. 12-184: Ladder diagram for the analog module example

PROFIBUS DP, Functions

The following Firmware Functions are available:

- Starting the bus communication: DPM_START
- Stopping the bus communication: DPM_STOP
- Program Example for Starting and Stopping the PROFIBUS
- Status information on PROFIBUS process data exchange: DPM_EXCHG

DPM_START

Starting the bus communication, PROFIBUS DP, Functions

Using this function, the PROFIBUS is switched to the OPERATE mode and communication between master and slaves is started.

Note: Using the Fieldbus IO Configurator FIOCon, it is possible to set the starting behavior of the PROFIBUS after system initialization. If ”Automatic enabling of communication by the system” is set, then the bus communication is automatically started after every PLC program download (CTRL-F9) and after every download of the **configuration** by the FIOCon. If the ”Controlled enabling of communication by the application program” setting is selected, the bus communication must be started explicitly with the block DP_START.

The bus communication is started if the input becomes START TRUE. If start is successful, the function result becomes TRUE.

Error variables

If a PROFIBUS interface is not provided, the error variables must be set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -244

START: Activating the OPERATE mode

Fig. 12-185: PROFIBUS DP, function DPM_START

DPM_STOP

Stopping the bus communication, PROFIBUS DP, Functions

Using this function, the PROFIBUS is switched to the STOP mode and communication between master and slaves is stopped.

The bus communication is stopped if the input becomes STOP TRUE. If the function is executed successfully, the function result becomes TRUE.

Error variables

If a PROFIBUS interface is not provided, the error variables must be set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -243

START: Activating the STOP mode

Fig. 12-186: PROFIBUS DP, function DPM_STOP

Program Example for Starting and Stopping the PROFIBUS

The PC104-PROFIBUS interface is fitted to slot 2. The bus can be started using the variable dp_start and can be stopped using the variable dp_stop.

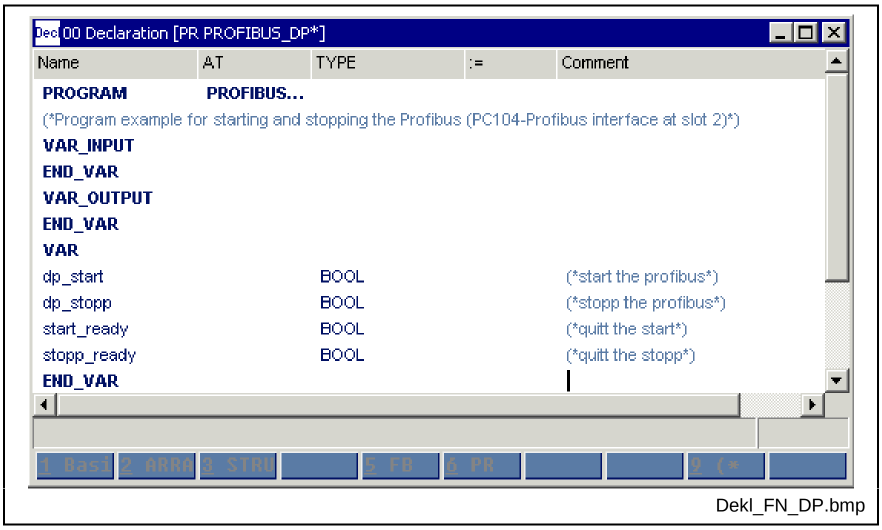


Fig. 12-187: Declaration part for the program example

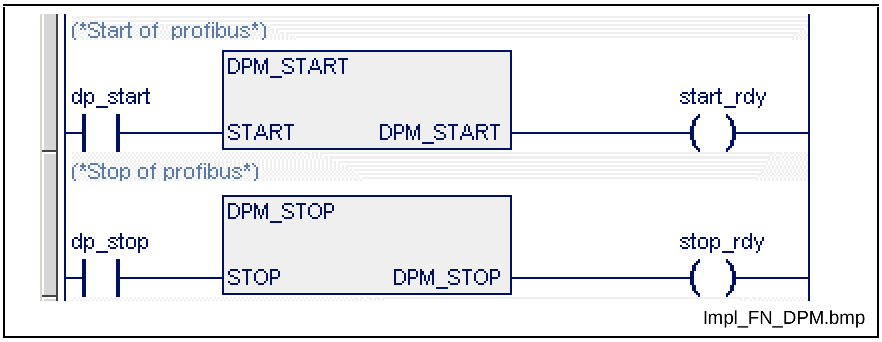


Fig. 12-188: Implementation part for the program example

DPM_EXCHG

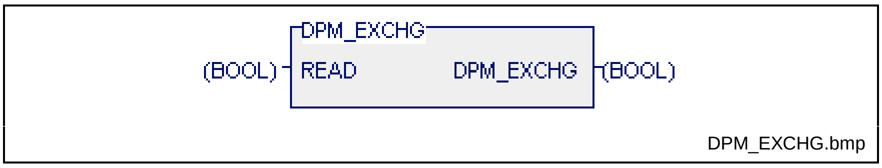
Status information on process data exchange, PROFIBUS DP, Functions

This function supplies the status information on the PROFIBUS process data exchange. If the data exchange is active, the function result is TRUE.

Error variables

If a PROFIBUS interface is not provided, the error variables must be set as follows:

S#ErrorFlg:	TRUE
S#ErrorNr:	235
S#ErrorTyp:	-245



READ: Read status

Fig. 12-189: PROFIBUS DP, function DP_EXCHG

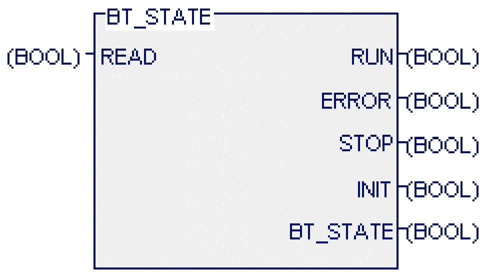
BT-Bus, Functions

Analogous to the other fieldbuses the following new system functions are provided for the BT bus:

- BT_STATE: BT bus diagnostic function
- BT_STOP: Stop I/O data exchange
- BT_START: Start I/O data exchange

BT_STATE

As diagnostic function system function BT_STATE shows the current operating state of the BT bus. (BT-Bus, Functions)



BT_STATE.bmp

READ:	0 - Function not active
	1 - Read operating state
RUN:	0 - I/O data exchange not active
	1 - I/O data exchange active
ERROR:	1 - Error on BT bus
STOP:	1 - BT bus switched to STOP
INIT:	1 - BT bus is initialized
BT_STATE:	State of the READ input

Fig. 12-190: BT bus: Function BT_STATE

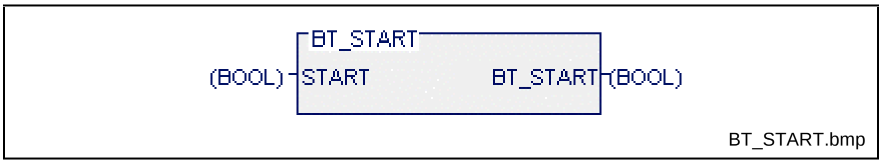
Error variables

If there is no BT bus connection, the error variables are set as follows:

S#ErrorFlg:	TRUE
S#ErrorNr:	235
S#ErrorTyp:	-219

BT_START

System function BT_START enables the PLC user to switch the BT bus from state 'STOP' to state 'Run'. This function is required to restart the communication after the detected bus error is eliminated (BT-Bus, Functions).



START:	0 - Function not active 1 - Switch BT bus in state 'RUN'
BT_START:	0 - 'STOP' (I/O data exchange not active) 1- 'RUN' (I/O data exchange active)

Fig. 12-191: BT bus: Function START

Error variables

If there is no BT bus connection, the error variables are set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -217

BT_STOP

System function BT_STOP enables the PLC user to switch the BT bus in state 'STOP'. In this state no data exchange with the operator terminals occurs (BT-Bus, Functions).

STOP:	0 - Function not active 1 - Switch BT bus in state 'STOP'
BT_STOP:	1 - 'STOP' (I/O data exchange not active)

Fig. 12-192: BT bus: Function STOP

Error variables

If there is no BT bus connection, the error variables are set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -218

ASI Bus, Functions

Analogous to the other fieldbusses the following system functions for the ASI bus are provided:

- ASIM_STATE_CH*: Diagnostic function for the channels 1 and 2, ASI bus
- ASIM_STOP: Stop I/O data exchange
- ASIM_START: Start I/O data exchange
- ASIM_RESET: Reset ASI bus
- ASIM_SLDIAG Diagnosis of single slaves

ASIM_START

System function ASIM_START enables the PLC user to switch the data exchange with the slaves on channel 1 and channel 2 of the ASI bus from state 'STOP' to state 'RUN'.

Note: If the process data exchange with system configurator SyCon (menu Online / Stop communication) was stopped, the process data exchange can only be restarted via the SyCon.

START: 0 - Function not active
1 – Switch BT bus to state 'RUN'
ASIM_START: 0 – 'STOP' (I/O data exchange not active)
1 – 'RUN' (I/O data exchange active)

Fig. 12-193: ASI bus: Function START

Error variables

If there's no ASIM bus connection, the error variables are set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -331

The error occurs also, if

- there's no ASI master connection programmed in the I/O editor or
- the address setting of the PC104 ASI master connection is incorrect.

ASIM_STOP

System function ASIM_STOP enables the PLC user to switch the data exchange with the slaves to channel 1 and channel 2 of the ASI bus to state 'STOP'. In this state no I/O data exchange occurs.

ASIM_STOP.bmp

STOP:	0 - Function not active
	1 – Switch BT bus to state 'STOP'
ASIM_STOP:	1 – 'STOP' (I/O data exchange not active)

Fig. 12-194: ASIM bus: Function STOP

Error variables

If there's no ASIM bus connection, the error variables are set as follows:

S#ErrorFlg:	TRUE
S#ErrorNr:	235
S#ErrorTyp:	-332

The error occurs also, if

- there's no ASI master connection programmed in the I/O editor or
- the address setting of the PC104 ASI master connection is incorrect.

ASIM_RESET

System function ASIM_RESET enables the PLC user to switch the data exchange with the slave on channel 1 and channel 2 of the ASI bus on and off.

If input RESET is set to TRUE, the master connection is reset, until this input is set to FALSE again. The input is only mapped to the output and provides therefore no further information.

During resetting the master connection, the outputs of the slave are switched to a safe state. If the RESET input is set to FALSE again, the initialization of the master occurs, whereby the process data exchange is started by means of SyCon depending on the settings. The initialization can last several seconds.

Note: If in the system configurator SyCon was set, that the process data exchange is stopped as soon as the communication with the slave is interrupted, the ASI master has to be reset by means of this function.

ASIM_RESET.bmp

RESET:	Executes a warm start of the ASI master connection
ASIM_RESET:	Follows the input

Fig. 12-195: ASIM bus: Function RESET

Error variables

If there’s no ASIM bus connection, the error variables are set as follows:

S#ErrorFlg:	TRUE
S#ErrorNr:	235
S#ErrorTyp:	-334

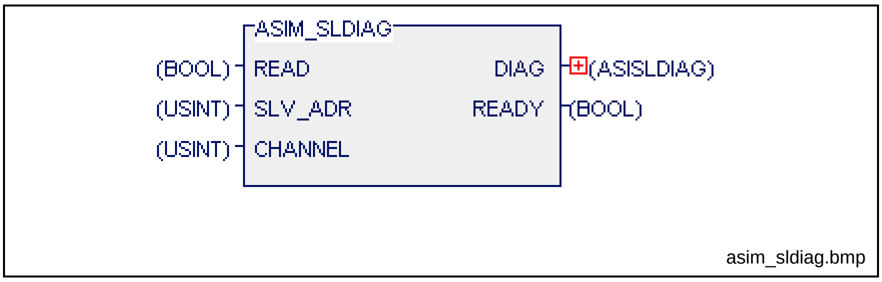
The error occurs also, if

- there’s no ASI master connection programmed in the I/O editor or
- the address setting of the PC104 ASI master connection is incorrect.

ASIM_SLDIAG

System function block ASIM_SLDIAG serves to transmit the diagnostic information to the slave, that is addressed via the inputs SLV_ADR and CHANNEL. If the information is valid, output READY becomes active.

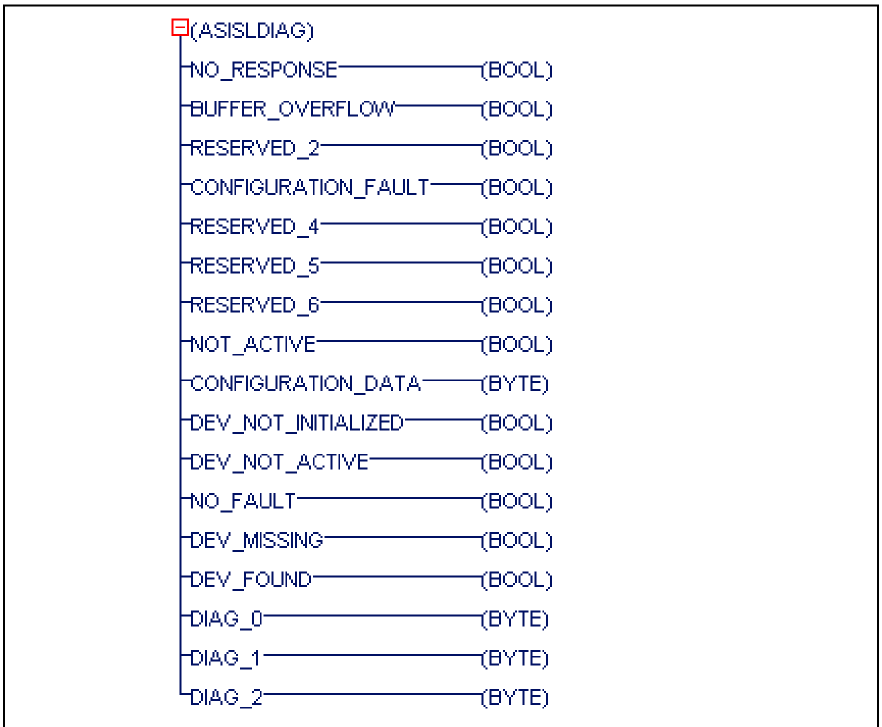
The diagnostic information is listed in structure ASISLDIAG.



READ:	Request of the diagnostic information
SLV_ADR:	Address of the slaves. Valid values: 1...62
CHANNEL:	Channel number. Valid values: 1, 2
DIAG:	Diagnostic information of the slave; see type description
READY:	For TRUE the information of the slave is valid

Fig. 12-196: ASIM-Bus: Function block ASIM_SLDIAG

Type description ASISLDIAG



typ_asisldiag.bmp

NO_RESPONSE:	Device does not respond or is not available.
BUFFER_OVERFLOW:	The number of the entries in the error buffer exceeds the maximum possible number.
CONFIGURATION_FAULT:	The determined IO or ID code differs from the configured code.
NOT_ACTIVE:	The slave is not active in the current configuration.
CONFIGURATION_DATA:	Retrieved IO/OD code
DEV_NOT_INITIALIZED:	Slave was not initialized
DEV_NOT_ACTIVE:	Slave not active
NO_FAULT:	Slave indicates no error
DEV_MISSING:	Slave not available
DEV_FOUND:	At the moment not supported
DIAG_0/1/2:	Slave-specific diagnostic information. See manual of the manufacturer.

Fig. 12-197: Data type "Diagnostic information of a SLAVE"

Note: These blocks may not be declared in the retain area.

Error variables

For all errors of the block applies:
S#ErrorFlg: TRUE, S#ErrorTyp: -336,

S#ErrorNr: 235 occurs, if

- there's no ASI master connection programmed in the I/O editor,
- there's no PC104 ASI master connection plugged in the control,
- the address setting of the PC104 ASI master connection is incorrect.

S#ErrorNr: 1 (invalid input parameter) occurs, if

- the slave address is greater 62 or equal to 0,
- the channel number is unequal to 1 or 2.

S#ErrorNr: 6 (internal transmission error) occurs, if during the diagnostic request an error, e.g. timeout, occurred.

ASIM_STATE_CH*

The system functions ASIM_STATE_CH1 or ASIM_STATE_CH2 enable to read the status information of the ASI master for the respective channel.

The information is as long valid as the main output is active.

Error variables

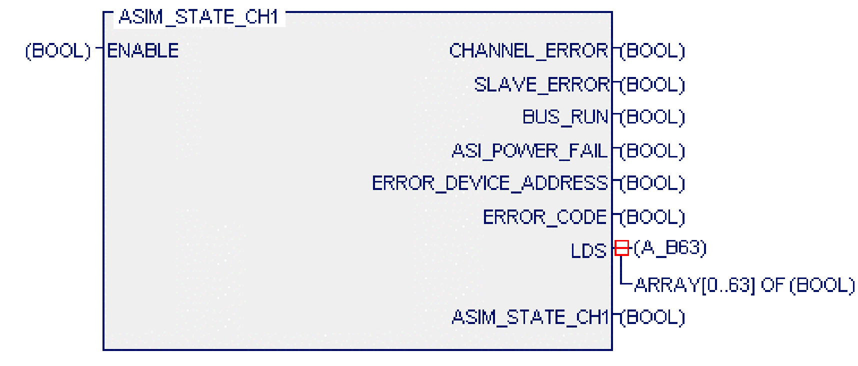
S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -333 (channel 1) or -335 (channel 2),

The error occurs also, if

- there's no ASI master connection programmed in the I/O editor,
- there's no PC104 ASI master connection plugged in the control,
- the address setting of the PC104 ASI master connection is incorrect.



ASIM_STATE_CH1.bmp

ENABLE:	Retrieving the status information
CHANNEL_ERROR:	If this output is set, an error occurred, that deactivates the process data exchange with all slaves.
SLAVE_ERROR:	The communication with at least one slave can not be started. See also: ERROR_CODE and ERROR_DEVICE_ADDRESS.
BUS_RUN:	Process data exchange with at least one slave is active.
ASI_POWER_FAIL:	This output is set, if the voltage supply of the ASI bus is insufficient. Output CHANNEL_ERROR becomes also TRUE.
ERROR_DEVICE_ADDRESS:	Address of the slave, that indicated the last diagnostic message.
ERROR_CODE:	See table below.
LDS:	List of slaves with diagnosis. Table of the slaves with pending diagnosis. Every slave is assigned to a corresponding ARRAY element.
ASIM_STATE_CH1/2:	TRUE, as long as ENABLE is TRUE. All outputs are only valid, if this output is set.

Fig. 12-198: ASIM_ bus: Functions ASIM_STATE_CH1 / ASIM_STATE_CH2

Note: CHANNEL_ERROR: If this output is set, an error occurred, that deactivates the process data exchange. Error causes can be:

- Soft/Hardware errors on the PC104 controller board
- Unplugged bus cables
- Insufficient voltage supply of the ASI bus

Additional information about the error cause is indicated in the outputs ERROR_CODE and ASI_POWER_FAIL.

Note: If the process data exchange via SyCon or ASIM_STOP was deactivated, output CHANNEL_ERROR is not set.

Value	Explanation
0	No Error / Success No error detected.
	Initialization error
50	No User Task. The user task could not be found (internal error).
51	No Global Data. The global data area could not be accessed (internal error).
52	No PLC Task. The PLC task could not be found (internal error).
53	Unknown Mode. The detected data exchange mode is not supported.
54	No Protocol Chip Found. The expected protocol chip could not been found (System Failure).
55	Channel #1 Not Found. The protocol chip for channel 1 does not respond.
56	Channel #2 Not Found. The protocol chip for channel 2 does not respond.
57	Initialization Failure Master. A failure during scanning the master table has occur.
58	Baudrate Not Supported. The detected baudrate is not supported.
59	Unknown Data Format. The detected data format is none of the defined (Intel / Motorola).
60	Unknown AutoClear Mode. The detected AutoClear mode is none of the defined.
61	Start-Option Not Supported. The detected Start-option is none of the defined.
62	AutoClear Mode - Automatic Address Assignment Conflict. The AutoClear mode conflicts to the automatic address assignment.
63	Initialization Failure Slave-Device. A failure during scanning the slave entry table has occurred.
64	Invalid ASi Channel. The detected channel number is invalid.
65	Maximum Slave Entries Reached. The detected count of slave parameter sets exceed the max. defined count of slaves.
66	Invalid Address Mapping. Address-overlapping was detected by the ASi Master during screening the input / output offsets.
67	Invalid Slave Address. The detected slave address is invalid.
68	Invalid Device Parameter. At least one of the detected device parameter is invalid.

Value	Explanation
69	Parameter Value Out Of Range. The detected parameter value is out of range.
70	IO Code Out Of Range. The detected IO code is not supported.
71	ID Code Out Of Range. The detected ID code is not supported.
72	Device Address Double. The detected slave address already exists.
73	Module Count Failure. The number of input / ouput modules exceeds the max. defined number.
74	Device Not Activated. The requested slave is not activated.
75	No Configuration Data Found. No configuration data in the data base found.
76	Version Incompatible. The detected version of the ASi master state machine does not match the expected one (internal error).
77	Unknown Initialization Error. Unknown initialization error detected.

Fig. 12-199: Error code at output "ERROR_CODE": **Initialization error**

	Runtime error
110	Watchdog Failure A watchdog failure occurred. The ASi Master will set both channels to OFFLINE state.
111	No Data Acknowledge. The user has failed to acknowledge the data cycle (in data exchange mode 0) .The ASi Master will set both channels to OFFLINE state.
112	Application Error. An application (USER) error has been detected by the task.
113	Unknown Command. The detected command is none of the defined.
114	Unexpected IX State. The fieldbus interface processor (IX1) reports an state which does not match to the expected one (internal error).
115	IX Not Ready. The fieldbus interface processor (IX1) is not ready.
116	IX Not Active. The fieldbus interface processor (IX1) is not active.
117	Reserved ...
118	Invalid Channel. The detected channel number is invalid or the ASi Master reports an unrecoverable failure for this channel.
119	Invalid Slave Address. The requested slave address is out of range.
120	Parameter Value Out Of Range. The parameter value handed over by the application is out of range.
121	Slave Exists. The requested slave exists in the current configuration.
122	Slave Not Exists. The requested slave does not exists in the current configuration.
123	Slave is Projected. The requested slave is a member of the List of Projected Slaves.
124	Slave is Not Projected. The requested slave is not a member of the List of Projected Slaves.
125	Slave is Active. The requested slave is a member of the List of Activated Slaves.
126	Slave is Not Active. The requested slave is not a member of the List of Activated Slaves.
127	Slave is Detected. The requested slave is a member of the List of Detected Slaves.
128	Slave is Not Detected. The requested slave is not a member of the List of Detected Slaves.

	Runtime error
129	Invalid Mode. The detected mode is not supported by the requested function.
130	Invalid Area. The detected area code is not supported by the requested function.
131	Projected Slave is Missing. A projected slave is missing.
132	None-Projected Slave Found. An none-projected slave was found by the ASi Master.
133	Slave Configuration Fault. The configured IO-/ID-Code differs from the detected IO-/ID-Code.
134	Service Not Available. The requested service is not available.
135	AutoConfiguration Failure.An error during performing the function ’AutoConfiguration’ occurred.
136	Multiplexing in Process. The multiplexing sequence for analogue modules is still in process.
137	APF for Channel. ASi power failure (APF) was detected for the requested channel.
138	Unexpected Parameter. Slave responded with an unexpected parameter value.
139	Auto Clear State Reached. The device reached the Auto Clear State.
140	Unexpected Runtime Error

Fig. 12-200: Error codes at output "ERROR_CODE": **Runtime error**

INTERBUS, Function

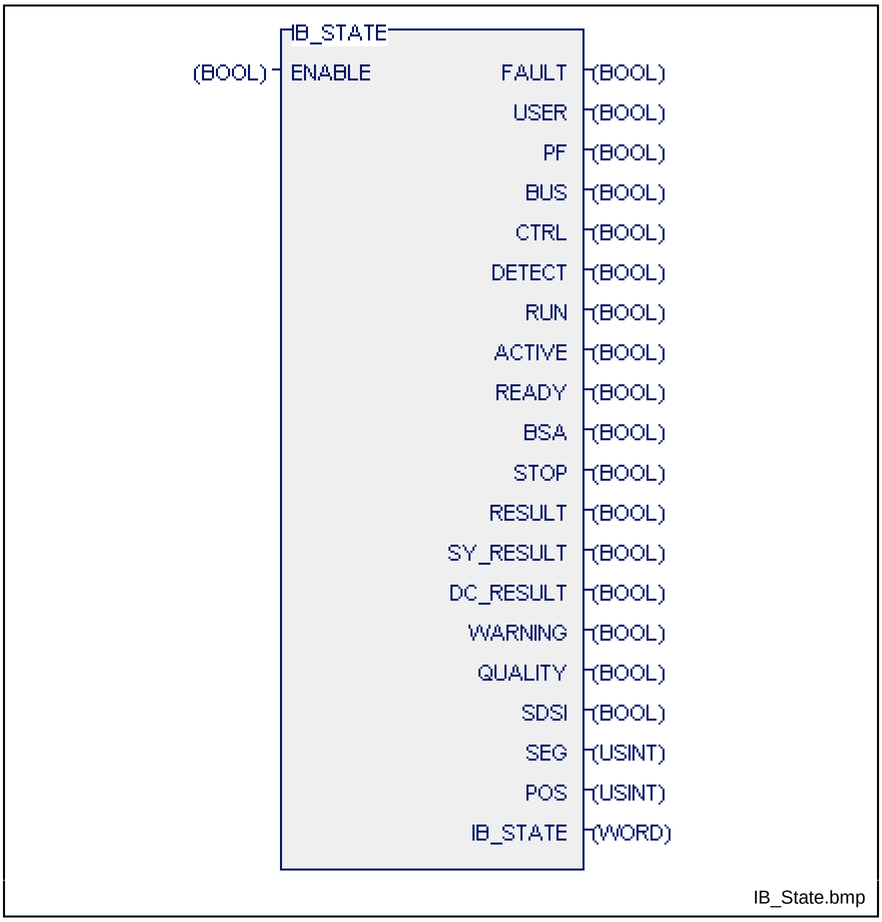
The diagnostic functions IB_STATE / IB_STATE2 for the INTERBUS retrieve the values of the diagnostic bit register and the diagnostic parameter register of the INTERBUS and provide them in a conditioned manner. The extension by the second INTERBUS master is only available after releasing version 06V02 / 23V02. Both busses provide the same functions. The two registers mentioned above map the diagnostic display to the control system and inform about the current status of the INTERBUS system.

Diagnostic bit register

The diagnostic bit register consists of 16 bits. Each bit in the diagnostic bit register is assigned to a state of the INTERBUS controller board. If there’s no malfunction, only the bits READY, ACTIVE and RUN are set. Then, the diagnostic parameter register contains value 0x0000.

Diagnostic parameter register

The states in the bits USER, PF, BUS and CTRL of the diagnostic bit register are specified via the diagnostic parameter register. The value of this register is always specified again, if one of the above mentioned bits is set. In case of a bus error (BUS == TRUE) or a peripheral fault (PF == TRUE) the diagnostic parameter register contains the specification of the error location (segment and position). In case of an user (USER == TRUE) or hardware error (CTRL == TRUE) an error code is indicated.



ENABLE	TRUE: Request of the diagnostic bit and diagnostic parameter register FALSE: Outputs are deleted
FAULT	Disjunction of the bits USER, PF, BUS and CTRL: If at least one of this signals is TRUE, the bit FAULT is set.
USER	User error / parameterization
PF	Peripheral fault
BUS	Bus error
CTRL	Error on the INTERBUS controller board / hardware
DETECT	Diagnostic routine is active
RUN	Data transmission is active, data cycles are executed
ACTIVE	Only ID cycles are executed
READY	Controller board is ready
BSA	One or more bus segments are switched off
STOP	Outputs are reset
RESULT	Standard function was negatively executed
SY_RESULT	Synchronization error occurred (only in operating mode "synchronous")
DC_RESULT	Incorrect data cycles (only in operating mode "synchronous")
WARNING	Bus warning time has expired (can be parameterized)
QUALITY	Specified error density exceeded (because of transmission fault)
SDSI	There’s a message in the standard signal interface
SEG	Segment of the error location during bus error or peripheral fault (corresponds to the diagnostic parameter register)
POS	Position of the error location during bus error or peripheral fault (corresponds to the diagnostic parameter register)
IB_STATE	Contents of the diagnostic parameter register

Fig. 12-201: INTERBUS – status display, equal for IB_STATE2

Error variables

For all errors of the function applies:
S#ErrorFlg: TRUE, S#ErrorTyp: -330/-337, S#ErrorNr: 235 occurs, if

- there’s no INTERBUS connection programmed in the I/O editor,
- there’s no PC104 INTERBUS connection plugged in the control,
- the address setting of the PC104 INTERBUS connection is incorrect.

12.4 User Functions

The programming system permits the user to write functions himself, which can be used as re-usable units in the form of a supplement to the standard and firmware functions. The user functions can import other user functions and use them in the same way as the standard and firmware functions. Structuring with sequential function chart elements and the use of external variables is not possible.

Import Rules for Functions

Standard, firmware and user functions can be used by means of a function .

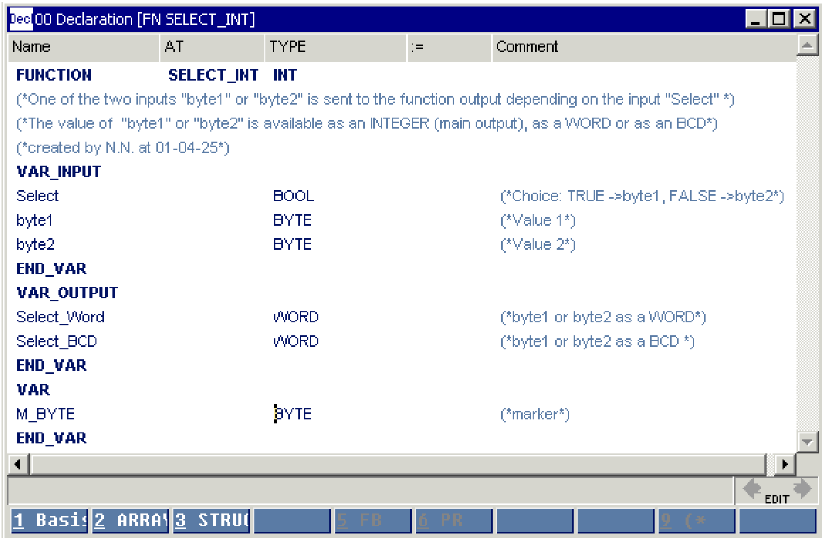
The required function is a standard or firmware function	The required function is a user function
It is included in the library of the programming system and is, thus, known. It does not require any memory that has to be kept permanently available. As a result, it can be simply used.	It is not included in the library and is, thus, not known to the programming system. It is made known by an automatic import of the function in the program, function block or function by which it is to be used. The declaration part of the function to be imported must at least be present. It does not require any memory that has to be kept permanently available. As a result, it can be simply used.

The nesting can be continued to any depth desired.

It is **forbidden** that function ‘A’ uses itself again (recursion) or that function ‘A’ uses function ‘B’ and the latter uses function ‘A’ again etc.

Program Example for User Function SELECT_INT

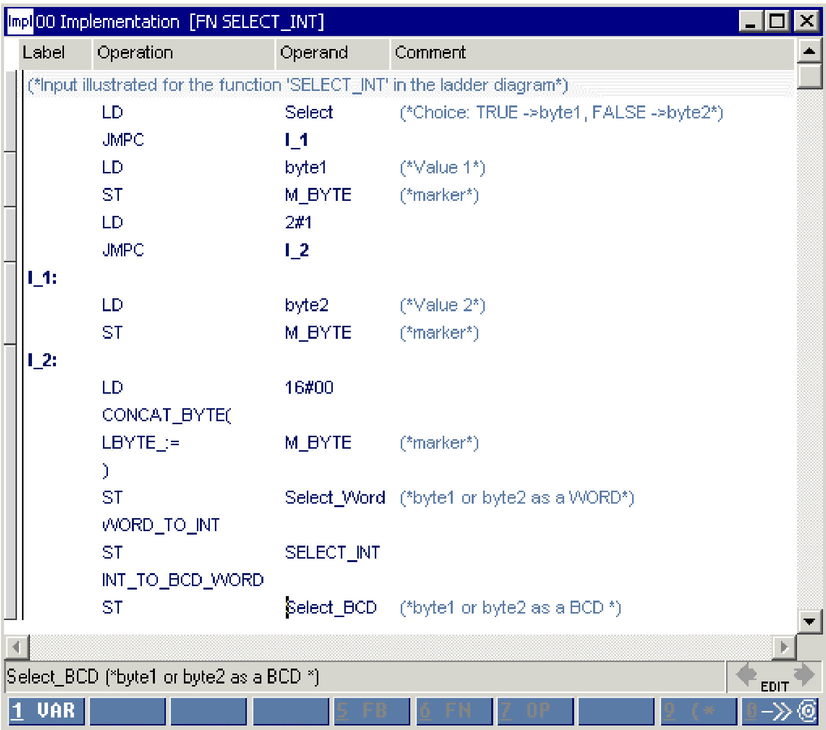
In addition to the main output, the function SELECT_INT is provided with two further outputs. It imports, i.e. uses, several other functions.



select_int_dekl.bmp

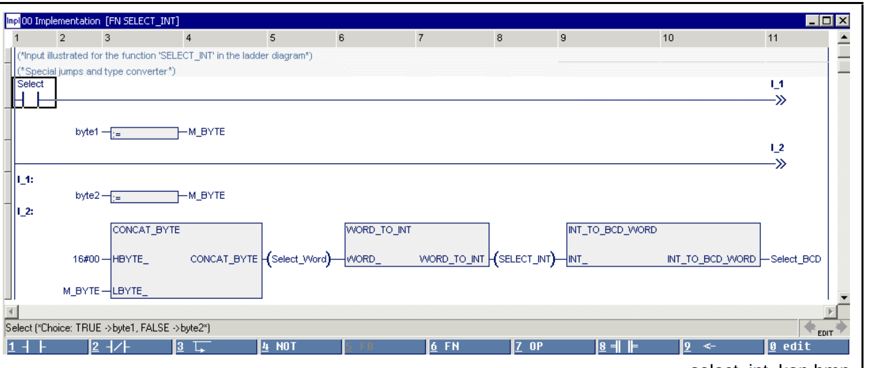
Fig. 12-202: Declaration part of the function SELECT_INT

The implementation is depicted below in IL and LD.



select_int_awl.bmp

Fig. 12-203: Instruction list of the function SELECT_INT



select_int_kop.bmp

Fig. 12-204: Ladder diagram of the function SELECT_INT

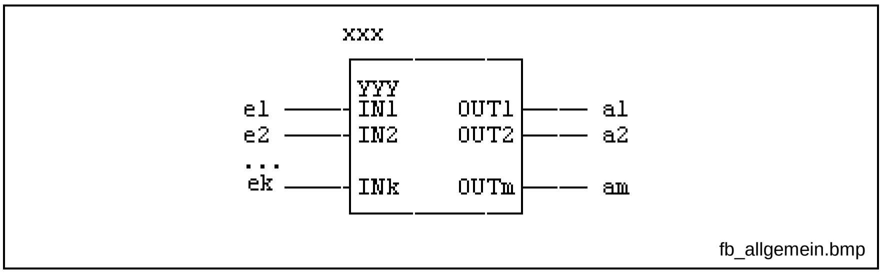
13 Function Blocks in WinPCL

13.1 Function Blocks, General Information

A function block (FUNCTION BLOCK, FB) is a program organization unit, which can have:

- 1...k inputs,
- 1...m outputs and
- internal variables

and can use external variables.



xxx	-	Instance name (name of the function block assignment)
yyy	-	Type name of the function block
ei	-	Inputs of the function block 1...k
aj	-	Outputs of the function block 1...m

Fig. 13-1: Function blocks, general interface

The **IEC concept** provides for a basic separation between the program code of the function block and the data storage necessary for storing the values of the variable.

The inputs and outputs of a function block are visible for the user. The internal variables remain secret to the user.
A distinction is made between standard and firmware function blocks as well as user-defined function blocks.

Standard Function Blocks

in accordance with EN 61131-3 (+ supplements)

Firmware Function Blocks

- Control of an INTERBUS
- PCP function blocks for the parameter channel of the INTERBUS
- Communication of the PLC with the CNC, via serial interfaces etc.
- Connection of miniature control panels and HMI
- Extension of the functional range of the ISP for Motion Control

User Function Blocks

written by the user himself.
Standard and firmware function blocks can be used, but <u>not modified</u>. Their interface is constant, even for further developed standard libraries and operating systems.

13.2 Standard Function Blocks

The standard function blocks for the Rexroth control system are based on EN 61131-3.
They are available in all programming languages of the system.
They can be used but not modified.

FlipFlops (bistable elements)

- SRflip-flop
- RSflip-flop
- TOGGLE

Edge evaluation

- R_TRIG, edge evaluation for rising edges
- F_TRIG, edge evaluation for falling edges

Collecting / splitting bit strings

- BOOL_BYTE, BOOL_WORD, BOOL_DW
- BYTE_BOOL, WORD_BOOL, DW_BOOL

Up-down counter DOS compatible

- CTUD_USINT_INDR, Counting range 0 ... 255
- CTUD_UINT_INDR, Counting range 0 ... 65535
- CTUD_INT_INDR, Counting range -32768 ...32767

Up-down counter EN 61131 compatible

- CTUD_USINT, Counting range 0 ... 255
- CTUD_UINT, Counting range 0 ... 65535
- CTUD_INT, Counting range -32768 ...32767

Time stages

- TP, - generation of PT-wide single pulses
- TON, - generation of PT-wide on-delay timer function block
- TOFF, - generation of PT-wide off-delay timer function block
- FLASH, - free running clock generator

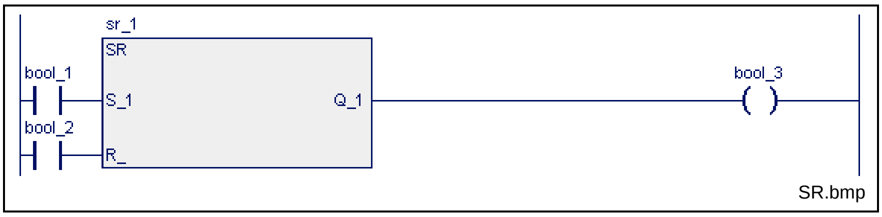
Date and time

- DATE_RD, read date
- TOD_RD, read time

Bistable Elements

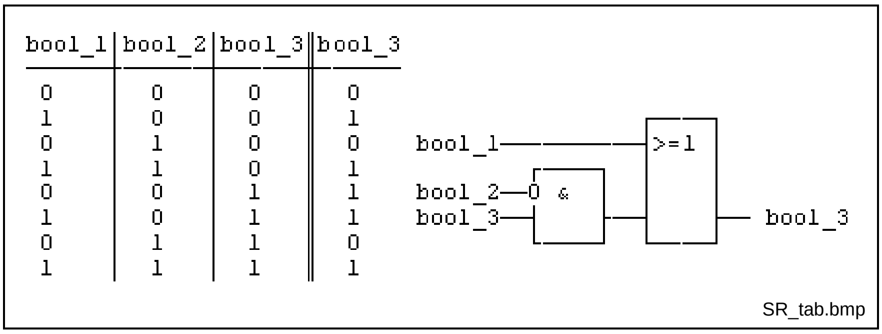
SR

The SR flip-flop (also see Standard Function Blocks, Bistable Elements) realizes a dominating setting of the memory.



S_1:	BOOL	Set input, dominating
R_:	BOOL	Reset input
Q_1:	BOOL	Output

Fig. 13-2: Standard function block SR

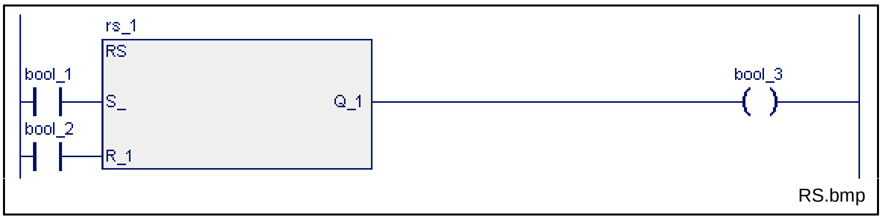


bool_1	bool_2	bool_3	bool_3
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	0
1	1	1	1

Fig. 13-3: Circuit diagram and replacement circuit SR

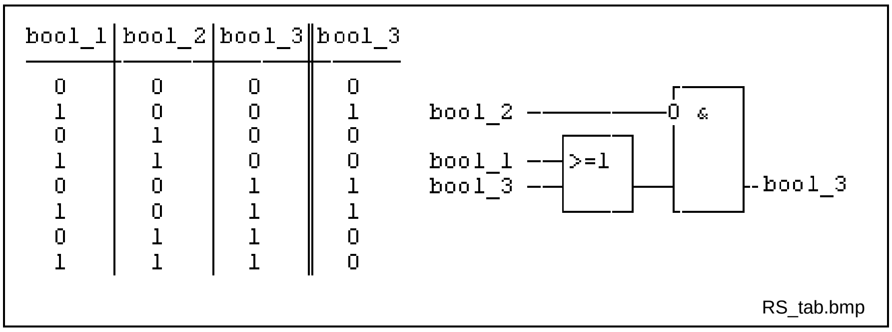
RS

An RS flip-flop (also see Standard Function Blocks, Bistable Elements) realizes a dominating resetting of the memory.



S_:	BOOL	Set input
R_1:	BOOL	Reset input, dominating
Q_1:	BOOL	Output

Fig. 13-4: Standard function block RS

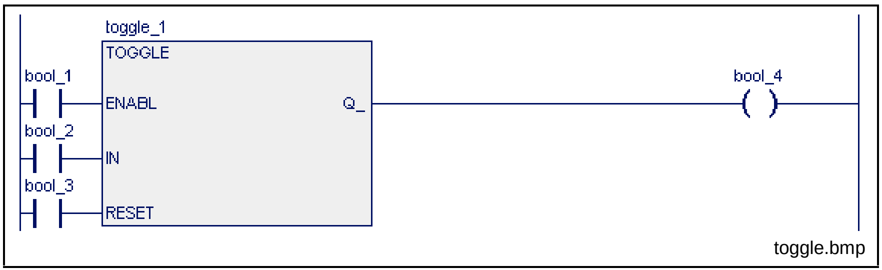


bool_1	bool_2	bool_3	bool_3
0	0	0	0
1	0	0	1
0	1	0	0
1	1	0	0
0	0	1	1
1	0	1	1
0	1	1	0
1	1	1	0

Fig. 13-5: Circuit diagram and replacement circuit RS

TOGGLE

The function block TOGGLE (also see Standard Function Blocks, Bistable Elements) inverts the previous state of the output Q_ whenever there is a positive edge at input IN. As long as the input RESET is logic 1, the output Q_ remains logic 0. The state of the output Q_ can be changed only if the input ENABLE is logic 1.



ENABLE:	BOOL	Enable input
IN:	BOOL	Switch over signal
RESET:	BOOL	Reset input
Q_:	BOOL	Output

Fig. 13-6: Standard function block TOGGLE

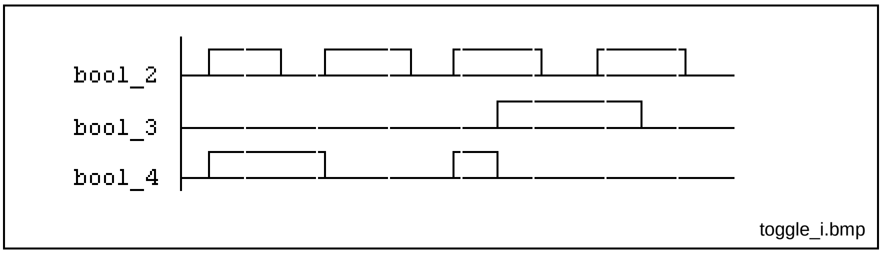


Fig. 13-7: Pulse diagram TOGGLE

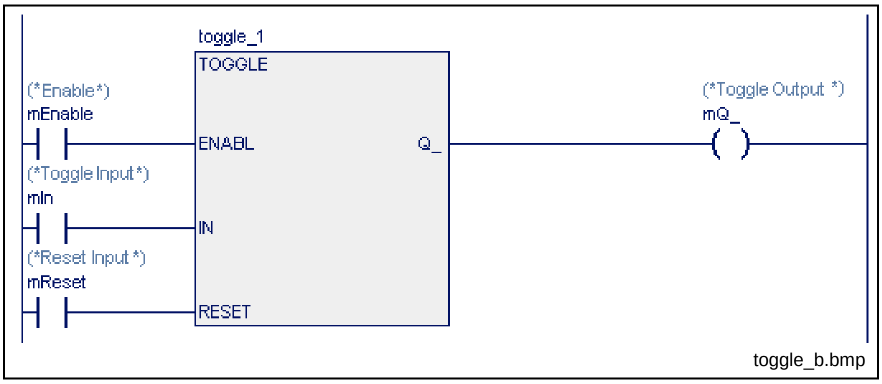


Fig. 13-8: TOGGLE application

Error handling

Errors cannot occur:

S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

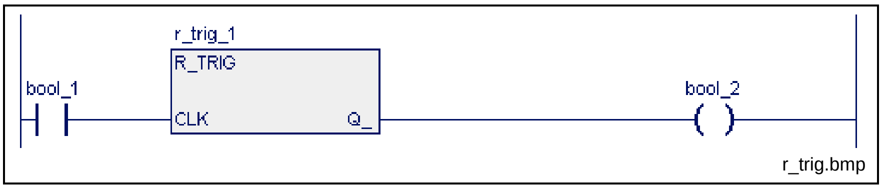
Edge Evaluation for Rising and Falling Edges

R_TRIG

The edge evaluation for rising R_TRIG edges (also see Standard Function Blocks, Edge Evaluation for Rising and Falling Edges) implements a 0-1-0 transition at the output when the input changes its assignment from 0 to 1.

The pulse duration results from the duration of the PLC cycle.

Note: Reprocessing of the function block in the PLC sequential cycle must be ensured after a 0-1 transition at the output!



CLK:	BOOL	Input signal for 0-1 edge evaluation
Q_:	BOOL	Output

Fig. 13-9: Standard function block R_TRIG

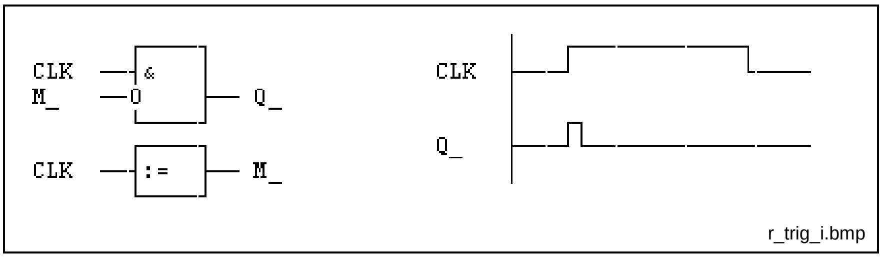


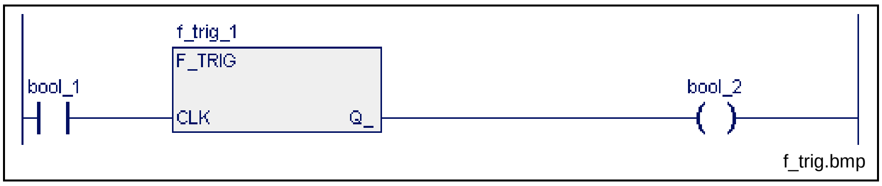
Fig. 13-10: Internal realization and pulse diagram

F_TRIG

The edge evaluation for falling F_TRIG edges (also see Standard Function Blocks, Edge Evaluation for Rising and Falling Edges) implements a 0-1-0 transition at the output when the input changes its value from 1 to 0.

The pulse duration results from the duration of the PLC cycle.

Note: Reprocessing of the function block in the PLC sequential cycle must be ensured after a 0-1 transition at the output!



CLK:	BOOL	Input signal for 10 edge evaluation
Q_:	BOOL	Output

Fig. 13-11: Standard function block F_TRIG

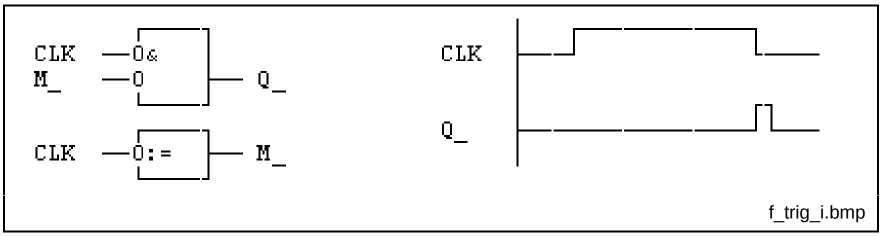


Fig. 13-12: Internal realization and pulse diagram

Collecting / Splitting Bit Strings

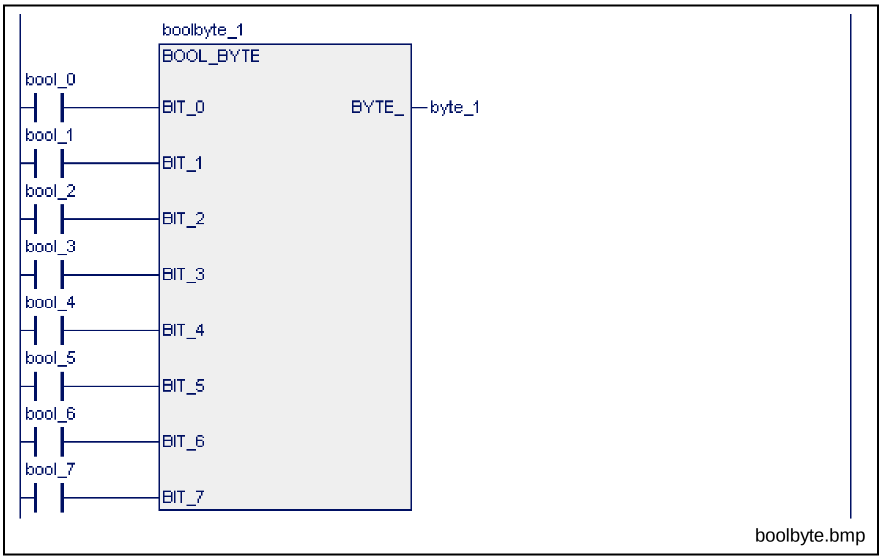
A set of function blocks is provided for collecting Boolean variables with following conversion into BYTE / WORD / DWORD.

Furthermore there is also a set of blocks for splitting BYTE / WORD / DWORD into Boolean variables.

BOOL_BYTE

The function block BOOL_BYTE (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts eight Boolean variables into a BYTE (and, analogously, BOOL_WORD 16 Boolean variables and BOOL_DW 32 Boolean variables). Unassigned inputs are interpreted as 0 (FALSE).

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



BIT_0:	BOOL	Input 2^0
BIT_1:	BOOL	Input 2^1
to		
BIT_7:	BOOL	Input 2^7
BYTE_:	BYTE	Output

Fig. 13-13: Standard function block BOOL_BYTE

bool_7	bool_6	bool_5	bool_4	bool_3	bool_2	bool_1	bool_0	byte_1
0	0	0	0	0	0	0	0	16#00
...	...	...	...	...	...	...	...	...
1	1	0	0	0	1	1	1	16#C7
...	...	...	...	...	...	...	...	...
1	1	1	1	1	1	1	1	16#FF

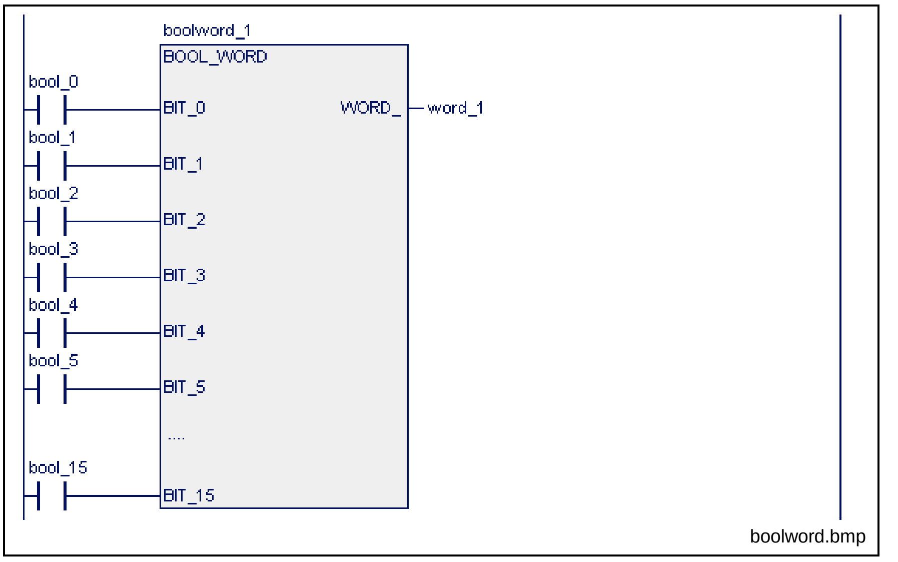
bool_byte_t.bmp

Fig. 13-14: Value assignment BOOL_BYTE

BOOL_WORD

The function block BOOL_WORD (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts 16 Boolean variables into a WORD (and, analogously, BOOL_BYTE 8 Boolean variables, BOOL_DW 32 Boolean variables). Unassigned inputs are interpreted as 0 (FALSE).

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



BIT_0:	BOOL	Input 2^0
BIT_1:	BOOL	Input 2^1
to		
BIT_15:	BOOL	Input 2^15
WORD_:	WORD	Output

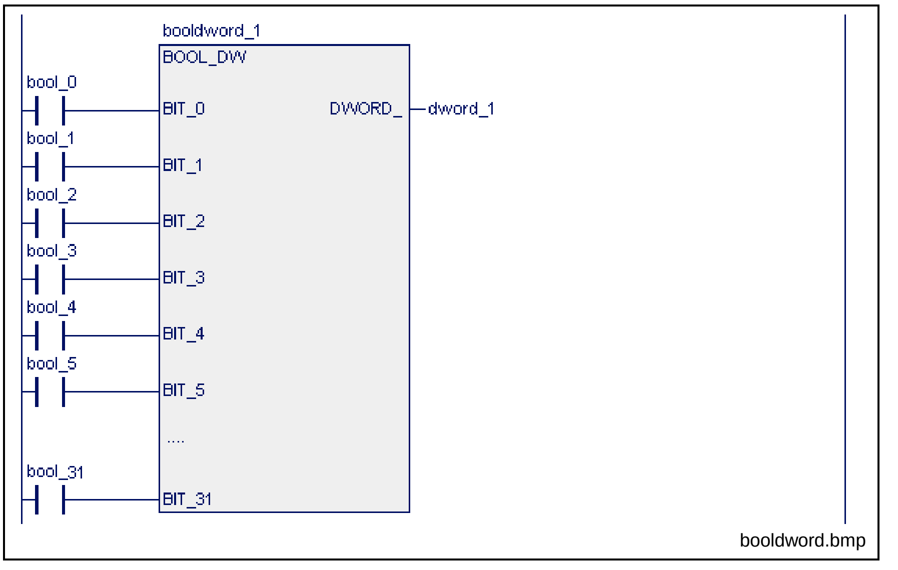
Fig. 13-15: Standard function block BOOL:_WORD

For further information see BOOL_BYTE.

BOOL_DW

The function block BOOL_DW (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts 32 Boolean variables into a DWORD (and, analogously BOOL_BYTE 8 Boolean variables and BOOL_WORD 16 Boolean variables). Unassigned inputs are interpreted as 0 (FALSE).

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



BIT_0:	BOOL	Input 2^0
BIT_1:	BOOL	Input 2^1
to		
BIT_31:	BOOL	Input 2^31
DWORD_:	DWORD	Output

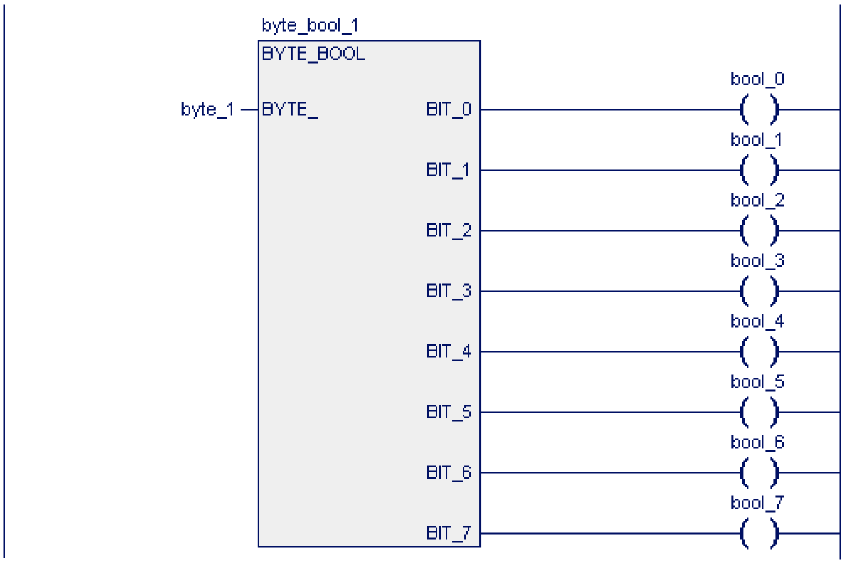
Fig. 13-16: Standard function block BOOL_DWORD

For further information see BOOL_BYTE.

BYTE_BOOL

The function block BYTE_BOOL (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts a byte into 8 Boolean variables (and, analogously, WORD_BOOL 16 Boolean variables, DWORD_BOOL 32 Boolean variables).

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



byte_bool.bmp

BYTE_:	BYTE	Input
BIT_0:	BOOL	Output 2^0
BIT_1:	BOOL	Output 2^1
to		
BIT_7:	BOOL	Output 2^7

Fig. 13-17: Value assignment BYTE_BOOL

byte_1	bool_7	bool_6	bool_5	bool_4	bool_3	bool_2	bool_1	bool_0
16#00	0	0	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...
16#C7	1	1	0	0	0	1	1	1
...	...	...	...	...	...	...	...	...
16#FF	1	1	1	1	1	1	1	1

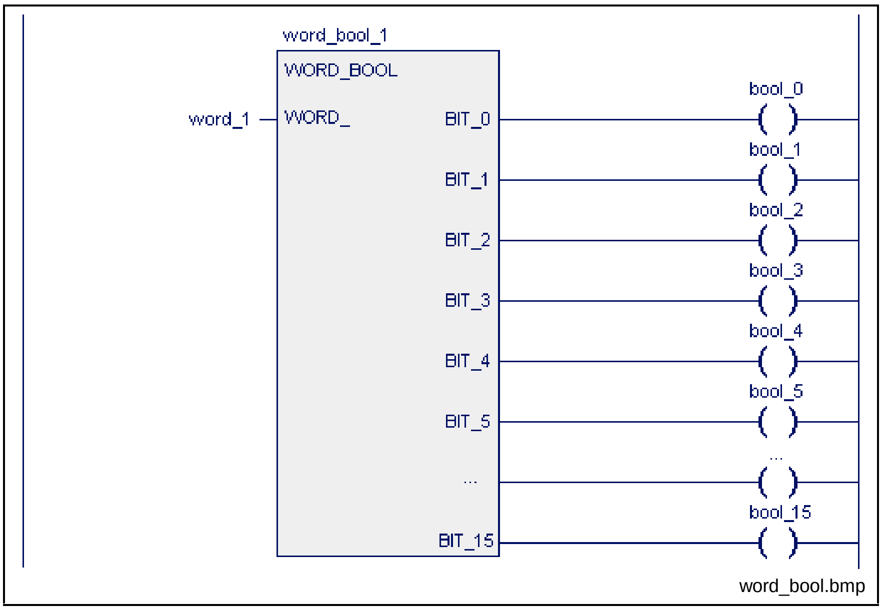
byte_bool_T.bmp

Fig. 13-18: Value assignment BYTE_BOOL

WORD_BOOL

The function block WORD_BOOL (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts a word into 16 Boolean variables (and, analogously, BYTE_BOOL 8 Boolean variables and DWORD_BOOL 32 Boolean variables).

Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



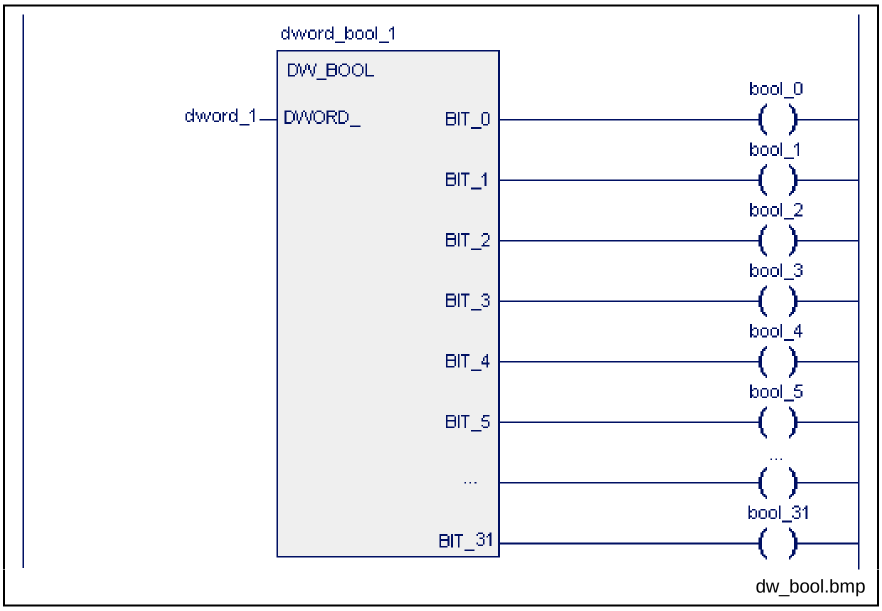
WORD_:	WORD	Input
BIT_0:	BOOL	Output 2^0
BIT_1:	BOOL	Output 2^1
to		
BIT_15:	BOOL	Output 2^15

Fig. 13-19: Standard function block WORD_BOOL

For further information see BYTE_BOOL.

DW_BOOL

The function block DW_BOOL (also see Standard Function Blocks, Collecting / Splitting Bit Strings) converts a doubleword into 32 Boolean variables (and, analogously, WORD_BOOL 16 Boolean variables and BYTE_BOOL 8 Boolean variables).
Errors cannot occur: S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0.



DWORD_:	DWORD	Input	
BIT_0:	BOOL	Output 2^0	
BIT_1:	BOOL	Output 2^1	to
BIT_31:	BOOL	Output 2^31	

Fig. 13-20: Standard function block DW_BOOL

For further information see BYTE_BOOL.

Up-Down Counter

Counters are available according to the standard defaults and as DOSPCL-compatible function blocks for the following three data types:

DOSPCL-compatible counters

New name	DOS name	Counter for	from	to
CTUD_USINT_INDR	CTUD_USI	USINT numbers	0	255
CTUD_UINT_INDR	CTUD_UIN	UINT numbers	0	65535
CTUD_INT_INDR	CTUD_INT	INT numbers	-32768	32767

EN-61131-3-compatible counters

Name	Counter for	from	to
CTUD_USINT	USINT numbers	0	255
CTUD_UINT	UINT numbers	0	65535
CTUD_INT	INT numbers	-32768	32767

The counters of a table merely differ in their counting ranges but not in their operating mode.

CTUD_USINT_INDR

CTUD_USINT_INDR (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input R_.

If TRUE is applied to it, this input is reset to CV_=0.

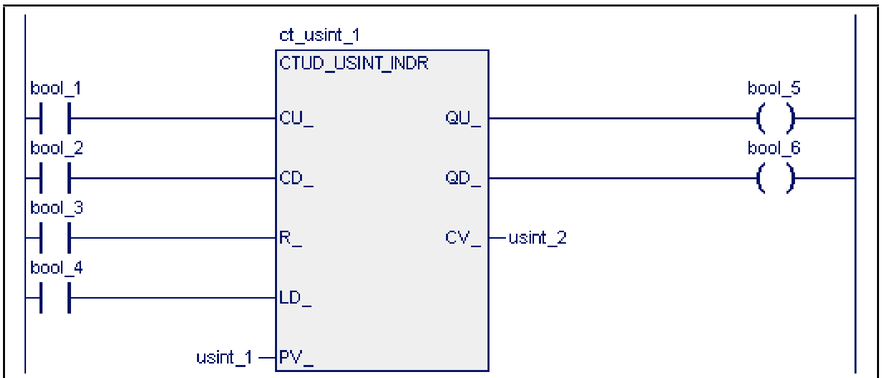
Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1. The usage of the inputs CU_ and CD_ is of no importance in this case.

Counting is possible under the marginal condition R_=0 and LD_=0.

- CV_ is incremented by 1 in every PLC cycle (contrary to EN 61131-3!) while CU_ is applied to 1 and CV_ < 255 .

Note: CV_ is decremented by 1 in every PLC cycle (contrary to EN 61131-3!) if CU_ is also applied to 0, while CD_ is 1 and CV_>0.

- Output QU_ triggers a 1 signal if CV_ >= PV_.
- Output QD_ triggers a 1 signal if CV_ = 0.



ctud_USINT_INDR.bmp

CU_:	BOOL	Up counter input (status-controlled)
CD_:	BOOL	Down counter input (status-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value at PV_)
PV_:	USINT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV_
QD_:	BOOL	TRUE if CV_=0
CV_:	USINT	Counter value

Fig. 13-21: Counter CTUD_USINT_INDR (DOS-PCL-compatible)

CTUD_UINT_INDR

CTUD_UINT_INDR (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input R_.

If TRUE is applied to it, this input is reset to CV_=0.

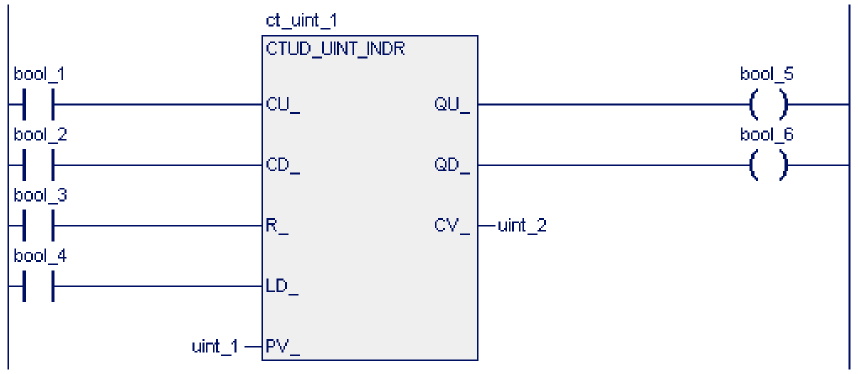
Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1. The usage of the inputs CU_ and CD_ is of no importance in this case.

Counting is possible under the marginal condition R_=0 and LD_=0.

- CV_ is incremented by 1 in every PLC cycle (contrary to EN 61131-3!) while CU_ is applied to 1 and CV_ < 65535 .

Note: CV_ is decremented by 1 in every PLC cycle (contrary to EN 61131-3!) if CU_ is also applied to 0, while CD_ is 1 and CV_>0.

- Output QU_ triggers a 1 signal if CV_ >= PV_.
- Output QD_ triggers a 1 signal if CV_ = 0.



ctud_UINT_INDR.bmp

CU_:	BOOL	Up counter input (status-controlled)
CD_:	BOOL	Down counter input (status-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value on PV_)
PV_:	UINT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV_
QD_:	BOOL	TRUE if CV_=0
CV_:	UINT	Counter value

Fig. 13-22: Counter CTUD_UINT_INDR (DOS-PCL-compatible)

CTUD_INT_INDR

The counter CTUD_INT_INDR (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input.

If TRUE is applied to it, this input is reset to CV_=0.

Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1.

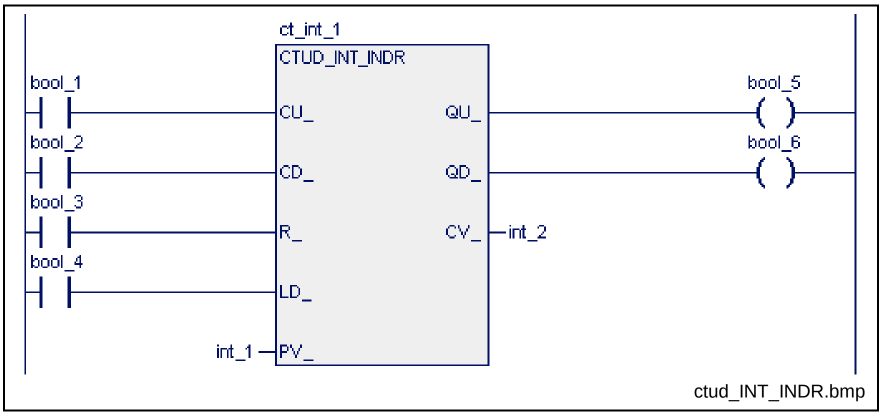
The usage of the inputs CU_ and CD_ is of no importance in this case.

Counting is possible under the marginal condition R_=0 and LD_=0.

- CV_ is incremented by 1 in every PLC cycle (contrary to EN 61131-3!) while CU_ is applied to 1 and CV_ < 32767 .

Note: CV_ is decremented by 1 in every PLC cycle (contrary to EN 61131-3!) if CU_ is also applied to 0, while CD_ is 1 and CV_>0.

- Output QU_ triggers a 1 signal if CV_ >= PV_.
- Output QD_ triggers a 1 signal if CV_ <= 0.



CU_:	BOOL	Up counter input (status-controlled)
CD_:	BOOL	Down counter input (status-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value on PV_)
PV_:	INT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV_
QD_:	BOOL	TRUE if CV_ <= 0
CV_:	INT	Counter value

Fig. 13-23: Counter CTUD_INT_INDR (DOS-PCL-compatible)

CTUD_USINT

CTUD_USINT (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input R_.

If TRUE is applied to it, this input is reset to CV_=0.

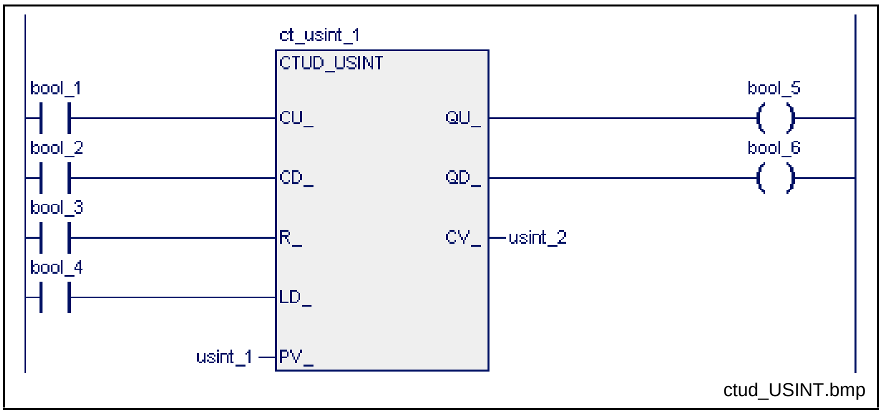
Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1. The usage of the inputs CU_ and CD_ is of no importance in this case.

Counting is possible under the marginal condition R_=0 and LD_=0.

- A 0-1 edge at CU_ increments the output CV_ by 1, as long as CV_ < 255.

Note: As long as CV_ > 0, the output CV_ is decremented by 1 with every 0-1 edge at CD_, if CU_ is also applied to 0.

- Output QU_ triggers a 1 signal if CV_ >= PV_.
- Output QD_ triggers a 1 signal if CV_ = 0.



CU_:	BOOL	Up counter input (edge-controlled)
CD_:	BOOL	Down counter input (edge-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value on PV_)
PV_:	USINT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV
QD_:	BOOL	TRUE if CV_=0
CV_:	USINT	Counter value

Fig. 13-24: Counter CTUD_USINT (EN-61131-3-compatible)

CTUD_UINT

CTUD_UINT (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input R_.

If TRUE is applied to it, this input is reset to CV_=0.

Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1. The usage of the inputs CU_ and CD_ is of no importance in this case.

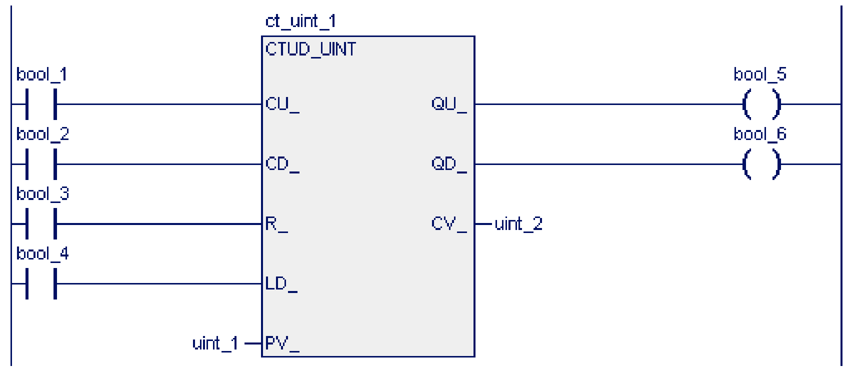
Counting is possible under the marginal condition R_=0 and LD_=0.

- A 0-1 edge at CU_ increments the output CV_ by 1, as long as CV_ < 65535.

Note: As long as CV_ > 0, the output CV_ is decremented by 1 with every 0-1 edge at CD_, if CU_ is also applied to 0.

Output QU_ triggers a 1 signal if CV_ >= PV_.

- Output QD_ triggers a 1 signal if CV_ = 0.



ctud_UINT.bmp

CU_:	BOOL	Up counter input (edge-controlled)
CD_:	BOOL	Down counter input (edge-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value on PV_)
PV_:	UINT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV_
QD_:	BOOL	TRUE if CV_=0
CV_:	UINT	Counter value

Fig. 13-25: Counter CTUD_UINT (EN-61131-3-compatible)

CTUD_INT

The counter CTUD_INT (also see Standard Function Blocks, Up-Down Counter) is provided with a dominating reset input.

If TRUE is applied to it, this input is reset to CV_=0.

Provided the marginal condition R_=0 is fulfilled, the preset value present at PV_ is applied as long as LD_ is 1.

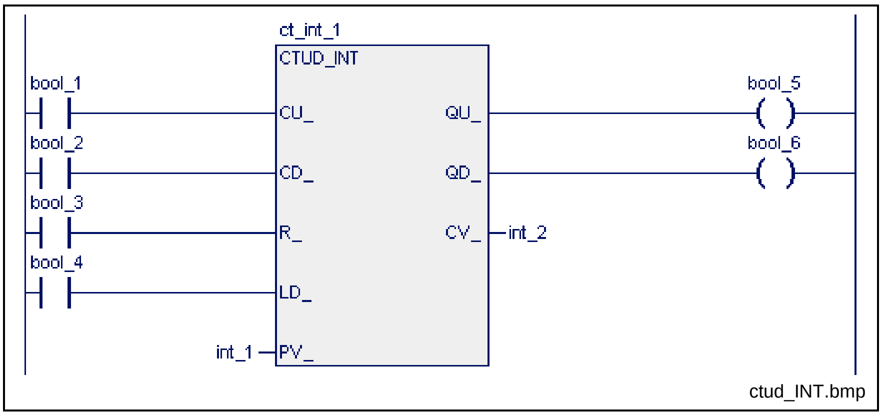
The usage of the inputs CU_ and CD_ is of no importance in this case.

Counting is possible under the marginal condition R_=0 and LD_=0.

- A 0-1 edge at CU_ increments the output CV_ by 1, as long as CV_ < 32767.

Note: As long as CV_ > -32768, the output CV_ is decremented by 1 with every 0-1 edge at CD_, if CU_ is also applied to 0.

- Output QU_ triggers a 1 signal if CV_ >= PV_.
- Output QD_ triggers a 1 signal if CV_ <= 0.



CU_:	BOOL	Up counter input (edge-controlled)
CD_:	BOOL	Down counter input (edge-controlled)
R_:	BOOL	Reset input (dominant)
LD_:	BOOL	Load input (for value on PV_)
PV_:	INT	Preset value
QU_:	BOOL	TRUE if CV_ >= PV_
QD_:	BOOL	TRUE if CV_ <= 0
CV_:	INT	Counter value

Fig. 13-26: Counter CTUD_INT (EN-61131-3-compatible)

Time Stages for Pulses, On-Delay and Off-Delay Timer Function Blocks

The following time stages are provided:

- TP - pulse
- TON - on-delay timer function block
- TOFF - off-delay timer function block
- FLASH - free running clock generator

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

TP

(Also see Standard Function Blocks, Time Stages for Pulses, On-Delay and Off-Delay Timer Function Blocks)

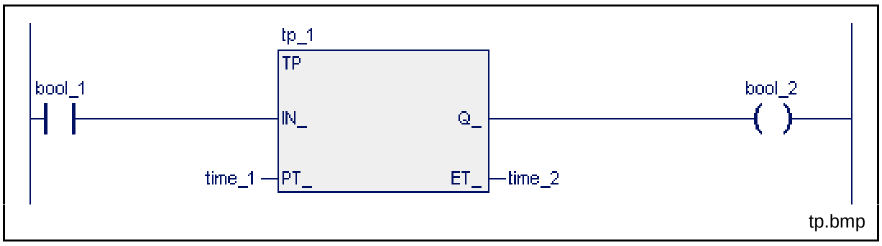
A single TP pulse appears a the output Q_, when a 0-1 transition is implemented at the input IN_.

The length of the input pulse is of no importance.

Retriggering of the time stage is not possible, i.e. pulses at the input are ignored as long as the pulse is applied to the output .

The current runtime of the pulse is counted at output ET_. The value remains active until the 1-0 transition takes place at the input.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).



IN_:	BOOL	Input
PT_:	TIME	Pulse width
Q_:	BOOL	Output
ET_:	TIME	Current time value

Fig. 13-27: Standard function block TP

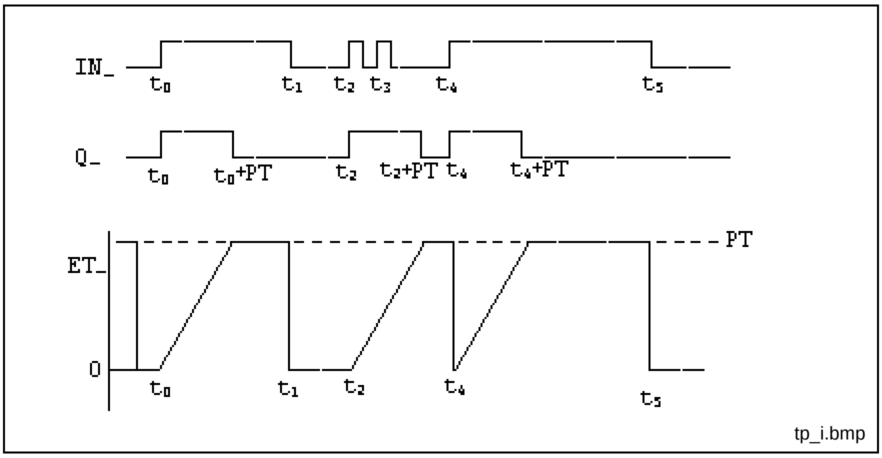


Fig. 13-28: Diagram of time stage TP (pulse)

TON

(Also see Standard Function Blocks, Time Stages for Pulses, On-Delay and Off-Delay Timer Function Blocks)

A 1-signal delayed by PT is applied to output Q after a 0-1 transition has been implemented at the input IN_.

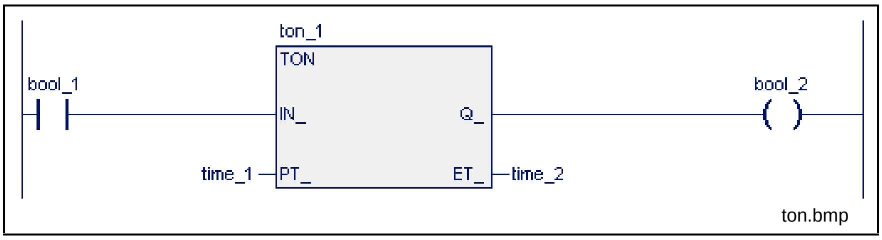
Output Q falls back to 0 if the input is applied to 0 again.

If the 1-signal at the input is shorter than PT, a 1-signal cannot be generated at the output.

The output ET indicates the current delay time.

The end value is preserved until the signal at the input is applied to 0 again.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).



IN_:	BOOL	Input
PT_:	TIME	On-delay timer function block
Q_:	BOOL	Output
ET_:	TIME	Current time value

Fig. 13-29: Standard function block TON

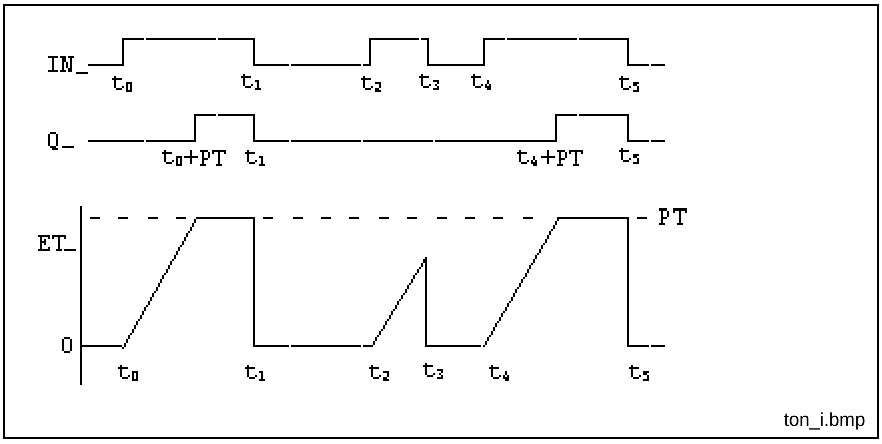


Fig. 13-30: Diagram of time stage TON (with on-delay)

TOFF

(Also see Standard Function Blocks, Time Stages for Pulses, On-Delay and Off-Delay Timer Function Blocks)

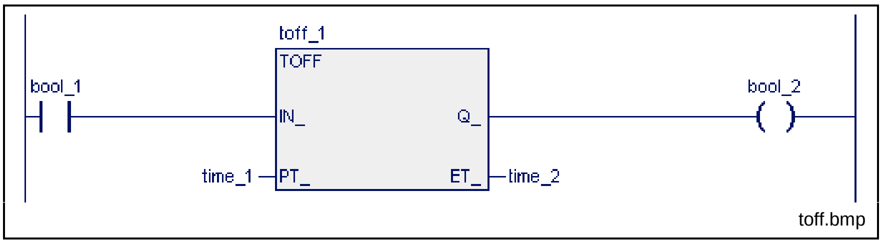
If a 0-1 transition is implemented at the input IN_, a 1-signal is applied to output Q.

If the signal at the input drops from 1 to 0, the 1-signal at output Q_ is still active for the time period PT and then falls back to 0.

The process restarts without any interruption if the input signal becomes 1 again during the delay time PT. Retriggering of the time stage is possible.

The output ET indicates the current delay time. The end value is preserved until the signal at the input is applied to 1 again.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).



IN_:	BOOL	Input
PT_:	TIME	Off-delay timer function block
Q_:	BOOL	Output
ET_:	TIME	Current time value

Fig. 13-31: Standard function block TOFF

Diagram of time stage TOFF (with off-delay)

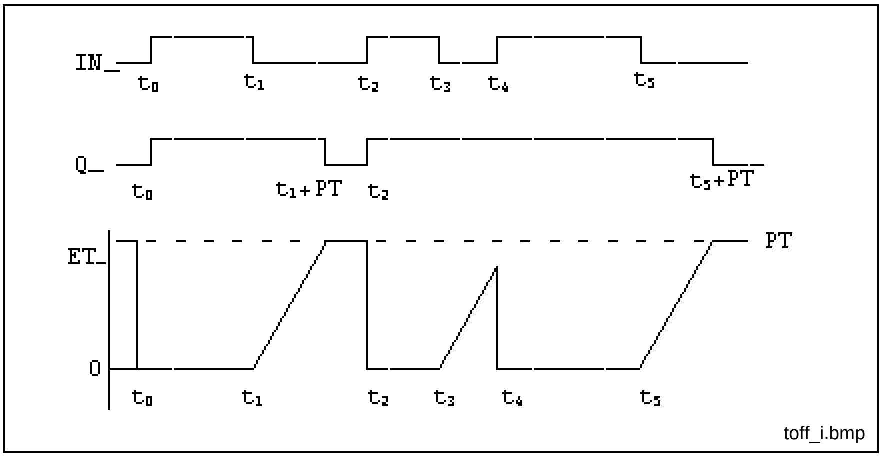
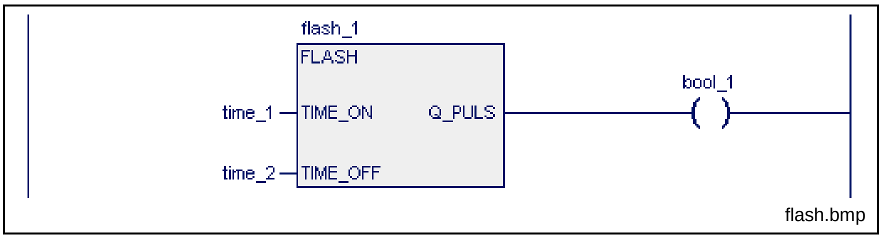


Fig. 13-32: Diagram of time stage TOFF (with off delay)

FLASH

The function block FLASH (also see Standard Function Blocks, Time Stages for Pulses, On-Delay and Off-Delay Timer Function Blocks) is operated as a free running clock generator. Pulse and pause times can be set using the input variables.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).



TIME_ON:	TIME	Switch-on time (PULSE)
TIME_OFF:	TIME	Switch-off time (PAUSE)
Q_PULSE:	BOOL	Output

Fig. 13-33: Standard function block FLASH

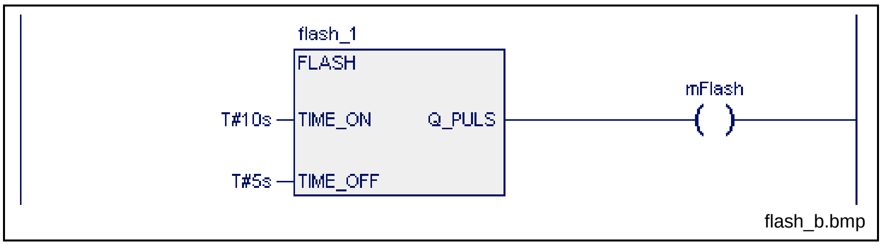


Fig. 13-34: FLASH application

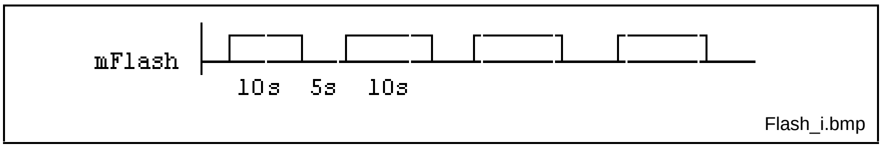


Fig. 13-35: Time course relating to the example above

Error handling

Errors cannot occur:

S#ErrorFlg: 0, S#ErrorNr: 0, S#ErrorTyp: 0

Function Blocks for Date and Time

The functions blocks

- DATE_RD and
- TOD_RD

serve for reading the current date and the current time. The time is provided with a resolution of one second.
The function block interfaces are exactly defined. When a function block is invoked, the programmer merely has to connect the individual signals. The function blocks are processed by an assembler program. What goes on inside the function blocks therefore cannot be represented by the programming languages IL, LD or FB.

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

DATE_RD

(Also see Standard Function Blocks, Function Blocks for Date and Time)

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Reading the date

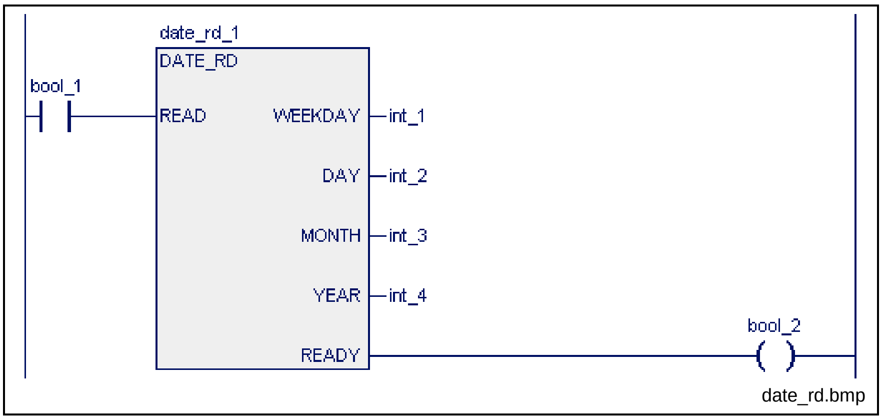


Fig. 13-36: Reading the date DATE_RD

Name	Type	Comment
READ:	BOOL	0 - FB not active 1 - Activation of reading the date
WEEKDAY:	INT	Weekday 0 - Sunday 1 - Monday 2 - Tuesday 3 - Wednesday 4 - Thursday 5 - Friday 6 - Saturday
DAY:	INT	Day (1…31)
MONTH:	INT	Month (1…12)
YEAR:	INT	Year (1980…2035)
READY:	BOOL	0 - Date invalid 1 - Date valid

Date operating principle

The date or time is read by setting the READ input. The result is made available as function block output.

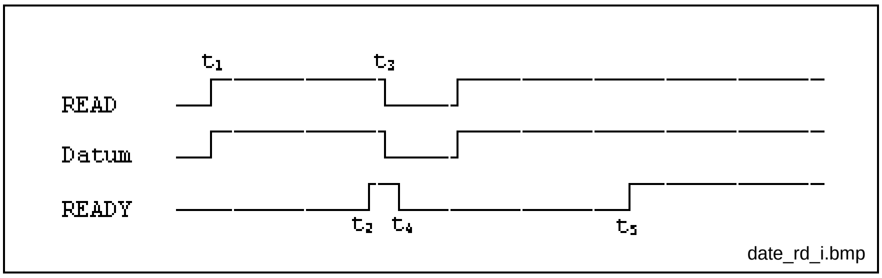


Fig. 13-37: Time course when reading the date DATE_RD

1. Reading of date (and time) is initialized by setting the READ input in the first PLC cycle.
2. The activated READY output indicates that transmission of date and time is completed.
3. If date and time are to be read only once, the READ input may now be cleared.
4. Clearing the READ input also clears the READY output of the function block.
5. If the READ input is still applied statically, date and time will be updated after one second.

Note: The period between setting of the READ input and setting of the READY output cannot exceed one second. This applies also after start of the PLC program.

Error handling for date (and time)

The function block DATE_RD does not generate any errors. S#ErrorFlg = 0, S#ErrorTyp = 0, S#ErrorNr = 0.

TOD_RD

(also see Standard Function Blocks, Function Blocks for Date and Time)

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Reading the time

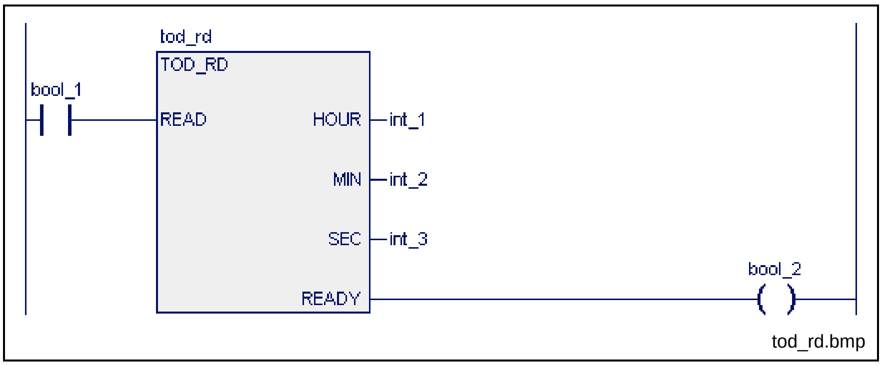


Fig. 13-38: Reading the time TOD_RD

Name	Type	Comment
READ:	BOOL	0 - FB not active 1 - Activation of reading the time
HOUR:	INT	Hours (0…23)
MIN:	INT	Minutes (0...59)
SEC:	INT	Seconds (0...59)
READY:	BOOL	0 - Time invalid 1 - Time valid

(Date and) time operating principle

Time is read by setting the READ input. The result is made available as function block output.

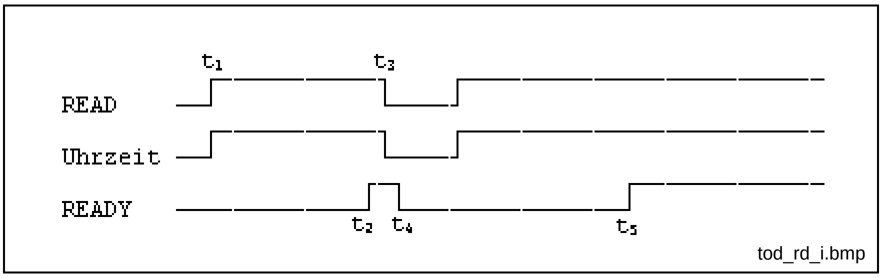


Fig. 13-39: Time course for reading the time TOD_RD

1. Reading of date and time is initialized by setting the READ input in the first PLC cycle.
2. The activated READY output indicates that transmission of date and time is completed.
3. If date and time are to be read only once, the READ input may now be cleared.
4. Clearing the READ input also clears the READY output of the function block.
5. If the READ input is still applied statically, date and time will be updated after one second.

Note: The period between setting of the READ input and setting of the READY output cannot exceed one second. This applies also after start of the PLC program.

Error handling for (date and) time

The function block TOD_RD does not generate any errors. S#ErrorFlg = 0, S#ErrorTyp = 0, S#ErrorNr = 0.

13.3 Firmware Function Blocks

INTERBUS, Function Blocks

The following user function blocks are available in the PLC programming interface for the control of the INTERBUS system (IBM2, G4):

- Preparation for Control of an INTERBUS
- CLR_DIAG: Clear diagnosis parameter register
- SEG_OFF: Deactivate segment
- SEG_ON: Activate segment
- **Fehler! Verweisquelle konnte nicht gefunden werden.**: Stop data transmission
- START_D: Start data transmission
- Excerpt from the Description of the Standard Registers
- Program Example for Control of an INTERBUS

These blocks permit INTERBUS modules to be exchanged during bus operation or INTERBUS sections to be put in and out of operation separately. Moreover, it is possible to clear the diagnosis parameter register, to allow a reaction to active INTERBUS messages (peripheral influences etc.).

PCP Function Blocks for the Parameter Channel of the INTERBUS

The PCP function blocks are provided on the basis of the PCP services.

- PCP_INITIATE: Establish the connection to a PCP slave
- PCP_READ: Read out object values
- PCP_WRITE: Change device parameters
- PCP_GET_OD: Read out several object descriptions
- PCP_IDENTIFY: Read out the "name plate"
- PCP_ABORT: Abbort a connection
- Error Messages of the Communication with PCP Function Blocks (General Information)

PROFIBUS DP, Function Blocks

The following Firmware Function Blocks are available in the PLC programming interface for controlling a PROFIBUS:

- Status information on the PROFIBUS master: DPM_STATE
- Single diagnosis of a PROFIBUS slave: DPM_SLDIAG
- Program Example for Control of a PROFIBUS

ASI Bus, Function Block with Data Type

The following Firmware Function Blocks are available in the PLC programming interface for controlling a ASI Bus:

- ASIM_SLDIAG Single diagnosis of an ASI Bus slave

Serial Interfaces, Function Blocks (also see firmware data types: COM)

The following firmware function blocks are available for control of serial interfaces:

- OPEN_COM Open serial interface
- CLOS_COM Close serial interface
- WR_BYTE Write data byte to serial interface
- RD_BYTE Read data byte from serial interface
- CTRL_COM - Determine status of serial interface
- WR_STR Write data string to serial interface
- RD_STR Read data string from serial interface
- CLR_COM Clear receive buffer and transmit buffer of serial interface
- Error Handling of Function Blocks for Serial Interfaces
- Program Example for Control of serial Interfaces

GUI_SK functionality

The GUI_SK functionality describes a mechanism for a screen-oriented machine operation. Using a list, the BTV20 / 30 machine function keys R1 to R8 and L1 to L8 can be assigned to any Boolean PLC variable upon each screen change (*soft keys*).

- Function Blocks for the HMI Interface (GUI_SK16))

Coupling miniature control panels to the PLC

WinPCL provides firmware blocks to couple miniature control panels to the PLC.

- BTXX, BTXX_2

MotionControl-Extension of the PLC

To extend the functional range of an ISP in relation to a Motion Control the PLC interface provides the following function blocks:

- CALC_LINEAR_Y - calculates linearized for a default X value of a predetermined X-Y value table the respective Y value.
- PID_CONTROL - PID-Control (P-, I-, PI-, PD-, PID-Control)
- AVERAGE_REAL - calculates the floating average value form maximum 64 REAL values.
- AVERAGE_DINT - calculates the floating average value from maximum 1023 DINT values.
- PT2_FILTER - PT2-Filter (low pass filtering of a signal)

INTERBUS, Function Blocks

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Preparation for Control of an INTERBUS

The files are residing in the following folders, as **archives to support the firmware functionality** for WinHMI and WinPCL:

- ...Mtgui\BasicData\TEMPLEATES\ibs_control.apv
- ...WinPCL\BasicData\TEMPLATES\ibs_control.apv

The application of the function blocks (also see INTERBUS, Function Blocks) requires the following activities:

- The current bus configuration has to be entered or read back in the bus configurator IBS CMD G4.

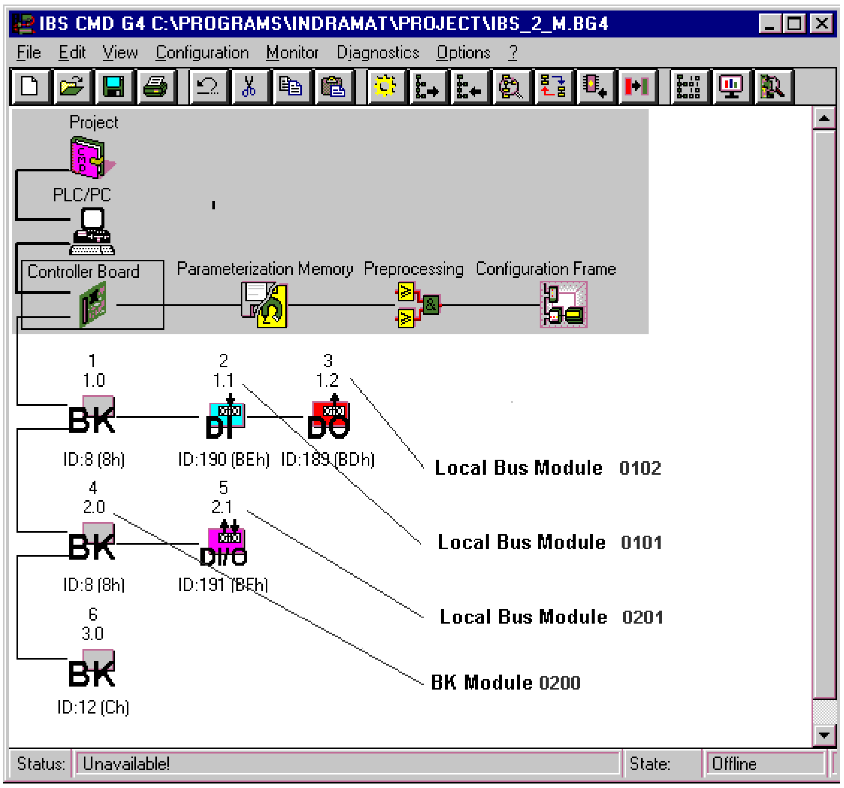
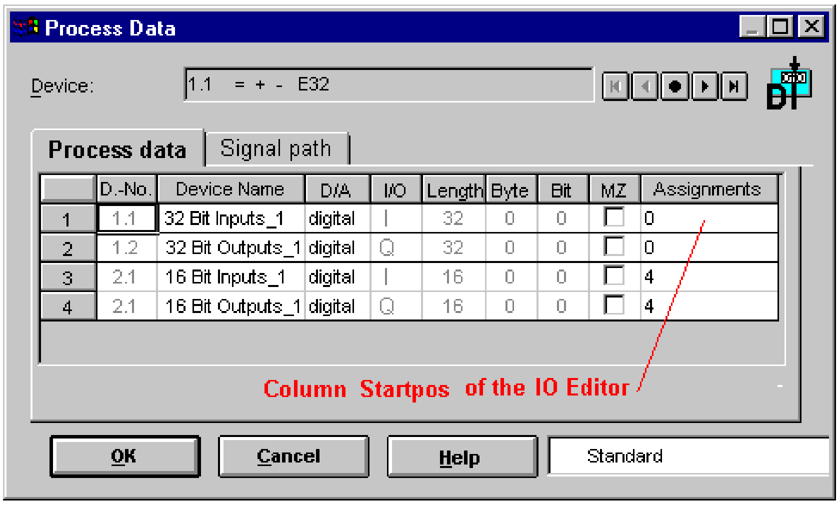


Fig. 13-40: Current bus configuration in IBS CMD G4 (example)

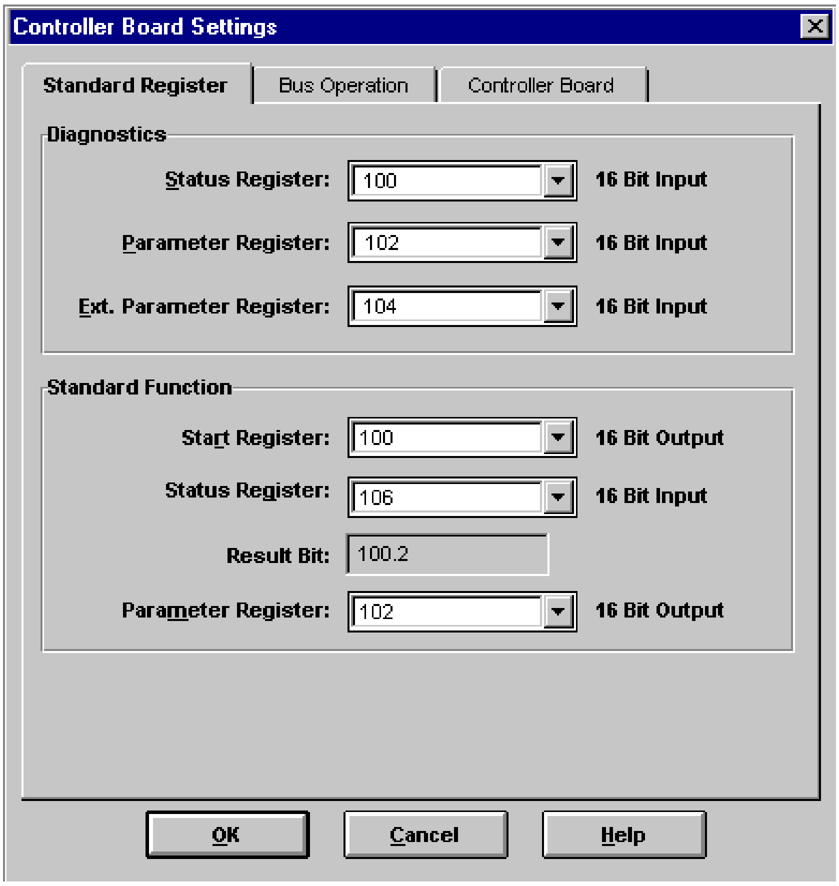
- Check of the (automatically) assigned process data



cmd_Prozessdaten.bmp

Fig. 13-41: Process data, addresses assigned automatically in the example

- The necessary addresses have to be defined for the standard registers of the bus.



Einstellung_CMD.bmp

Fig. 13-42: Addresses of the INTERBUS standard register (example)

Note: Please note, that the addresses of these registers must be entered with sufficient spacing for possible bus extensions after the last bus device.

- The logic numbers of the INTERBUS devices and the INTERBUS registers must be entered in the I/O editor of the resource:

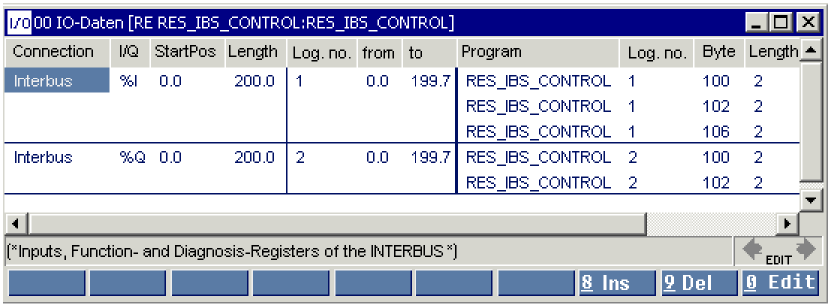


Fig. 13-43: IO editor of the resource with bus devices and registers

- The registers are to be declared as variables in the resource and to be enabled as global variables (copy from the sample resource!).

Note: The names of the variables (register) must be applied exactly, because they are accessed in the function blocks by means of VAR_EXTERNAL!

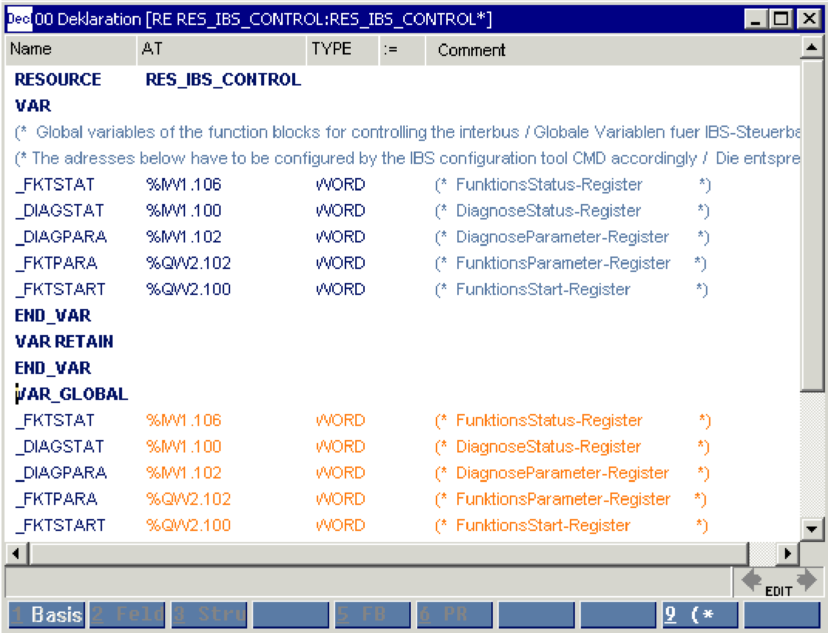


Fig. 13-44: Declaration of the registers at resource level and enable

- The blocks are available as user function blocks. They must be imported to the programming interface using the menu File / Archive / Load archive.
    - ...Mtgui\BasicData\TEMPLEATES\ibs_control.apv or
    - ...WinPCL\BasicData\TEMPLATES\ibs_control.apv
- The function blocks are declared in the declaration editor of the respective program intended to use them. The registers must be declared with same names in the VAR EXTERNAL area.

Continued in the section "Program Example for Control of an INTERBUS".

CLR_DIAG

(also see INTERBUS, Function Blocks)

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Clear diagnosis registers

If a rising edge is applied to the ENABLE input, a function is triggered on the interface module, which clears the diagnosis status and the diagnosis parameter register. If no error message or fault of the INTERBUS is active after clearing, the diagnosis parameter register has the value 16#0000, while the diagnosis status register shows the current status of the INTERBUS interface module. Otherwise the diagnosis status register contains the type of the fault and the diagnosis parameter register provides additional information on this fault.

clr_diag.bmp

ENABLE:	(BOOL)	Clearing the diagnosis register (edge-controlled)
OK:	(BOOL)	Function execution positive / negative
ACTIVE:	(BOOL)	Function execution active

Fig. 13-45: Rexroth function block CLR_DIAG

If a rising edge is applied to the ENABLE input, the function block is activated and the ACTIVE output becomes TRUE. The ACTIVE output becomes FALSE after successful execution of the function. With the falling edge, the result of the executed function is indicated in the OK signal for the duration of a PLC cycle . If TRUE is applied to OK, the two registers were successfully cleared; if not, execution of the function failed. This is indicated in the set bit USER of the diagnosis status register. The diagnosis parameter register then contains additional information on the active fault.

Note: The function block must be activated only if no other function block (SEG_OFF, SEG_ON, START_D, STOP_D) is active.

SEG_OFF

(also see INTERBUS, Function Blocks)

Note: This block may not be declared in the retain area (see also Limitation of the Declaration of Function Blocks in the Retain Area).

Deactivate segment

If a rising edge is applied to the ENABLE input, a function is triggered on the interface module, which deactivates the segment defined at SEG_POS during bus operation. The outputs of the disconnected devices are reset.

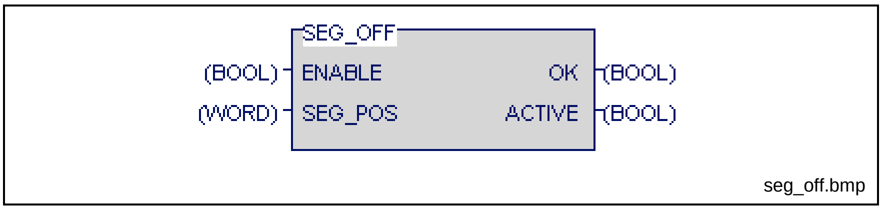
If a bus terminal (e.g. 16#0200) is indicated as parameter at the input SEG_POS, the local bus interface and the continuing remote bus interface are deactivated; this results in a deactivation of all physical devices behind this bus - that means also the devices of the corresponding local bus segment. The INTERBUS ring can be opened starting with the defined bus terminal, without generating a bus error.

If a local bus device (e.g. 16#0201) is indicated, only this local bus is deactivated. The continuing remote bus of this bus terminal remains active.

The BSA bit in the diagnosis status register is set after successful deactivation.

Note: A deactivation of individual local bus devices is not possible. Always all devices of the corresponding local bus are deactivated.

Devices 0.0 and 1.0 may not be deactivated. Indicating these devices as parameters results in the USER error 0x0A20. The USER bit in the diagnosis status register is set.



ENABLE:	(BOOL)	Deactivating the segment (edge-controlled)
SEG_POS:	(BOOL)	Segment SS position PP in format 16#SSPP
OK:	(BOOL)	Function execution positive / negative
ACTIVE:	(BOOL)	Function execution active

Fig. 13-46: Rexroth function block SEG_OFF

If a rising edge is applied to the ENABLE input, the function block is activated and the ACTIVE output becomes TRUE. The ACTIVE signal becomes FALSE after successful execution of the function. The result of the executed function is indicated in the OK signal for the duration of a PLC cycle with the falling edge of the ACTIVE signal. If TRUE is applied to OK, the segment was successfully deactivated; if not, execution of the function failed. This is indicated in the set USER bit of the diagnosis status register. The diagnosis parameter register then contains additional information on the active fault.

Note: The function block must be activated only if no other function block (CLR_DIAG, SEG_ON, START_D, STOP_D) is active.

SEG_ON

(also see INTERBUS, Function Blocks)

Note: This block may not be declared in the retain area (see also Limitation of the Declaration of Function Blocks in the Retain Area).

Switching on segment

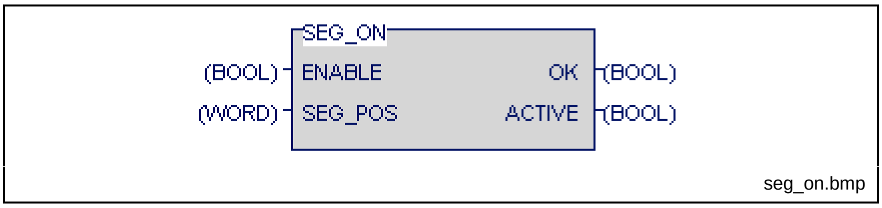
A function is released on the interface module with rising edge on the input ENABLE, which activates the segment defined at SEG_POS during the bus operation.

If a bus terminal (e.g. 16#0200) is defined as a parameter on input SEG_POS, the local bus interface and the continuing remote bus interface are activated. This results in an activation of all physical users behind the bus - that means the users of the corresponding local bus segment are activated, too.

Note: Connection of individual local bus devices is not possible. Always all devices of the corresponding local bus are connected.

Devices 0.0 and 1.0 may not be connected. Indicating these devices as parameters results in the USER error 0x0A20. The USER bit in the diagnosis status register is set.

Before activating a segment it has to be ensured that the bus structure corresponds to that bus which was available before the segment was deactivated. Otherwise a bus error is generated which leads to a deactivation of the bus.



ENABLE:	(BOOL)	Activating the segment (edge-controlled)
SEG_POS:	(BOOL)	Segment SS position PP in format 16#SSPP
OK:	(BOOL)	Function execution positive / negative
ACTIVE:	(BOOL)	Function execution active

Fig. 13-47: Rexroth function block SEG_ON

If a rising edge is applied to the ENABLE input, the function block is activated and the ACTIVE output becomes TRUE. The ACTIVE signal becomes FALSE after successful execution of the function. The result of the executed function is indicated in the OK signal for the duration of a PLC cycle with the falling edge. If TRUE is applied to OK, the segment was successfully activated; if not, execution of the function failed. This is indicated in the set USER bit of the diagnosis status register. The diagnosis parameter register then contains additional information on the active fault.

Note: The function block must be activated only if no other function block (SEG_OFF, CLR_DIAG, START_D, STOP_D) is active.

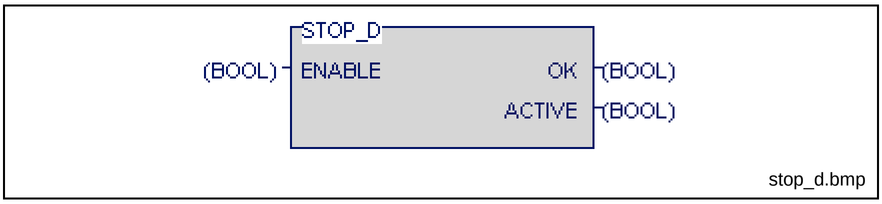
STOP_D

(also see INTERBUS, Function Blocks)

Note: This block may not be declared in the retain area (see also Limitation of the Declaration of Function Blocks in the Retain Area).

Stop data transmission

If a rising edge is applied to the ENABLE input, a function is triggered on the interface module, which stops the data transmission and resets the outputs.



ENABLE:	(BOOL)	Stopping the data transmission (edge controlled)
OK:	(BOOL)	Function execution positive / negative
ACTIVE:	(BOOL)	Function execution active

Fig. 13-48: Rexroth function block

If a rising edge is applied to the ENABLE input, the function block is activated and the ACTIVE output becomes TRUE. The ACTIVE signal is deleted after successful execution of the function. The result of the executed function is indicated in the OK signal with this falling edge. If TRUE is applied to OK, the data transmission was successfully stopped; if not, execution of the function failed. This is indicated in the set USER bit of the diagnosis status register. The diagnosis parameter register then contains additional information on the active fault.

Note: The function block must be activated only if no other function block (SEG_OFF, SEG_ON, START_D, CLR_DIAG) is active.

START_D

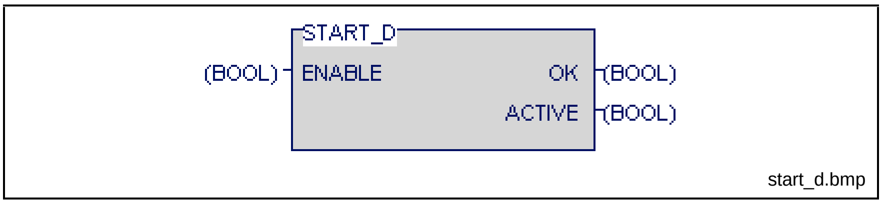
(also see INTERBUS, Function Blocks)

Note: This block may not be declared in the retain area (see also Limitation of the Declaration of Function Blocks in the Retain Area).

Start data transmission

If a rising edge is applied to the ENABLE input, a function is triggered on the interface module, which starts the data transmission.

Note: Activating this function block while data transmission has already been started (RUN bit is set), results in a USER error (USER bit set). The diagnosis parameter register then contains the value 0x0A02.



ENABLE:	(BOOL)	Starting the data transmission (edge controlled)
OK:	(BOOL)	Function execution positive / negative
ACTIVE:	(BOOL)	Function execution active

Fig. 13-49: Rexroth function block START_D

If a rising edge is applied to the ENABLE input, the function block is activated and the ACTIVE output becomes TRUE. The ACTIVE signal is deleted after successful execution of the function. The result of the executed function is indicated in the OK signal with this falling edge. If TRUE is applied to OK, the data transmission was successfully started; if not, the execution of the function failed. This is indicated in the set USER bit of the diagnosis status register. The diagnosis parameter register then contains additional information on the active fault.

Note: The function block must be activated only if no other function block (SEG_OFF, SEG_ON, CLR_DIAG, STOP_D) is active.

Excerpt from the Description of the Standard Registers

(Also see INTERBUS, Function Blocks)

Note: Read only access is allowed to standard registers only. If the registers are overwritten, correct execution of the function blocks cannot be ensured, and the bus system may show an unpredictable behavior.

General information

The diagnosis status register and the diagnosis parameter register are available on the PLC programming interface for an INTERBUS diagnosis. They map the current status of the INTERBUS system in the user program, so that the status of the bus system, reasons for the error and further information can be evaluated. Furthermore three standard function registers are provided to allow the execution of predefined functions on the interface module.

The following standard registers are made available by the INTERBUS connection module:

Diagnosis status register / diagnosis parameter register

A status of the INTERBUS interface module is assigned to each bit in the diagnosis status register. The statuses in the error bits (USER, PF, BUS, CTRL) are explained in more detail in the description of the diagnosis parameter register. This register is always written again if one of the above mentioned error bits is active. In case of an error the diagnosis parameter register either contains the error location, segment and position of the IBS number or the error type. Otherwise the diagnosis parameter register has the value 0x0000.

Bit (15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0)		Bit in the diagnostic status register
0	USER	User error
1	PF	Peripheral failure
2	BUS	Bus failure
3	CTRL	Error in connection module / hardware
4	DETECT	Diagnosis routine is active
5	RUN	Data transmission is active
6	ACTIVE	Chosen configuration is ready for operation
7	READY	Connection module is ready for operation
8	BSA	Bus segment deactivated
9	BASP/SYS_FAIL/CLAB/STOP	Stop control system, reset outputs
10	RESULT	Negative processing of standard functions
11	SY_RESULT	Synchronization error occurred
12	DC_RESULT	Faulty data cycles
13	WARNING	Fixed waiting time exceeded
14	QUALITY	Fixed error density exceeded
15	SDSI	Active messages for control

Fig. 13-50: Structure of the diagnosis status register

Possible use

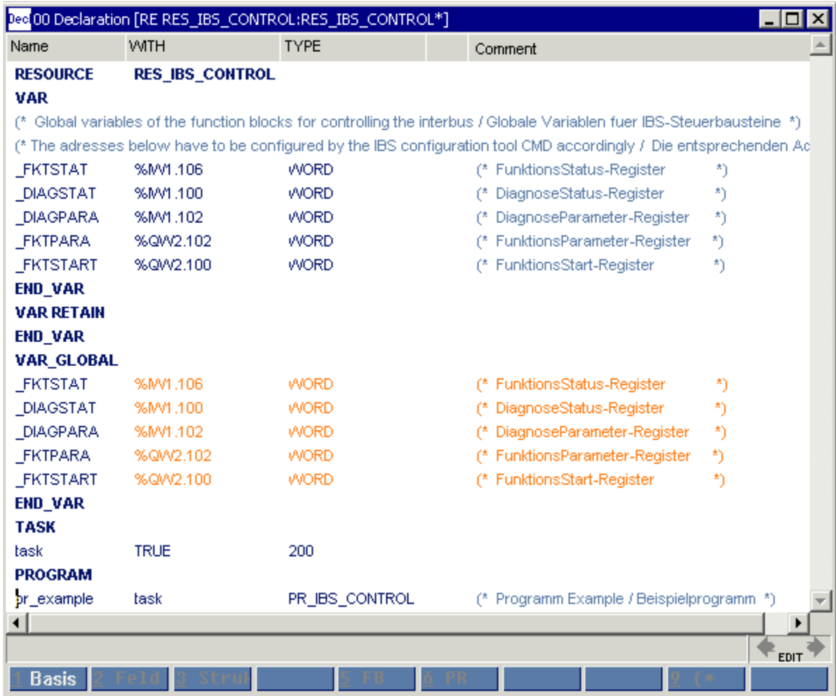
The bits which are available in the diagnosis status register can be used for monitoring of the INTERBUS. Bit PF, for example, is set if a module signals a failure in the peripheral equipment. In addition to other causes, failures in the peripheral equipment may be triggered, if the external voltage supply is missing at an output module or if an INTERBUS ring fails in connection with a gateway. The module which initiated this failure can be read via the diagnosis parameter register.

Standard function register

By these registers, predefined functions on the INTERBUS interface module can be executed and monitored by setting of a certain bit. The required function is selected via the function start register, while the corresponding parameters have to be transmitted into the function parameter register in relation to the selected function. The execution of the function is indicated in a bit of the function status register.

Program Example for Control of an INTERBUS

This program example continues the section "Preparation for Control of an INTERBUS ". It uses the resource programmed there (also see INTERBUS, Function Blocks).



Dekl_IBS_Reg_01.bmp

Fig. 13-51: Resource complete for INTERBUS example

For the program "interbus: IBS_CMD_PR", the declaration on the following page is used.

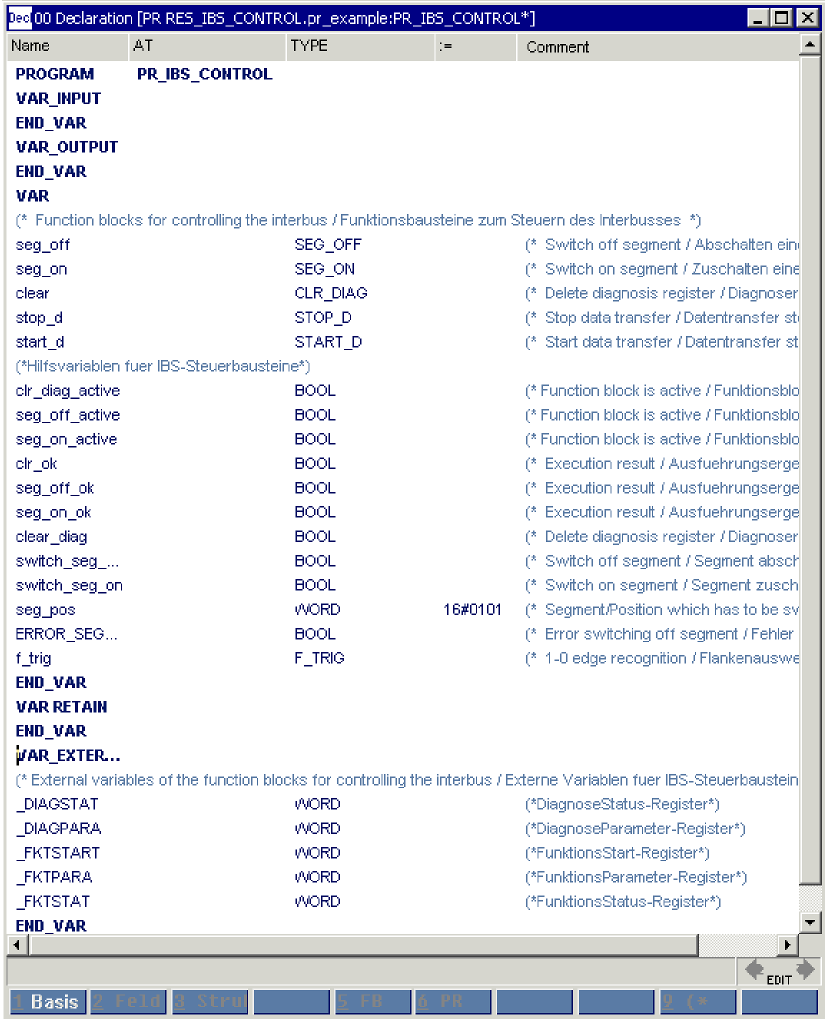
It contains the declaration of the instances of the IBS function blocks and the variable for their activation.

The INTERBUS registers are not required for the example at program level.

The INTERBUS registers, however, are evaluated in the following function blocks:

- clear: CLR_DIAG
- seg_off: SEG_OFF
- seg_on SEG_ON
- stop_d STOP_D
- start_d START_D

For that reason, they are declared there by means of VAR EXTERNAL.



ibs_cmd_pr_dekl.bmp

Fig. 13-52: Declaration for the program "interbus: IBS_CMD_PR"

The implementation of the program is attached as ladder diagram on the next page.

The first network serves for deleting the diagnosis register.

The second and third networks are used for activating and deactivating the INTERBUS device 0101 (I/O editor, logic number 2).

The forth network evaluates the errors.

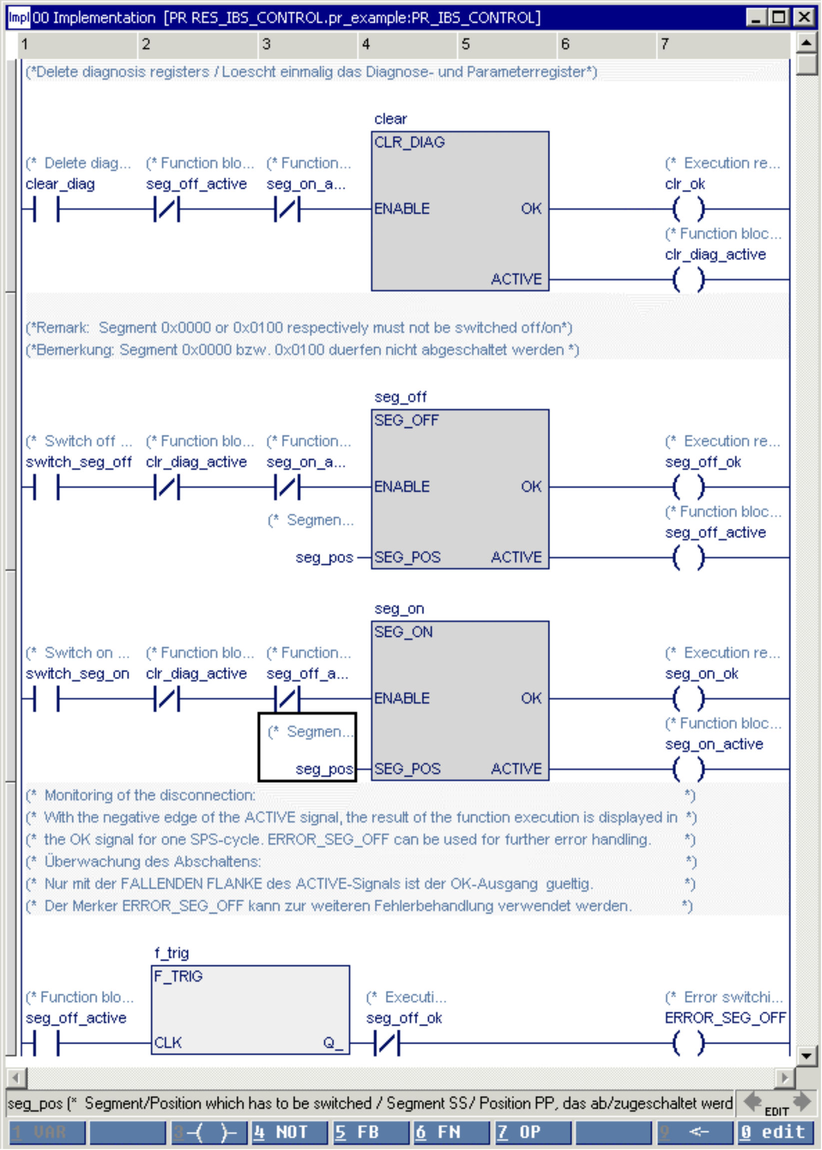


Fig. 13-53: Implementation for the program "interbus: IBS_CMD_PR"

PCP Function Blocks for the Parameter Channel of the INTERBUS

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Process data channel and parameter data channel

Process data and parameter data are transmitted in the INTERBUS system via this two independently transferring data transmission channels. Depending on its function not every slave has to support both channels. However, intelligent devices that transmit process and parameter data, like frequency converters, drives etc., need both channels. The process data channel allows to access to the cyclic process data. If required, the parameter data are transmitted acyclically via the parameter data channel. The parameter data channel is integrated in the transmission protocol.

In the summation frame of the INTERBUS remain gaps at the locations at which the PCP devices are addressed. If a transmission of parameter data is necessary, the data block is divided up in individual segments, that are as large as the gap. Up to now segment sizes of 1, 2 or 4 words are possible. One of these segments is transmitted in each INTERBUS cycle until the whole data block has been sent.

Implemented PCP function blocks

The PCP function blocks are provided on the basis of the PCP services.

- PCP_INITIATE: Establish the connection to a PCP slave
- PCP_READ: Read out object values
- PCP_WRITE: Change device parameters
- PCP_GET_OD: Read out several object descriptions
- PCP_IDENTIFY: Read out the "name plate"
- PCP_ABORT: Abort a connection

Note: A second service request must not be send to the same device, if the acknowledgement of the first service request was not received yet (parallel services).

However, it is possible to transmit PCP services simultaneously to different devices, if an own FB instance was declared for each device.

If several instances of the same PCP block type addressing the same device (same communication reference CR) are used, at each case, the input INVOKE_ID is to be wired with an unambigous number.

It has to be ensured that the execution of the PCP FBs with PCP FBs that are implemented in other tasks is not interrupted.

Operating principle

The operating principle is the same for all FBs.

With the rising edge of input EXECUTE the corresponding function block is activated, the output ACTIVE is set and the service request is sent via the INTERBUS master to the PCP device specified at input COMM_REF.

The response was received with the rising edge of the READY output. The output data of the FB are valid. At the same time, ACTIVE changes to FALSE. As long as ACTIVE is set, the outputs of the FB remain reset.

If input EXECUTE is reset as long as ACTIVE is set, the FB remains active (output ACTIVE set), until the response has been received. Therefore, it is recommended to reset the EXECUTE signal only with the set READY signal. This is necessary as the internal communication to the master connection must not be interrupted before the response has been received.

PCP_INITIATE

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

With this service a logic connection between master and slave is established. It is verified, if the settings for the seizes of transmitter and receiver buffer are equal to the supporting services.

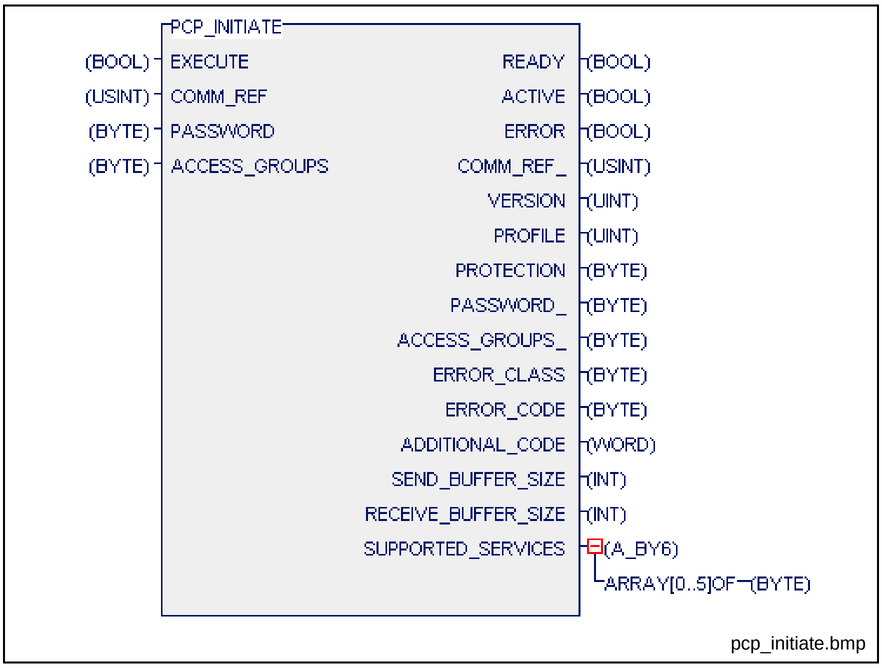


Fig. 13-54: Function block PCP_INITIATE

EXECUTE

Activation of the service INITIATE

COMM_REF

Communication reference between controller board and slave

PASSWORD

Password, that is defined for access to objects of the device. You can find it in the device documents. In some profiles no password is provided. In this case, value 16#00 has to be filled in.

ACCES_GROUPS

Manufacturer-specific assignment of the controller board to an access group. In some profiles access groups are not provided. In this case value 16#00 has to be filled in.

READY

Response received, service request terminated

ACTIVE

Service request is executed

ERROR

Error in service request

COMM_REF_

Communication reference between controller board and slave

VERSION

Version identification of the object directory in 2 bytes. It is device-specific and is read out of the object directory by the system, e.g. 16#0000.

PROFILE

Identification of the device profile, i.e. the number of the user-specific definitions is indicated (16#xxxx).

PROTECTION

Contains the attribute "Access_Protection_Supported" from the device documents. The parameter indicates, if access rights have been verified during the access to objects:

16#FF Access rigths have been verified (TRUE),

16#00 Access rights have not been verified (FALSE).

PASSWORD_

Manufacturer-specific password. Normally, it is not used. In this case the parameter "Password" contains value 16#00.

ACCESS_GROUPS_

Manufacturer-specific assignment of the controller board to an access group. In some profiles access groups are not provided. In this case this parameter contains value 16#00.

ERROR_CLASS

Identification for an error number of the initiate service

Specifies the error; Error Message, Service-Specific, Referring to PCP Blocks

ERROR_CODE

Specifies the error:

16#01 The seizes of the transmitter and receiver buffer of the two devices do not match.

16#02 The supporting services of the two devices do not match.

16#04 Service rejected by the user program; the error cause is manufacturer-specific.

ADDITIONAL_CODE

Manufacturer-specific information on the error cause:

16#xxxx Please look at your device documents. Maybe the device is not ready yet.

16#0000, if parameter "ERROR_CODE" contains the error codes 16#01 or 16#02.

SEND_BUFFER_SIZE / RECEIVE_BUFFER_SIZE

Buffer seizes (transmitter / receiver buffer) of the remote device.

SUPPORTED_SERVICES

Coding of the supported services that the remote device can process. The coding always occurs in 6 bytes:

The supporting user services of a PCP device are represented hexadecimally in a bit pattern of 48 bits, whereby the bits 0 ... 23 indicate the supporting services as client and the bits 24 ... 47 the supporting services as server. Only some of these 48 bits are really used.

The bits 3, 4, 8, 27, 28 and 32 represent each several services.

All unused bits as well as the bits of the not supported services are set on "0". Then, the used bits are set on "1", if the service is supported.

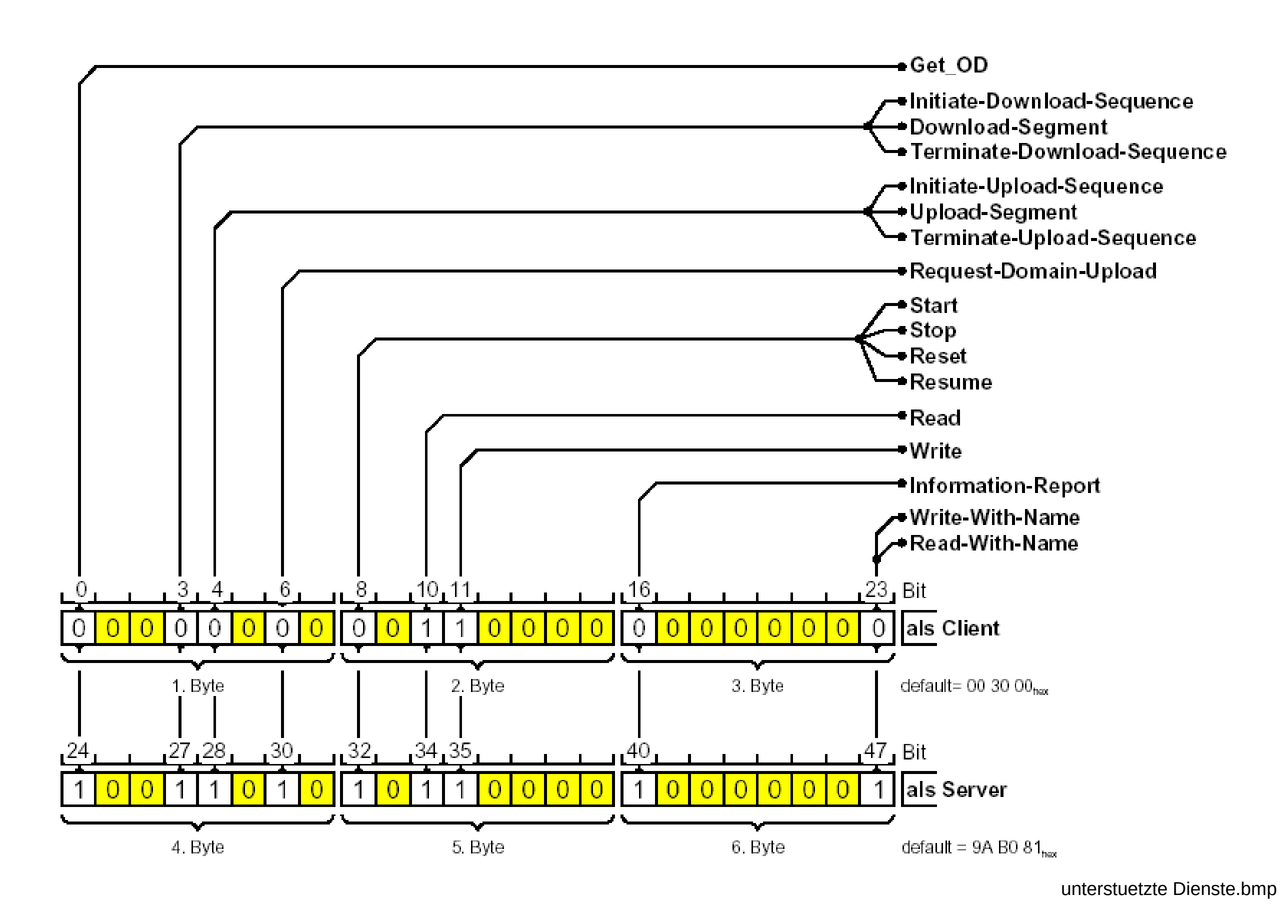


Fig. 13-55: Supported services

Function block PCP_INITIATE causes the following error messages:

S#ErrorNr: -324 with

- S#ErrorTyp: 6 if an internal transmission error occurs or
- S#ErrorTyp: 234 if the memory is not available

PCP_READ

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

With this service object values can be read out, if the slave is an PPC slave. As for ARRAYs and records you can specify, if the whole object or only one element of the object is to be read out.

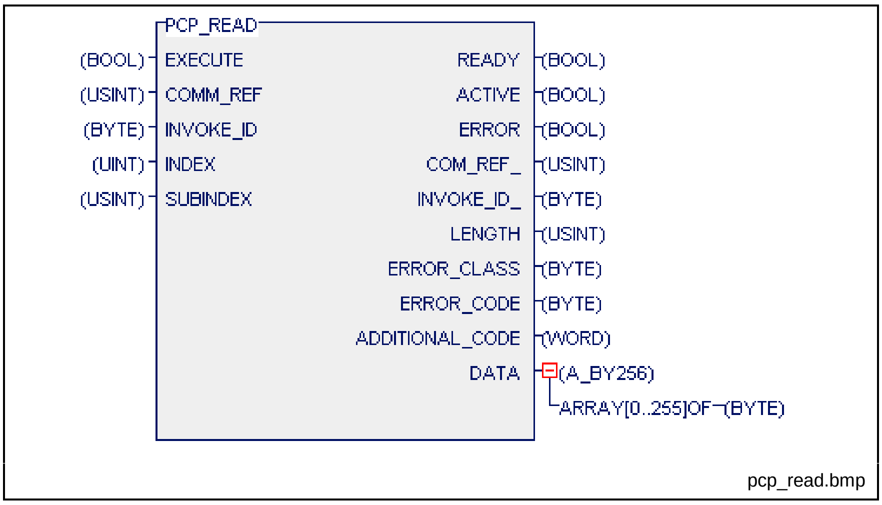


Fig. 13-56: Function block PCP_READ

EXECUTE

Activation of service READ

COMM_REF

Communication reference between controller board and slave

INVOKE_ID

Order number for parallel services (default value = 16#00). This parallel services have to be supported by the corresponding device. Please, look at the device description.

If several instances of the same PCP block type addressing the same device (same communication reference CR) are used, at each case, the input INVOKE_ID is to be wired with an unambigous number.

INDEX

Index that is – in the device documents – assigned to the object to be read out. The index is the logic address of the object. You will find the index in the device documents.

SUBINDEX

A subindex – a logic subaddress – is assigned to each element of an object (ARRAY or record). You will find the subindex in the device documents. If the whole object is to read out, fill in 16#00.

READY

Response received, service request terminated

ACTIVE

Service request is executed

ERROR

Error in service request

COMM_REF_

Copy of **COMM_REF**

INVOKE_ID_

Copy of **INVOKE_ID**

LENGTH

Number of the following data bytes (user data). It depends on the read out object, e.g. if only one element or the whole object has been read.

ERROR_CLASS / ERROR_CODE

Specifies the error; Error Message, Service-Specific, Referring to PCP Blocks

ADDITIONAL_CODE

Manufacturer-specific information on the error cause:

16#xxxx Please, look at your device documents. Maybe the device is not ready yet.

16#0000 , if parameter "ERROR_CODE" contains the error codes 16#01 or 16#02.

DATA

User data of the object

Function block PCP_READ causes the following error messages:

S#ErrorNr: -326 with

- S#ErrorTyp: 6 if an internal transmission error occurs or
- S#ErrorTyp: 234 if the memory is not available

PCP_WRITE

Note:	This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

With this service the set device parameters of an object can be modified. As for ARRAYs and records you can specify, if the whole object or only one element of the object is to be modified.

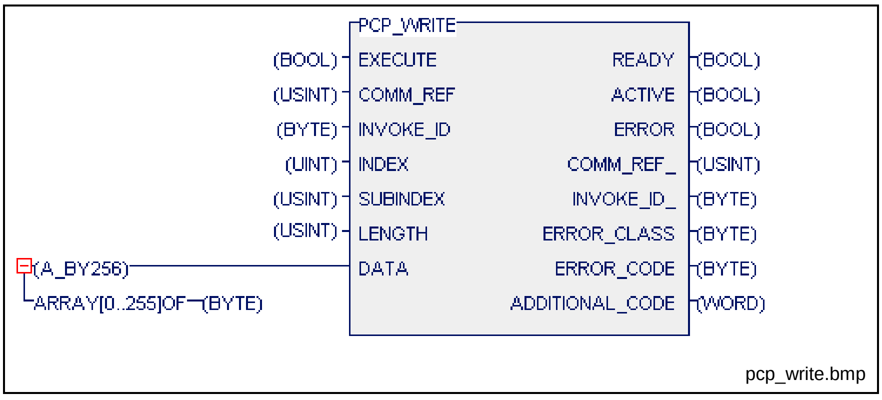


Fig. 13-57: Function block PCP_WRITE

EXECUTE

Activation of service WRITE

COMM_REF

Communication reference between controller board and slave

INVOKE_ID

Order number for parallel services (default value = 16#00). This parallel services have to be supported by the corresponding device. Please, look at the device description.

If several instances of the same PCP block type addressing the same device (same communication reference CR) are used, at each case, the input INVOKE_ID is to be wired with an unambigous number.

INDEX

Index that is – in the device documents - assigned to the object to be written. The index is the logic address of the object. You will find the index in the device documents.

SUBINDEX

A subindex – a logic subaddress – is assigned to each element of an object (ARRAY or record). You will find the subindex in the device documents. If the whole object is to be described, fill in 0.

LENGTH

Number of the following data bytes. It depends on the written object, e.g. if only one element or the whole object has been read.

DATA

At this place the real user data are filled in, i.e. the values that have to be newly written.

READY

Response received, service request terminated

ACTIVE

Service request is executed

ERROR

Error in service request

COMM_REF_

Copy of **COMM_REF**

INVOKE_ID_

Copy of **INVOKE_ID**

ERROR_CLASS / ERROR_CODE

Specifies the error; Error Message, Service-Specific, Referring to PCP Blocks

ADDITIONAL_CODE

Manufacturer-specific information on the error cause:

16#xxxx Please, look at your device documents. Maybe your device is not ready yet.

16#0000, if parameter "ERROR_CODE" contains the error codes 16#01 or 16#02.

Function block PCP_WRITE causes the following error messages:

S#ErrorNr: -327 with

- S#ErrorTyp: 6 if an internal transmisson error occurs or
- S#ErrorTyp: 234 if the memory is not available

PCP_GET_OD

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

With this service one or several object descriptions of a device can be read out. If the whole object directory is to be read out, this service has to be used several times depending on its seize.

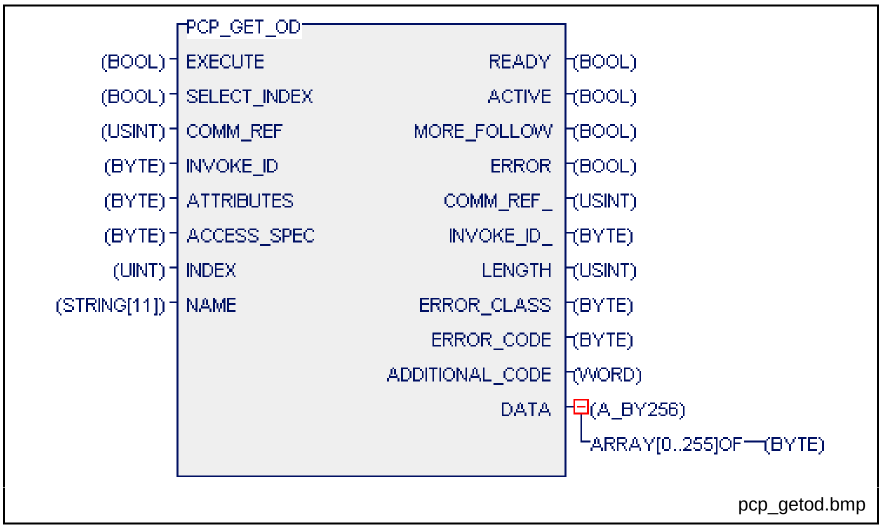


Fig. 13-58: Function block PCP_GET_OD

EXECUTE

Activation of service GET_OD

SELECT_INDEX

Selection, if the object can be accessed via index addressing (TRUE) or via the name (FALSE).

COMM_REF

Communication reference between controller board and slave

INVOKE_ID

Order number for parallel services (default value = 16#00). This parallel services have to be supported by the corresponding device. Please, look at the device description.

If several instances of the same PCP block type addressing the same device (same communication reference CR) are used, at each case, the input INVOKE_ID is to be wired with an unambigous number.

ATTRIBUTES

Selection between object description in short form (16#00) or long form (16#01)

ACCESS_SPEC

Indicates which object is to be accessed.

16#01 Index of an object

16#02 Name of a variable

16#05 Name of a program sequence (PI)

16#07 Objects as from this starting index are read out

INDEX

Specify the index of the object at this place

NAME

Specify the name of the object at this place

READY

Response received, service request terminated

MORE_FOLLOW

If the access is executed via the starting index, MORE_FOLLOW indicates that the seize of the requested object descriptions is greater than the transmitter buffer and therefore, not all of the requested data could be read out.

FALSE: No further data.

TRUE: There are still further values that you can read out with a further Get_OD.

If the access occurs via index or name, the value is always FALSE.

ACTIVE

Service request is executed

ERROR

Error in service request

COMM_REF_

Copy of **COMM_REF**

INVOKE_ID_

Copy of **INVOKE_ID**

LENGTH

Number of the following data bytes. It depends on the read out object, i.e. if only one element or the whole object has been read out.

ERROR_CLASS / ERROR_CODE

Specifies the error; Error Message, Service-Specific, Referring to PCP Blocks

ADDITIONAL_CODE

Manufacturer-specific information on the error cause:

16#xxxx Please, look at the device documents. Maybe the device is not ready yet.

16#0000, if parameter "ERROR_CODE" contains the error codes 16#01 or 16#02.

Function block PCP_GET_OD causes the following error messages:

S#ErrorNr: -329 with

- S#ErrorTyp: 6 if an internal transmission error occurs or
- S#ErrorTyp: 234 if the memory is not available

PCP_IDENTIFY

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

With this service the name plate of the device can be read out.

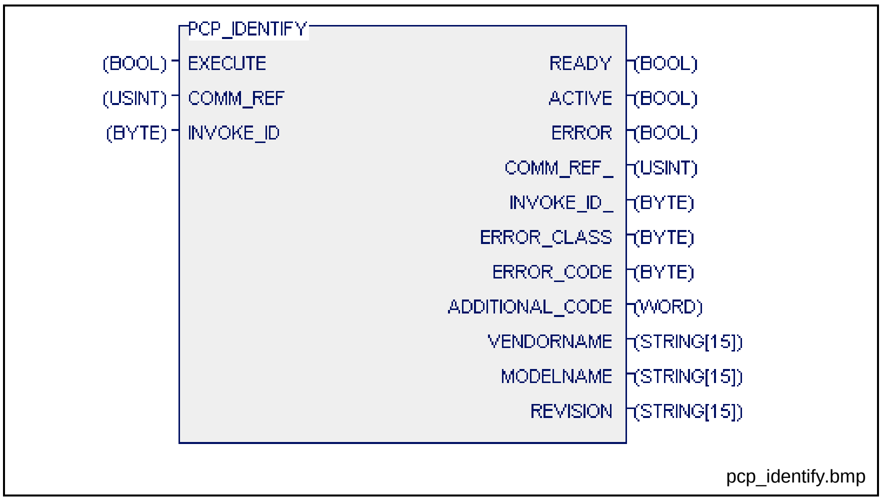


Fig. 13-59: Function block PCP_IDENTIFY

EXECUTE

Activation of service IDENTIFY

COMM_REF

Communication reference between controller board and slave

INVOKE_ID

Order number for parallel services (default value = 16#00). This parallel services have to be supported by the corresponding device. Please, look at the device description.

If several instances of the same PCP block type addressing the same device (same communication reference CR) are used, at each case, the input INVOKE_ID is to be wired with an unambigous number.

READY

Response received, service request terminated

ACTIVE

Service request is executed

ERROR

Error in service request

COMM_REF_

Copy of **COMM_REF**

INVOKE_ID_

Copy of **INVOKE_ID**

ERROR_CLASS / ERROR_CODE

Specifies the error: Error Message, Service-Specific, Referring to PCP Blocks

ADDITIONAL_CODE

Manufacturer-specific information on the error cause:

16#xxxx Please, look at the device documents. Maybe the device is not ready yet.

16#0000, if parameter "ERROR_CODE" contains error code 16#01 or 16#02.

VENDORNAME

Manufacturer name of the device

MODELNAME

Device name

REVISION

Revision number of the device

Function block PCP_IDENTIFY causes the following error messages:

S#ErrorNr: -328 with

- S#ErrorTyp: 6 if an internal transmission error occurs or
- S#ErrorTyp: 234 if the memory is not available

PCP_ABORT

Note:	This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

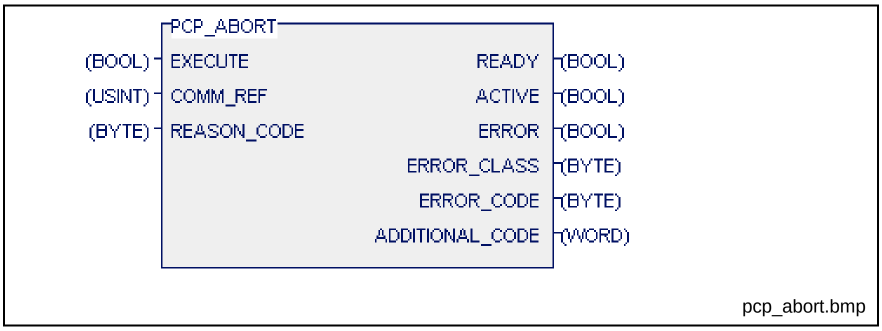


Fig. 13-60: Function block PCP_ABORT

EXECUTE

Activation of service ABORT

COMM_REF

Communication reference between controller board and slave

REASON_CODE

Cause for abortion of the connection. Default: 16#00

READY

Response received, service request terminated

ACTIVE

Service request is executed

ERROR

Error in service request

ERROR_CLASS / ERROR_CODE

Specifies the error; Error Message, Service-Specific, Referring to PCP Blocks

ADDITIONAL_CODE

Manufacturer-specific information on error cause:

16#xxxx Please, look at the device documents. Maybe the device is not ready yet.

16#0000, if parameter "ERROR_CODE" contains the error codes 16#01 or 16#02.

Function block PCP_ABORT causes the following error messages:

S#ErrorNr: -325 with

- S#ErrorTyp: 6 if an internal transmission error occurs or
- S#ErrorTyp: 234 if the memory is not available

Error Messages of the Communication with PCP Function Blocks (General Information)

If a service can't be executed as provided, an internal error message is generated. There are three groups of error messages:

- Error messages in conjunction with a connection abortion (Abort_Indication for PCP Blocks). In this case the output ADDITIONAL_CODE contains value 0x488D.
- Error messages after rejection of the sent service by the reject service (Reject_Indication for PCP Blocks). In this case output ADDITIONAL_CODE contains the value 0x488E.
- Error messages after sending the confirmed services, that could not be executed (Error Message, Service-Specific, Referring to PCP Blocks).

Abort_Indication for PCP Blocks

Meaning of the parameters ERROR_CLASS/ERROR_CODE in case of a connection abortion (see also Error Messages of the Communication with PCP Function Blocks (General Information)):

16#00 | 16#01 (Disconnect)

Meaning: The user program of the PCP device interrupted the connection.

Cause: -

Remedy: Inform the manufacturer of the PCP device.

16#01 | 16#01 (CRL-Error)

Meaning: CRL entry not OK

Cause: The control has sent the service request Initiate_Request", but the CRL for the device is not available or the CR is not assigned to a device.

Remedy: Verify the CR entries in the CRL.

16#01 | 16#02 (User Error)

Meaning: The PCP device has received an impermissible or faulty service.

Cause:

Possibility 1: The connection has already been established. You have tried a second time to establish a connection with the service request "Initiate_Request" and thus caused the abortion of the connection.

Possibility 2: You have sent a service without establishing a connection before.

Remedy:

Possibility 1: Re-establish the connection.

Possibility 2: Establish the connection and send the service once again.

16#01 / 16#03 ... 16#09, 16#10 (System error)

Meaning: Error concerning the PCP device.

Cause: -

Remedy: Inform the manufacturer of the PCP device.

16#01 | 16#13 (No CRL available)

Meaning: No CRL available.

Cause:

Possibility 1: You have not load a CRL.

Possibility 2: An existing connection has been interrupted, because you have re-loaded the CRL.

Remedy: Project a CRL, re-establish the connection.

16#02 | 16#00 (LLI-Context-Check-Fail)

Meaning: The connection parameters between your controller board and the PCP device do not match.

Cause: The number of the parallel services or the connection monitoring is projected differently on the device and on the controller board.

Remedy: Correct the corresponding parameter in the CRL of the controller board.

16#02 | 16#01 (Invalid-LLI-PDU)

Meaning: Impermissible service while establishing or interuppting the connection.

Cause: A device received a PCP service (e.g. Read or Write), although the connection was not established.

Remedy: Establish the connection.

16#02 | 16#02 (Invalid LLI-PDU)

Meaning: Impermissible service during data transfer

Cause: You have switched off the control without interrupting the connection before. The connection with the communication partner was still active. To re-establish the connection with service "Initiate" failed. Now, the connection was interrupted.

Remedy: Re-establish the connection with the service "Initiate".

16#02 | 16#08 (Local Error)

Meaning: System error.

Cause: -

Remedy: Inform the manufacturer of the PCP device.

16#02 | 16#09 (Associate-Timeout)

Meaning: Waiting time for establishing the connection has elapsed.

Cause:

Possibility 1: Defective device.

Possibility 2: INTERBUS inactive.

Remedy:

Possibility 1: Change device.

Possibility 2: Put the INTERBUS to the status RUN.

16#02 / 16#11 (Invalid-LLI-PDU)

Meaning: Invalid service while interrupting the connection.

Cause: You tried to re-establish the connection during the interruption of the connection.

Remedy: Wait approximately 30 to 100 ms before starting to establish a new connection.

(The waiting time depends on the number of the INTERBUS modules.)

16#02 / 16#12, 16#14 (Invalid-LLI-PDU)

Meaning: System error.

Cause: -

Remedy: Inform the manufacturer of the PCP device.

16#03 | 16#02 (Remote-Resource)

Meaning: The receiver buffers on the PCP device are full.

Cause: The PCP device does not respond or is not available.

Remedy: Verify the remote address in the CRL.

16#03 | 16#11 (PDL-Timeout)

Meaning: Internal communication acknowledgement not received within the waiting time.

Cause: The PCP device could be defective.

Remedy: Inform the manufacturer of the PCP device.

16#03 | 16#12 (PDL disconnect)

Meaning: Repeated error during data transmission.

Cause: The PCP devices try to synchronize, but without success.

Remedy: Repeat service after approximately 30 to 100 ms. The waiting time depends on the number of the INTERBUS modules. If the error occurs several times, it is a system error. In this case inform the manufacturer.

16#03 / 14hex, 15hex (PDL invalid)

Meaning: System error.

Cause: -

Remedy: Inform the manufacturer of the PCP device.

16#03 | 20hex (PDL-Cycle-Error)

Meaning: Fatal bus error.

Cause: -

Remedy: Verify the wiring. Provide for reliability of the bus (see Application Description).

Reject_Indication for PCP Blocks

Meaning of parameter ERROR_CODE in case of a service rejection (see also Error Messages of the Communication with PCP Function Blocks (General Information))

ERROR_CLASS

Specifies the type of the rejected message:

16#01: Error when requesting (***Request***) a confirmed service

16#02: Error when confirming (***Response***) an activated service

16#01 (Invoke-ID exists)

Meaning: The invoke ID already exists.

Cause: A parallel service with identic invoke ID has been sent.

Remedy: Use a free invoke ID.

16#02 (Max Services Overflow)

Meaning: To many service requests have been sent to a device.

Cause: A second service request has been sent to a device without waiting for the service acknowledgement (confirmation) of the first service request to this device. Or a CR was assigned mistakenly two times.

Remedy: Re-send the service request, when the confirmation arrived. Verify if the CR exists several times.

16#03 (Service-Not-Supported-Connection-Oriented)

Meaning: Service is not supported as client.

Cause: A service that is not projected in the CRL was used in the user program.

Remedy: Add the service by means of service "Load_CRL_Attribute_Loc" to the supported services as client.

16#05 (PDU-Size)

Meaning: The maximum message length (PDU size) was exceeded.

Cause: You have send one of the services "Write" or "Write_With_Name". However, this client contained to many data for the PCP device.

Remedy: Verify the length parameter (length) in the object description of the device.

16#07 (Max-Unconfirmed-Services-Overflow)

Meaning: Maximum number of unconfirmed services exceeded.

Cause: The primarily sent service is not completely executed yet (only for unconfirmed services).

Remedy: Re-send the service.

Error Message, Service-Specific, Referring to PCP Blocks

Meaning of the parameters ERROR_CLASS/ERROR_CODE in case of a service-specific error message (see also Error Messages of the Communication with PCP Function Blocks (General Information)):

16#00 | 16#01 (Max. PDU-Size insufficient)

Meaning: The sizes of the transmitter and receiver buffer of the two communication devices do not match.

Cause: -

Remedy: Adapt the buffer sizes of the controller board with service "Load_CRL_Attribute_Loc_Request" to those of the communication partner.

16#00 | 16#02 (Feature not supported)

Meaning: The desired service is not supported.

Cause: The supported services of the two communication devices do not match.

Remedy: Change the supported services of the controller board with service "Load_CRL_Attribute_Loc".

16#00 | 16#04 (User Initiate denied)

Meaning: This error message depends on the manufacturer.

Cause: -

Remedy: Please, look at the device description.

16#05 | 16#01 (State-Conflict)

Meaning: A start or stop command was sent two times.

Cause: Error occurs only while the start or stop service is activated. As the start or stop service has already been executed, the service can not be executed once again.

Remedy: No remedial action necessary.

16#05 | 16#05 (Service-Parameter)

Meaning: An impermissible value was specified for the parameter Access_Specification or the access occured with a too long name.

Cause: This error occurs only when service Get_OD is active.

Remedy: Please, look up the valid values in the device description and re-send the service once again.

16#06 | 16#02 (Hardware Fault)

Meaning: The access on the object failed because of a hardware error.

Cause: E.g. missing peripheral voltage.

Remedy: Eliminate the hardware error.

16#06 | 16#03 (Object-Access-Denied)

Meaning: The object has restricted access rights.

Cause: Possibly, the object can only be read, but not be described or it is password-protected.

Remedy: Please, look up the access rights in the object description.

16#06 | 16#05 (Object-Attribute-Inconsistent)

Meaning: A service parameter was indicated with an impermissible value.

Cause: E.g. an incorrect length indication or an impermissible subindex.

Remedy: Verify the parameter referring to the object description and re-send the service with the corrected values.

16#06 | 16#06 (Object-Access-Unsupported)

Meaning: The used service can not be applied to this object.

Cause: E.g. a program sequence can be started or stopped, but not read.

Remedy: Please, look at the object description, which services are permitted for this object.

16#06 | 16#07 (Object-Non-Existent)

Meaning: The object does not exist.

Cause: Probably, parameter "Index" has an incorrect value.

Remedy: Verify the index of the object referring to the object description and re-send the service.

08hex | 16#00 (Application-Error)

Meaning: Device-specific error message; no error in communication.

Cause: -

Remedy: Please, look at your device description.

16#09 | 16#xx (Firmware-Error)

Meaning: You find the decription of this error message in the general INTERBUS documentation "Services and error messages of the firmware". In section "Error codes to user errors" you will find under code 16#09xx all error codes of error class 16#09.

Cause: -

Remedy: Please, look at your device description.

PROFIBUS DP, Function Blocks

The following Firmware Function Blocks are available in the PLC programming interface for controlling a PROFIBUS:

- Status information on the PROFIBUS master: DPM_STATE
- Single diagnosis of a PROFIBUS slave: DPM_SLDIAG
- Program Example for Control of a PROFIBUS

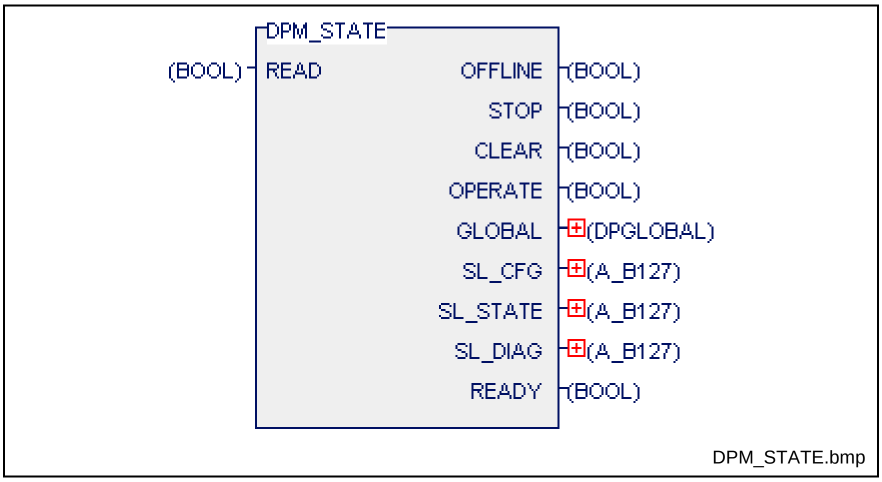
Note: These blocks may not be declared in the retain are (see section Limitation of the Declaration of Function Blocks in the Retain Area).

DPM_STATE

Status information on the PROFIBUS master, also see PROFIBUS DP, Function Blocks

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area)

This function block supplies status information of the PROFIBUS DP master, if the Boolean input READ is set to TRUE.



READ: Read status
OFFLINE: Operating state OFFLINE
STOP: Operating state STOP
CLEAR: Operating state CLEAR
OPERATE: Operating state OPERATE
GLOBAL: Global status bits (firmware data type DPGLOBAL)
SL_CFG: Table of configured slaves
SL_STATE: Table of active slaves
SL_DIAG: Table of slaves with diagnosis
READY: Function block is processed

Fig. 13-61: Firmware function block DP_STATE

Note: SL_DIAG contains the list of the slaves with active diagnosis. The entries are cleared with scanning of the slave single diagnosis; single diagnosis of a PROFIBUS slave DPM_SLDIAG.

Rexroth
Bosch Group

Error variables

If a PROFIBUS interface is not provided, the error variables must be set as follows:

S#ErrorFlg: TRUE

S#ErrorNr: 235

S#ErrorTyp: -242

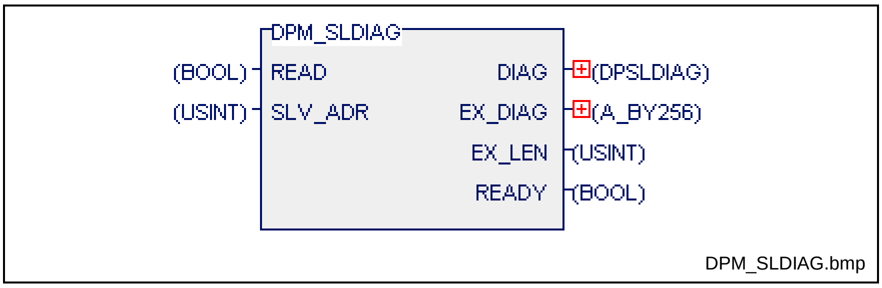
DPM_SLDIAG

Single diagnosis of a PROFIBUS slave, also see PROFIBUS DP, Function Blocks

---

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

The diagnosis information of a DP slave consists of standard diagnosis information and - if provided - user-specific diagnosis information. At the DIAG output, this function block supplies the standard diagnosis of the slave which is addressed by the SLV_ADR input. The user-specific diagnosis information is made available at the EX_DIAG output, with the length (in bytes) of this information being specified at the EX_LEN output.



READ:	Read diagnosis
SLV_ADR:	Slave address
DIAG:	Diagnosis of the addressed slave (firmware data type DPSLDIAG)
EX_DIAG:	Extended diagnosis data
EX_LEN:	Length of the extended diagnosis data
READY:	Function block is processed

Fig. 13-62: Firmware function block DPM_SLDIAG

Note: In order to avoid an unnecessary load on the bus, diagnosis should be requested only if the function block DPM_STATE activated the corresponding bit in the diagnosis field. This bit in the diagnosis field is cleared with reading of the diagnosis.

Furthermore this function block must only be implemented in controls with DP master configuration.

Error variables

If a PROFIBUS interface is not provided, the error variables must be set as follows:

S#ErrorFlg: TRUE

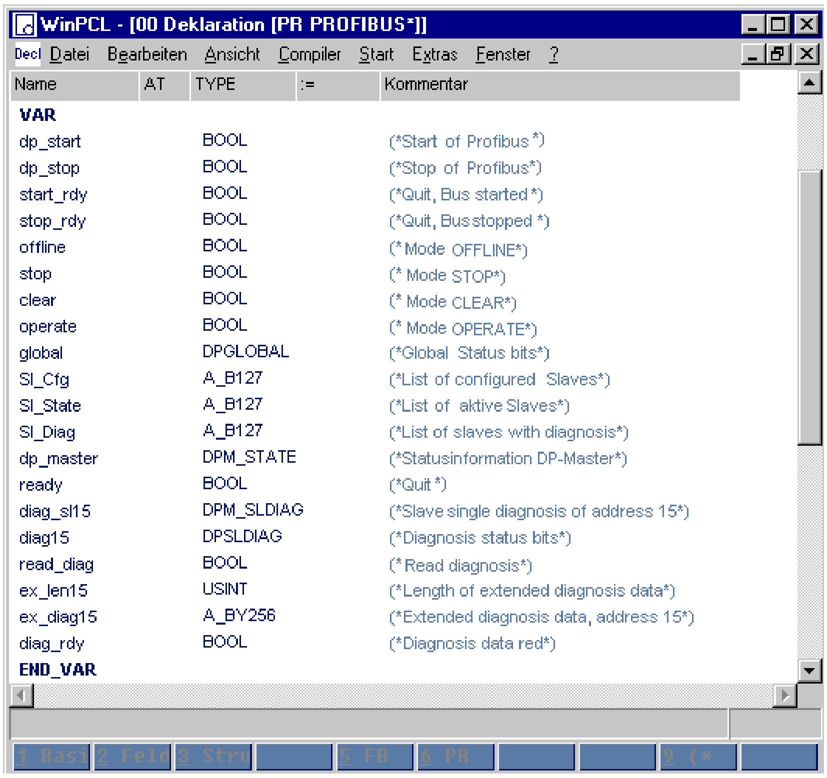
S#ErrorNr: 235

S#ErrorTyp: -239

Program Example for Control of a PROFIBUS

(Also see PROFIBUS DP, Function Blocks)

In this program example, the single diagnosis of the slave with address 15 is read. The PC104-PROFIBUS interface is fitted to slot 2. The bus can be started using the variable dp_start and can be stopped using the variable dp_stop. If the master identifies a diagnosis of the slave, bit SL_Diag[15] is activated. The status bits of the diagnosis, which are set in field diag15, then can be read via the variable read_diag. The bit SL_Diag[15] is cleared with reading of the diagnosis.



Dekl_FB_DP.bmp

Fig. 13-63: Declaration part for the program example

Rexroth Bosch Group

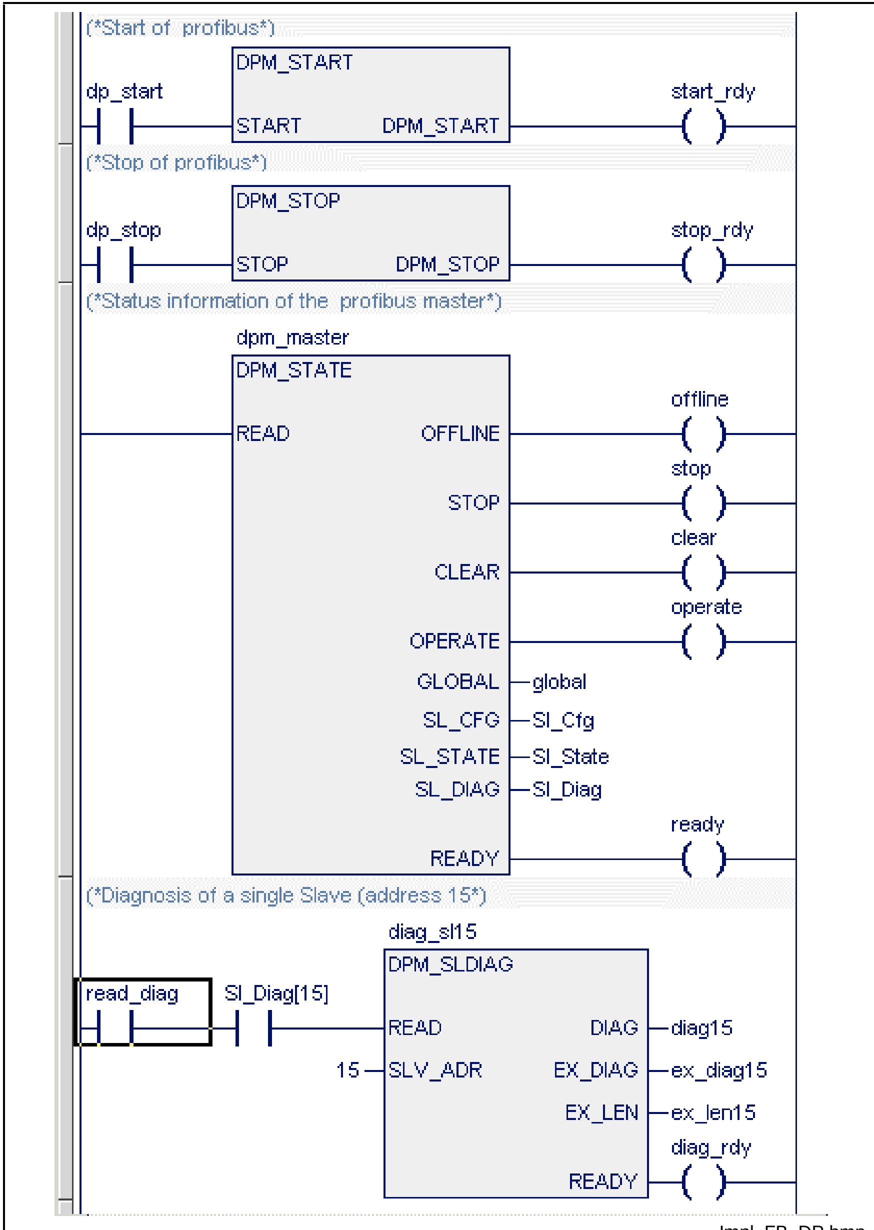


Fig. 13-64: Implementation part for the program example

ASI Bus, Function Block with Data Type

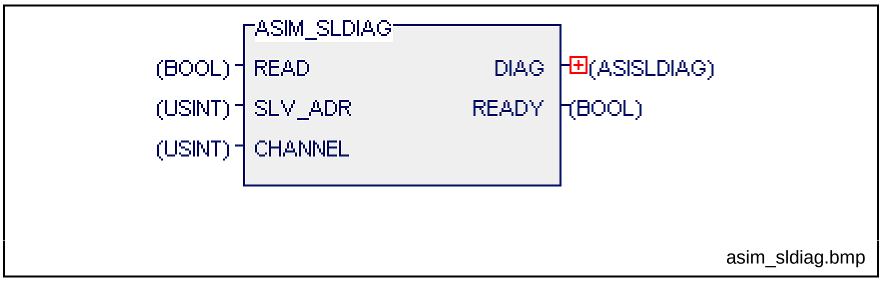
The following Firmware Function Blocks are available in the PLC programming interface for controlling a ASI Bus.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

ASIM_SLDIAG

System function block ASIM_SLDIAG serves to transmit the diagnostic information to the slave, that is addressed via the inputs SLV_ADR and CHANNEL. If the information is valid, output READY becomes active.

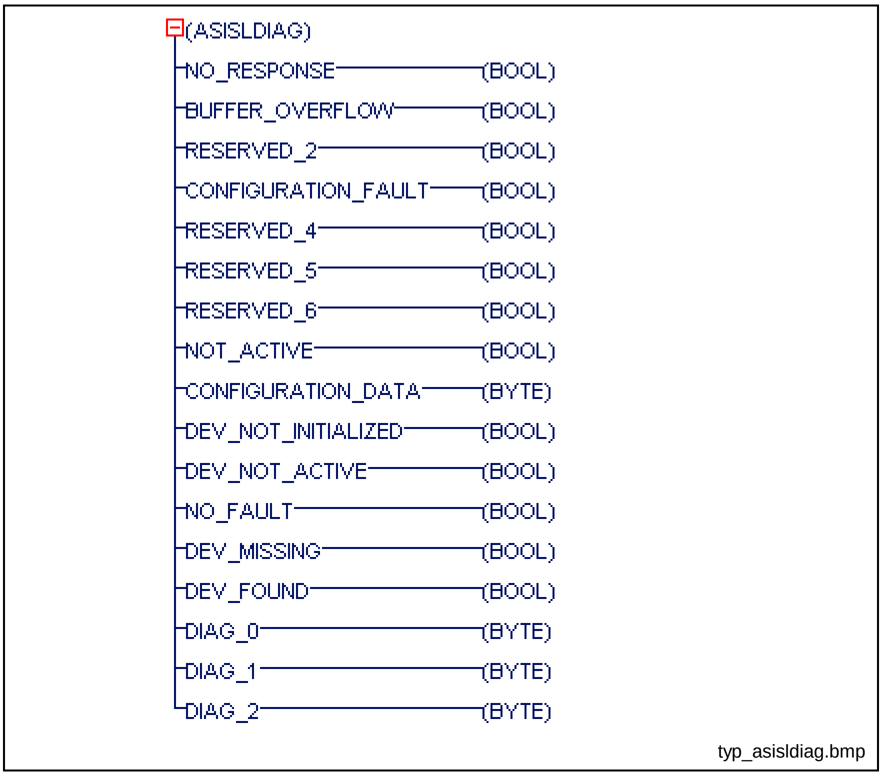
The diagnostic information is listed in structure ASISLDIAG.



READ:	Request of the diagnostic information
SLV_ADR:	Address of the slaves. Valid values: 1...62
CHANNEL:	Channel number. Valid values: 1, 2
DIAG:	Diagnostic information of the slave; see type description
READY:	For TRUE the information of the slave is valid

Fig. 13-65: ASIM-Bus: Function block ASIM_SLDIAG

Type description ASISLDIAG



NO_RESPONSE:	Device does not respond or is not available.
BUFFER_OVERFLOW:	The number of the entries in the error buffer exceeds the maximum possible number.
CONFIGURATION_FAULT:	The determined IO or ID code differs from the configured code.
NOT_ACTIVE:	The slave is not active in the current configuration.
CONFIGURATION_DATA:	Retrieved IO/OD code
DEV_NOT_INITIALIZED:	Slave was not initialized
DEV_NOT_ACTIVE:	Slave not active
NO_FAULT:	Slave indicates no error
DEV_MISSING:	Slave not available
DEV_FOUND:	At the moment not supported
DIAG_0/1/2:	Slave-specific diagnostic information. See manual of the manufacturer.

Fig. 13-66: Data type "Diagnostic information of a SLAVE"

Error variables

For all errors of the block applies:
S#ErrorFlg: TRUE, S#ErrorTyp: -336,

S#ErrorNr: 235 occurs, if

- there's no ASI master connection programmed in the I/O editor,
- there's no PC104 ASI master connection plugged in the control,
- the address setting of the PC104 ASI master connection is incorrect.

S#ErrorNr: 1 (invalid input parameter) occurs, if

- the slave address is greater 62 or equal to 0,
- the channel number is unequal to 1 or 2.

S#ErrorNr: 6 (internal transmission error) occurs, if during the diagnostic request an error, e.g. timeout, occurred.

Serial Interfaces, Function Blocks

Serial interfaces exist on different peripheries of the control. Special firmware function blocks are available for addressing these interfaces via PLC programs.

The following firmware function blocks are available for control of serial interfaces:

- OPEN_COM Open serial interface
- CLOS_COM Close serial interface
- WR_BYTE Write data byte to serial interface
- RD_BYTE - Read data byte from serial interface
- CTRL_COM - Determine the state of the interface
- WR_STR - Write data string to serial interface
- RD_STR - Read data string from serial interface
- CLR_COM - Clear receive buffer and transmit buffer of serial interface

---

Note: These blocks may not be declared in the retain area (see also Limitation of the Declaration of Function Blocks in the Retain Area).

OPEN_COM

OPEN_COM (Firmware Function Blocks, Serial Interfaces, Function Blocks, COM data type) initializes the transfer channel to a general serial interface, if the edge at the FB input OPEN is positive. From this interface, data can be transmitted or received only if the output READY of the block is logic one.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

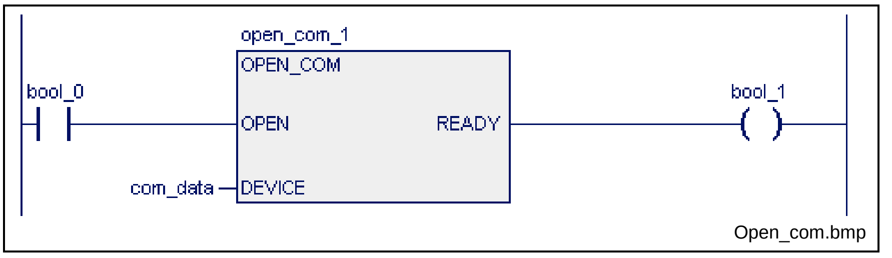


Fig. 13-67: Firmware function block OPEN_COM

Name	Type	Comment
OPEN:	BOOL;	0 - FB not active 1 - Open interface
DEVICE:	COM;	Parameter of the serial interface
READY:	BOOL;	0 - Interface is not opened or FB is not active 1 - Interface is open

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

CLOS_COM

With the edge at the FB input CLOS being positive, CLOS_COM (Firmware Function Blocks, Serial Interfaces, Function Blocks) closes the transfer channel to a general serial interface, thus clearing the transmitter and receiver buffers of the interface. Data residing in the buffers after the closing process has been initiated will get lost. The output READY becomes logic one, only after the serial interface has been closed.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

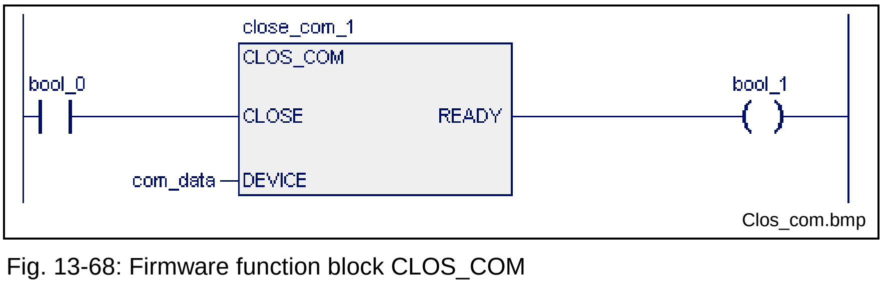


Fig. 13-68: Firmware function block CLOS_COM

Name	Type	Comment
CLOSE:	BOOL;	0 - FB not active 1 - Close interface
DEVICE:	COM;	Parameter of the serial interface
READY:	BOOL;	0 - Interface is not closed or FB is not active 1 - Interface is closed

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

WR_BYTE

Using WR_BYTE (Firmware Function Blocks, Serial Interfaces, Function Blocks), a data byte is written to the transmitter buffer of the interface selected. As long as the FB input WRITE is logic one and the transmitter buffer can still take up characters, the data byte, which is active at the FB input DATA whenever the block is called up, is written to the buffer. The data is emitted from the transmitter buffer by the PLC firmware, via the appropriate serial interface.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

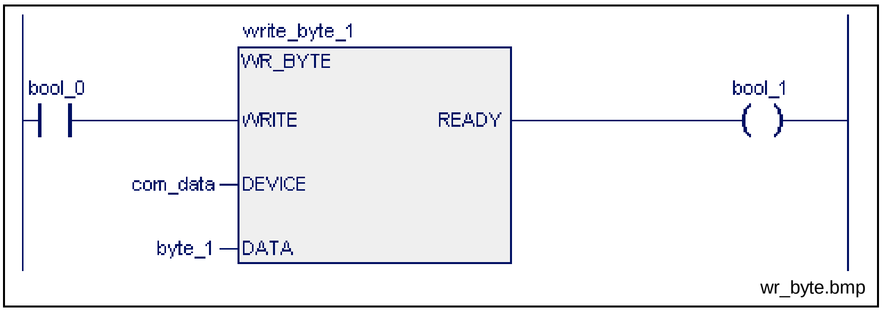


Fig. 13-69: Firmware function block WR_BYTE

Name	Type	Comment
WRITE:	BOOL;	0 - FB not active 1 - Write data byte to the transmitter buffer
DEVICE:	COM;	Parameter of the serial interface
DATA:	BYTE;	Data byte to be sent
READY:	BOOL;	0 - Data byte has not yet been written to the transmit buffer or the FB is not active 1 - Data byte has been written into the transmit buffer

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

RD_BYTE

Using RD_BYTE (Firmware Function Blocks, Serial Interfaces, Function Blocks), a data byte is read from the receiver buffer of the interface selected. As long as the FB input READ is logic one and the receiver buffer is not empty, a data byte is read from the buffer and assigned to the FB output DATA whenever the block is called up. The data is accepted from the appropriate serial interface to the receiver buffer by the PLC firmware.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

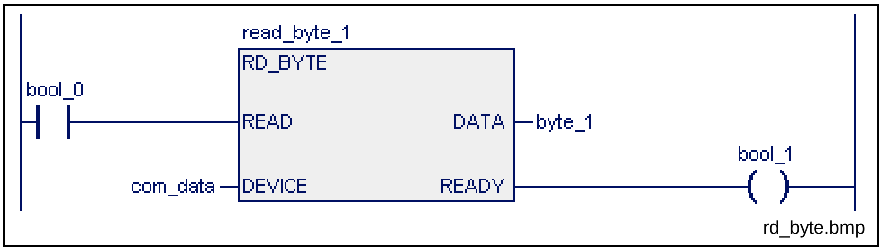


Fig. 13-70: Firmware function block RD_BYTE

Name	Type	Comment
READ:	BOOL;	0 - FB not active 1 - Read data byte from the receiver buffer
DEVICE:	COM;	Parameter of the serial interface
DATA:	BYTE;	Received data byte
READY:	BOOL;	0 - Data byte has not yet been read from the receiver buffer or the FB is not active 1 - Data byte has been read from the receiver buffer

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

CTRL_COM

The function block CTRL_COM (Firmware Function Blocks, Serial Interfaces, Function Blocks) is intended to return the state of a serial interface. To achieve this, the interface parameters are applied to the FB input DEVICE in the form of a COM structure. If the input CTRL is TRUE, the interface parameters are determined and applied to the outputs. The output READY becomes TRUE, if all information is provided.

The following status information can be evaluated:

- serial interface open or closed,
- number of characters residing in the receiver or transmitter buffer of the serial interface.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

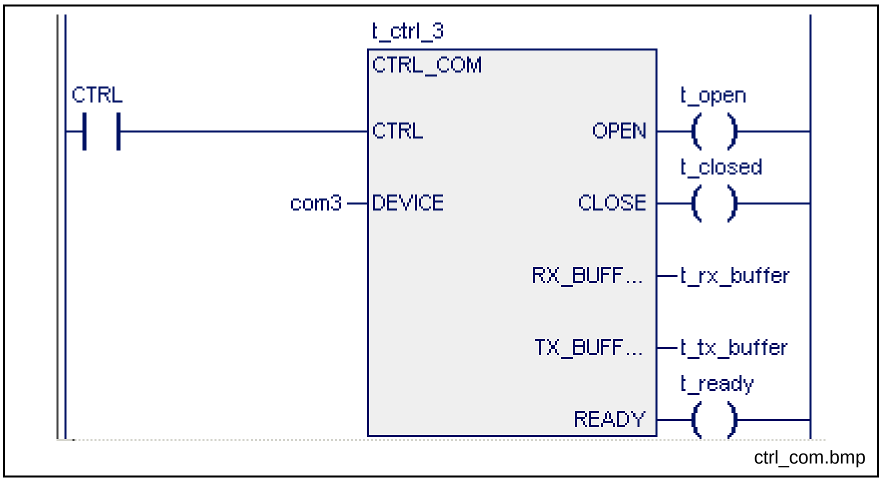


Fig. 13-71: Firmware function block CTRL_COM

Name	Type	Comment
CTRL:	BOOL;	0 - FB not active 1 - Determine state
DEVICE:	COM;	Parameter of the serial interface
OPEN:	BOOL;	0 - Interface not open 1 - Interface open
CLOSE	BOOL	0 - Interface not closed 1 - Interface closed
RX_BUFFER	INT	0 - No characters in receiver buffer or interface closed >0 - At least one character in receiver buffer
TX_BUFFER	INT	0 - No characters in transmitter buffer or interface closed >0 - At least one character in transmitter buffer
READY:	BOOL;	0 - State of interface not evaluated 1 - State of interface evaluated

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

WR_STR

Using WR_STR (Firmware Function Blocks, Serial Interfaces, Function Blocks), a data string is written to the transmitter buffer of the interface selected. The character active at the input STR_END is added to the data string as the string delimiter. If the input is not activated, the character with the ASCII code 0 is added to the data string. The string is written to the transmitter buffer, if the buffer provides enough unassigned memory for this string. The data is emitted from the transmitter buffer by the PLC firmware, via the appropriate serial interface.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

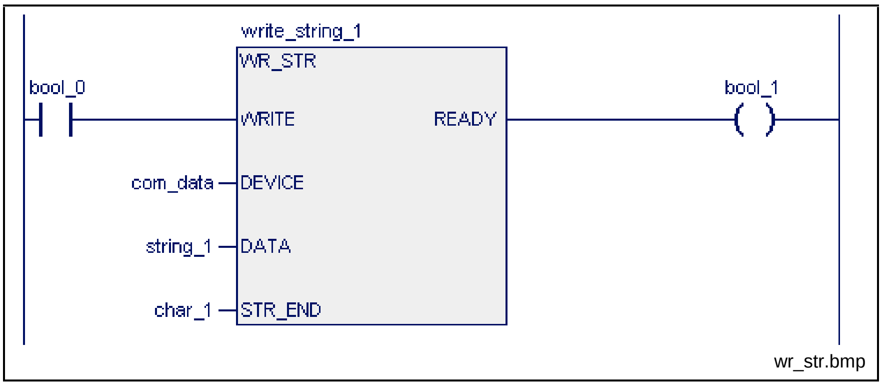


Fig. 13-72: Firmware function block WR_STR

Name	Type	Comment
WRITE:	BOOL;	0 - FB not active 1 Write data string into the transmitter buffer
DEVICE:	COM;	Parameter of the serial interface
DATA:	BYTE;	Data string to be sent
STR_END:	CHAR;	Data string delimiter
READY:	BOOL;	0 - Data string has not yet been written to the transmitter buffer or the FB is not active 1 - Data string has been written to the transmitter buffer

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

RD_STR

Using RD_STR (Firmware Function Blocks, Serial Interfaces, Function Blocks), a string is read from the receiver buffer of the interface selected. The end of the string is reached when the character read from the receiver buffer is equal to the string delimiter active at the input STR_END. In this case, the output READY becomes logic one. If the input STR_END is not activated, the ASCII code 0 is assigned to the string delimiter as a standard.

A string variable which is preset at the DATA output is filled with characters until the string delimiter has been read out. The string delimiter does not form a part of the output string. The content of the string variable is cleared, if no string delimiter was received before 255 characters were read. If there are still characters in the transmitter buffer, the string variable is filled again with these characters until the string delimiter is reached.

The data of the serial interface is applied in the receiver buffer through the PLC firmware.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

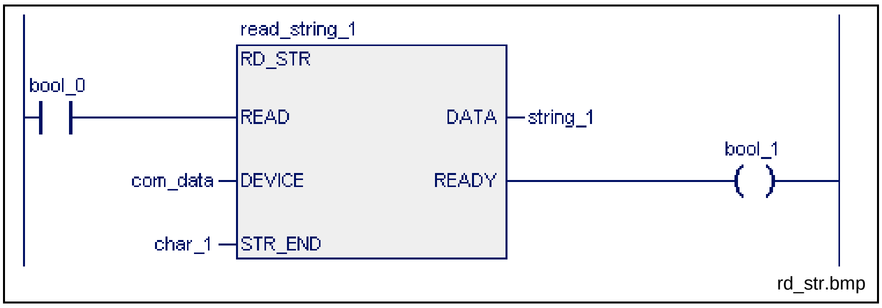


Fig. 13-73: Firmware function block RD_STR

Name	Type	Comment
READ:	BOOL;	0 - FB not active 1 - Read data string from receiver buffer
DEVICE:	COM;	Parameter of the serial interface
STR_END:	CHAR;	String delimiter
DATA:	BYTE;	Received data string
READY:	BOOL;	0 - Data string has not yet been read completely from the receiver buffer or the FB is not active 1 - Data string has been read from the receiver buffer

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

CLR_COM

Using the function block CLR_COM (Firmware Function Blocks, Serial Interfaces, Function Blocks), the receiver and transmitter buffers of an open serial interface are cleared. Data residing in the buffers after the clearing process has been initiated will get lost.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

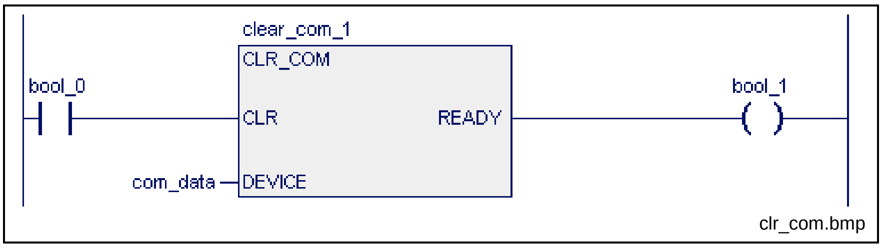


Fig. 13-74: Firmware function block CLR_COM

Name	Type	Comment
CLR:	BOOL	0 - FB not active 1 - Clear receiver and transmitter buffers
DEVICE:	COM	Parameter of the serial interface
READY:	BOOL	0 - Receiver or transmitter buffers were not cleared or the FB is not active 1 - Receiver and transmitter buffers were cleared

For handling of errors see Error Handling of Function Blocks for Serial Interfaces.

Error Handling of Function Blocks for Serial Interfaces

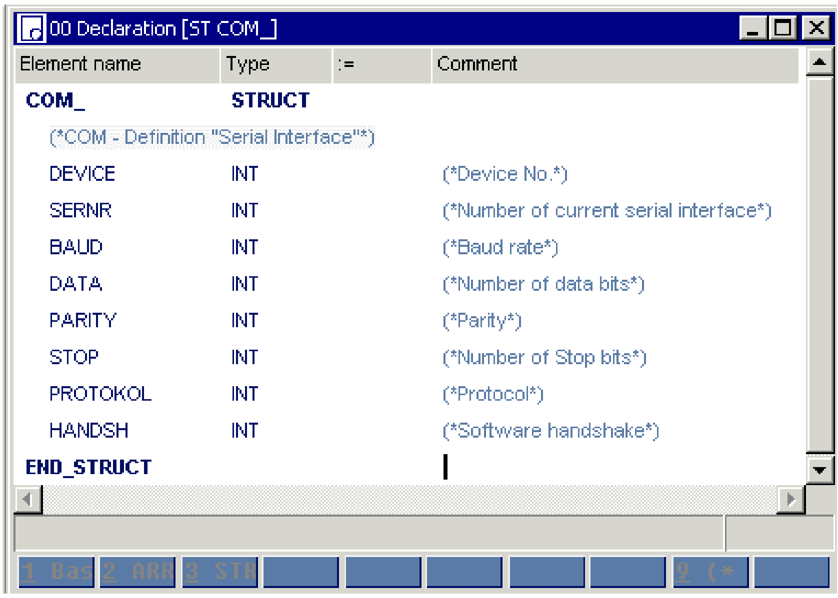
The function blocks which have been written to cannot be executed correctly because of programming or hardware errors. In such a case, error handling reports the cause of the error.

S#ErrorTyp	Function block
-106	OPEN_COM
-107	CLOS_COM
-110	WR_BYTE
-111	RD_BYTE
-112	CTRL_COM
-203	RD_STR
-204	WR_STR
-227	CLR_COM

S#ErrorNr	Error
238	Interface not opened
240	Invalid input parameter DEVICE, COM
241	Invalid input parameter SERNR, COM
242	Invalid input parameter BAUD, COM
243	Invalid input parameter DATA, COM
244	Invalid input parameter PARITY, COM
245	Invalid input parameter STOP, COM
246	Invalid input parameter PROTOKOL, COM
247	Invalid input parameter HANDSH, COM
248	Interface does not exist
249	All COM interfaces opened
252	General interface error
253	Transmitter buffer full
254	Receiver buffer full
255	Timeout acknowledgement telegram

Program Example for Control of serial Interfaces

Data are to be exchanged between two serial interfaces of the PLC card: After activation of the OPEN switch, byte 16#F8 is sent by interface 3 and the string ABCDE is sent by interface 4, with character F being the end character. The transmitter and receiver buffers of the interface are cleared with activation of the CLEAR switch. The transmission is completed with activation of the CLOSE switch.

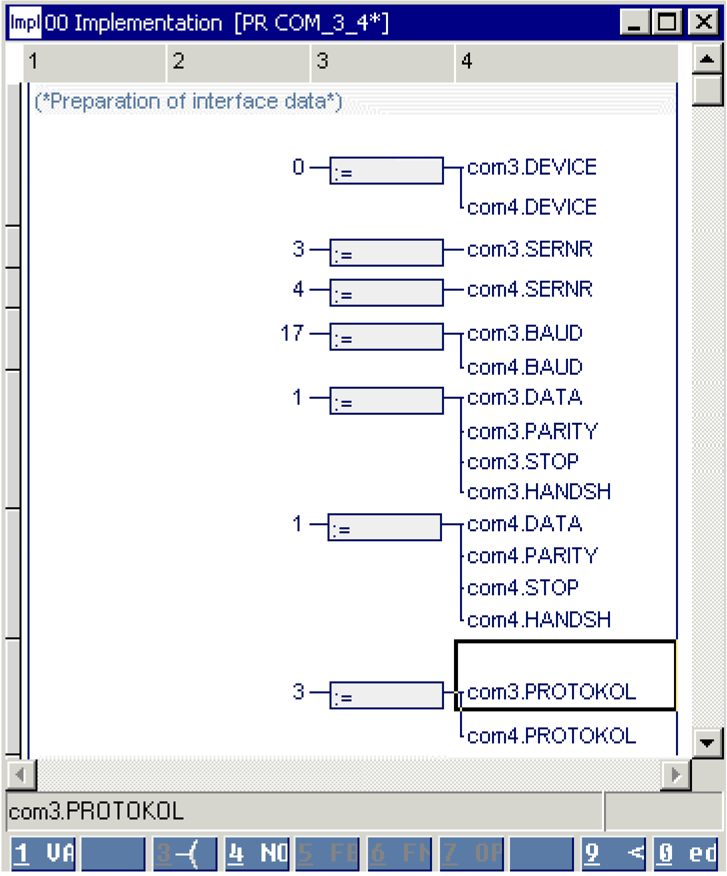


ST COM.bmp

Fig. 13-75: Definition of the serial interfaces in the structure "COM"

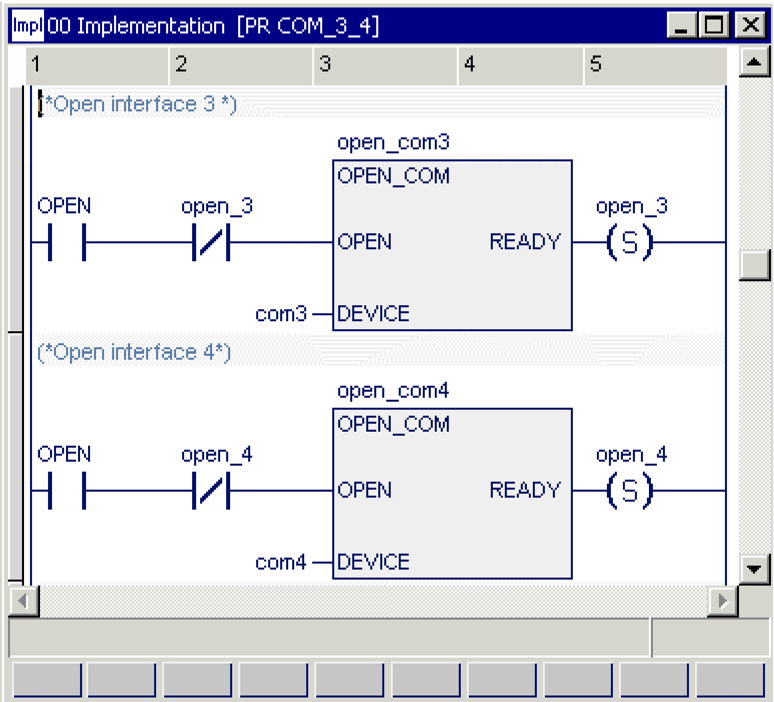
Setting of the interfaces is done with the first step. This is effected by transmission of the values to the individual interfaces.

- com3: COM (*interface COM 3*)
- com4: COM (*interface COM 4*)



Serielle Schnittst_01.bmp

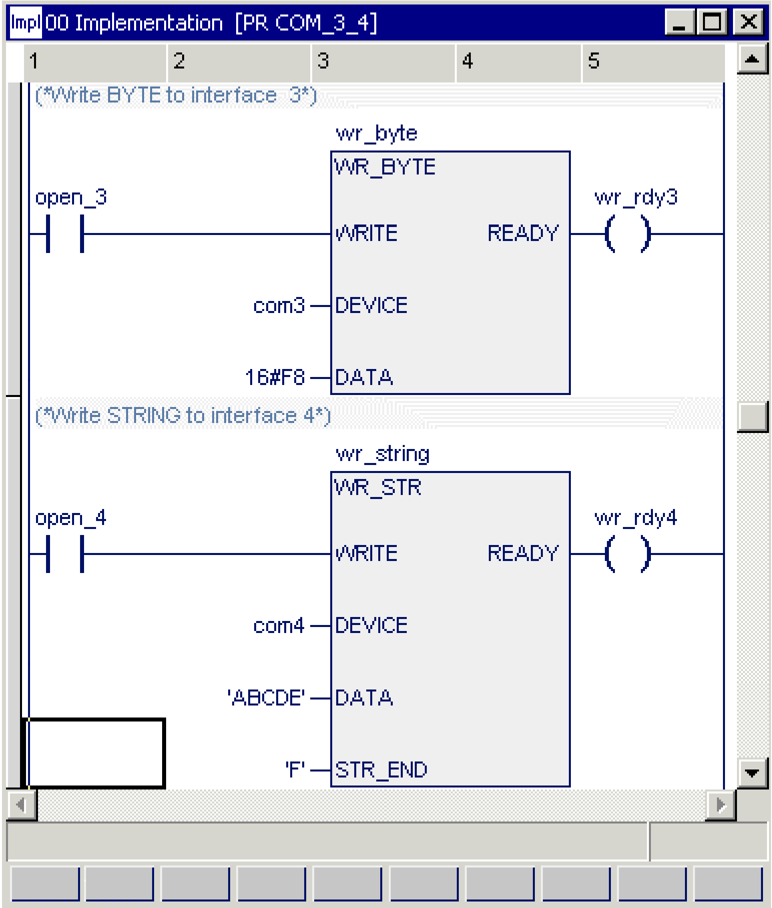
Fig. 13-76: Setting the serial interfaces



Oeffn_Serielle Schnittst.bmp

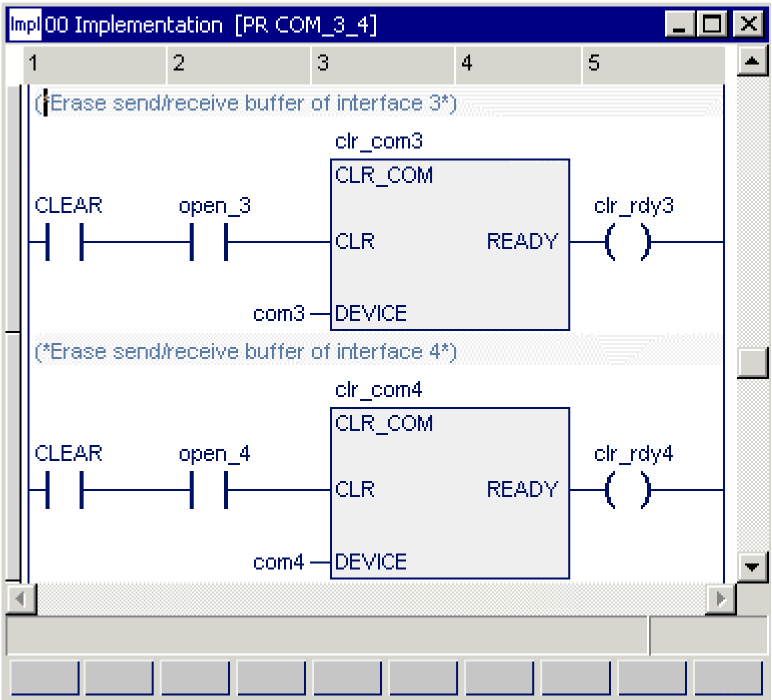
Fig. 13-77: Opening the serial interfaces

Rexroth
Bosch Group



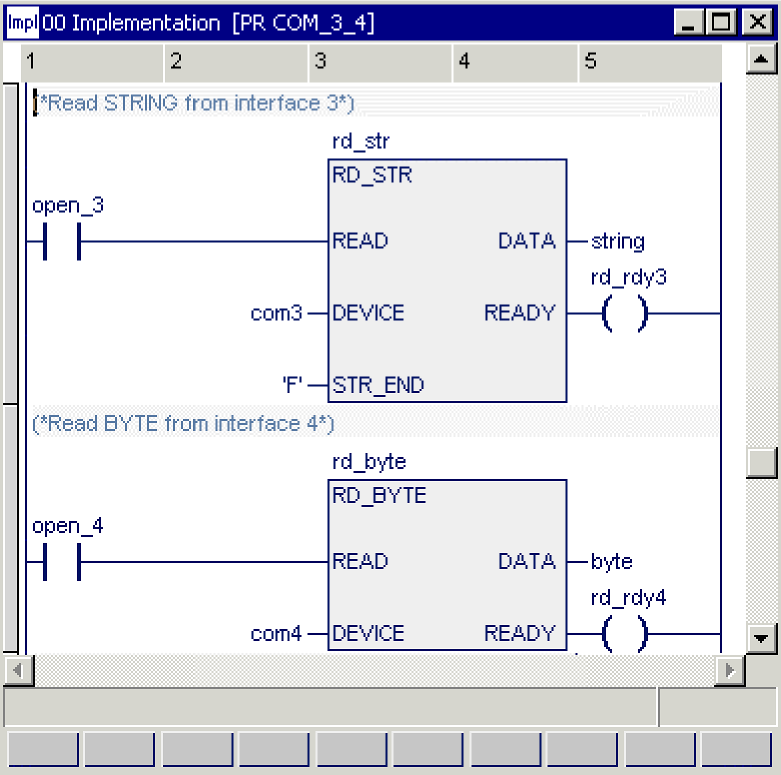
Schreib_Serielle Schnittst.bmp

Fig. 13-78: Writing to serial interfaces



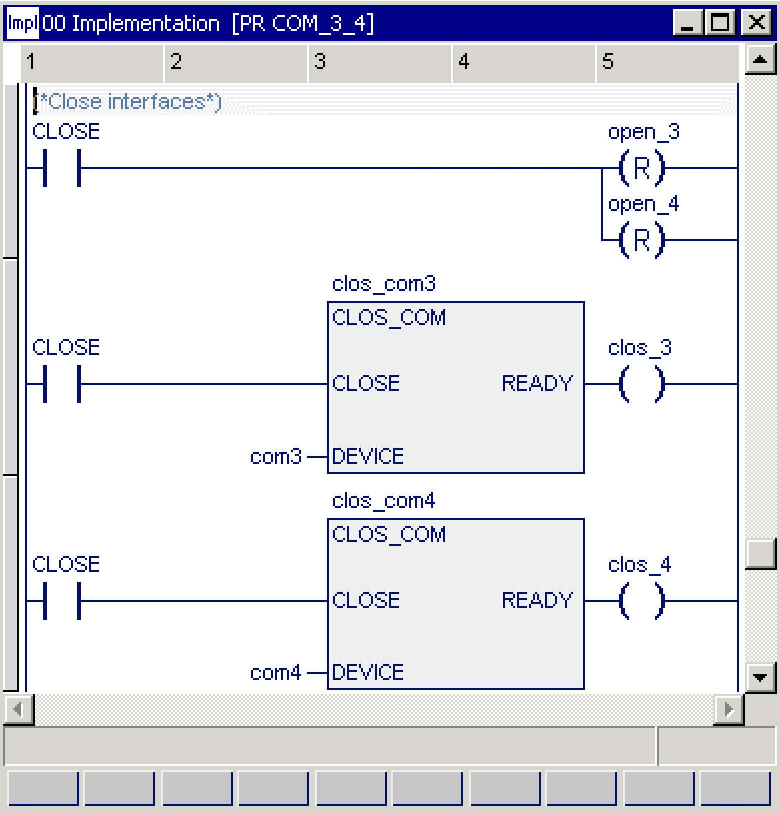
Löschen_Serielle Schnittst.bmp

Fig. 13-79: Clearing the transmitter and receiver buffers of the serial interfaces



Lesen_Serielle Schnittst.bmp

Fig. 13-80: Reading a serial interface



Schliessen_Serielle Schnittst.bmp

Fig. 13-81: Closing the serial interfaces

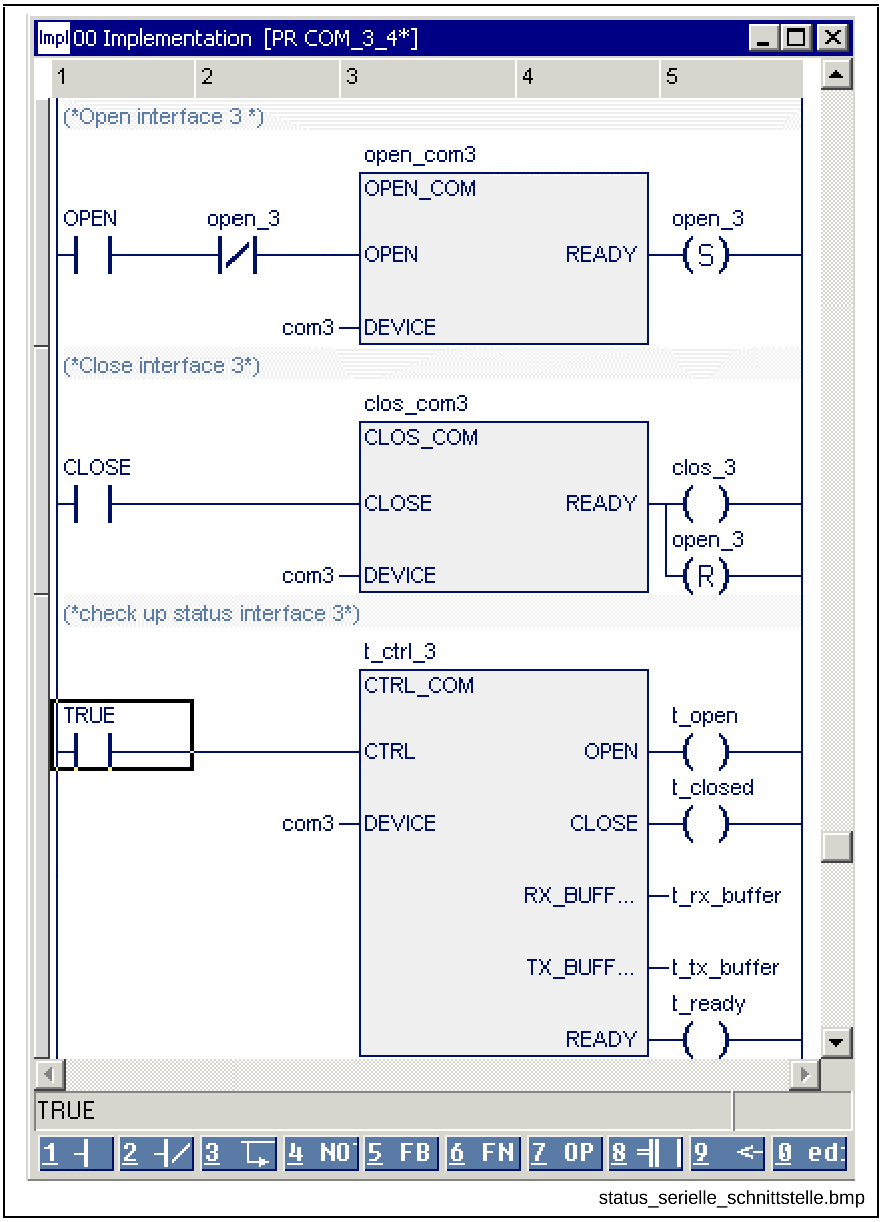


Fig. 13-82: Status test of a serial interface

This results in the following signal play for the present example:

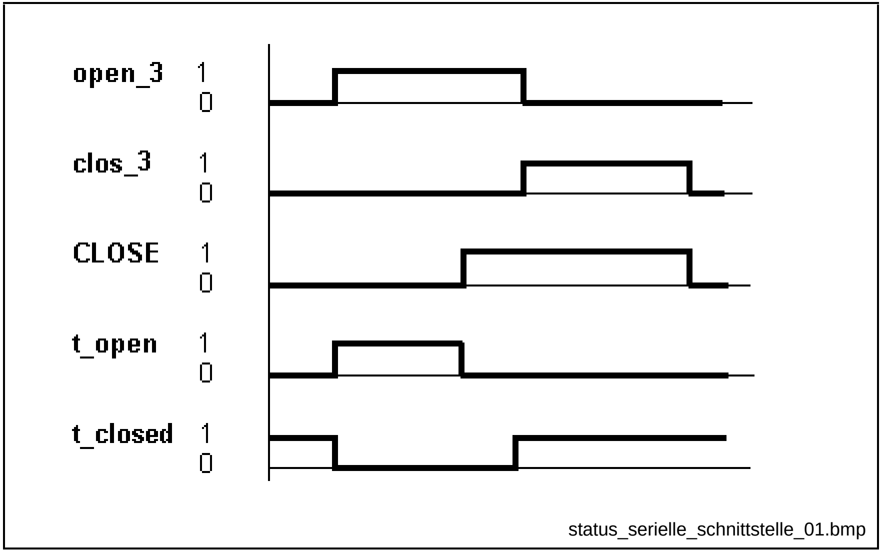


Fig. 13-83: Status inquiry for the above example

Function Blocks for the HMI Interface (GUI_SK16)

The GUI_SK functionality describes a mechanism for a screen-oriented machine operation.

Operating principle

A soft-key bar is assigned to each screen in the "graphical user interface".

Three bits of information are assigned to each soft key:

- Input, flag or output, which is affected with actuation of the corresponding soft key.
- Input, flag or output, whose status information causes the soft key to appear normal or pressed.
- Input, flag or output, whose status information causes the soft key to appear normal or lit.

Depending on the active screen, the graphical user interface delivers the addresses of the outputs or flags, which are to be affected by pressing the soft keys, to the PLC.

With execution of the FB in the PLC program, these outputs or flags are activated when the soft keys are actuated. These outputs or flags are cleared if the soft keys are no longer actuated or if the screen is changed.

The outputs and flags are processed in the PLC program and the relevant machine functions are executed.

’GUI_SK’/GUI_SK16 serve for transmitting the signal status of the soft keys ’SK1 ... SK8(...SK16)’. The softkeys can be derived from any input. A typical example is the derivation from the machine function key of a BTV as shown in the following example.

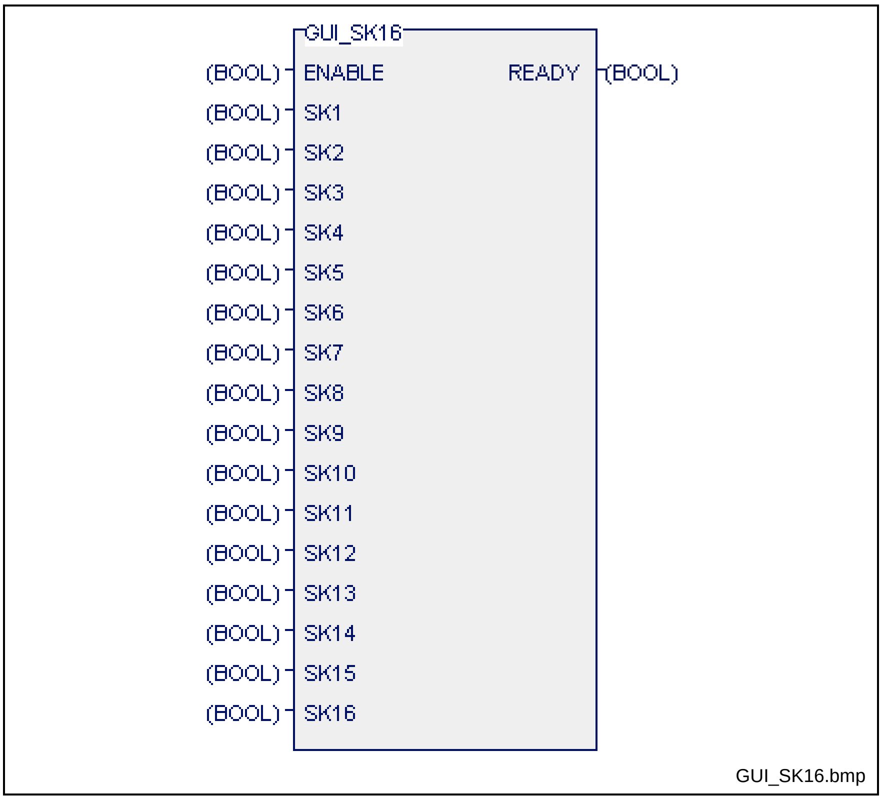
GUI_SK16

For operating the machine, the BTV machine keys R1 to R8 (and L1 to L8) are linked to the function block inputs SK1 to SK16 in the PLC program.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

The block itself is enabled with the "ENABLE" input.

The READY signal is applied to its output.



ENABLE: Enable (FALSE - no execution, TRUE - execution)
SK1: Connection for machine function key 1
to
SK16: Connection for machine function key 16
READY: Acknowledgement (FALSE - no execution, TRUE - execution)

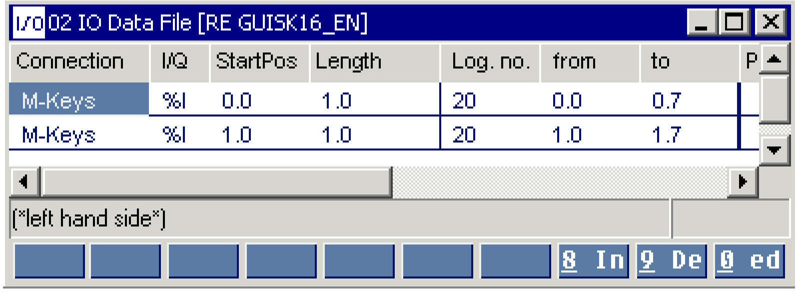
Fig. 13-84: Firmware function block GUI_SK16

Note: Only one instance of the function block GUI_SK16 must be active in a PLC resource.

This instance should be programmed in the leading program in the leading cyclical task at the leading position.

Program Example of HMI Link via GUI_SK16

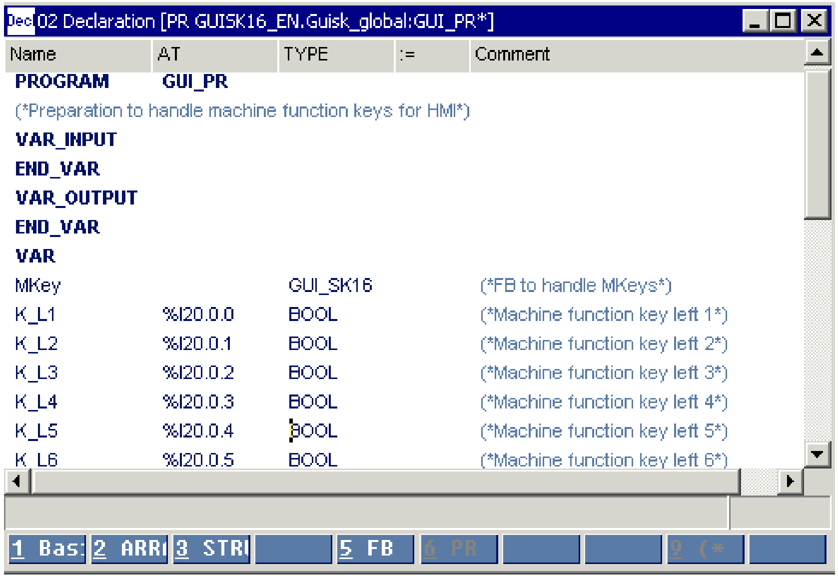
The access to the machine function keys is enabled at resource level in the IO table.



mleys_io.bmp

Fig. 13-85: IO table with access to machine function keys

The variables can either be distributed via VAR GLOBAL / VAR EXTERNAL or - as seen in the example - declared in the program.



mleys_dekl.bmp

Fig. 13-86: Declaration of the machine function keys in a program

They are used in the first network of the implementation of the program.

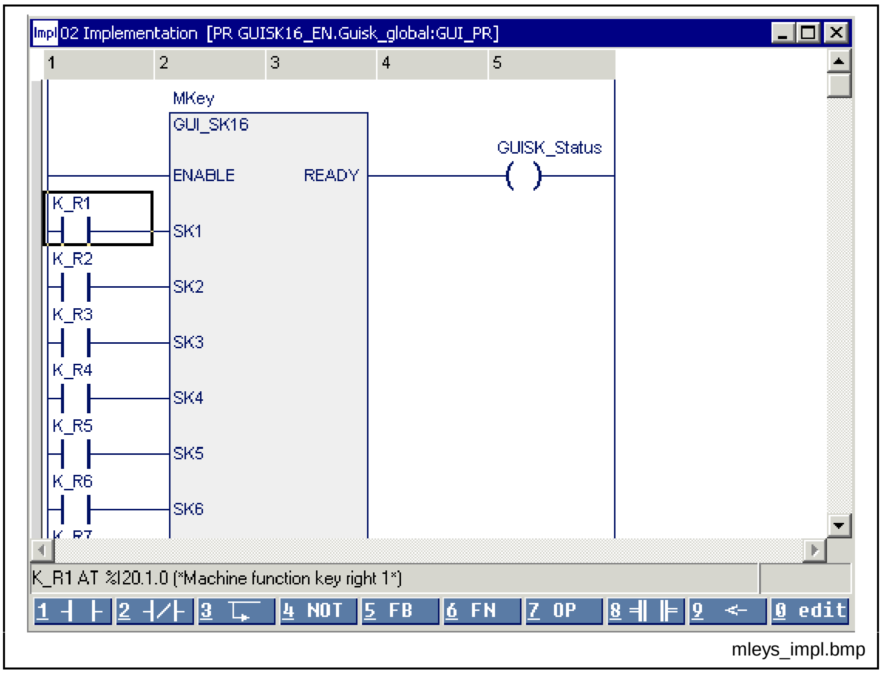


Fig. 13-87: GUI_SK16 - use in the implementation

Miniature Control Panels, Function Blocks for Data Exchange with the PLC

WinPCL provides firmware blocks to couple miniature control panels to the PLC.

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

BTXX, BTXX_2

The function blocks (FB) BTXX and BTXX_2 realize the communication between the PLC and the HMI control panels BTV04, BTV05 and BTC06 via a serial interface. Thereby, data and visualization data are cyclically exchanged. The two blocks differ from eachother in the quantity of the data to be transmitted:

- BTXX: 32 bits
- BTXX_2: 64 bits

The I/O data are up-dated in each PLC cycle.

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

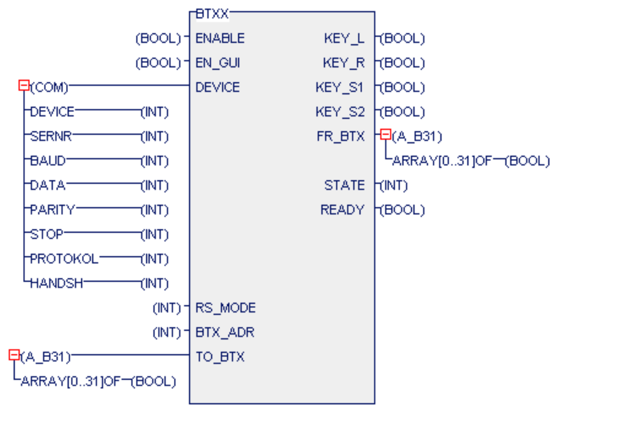
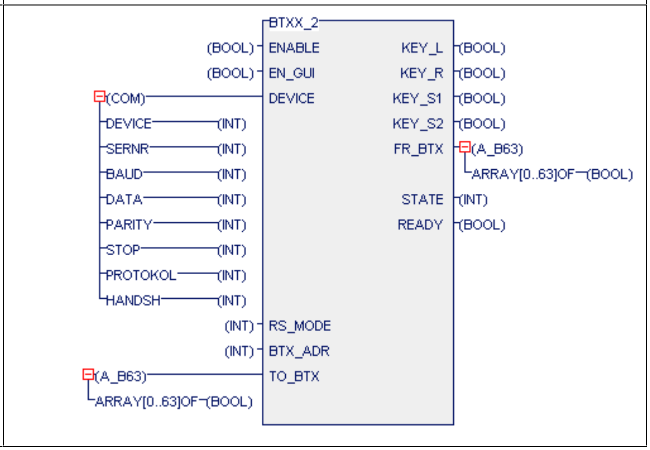
BTXX	BTXX_2
	

Fig. 13-88: Interfaces of the blocks

Input / output variables	Description
ENABLE	The execution of the FB is activated when the ENABLE input is set. The interface is automatically opened. It is not allowed to program an OPEN_COM for the selected interface before. This functionality is integrated in the FB. If this input is deleted, the connection to the control panel is interrupted and the interface is closed again (no CLOS_COM).
EN_GUI	Input EN_GUI serves to activate the GUI_SK functionality for the two machine control keys at the respective control panel.
DEVICE	Input "Device" contains the parameter set of the used serial interface. Due to the used Rexroth communication protocol (SIS = **S**erial **I**ndramat **I**nterface) the following values are predefined: BAUD: **19** (38400 bauds) PARITY: **3** (EVEN) PROTOKOL: **4** (SIS protocol).
RS_MODE	**0** : RS-232, **1** : RS-485, **2** : RS-422
BTX_ADR	Station address of the connected control panel
TO_BTX	32 bits or 64 bits input memory for PLC function keys The applied signals are send to the control panel as Boolean inputs for the PLC function keys.
KEY_L	Status 'Control panel' 'Machine control key' on the left
KEY_R	Status 'Control panel' 'Machine control key' on the right
KEY_S1	Control panel-specific status 1 (optional e.g., life-man-switch)
KEY_S2	Control panel-specific status 2 (optional) The use of the status bits are independant from the connected control panel.
FR_BTX	32 bits or 64 bits output memory for PLC function keys The applied signals are send by the control terminal as Boolean outputs for the PLC function keys.

Input / output variables	Description
STATE	Function block status: 0: Reset status 1: Verify if the FB input parameters are valid 2: Initialize interface 3: Initialize SIS communication distributor 4: Send initaliszation telegram to control terminal 5: Wait for initalization confirmation of the control terminal 6: Communication active
READY	Communication with control panel active

Fig. 13-89: Input / output variables BTXX and BTXX_2

MotionControl Extension of the PLC

To extend the functional range of an ISP in relation to a Motion Control the PLC interface provides the following function blocks:

Function block	Brief description	S#ErrorTyp
CALC_LINEAR_Y	Calculation of the Y value of an default X value when the X-Y value table is predetermined	-303
PID_CONTROL	PID control (P, I, PI, PD, PID control)	-304
AVERAGE_REAL	Calculation of the floating average value with 64 REAL values	-305
AVERAGE_DINT	Calculation of the floating average value with 1023 DINT values	-306
PT2_FILTER	PT2 filter (low pass filter)	-310

Fig. 13-90: Survey of firmware function blocks to extend the functional range of an ISP in relation to a Motion Control

Note: These blocks may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

CALC_LINEAR_Y

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Brief Description

Function block CALC_LINEAR_Y calculates linearized for a default X value of a predetermined X-Y value table the respective Y value.

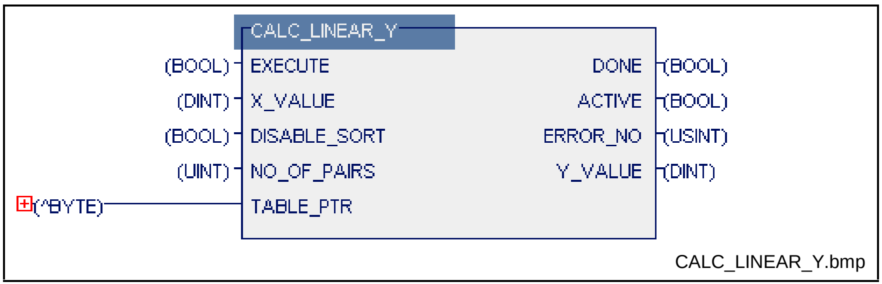


Fig. 13-91: FW function block CALC_LINEAR_Y

Input / output variables	Description
EXECUTE	Processing release of the function block
X_VALUE	Default X value X_{SEARCH}
DISABLE_SORT	Necessity to sort the X-Y value table (FALSE == required, TRUE == not required)
NO_OF_PAIRS	Number of valid X-Y value pairs in the value table (valid: >= 2)
TABLE_PTR	Pointer on value table (Array of Struct)
DONE	Calculation of the Y value completed, output variables valid
ACTIVE	Calculation of the Y value active, output variables invalid
ERROR_NO	Equivalent to S#ErrorNr (error-free == 0)
Y_VALUE	Searched Y value Y_{SEARCH}

Fig. 13-92: Input / output variables CALC_LINEAR_Y

Note: Pointer TABLE_PTR has to refer to an ARRAY VALUE_TABLE whose error message type is a structure COORDINATE_PAIR consisting of two DINT types X and Y.

VALUE_TABLE[Number_of_Pairs] OF COORDINATE_PAIR

with

COORDINATE_PAIR [X; Y] with X {DINT}; Y {DINT}

The number of elements in the value table "Number_of_Pair" is not limited, however, if DISABLE_SORT was not activated, the number should not be too high so that the cycle time is not too heavily charged.

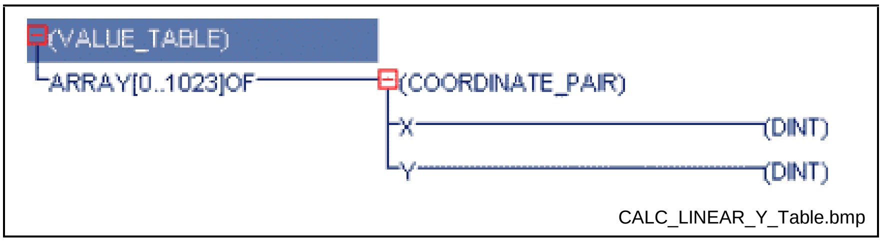


Fig. 13-93: Structure of the value table

Functional Description

Depending on the input DISABLE_SORT and after the processing release of the function block by input EXECUTE function block CALC_LINEAR_Y conditionally arranges the input data array in ascending order according to the X values and writes these back on the input data array.

Note: Therefore, the sorting algorithm is completed within a PLC cycle.

It is required to predetermine an already sorted value table and set the entry DISABLE_SORT on TRUE in order to avoid a heavily increased cycle time when using a larger value table. If this is not possible because of the present data or can not be ensured, it is required to activate the sorting algorithm only for the first application of the value table.

After the conditionally executed sorting of the input value pairs the corresponding Y value is calculated for the X value predetermined at entry X_VALUE. Thereby, the X value of the searched Y value has to be between two default X-Y value pairs.

Note: The AB link between two default X-Y value pairs (X_A;Y_A) → (X_B;Y_B) is always interpreted as straight line AB.

Note: The value table, i. e. the ARRAY is always evaluated as of element 0 and thus, no leading element can be ignored.

Note: With regard to the charge of the PLC cycle time the calculation of the Y value is divided in two PLC cycles time when sorting is activated (DISABLE_SORT == FALSE).

Error Handling

Function block CALC_LINEAR_Y can generate the following system error codes:

System variable	Value	Description
S#ErrorFlg	TRUE	
S#ErrorTyp	-303	
S#ErrorNr	1	Invalid input parameter
S#ErrorNr	2	Overrange (X_{SEARCH} > X_{MAX})
S#ErrorNr	3	Overrange (X_{SEARCH} < X_{MIN})
S#ErrorNr	7	Index error (X_A == X_B)
S#ErrorNr	208	System error (State Machine)

Fig. 13-94: Error codes CALC_LINEAR_Y

The I/O data are up-dated in each PLC cycle.

Application Example

For a predetermined value table the Y value Y_{SEARCH} is searched for a default X value X_{SEARCH} = 12.327. The value table contains four value pairs to be considered. Thus, input NO_OF_PAIRS is set to 4. As the value table is not given in an ascending order the sorting algorithm has to be activated (DISABLE_SORT == FALSE). After the processing release with EXECUTE the function block calculates a Y value Y_{SEARCH} = 23.031 for the default X value X_{SEARCH} = 12.327.

	Default value table (FLOAT)		ARRAY – unsorted (DINT)		ARRAY - ascending order (DINT)	
	X	Y	X	Y	X	Y
0	5.223	7.999	5223	7999	3712	11877
1	3.712	11.877	3712	11877	5223	7999
2	16.121	24.222	16121	24222	9653	22192
3	9.653	22.192	9653	22192	16121	24222

Fig. 13-95: Value table for application example

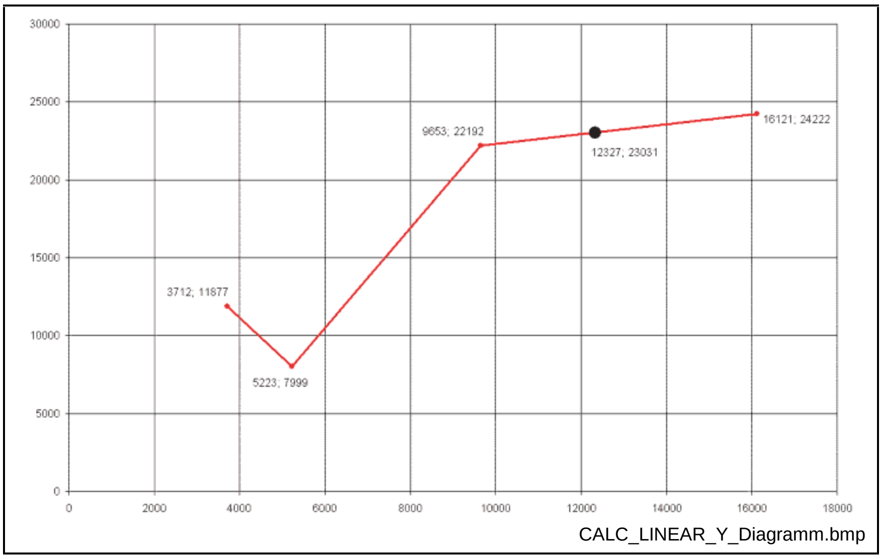


Fig. 13-96: Diagram for application example

PID_CONTROL

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Brief Description

Depending on the wiring of the input variables function block PID_CONTROL provides the control functionalities "P control", "PI control", "PD control", "I control" and "PID control".

Each illustrated control can feature not only a positive but also a negative sense of action. Besides the virtual control function each control type has the possibility to restrict the regulating variable, an anti-reset windup behavior, the capability to shut down the control, as well as a preset functionality for the jerk-free connection of the control.

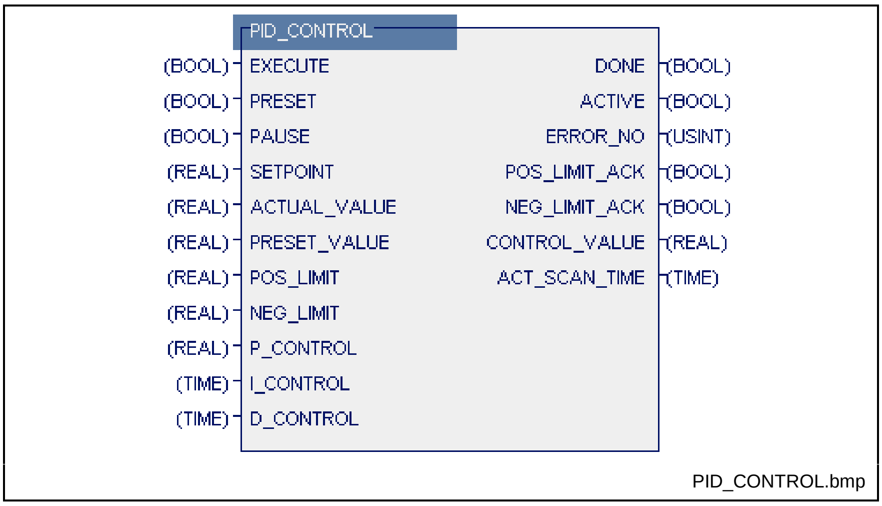


Fig. 13-97: FW function block PID_CONTROL

Input / output variables	Description
EXECUTE	Processing release of the function block
PRESET	Activation of the preset input as controlled variable x (FALSE == inactive; TRUE == active)
PAUSE	Activation of the control shutdown (FALSE == inactive; TRUE == active)
SETPOINT	Reference variable w (command value)
ACTUAL_VALUE	Controlled variable x (actual value)
PRESET_VALUE	Preset value of the controlled variable x (actual value)
POS_LIMIT	Restriction of the regulating variable - Maximum output value of regulating variable y
NEG_LIMIT	Restriction of the regulating variable - Minimum output value of regulating variable y
P_CONTROL	Proportional control gain K_R (P fraction; 0 == deactivated)
I_CONTROL	Control reset time T_N in [ms] (I fraction; 0 == deactivated)
D_CONTROL	Control rate time T_V in [ms] (D fraction; 0 == deactivated)

Input / output variables	Description
DONE	Calculation of regulating variable y completed, output variable valid
ACTIVE	Calculation of regulating variable y active
ERROR_NO	Equivalent to S#ErrorNr (error-free == 0)
POS_LIMIT_ACK	Signal "Maximum restriction of output variable" (FALSE == inactive; TRUE == active)
NEG_LIMIT_ACK	Signal "Minimum restriction of output variable" (FALSE == inactive; TRUE == active)
CONTROL_VALUE	Control output value – regulating variable y
ACT_SCAN_TIME	Control cycle time in [ms]

Fig. 13-98: Input / output variables PID_CONTROL

Note: Because of the cyclic activation of the function block and the following output of the regulating variable the behavior of output DONE and ACTIVE is as follows:

DONE==FALSE,	ACTIVE==FALSE	→	Inactive
DONE==TRUE,	ACTIVE==TRUE	→	Active
DONE==FALSE,	ACTIVE==TRUE	→	Pause
DONE==TRUE,	ACTIVE==FALSE	→	Error

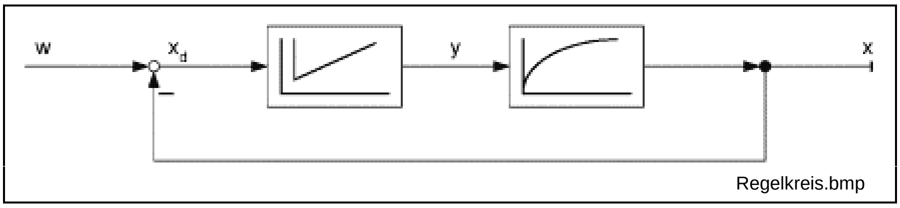


Fig. 13-99: Block diagram 'Control loop'

Theoretical bases

Provided that the scanning time T_A is negligible relative to the time constants T_N and T_V of the control loop an analog control can be transformed to a digital control. The integration of the analog control is approximated by an addition (trapeze approximation), the differentiation by a subtraction (difference formation backwards).

$$y(t) = K_R \cdot \left[x_d(t) + \frac{1}{T_N} \cdot \int x_d(t)\,dt + T_V \cdot \frac{d\,x_d(t)}{dt} \right]$$

Fig. 13-100: Analog PID control

$$y_k = K_R \cdot \left[x_{d,k} + \frac{T_A}{2 \cdot T_N} \cdot \sum_{i=0}^{k} \left(x_{d,i} + x_{d,i-1} \right) + \frac{T_V}{T_A} \cdot \left(x_{d,k} - x_{d,k-1} \right) \right]$$

Fig. 13-101: Digital PID control

The digital PID control can be transformed into a recursive form which constitutes the basis of function block PID_CONTROL.

$$y_{k-1} \quad \text{für} \quad (k-1) \cdot T_A$$

Fig. 13-102: Basis to transform into recursive form

$$y_k = y_{k-1} + K_R \cdot \left[\left(x_{d,k} - x_{d,k-1} \right) + \frac{T_A}{2 \cdot T_N} \cdot \left(x_{d,k} + x_{d,k-1} \right) + \frac{T_V}{T_A} \cdot \left(x_{d,k} - 2 \cdot x_{d,k-1} + x_{d,k-2} \right) \right]$$

Fig. 13-103: Digital PID control - Recursive form (parallel form)

$$y_k = a_1 \cdot y_{k-1} + b_0 \cdot x_{d,k} + b_1 \cdot x_{d,k-1} + b_2 \cdot x_{d,k-2}$$

Fig. 13-104: Digital PID control – General coefficient

Functional Description

In the first (inactive) POU cycle function block PID_CONTROL initializes the deviation $x_{d,k-1}$ and $x_{d,k-2}$ to $x_{d,k}$ as well as the regulating variables y_k and y_{k-1} to the preset value PRESET_VALUE.

If the control variables shall be reintialized for jerk-free connection of the control to the runtime of the POE input PRESET has to be set when the function block (EXECUTE == FALSE) is deactivated.

$$x_{d,k-1} = x_{d,k-2} = x_{d,k} = w - x$$
$$y_k = y_{k-1} = PRESET\_VALUE$$

Fig. 13-105: Initialization of the internal controlled variables

Note: At the time of the initialization of the control variables during the first POU cycle it has to be ensured that the input variables SETPOINT (reference variable w) and ACTUAL_VALUE (controlled variable x) have valid values (measuring ranges).

Note: The regulating variable restriction is also active during the initialization when the values are valid (POS_LIMIT > NEG_LIMIT).

The function block can be used as one of the illustrated controls depending on the connection of the input variables P_CONTROL, I_CONTROL and D_CONTROL. The function block allows to adapt the control type during the runtime of the control.

Control type	**P_CONTROL**	**I_CONTROL**	**D_CONTROL**
P control	Valid value	Zero	Zero
I control	Zero	Valid value	Zero
PI control	Valid value	Valid value	Zero
PD control	Valid value	Zero	Valid value
PID control	Valid value	Valid value	Valid value

Fig. 13-106: Possible control types PID_CONTROL

After the processing release with EXECUTE function block PID_CONTROL controls the controlled variable x on the basis of the selected control type on the reference variable w. Thereby, the regulating variable y can either have a positive or a negative sense of action.

If the regulating variable y reaches the regulating variable restriction POS_LIMIT or NEG_LIMIT, thus, the regulating variable is limited to the respective regulating variable restriction and anti reset windup measures for the integral fraction of the control are started. The anti reset windup measures ensure that in case of a stationary remaining control deviation $x_{d,k}$ the integral fraction is not further integrated and therefore, the reaction time of the control is decelerated when the control deviation changes (sign change). The limitation of the regulating variable y to the regulating variable restriction POS_LIMIT or NEG_LIMIT is signaled at the control by the Boolean output POS_LIMIT_ACK or NEG_LIMIT_ACK.

If the input PAUSE is activated during the runtime of function block PID_CONTROL, the current control deviation $x_{d,k}$ is set to zero and the control is thus stopped. In the second activation cycle input PAUSE affects indirectly on $x_{d,k-1}$ and in the third activation cycle on $x_{d,k-2}$. Input PAUSE is suited to keep the regulating variable y in the settled status.

Output ACT_SCAN_TIME specifies the time difference between the calculation time of the control difference $x_{d,k}$ and the calculation time of the control difference $x_{d,k-1}$ and allows thus to provide the regulating variable with a time marker.

Note: Output ACT_SCAN_TIME is also carried along during the activation of input PAUSE.

Note: The control coefficients are calculated when the control type or the time difference change.

In order to avoid that the PLC cycle time is charged by a permanent calculation of the coefficient, it is required to use the function block in a time-controlled task.

Error Handling

Function block PID_CONTROL can generate the following system error codes:

System variable	Value	Description
S#ErrorFlg	TRUE	
S#ErrorTyp	-304	
S#ErrorNr	1	Invalid input parameter
S#ErrorNr	208	System error (State Machine)

Fig. 13-107: Error codes PID_CONTROL

The I/O data are up-dated in each PLC cycle.

AVERAGE_REAL

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Brief Description

Function block AVERAGE_REAL calculates the floating average value form maximum 64 REAL values.

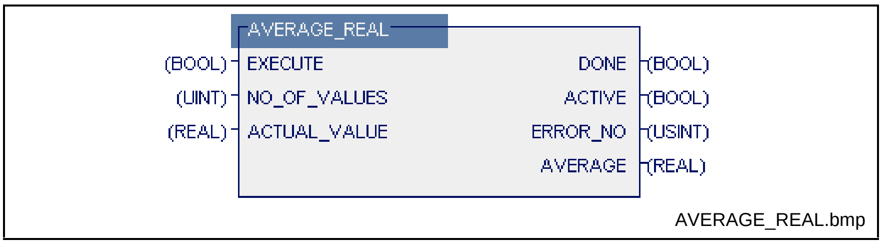


Fig. 13-108: FW function block AVERAGE_REAL

Input / output variables	Description
EXECUTE	Processing release of the function block
NO_OF_VALUES	Number of values with which the average value is to be calculated (valid: 1..64)
ACTUAL_VALUE	Current input value
DONE	Calculation of the floating average value completed, output variables valid
ACTIVE	Calculation of the floating average value active
ERROR_NO	Equivalent to S#ErrorNr (error-free == 0)
AVERAGE	Floating average value

Fig. 13-109: Input / output variables AVERAGE_REAL

Note: Because of the cyclic activation of the function block and the following output of the regulating variable the behavior of output DONE and ACTIVE is as follows:

DONE==FALSE,	ACTIVE==FALSE	→	Inactive
DONE==TRUE,	ACTIVE==TRUE	→	Active
DONE==TRUE,	ACTIVE==FALSE	→	Error

Functional Description

After the processing release with EXECUTE function block AVERAGE_REAL reads the REAL value at the input ACTUAL_VALUE in an internal FIFO register. Then, the average value of the whole FIFO register whose data width was determined by means of NO_OF_VALUES is calculated and emitted at output AVERAGE.

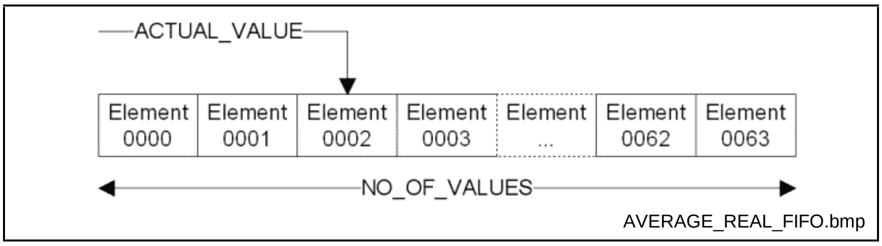


Fig. 13-110: FIFO register AVERAGE_REAL

Error Handling

Function block AVERAGE_REAL can generate the following system error codes:

System variable	Value	Description
S#ErrorFlg	TRUE	
S#ErrorTyp	-305	
S#ErrorNr	1	Invalid input parameter
S#ErrorNr	208	System error (State Machine)

Fig. 13-111: Error codes AVERAGE_REAL

The I/O data are up-dated in each PLC cycle.

AVERAGE_DINT

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Brief Description

Function block AVERAGE_DINT calculates the floating average value from maximum 1023 DINT values.

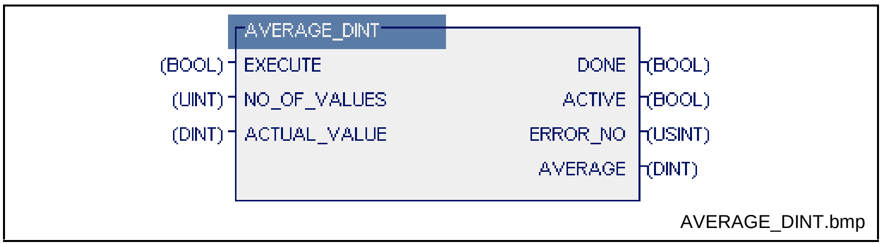


Fig. 13-112: FW function block AVERAGE_DINT

Input / output variables	Description
EXECUTE	Processing release of the function block
NO_OF_VALUES	Number of values with which the average value is to be calculated (valid: 1..1023)
ACTUAL_VALUE	Current input value
DONE	Calculation of the floating average value completed, output variable valid
ACTIVE	Calculation of the floating average value active
ERROR_NO	Equivalent to S#ErrorNr (error-free == 0)
AVERAGE	Floating average value

Fig. 13-113: Input / output variable AVERAGE_DINT

Note: Because of the cyclic activation of the function block and the following output of the regulating variable the behavior of output DONE and ACTIVE is as follows:

DONE==FALSE,	ACTIVE==FALSE	→	Inactive
DONE==TRUE,	ACTIVE==TRUE	→	Active
DONE==TRUE,	ACTIVE==FALSE	→	Error

Functional Description

After the processing release with EXECUTE function block AVERAGE_DINT reads cyclically the DINT value at input ACTUAL_VALUE in an internal FIFO register. Then, the average value of the whole FIFO register whose data width was determined by means of NO_OF_VALUES is calculated and emitted at output AVERAGE.

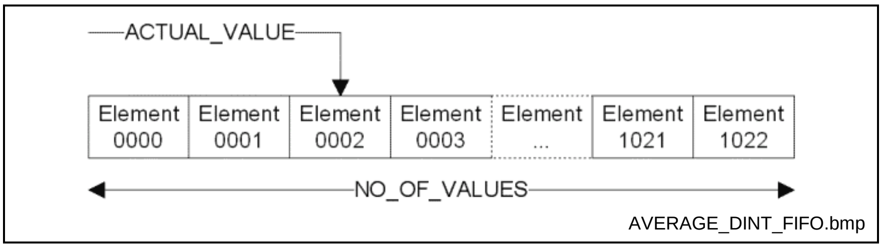


Fig. 13-114: FIFO register AVERAGE_DINT

Error Handling

Function block AVERAGE_DINT can generate the following system error codes:

System variable	Value	Description
S#ErrorFlg	TRUE	
S#ErrorTyp	-306	
S#ErrorNr	1	Invalid input parameter
S#ErrorNr	208	System error (State-Machine)

Fig. 13-115: Error codes AVERAGE_DINT

The I/O data are up-dated in each PLC cycle.

PT2_FILTER

Note: This block may not be declared in the retain area (see section Limitation of the Declaration of Function Blocks in the Retain Area).

Brief Description

Function block PT2_FILTER serves for low pass filtering of a signal, i. e. in its function as transfer element the signal fractions with low frequencies are unchanged transmitted and the fractions with high frequencies are attenuated or blocked.

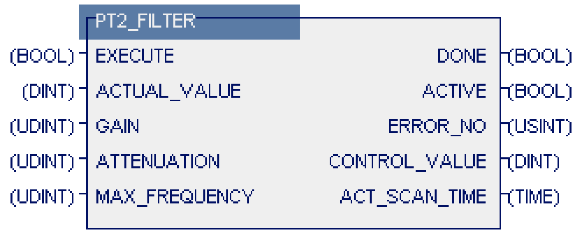


Fig. 13-116: FW function block PT2_FILTER

Input / output variables	Description
EXECUTE	Processing release of the function block
ACTUAL_VALUE	Current input value
GAIN	Gain K_P
ATTENUATION	Attenuation D
MAX_FREQUENCY	Limit frequency f_{Grenz}
DONE	Calculation of filtered value completed, output variables valid
ACTIVE	Calculation of filtered value active
ERROR_NO	Equivalent to S#ErrorNr (error-free == 0)
CONTROL_VALUE	Filtered value
ACT_SCAN_TIME	Filter cycle time t_{Abtast} in [ms]

Fig. 13-117: Input / output variables PT2_FILTER

Note: Because of the cyclic activation of the function block and the following output of the regulating variable the behavior of output DONE and ACTIVE is as follows:

DONE==FALSE, ACTIVE==FALSE → Inactive
DONE==TRUE, ACTIVE==TRUE → Active
DONE==TRUE, ACTIVE==FALSE → Error

Functional Description

After the processing release with EXECUTE function block 0PT2_FILTER calculates cyclically the filtered value CONTROL_VALUE.

For this purpose, the analog transfer function of the PT_2 element is calculated time-restricted in compliance with Shannon’s theorem which restricts the maximum usable frequency range to half of the scanning frequency f_{Abtast}.

$$f_{Abtast} > 2 \cdot f_{Grenz}$$

Fig. 13-118: Shannon’s theorem

$$G(s) = \frac{x_a(s)}{x_e(s)} = \frac{K_P}{1 + 2 \cdot D \cdot \frac{s}{\omega_0} + \frac{s^2}{{\omega_0}^2}}$$

Fig. 13-119: Transfer function of a PT_2 element

Note: The filter coefficients are recalculated when the scanning time changes, whereby the scanning time corresponds to the POU cycle time.

To avoid to charge the PLC cycle time by permanent calculation of the coefficients, it is required to use the function block in a time-controlled task.

Error Handling

Function block PT2_FILTER can generate the following system error codes:

System variable	Value	Description
S#ErrorFlg	TRUE	
S#ErrorTyp	-310	
S#ErrorNr	1	Invalid input parameter
S#ErrorNr	2	Overrange (f_{Abtast} <= 2 * f_{Grenz})
S#ErrorNr	208	System error (State-Machine)

Fig. 13-120: Error codes PT2_FILTER

The I/O data are up-dated in each PLC cycle.

13.4 User Function Blocks

The programming system permits the user to write function blocks himself, which can be used as re-usable units in the form of a supplement to the standard and firmware function blocks. The user function blocks can import other user functions or user function blocks and use them in the same way as the standard and firmware functions or function blocks.

A structuring with SFC sequential function chart elements is possible, external variables can be used.

Import Rules, Function Blocks

A user function block can use user, standard and firmware functions as well as user, standard or firmware function blocks:

The used function is a standard- or firmware function	**The used function is a user function**	**The used function block is a standard or firmware function block**	**The used function block is a user function block**
Direct use without import or declaration according to the selected input language.	Automatic import of the function, at least its declaration should exist. Use according to the selected input language.	No import necessary, as contained in the standard library. A function block requires space for the data pertaining to its assignment (counter value, runtime, and the like). The required space has to be reserved by the unit wishing to use it. Declaration of the assignment of the FB in a separate declaration editor. Use according to the selected input language.	Automatic import of the function block, at least its declaration should exist. Declaration of the assignment of the FB. Use according to the selected input language.

The nesting can be continued to any depth desired.

It is **forbidden** that function block ‘A’ uses itself again (recursion) or that function block ‘A’ uses function block ‘B’ and the latter uses function block ‘A’ again, etc.

13.5 Limitation of the Declaration of Function Blocks in the Retain Area

In principle, the declaration of function blocks in the retain area is permitted according to IEC-61131-3.

However, there are limitations which must be observed whenever Rexroth system function blocks are used. The following list specifies the function blocks which may not be declared in the retain area.

Note: These blocks may neither be declared directly in the retain area nor indirectly via blocks which are declared in the retain segment themselves.

Timer function blocks

Programming of timer stages is not permitted in the retain area.

- TP
- TON
- TOFF
- FLASH

Serial interface

Function blocks for supporting serial interfaces in the PLC application program:

- OPEN_COM
- CLOS_COM
- BTXX
- BTXX_2

Bus communication function blocks

Communication with other control components

Data exchange with other control components is achieved using function blocks via special data channels in the common dual-ported RAM. Usually, more than one communication cycle is required for data exchange with the CNC.

The progress made in data exchange is mapped as state-machine in the particular function block concerned. In other words, the internal state of the function block depends on the state of the DPR. However, the DPR is always re-initialized after a reset.

Current communication cycles will be lost. As a result, the state-machine in the FB is invalid, if this FB has been declared to be remanent.

The following function blocks may not be declared to be remanent:

MTCNC	**SYNAX**
NC memory selection	MC_CHANGE_PHASE
SEL_MEM	MC_RD_PARAMETER
ACT_MEM	MC_WR_PARAMETER
	MC_WR_LISTDATA
Process data channel	MC_DIAGNOSIS
AXD_WR	MC_RD_LISTDATA
AXD_RD	MC_RD_DATASTATUS
DCD_RD	MC_ABORT_TRANSMISSION
DCD_WR	MC_RW_PTR_TLG
MTD_WR	MC_RD_PHASE
MTD_RD	MC_RD_ATTRIBUTE
NCVAR_RD	MC_RD_NAME
NCVAR_WR	MC_RD_UNIT
OTD_WR	MC_RD_MIN_VALUE
OTD_RD	MC_RD_MAX_VALUE
TL_DELETE	MC_RD_ELEMENT
TL_ENABLE	MC_WR_ELEMENT
TL_MOVE	MC_RD_ARRAY
TL_RESET	MC_WR_ARRAY
TLBD_RD	MC_RW_ARRAY_TLG
TLBD_WR	
TLD_WR	
TLD_RD	
TLED_RD	
TLED_WR	

Fig. 13-121: Coupling with other control components

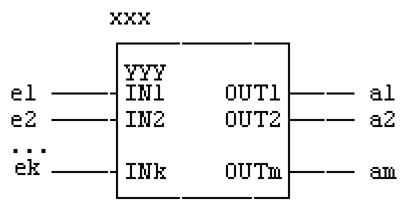
14 Programs and Resources in WinPCL

14.1 Programs, General Information

As is the case with function blocks, a program (PROGRAM, PR) is a program organization unit which provides the following:

- 1...k inputs,
- 1...m outputs and
- internal variables,
- and can use external variables.

A program is the smallest unit in the programming system which can be loaded and started.



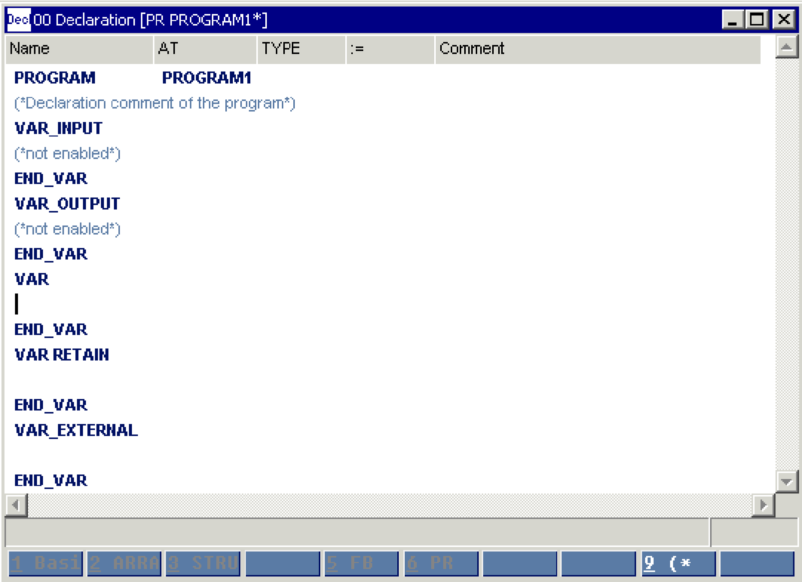
pr_allgemein.bmp

xxx	-	Instance name (name of the PR assignment)
yyy	-	Type name of the PR
ei	-	Inputs of program 1...k
aj	-	Outputs of program 1...m

Fig. 14-1: Program, general interface

Delivery of values to the input variable and transmission of the values to the output variable has to take place at resource level. This possibility is not available at present!

Note: Since use of VAR_INPUT and VAR_OUTPUT is still disabled at the moment, it is prohibited to enter these variables in the declaration part.



dekl_pr.bmp

Fig. 14-2: Declaration part of a program

Contrary to the function block, it is permitted to use absolutely addressed variables in the program.

Area	Variable type	Use
VAR	%I	Input variable, write protection
VAR	%Q	Output variable
VAR	%M	Flag variable
VAR RETAIN	%R	Buffered flag

Fig. 14-3: Permitted absolutely addressed variables

The user can decide how to obtain absolute addressed variables:

- At program level: absolute variables declared in VAR or VAR RETAIN: these variables should only be used in this specific program.
- At resource level: absolute variables declared in VAR or VAR RETAIN, which are globally enabled: these variables can be used in several programs via VAR_EXTERNAL.

Note: Repeated declarations of variables with same addresses in different programs should be avoided for a better program transparency.

The **IEC concept** provides for a basic separation between the program code of the program and the data memory required for saving the values of the variables.

The programming system allows the user to write programs himself and to use them repeatedly. The user programs can import other user functions or user function blocks and use them in the same way as the standard and firmware functions or function blocks.

A structuring with SFC sequential function chart elements is possible, external variables can be used.

Import Rules

A user program can use user, standard and firmware functions as well as user, standard or firmware function blocks:

The used function is a standard- or firmware function	**The used function is a user function**	**The used function block is a standard or firmware function block**	**The used function block is a user function block**
Direct use without import or declaration according to the selected input language.	Automatic import of the function, at least its declaration should exist. Use according to the selected input language.	No import necessary, as contained in the standard library. A function block requires space for the data which belong to the assignment (counter value, runtime, or the like). The required space has to be reserved by the unit wishing to use it. Declaration of the assignment of the FB in a separate declaration editor. Use according to the selected input language.	Automatic import of the function block, at least its declaration should exist. Declaration of the assignment of the FB. Use according to the selected input language.

The nesting can be continued to any depth desired.

14.2 Resources

A resource is the uppermost level in the programming system.

Note: The name of a resource may not exceed a length of 32 characters.

If this length is exceeded, excess characters may be cut off outside of Win PCL.

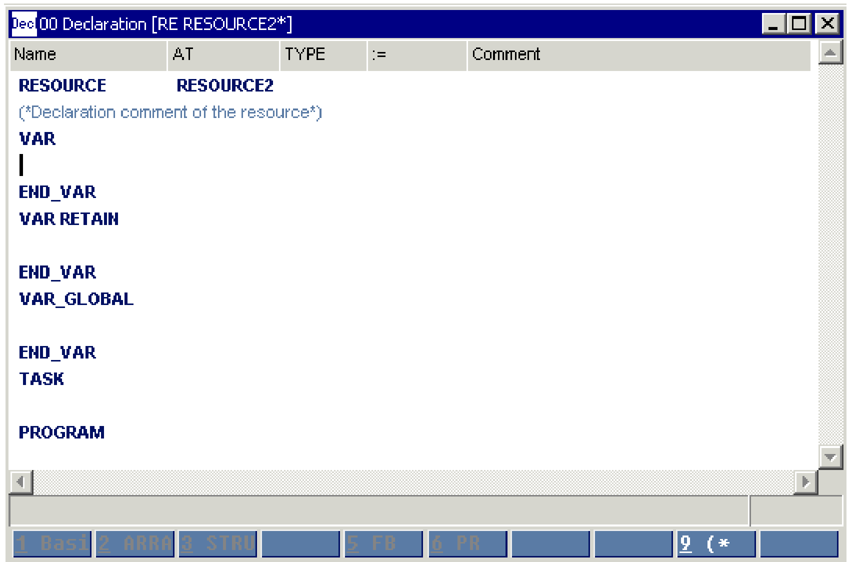
In the current version, the resource must fulfill the following tasks:

- Establishment of a connection to the IO interface of the control ("View / IO editor").
- Providing the information, which is assigned to BTV file(s) of the resource ("Extras / Miniature control panels" menu item).
- Providing the diagnosis module assignment data ("Extras / Diagnosis Module Assignment").
- Declaration of variables in the VAR and VAR RETAIN areas. Absolute addresses can be assigned to these variables .

Area	Type	Use
VAR	%I	Input variable, write protection
VAR	%Q	Output variable
VAR	%M	Flag variable
VAR RETAIN	%R	Buffered flag

Fig. 14-4: Permitted absolutely addressed variables

- Enabling of declared variables for global application is allowed; the name of the variable is significant.



dekl_re.bmp

Fig. 14-5: Declaration part of a resource

- Moreover, the resource level is used to declare the tasks available for the resource. The header changes as shown in the table below if the cursor is positioned below the keyword "TASK" (Tasks, Time Diagrams of the Execution):

Name	ENABLE	PRIORITY	INTERVAL	Comment
Slow	TRUE	250		Cyclic task
Fast	FastEnable	1	T#2ms	Time-controlled task

Fig. 14-6: Possible tasks of a resource

Column 1:

Entry of the task name.

Column 2:

ENABLE, Boolean variable or TRUE for controlling the task enabling.

The variable has to be declared globally and can be controlled via VAR-EXTERNAL by a program or a function block.

Note: The task can be stopped by a program instance or a function block, that runs under its control, but cannot be restarted.

Column 3:

PRIORITY, the priority of a task can be between 0, highest priority, and 65535, lowest priority. A program under a task with a higher priority interrupts a program under a task with a lower priority. Time-controlled or edge-controlled tasks should always have a higher priority than cyclic tasks so that they can be activated for execution.

Column 4:

INTERVAL, indicates the time frame for starting the task. If this column is empty, the task is restarted immediately after completed execution, otherwise after the defined interval.

Note: A total of eight tasks are enabled:

- cyclic tasks: at least one task
- time-controlled tasks: no more than seven tasks
- edge-controlled tasks: not enabled so far

The last section in the declaration contains the declaration of program instances and their assignment to already declared tasks. The header changes as shown in the table below if the cursor is below the keyword "PROGRAM":

Name	WITH	TYPE	Comment
pr_instance_1	Slow	PR_TYPE	Background program
pr_fast_inst	Fast	PR_FAST_TYPE	Fast cyclic program

Fig. 14-7: Assignment of program instances to tasks

Column 1:

Contains the instance name of the program.

Column 2:

Contains the name of the task under the management of which the program has to run.

Column 3:

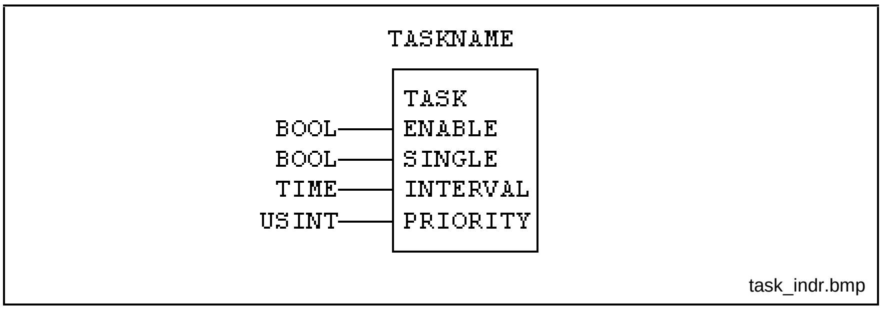
Contains the type name of the program.

14.3 Tasks, Time Diagrams of the Execution

A **TASK** is defined as an execution control element, which is able to initiate the execution of a set of program instances on cyclic or time-controlled basis or, in future, when the rising edge of a specific Boolean variable occurs.

Tasks and their connections to program instances are represented textually as a part of the resource (cf. Resources).

A task is implicitly permitted through the connected resource. In addition to the definitions in EN 61131-3, it can be enabled by means of the "ENABLE" input or its execution can be disabled.



ENABLE:	Enable input:	FALSE, task disabled (standard)
	TRUE, task enabled	
SINGLE:	Execution with rising edge, not connectable so far	
INTERVAL:	Cyclic execution if not connected (standard), interval for time-controlled execution, if connected	
PIORITY:	Priority input 0 (highest priority)... 65535 (lowest priority)	

Fig. 14-8: Task, shown as graphical diagram

If the task is enabled, control of the program instances has preemptive scheduling according to the following rules:

- preemptive scheduling: A task scheduled for execution with higher priority interrupts the execution of another task with lower priority, that means the task with the lower priority is not executed until the execution of the tasks with the higher priority is completed. A task cannot interrupt tasks with same or higher priority.

Note: Dependent on the scheduled priorities it is possible, that a task cannot start the execution at the moment it was scheduled for execution.

- **Time-controlled task** (can be disabled via ENABLE=FALSE):
  - Start takes place equidistantly; the interval time is specified through the INTERVAL input.
  - If a time-controlled task of low priority meets a higher-priority cyclic or time-controlled task, *it is placed in a waiting list until the higher-priority task is completed. If the tasks meet several times, processing is done only once.*

- **Cyclic task** (can be disabled via ENABLE=FALSE):
  - Each cyclic task is executed once in every cycle of the resource.
  - The execution order is determined by the visible order in the declaration in the resource file. Therefore, it is independent of the priority.
  - The priority is decisive with respect to interruptions by time-controlled tasks.
- Each task organizes the **updating** of its **I/O map**.
- In addition, the **I/O map of the resource**, i.e. the global absolutely addressed variables are updated by the task.

Note: The global variables for the remaining execution of an interrupted program instance can be changed if the execution of a task was interrupted !

Example of the execution of a task, Start with t=0:

Name	ENABLE	PRIORITY	INTERVAL	Comment
High	FastEnable	2	T#6ms	Time-controlled task, high priority
Medium	TRUE	5	T#18ms	Time-controlled task, low priority
Low	TRUE	100		Cyclic task, high priority
Basic	TRUE	250		Cyclic task, low priority

Fig. 14-9: Task declaration in the declaration part of the resource

Name	WITH	TYPE	Comment
pr_B1	High	PR_TYPE_4	Execution time: 2ms
pr_A1	Medium	PR_TYPE_2	Execution time: 2ms
pr_A2	Medium	PR_TYPE_3	Execution time: 2ms
pr_0	Low	PR_TYPE_0	Execution time: 4ms
pr_1	Basic	PR_TYPE_1	Execution time: 4ms

Fig. 14-10: Assignment of program instances to tasks

Note: By the order selected for program instances (descending priority of the tasks), cyclic tasks can be executed by descending priority.

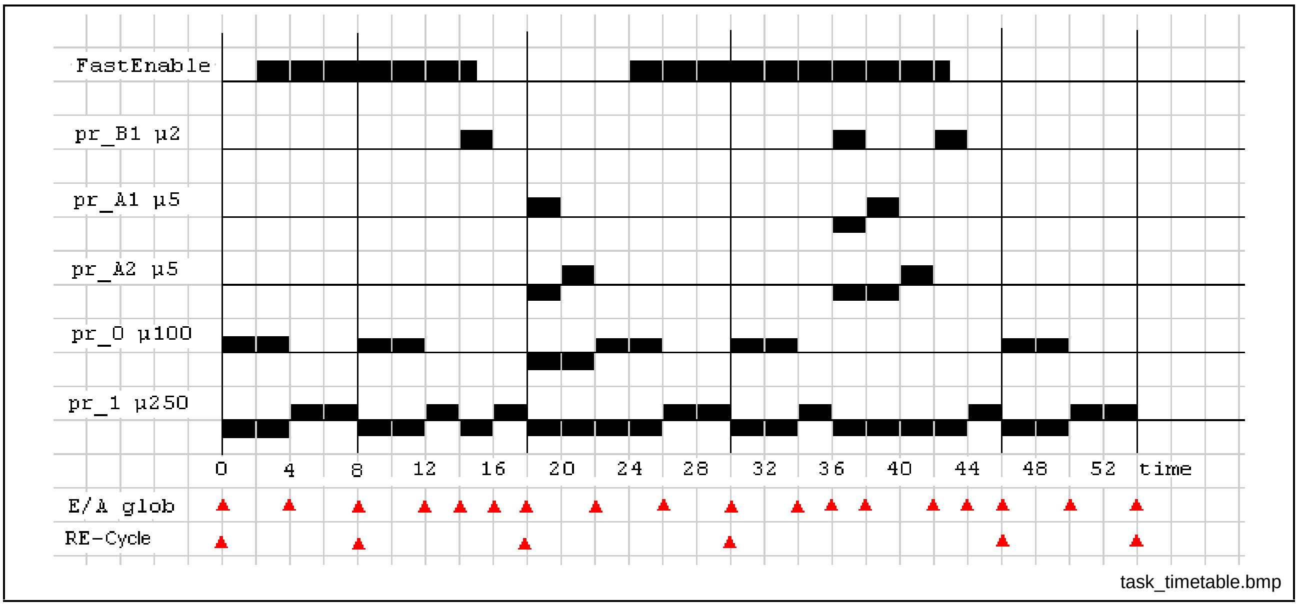
The Boolean variable "FastEnable" is assumed to be globally enabled with the preassignment "FALSE". After t=2ms and t=24ms, it is deactivated by the program pr_1 applied to "TRUE" and, after 15 ms and 43 ms, by the program pr_B1 (self-deactivation). FastEnable is contained in these programs as VAR_EXTERNAL!

The order of executing the programs pr_A1 and pr_A2, same task, complies with the visible order of the entry in section Resources.

Notation "X µ Y" in the following time schedule indicates that the POU "X" is scheduled with priority "Y" and is being executed.

t/ ms	Executing	Waiting
0	pr_0 µ100	(Cycl), pr_1 µ250 (0ms)
2	pr_0 µ100	(FastEnable= TRUE), pr_1 µ250 (0ms)
4	pr_1 µ250	
8	pr_0 µ100	(Cycl), pr_1 µ250 (0ms)
12	pr_1 µ250	
14	pr_B1 µ2	pr_1 µ250 (2ms)
15	pr_B1 µ2	(FastEnable= FALSE), pr_1 µ250 (2ms)
16	pr_1 µ250	
18	pr_A1 µ5	(Cycl),pr_A2 µ5, pr_0 µ100 (0ms), pr_1 µ250 (0ms)
20	pr_A2 µ5	pr_0 µ100 (0ms), pr_1 µ250 (0ms)
22	pr_0 µ100	pr_1 µ250 (0ms)
24	pr_0 µ100	(FastEnable= TRUE), pr_1 µ250 (0ms)
26	pr_1 µ250	
30	pr_0 µ100	(Cycl), pr_1 µ250 (0ms)
34	pr_1 µ250	
36	pr_b1 µ2	pr_A1 µ5, pr_A2 µ5, pr_1 µ250 (2ms)
38	pr_A1 µ5	pr_A2 µ5, pr_1 µ250 (2ms)
40	pr_A2 µ5	pr_1 µ250 (value: 2ms)
42	pr_b1 µ2	pr_1 µ250 (value: 2ms)
43	pr_b1 µ2	(FastEnable= FALSE), pr_1 µ250 (value: 2ms)
44	pr_1 µ250	
46	pr_0 µ100	(Cycl), pr_1 µ250 (0ms)
50	pr_1 µ250	

Fig. 14-11: Time schedule



X µY:	Instance X is scheduled or executed with priority Y
E/A global:	Updating of the global variables
RE-Cycle:	Flags indicate the cycle start of the resource

Fig. 14-12: Time table for program execution

A FALSE / TRUE status change of the Boolean variable FastEnable (task enable high) is identified at the beginning of the next resource cycle. The first activation takes place one interval time (here 6 ms) later.

- Edge: t=2ms, cycle start: t=8ms, activation t=14ms
- Edge: t=24ms, cycle start: t=30ms, activation t=36ms

The cycle of the resource is completed always after execution of all cyclic programs (pr_0/priority 100, pr_1 /priority 250 in the example), that means at the instant: 8 msec, 18 msec, 30 msec, 46 msec, etc.

14.4 Management of Global and Local Data

User data are managed in the PLC in data segments. Thereby, it is distinguished between compiler-assigned and absolute data as well as between local and global data.

Local Data

The validity area of the variables is restricted to the respective program organization unit (POU).

Compiler-assigned local data

Declared variables without absolute reference are compiler-assigned data, that are specified free from conflicts by the compiler. Only volatile and non-volatile (remanent) data are distinguished:

```
VAR
fvar    :    BYTE;  (* compiler-assigned volatile data *)
END_VAR
```

```
VAR RETAIN
rvar    :    BYTE;  (* compiler-assigned remanent data *)
END_VAR
```

Absolute data

Declared variables with absolute reference are absolute data. They are linked with the aid of the absolute reference to a concrete memory address by the user. Thereby, no conflict test is executed. Either several symbolic variables of the same data type can be linked to one reference or memory areas can be overlayed by variables with different data types. Volatile and non-volatile absolute (remanent) data are distinguished.

Note: Compiler-assigned and absolute data never coincide.

Absolute data are permitted in resources and programs.

```
VAR
fabs    AT    %MB4   :    BYTE;  (* absolute volatile data *)
END_VAR

VAR RETAIN
rabs    AT    %RB6   :    BYTE;  (* absolute remanent data *)
END_VAR
```

Global Data

Basically, global variables are assigned to the resource. The validity area of the global variables is the resource and each program organization unit (POU) used by this resource and in which they are declared as external variables. Compiler-assigned and absolute global variables are distinguished.

Compiler-assigned global data

Compiler-assigned global variables are declared in the declaration editor of the resource in the area VAR - END_VAR or VAR RETAIN - END_VAR. So, they are initially local variables of the resource. To assign to this variables the property global, they have to be entered in the area VAR GLOBAL - END_VAR of resource’s declaration editor (To copy the block is possible.)

```
VAR
fglvar  :    BYTE;  (* compiler-assigned volatile data *)
END_VAR

VAR RETAIN
rglvar  :    BYTE;  (* compiler-assigned remanent data *)
END_VAR

VAR GLOBAL
fglvar  :    BYTE;  (* global volatile data *)
rglvar  :    BYTE;  (* global remanent data *)
END_VAR
```

Absolute global data

If you extend the declaration of a local variable by an absolute reference, you obtain an absolute local variable. With the specification of the reference the variable is linked to a concrete memory address by the user. To assign this variables the property global, they have to be entered in the area VAR GLOBAL - END_VAR of the resource's declaration editor.

```
VAR
fglabs    AT  %MB4 :      BYTE;   (* absolute volatile flags *)
END_VAR

VAR RETAIN
rglabs    AT  %RB6 :      BYTE;   (* absolute remanent flags *)
END_VAR

VAR GLOBAL
fglvar    AT  %MB4 :      BYTE;   (* global absolute variables *)
rglvar    AT  %RB6 :      BYTE;   (* global absolute remanent variables *)
END_VAR
```

Note: Compiler-assigned global and absolute global data never coincide.

Absolute Data

Basically, absolute data are, like the global data, assigned to the resource. They can be declared and used in the resource and in programs. Thereby, each declaration on an absolute reference is a reference on the same memory address, i.e. a local absolute variable with the reference %MB4 coincides with a global variable with reference %MB4. Thus, absolute variables can be used in the resource and the programs like global variables without being explicitly declared under VAR GLOBAL and VAR EXTERNAL.

```
RESOURCE R1
VAR
glabs     AT  %MB4 :      BYTE;   (* absolute volatile flags *)
END_VAR

VAR GLOBAL
glabs     AT  %MB4 :      BYTE;   (* global absolute volatile flags *)
END_VAR

PROGRAM P1
VAR
labs      AT  %MB4 :      BYTE;   (* local absolute volatile flags *)
END_VAR

VAR EXTERNAL
glabs     :          BYTE;   (* reference on global absolute volatile flags *)
END_VAR
```

In program P1 the variables "labs" and "glabs" are two references on the same memory address.

14.5 Start of the PLC

User data are initialized in the PLC at specified points of time. These points of time are designated as cold start and warm start. If a cold start is executed, it always contains all activities of the warm start.

Basically, all variables are initialized with the standard initial value. The standard initial value is zero for all data types.

Cold start

After downloading the files of a current resource in the PLC a cold start is executed under the following conditions:

1. A change of the current resource has occurred. As criteria solely the name of the current resource is used and not the name of the variant etc. So, to retrieve an archive with the same current resource name is no change of the current resource.
2. The PLC program was invalid before downloading (e.g. interruption of the download in the PLC renders the program invalid or if an transmission error occurs.)
3. The data in the safety directory "Downloaded Files" do not fit to the data in the PLC.

During a cold start all volatile and remanent data in the PLC are first initialized with the standard initial value and then with the single specified initial values.

Warm start

A warm start is executed:

1. after every download,
2. after switching on the PLC,
3. after each soft or hard reset of the PLC.

During a warm start all volatile data in the PLC are first initialized in the PLC with the standard initial value and then with the single specified initial values.

Additionally, all remanent data are initialized, that are used for the first time after the last download or if its name, data type or initial value has changed.

14.6 Initialization of the Data

Initialization of global data

Global data are assigned to the resource. They are treated like local data of the resource and initialized with the other data of the resource according to the rules for cold and warm start.

Initialization of absolute data

Absolute data are assigned to the resource. They are treated like local data of the resource and initialized with the other data of the resource according to the rules for cold and warm start.

Additionally, absolute data can be declared in a program and can be provided with single specified initial values. They are initialized during the initialization of the program with the single specified initial values (not once more with the standard initial value). Thereby, multiple assigned single initial values can superpose on the same absolute reference. The order of this initialization depends on the order of the activation of the program instances from the tasks.

Remanent absolute data are not initialized after deleting a declaration, as it could be also used otherwise.

15 Error Management

15.1 S#ErrorFlg

Fundamentals of the Error Management Concept

The execution of programs on the PLC can result in the calculation of variable values, for which the downstream functions and function blocks are not defined.

Example:

The calculation of the divisor of an integer division results in the value zero, a value which normally cannot be accepted.

Name	AT	TYPE	:=	Comment
VAR				
real1		REAL	3.0	
real2		REAL	0.0	
real3		REAL	2.0	
END_VAR				

Fig. 15-1: Declaration of the variables

Label	Operation	Operand	Comment
	LD	real1	
	DIV	real2	
	ST	real3	

Fig. 15-2: Implementation

The result of this division is not correct. For that reason, the calculation is not carried out, so that the initial value of real3 (3.0) remains the same. However, the error variables are set:

S#ErrorFlg= TRUE, S#ErrorTyp= -10038, S#ErrorNr= 8

Goal

To allow the user to localize the above mentioned errors and to react accordingly.

To achieve this, three error variables were automatically added in each function, each function block and in each program, in addition to the known function contents.

An execution of the user program without any interruptions is ensured in any case.

The program continues as expected when the error message is ignored.

S#ErrorFlg (BOOL) Standard: FALSE	S#ErrorNr(USINT) Standard: 0	S#ErrorTyp (INT) Standard: 0
S#ErrorFlg= 0, there was no error during the present execution (Standard),S#ErrorFlg= 1, there was at least one error which is specified in more detail	Detailed information on the error can be taken from a number > 0.	The PLC manufacturer reserved negative error types for standard and firmware functions and function blocks. Positive error types are available to the user for his own work.

Fig. 15-3: S#ErrorFlg, S#ErrorNr and S#ErrorTyp

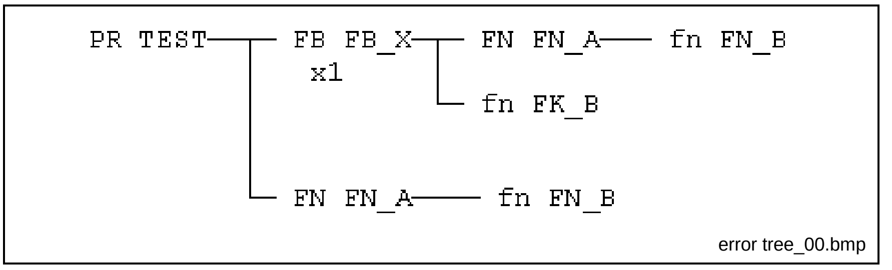
Occurred errors are handed upward to the respective unit that initialized the call up to the program.

The three variables are also declared in self-written functions, function blocks and programs and are available for access.

15.2 Error Management Sequence

After the first reload of a program, plus user FB / FN, all error variables of the individual program organization unit are set to their standard value 0 when the program is started.

The program should be structured as follows:



PR: Program
FB: User function block
fb: Standard function block
FN: User function
fn: Standard function

Fig. 15-4: Structure of the program "Test"

The following sequence is passed during the execution:

Name	Section	S#ErrorFlg	S#ErrorNr	S#ErrorTyp
PR TEST	(1)	0	0	0
FB FB_X/x1	(1)	0	0	0
FN FN_A	(1)	0	0	0
fn FN_B		0	0	0
FN FN_A	(2)	0	0	0
FB FB_X/x1	(2)	0	0	0
fn FN_B		0	0	0
FB FB_X/x1	(3)	0	0	0
PR TEST	(2)	0	0	0
FN FN_A	(1)	0	0	0
fn FN_B		0	0	0
FN FN_A	(2)	0	0	0
PR TEST	(3)	0	0	0

Fig. 15-5: Correct run of the program

If there are no errors PR TEST (cycle 3) is closed with assignment S#ErrorFlg= 0, S#ErrorNr= 0, S#ErrorTyp= 0.

Different variables changed during the calculation. The next PLC cycle follows.

The changed variables cause an error in the standard function fn FN_B. The function fn FN_B calculates the values and additionally changes the error variables.

The error message is then forwarded to the respective initiating file, on the left in the import tree, until it reaches the program itself.

The error message is not forwarded to the right in the import tree.

If you go on further to the right in the import tree, the three variables for the new POU are reset to zero.

There should not be any user reaction yet.

Name	Section	S#ErrorFlg	S#ErrorNr	S#ErrorTyp	Comment
PR TEST	(1)	0	0	0	Without errors
FB FB_X/x1	(1)	0	0	0	Without errors
FN FN_A	(1)	0	0	0	Without errors
fn FN_B		1	23	-1	Place of error
FN FN_A	(2)	1	23	-1	Error taken to the left
FB FB_X/x1	(2)	1	23	-1	Error taken to the left
fn FN_B		0	0	0	To the right without error
FB FB_X/x1	(3)	1	23	-1	Error remains stored
PR TEST	(2)	1	23	-1	Error taken to the left
FN FN_A	(1)	0	0	0	To the right without error
fn FN_B		0	0	0	To the right without error
FN FN_A	(2)	0	0	0	Back without error
PR TEST	(3)	1	23	-1	Error remains stored

Fig. 15-6: Error in fn FN_B without any reaction

The active error is taken over in the program to the next PLC cycle.

In the next cycle you go to the right with reset error variables.

- Elimination of the error causes results in the table below, "Error in fn FN_B does not exist any longer, no user reaction".
- If the error still exists, the table "Error in fn FN_B without reaction of the user" comes up again.

Name	Section	S#ErrorFlg	S#ErrorNr	S#ErrorTyp	Comment
PR TEST	(1)	1	23	-1	Error remains stored
FB FB_X/x1	(1)	0	0	0	To the right without error
FN FN_A	(1)	0	0	0	To the right without error
fn FN_B		0	0	0	Without errors again
FN FN_A	(2)	0	0	0	Back without error
FB FB_X/x1	(2)	0	0	0	Back without error
fn FN_B		0	0	0	To the right without error
FB FB_X/x1	(3)	0	0	0	Back without error
PR TEST	(2)	1	23	-1	Error remains stored
FN FN_A	(1)	0	0	0	To the right without error
fn FN_B		0	0	0	To the right without error
FN FN_A	(2)	0	0	0	Back without error
PR TEST	(3)	1	23	-1	Error remains stored

Fig. 15-7: Error in fn FN_B does not exist any longer, no user reaction

The error at program level has to be cleared actively by the user.

The user can also interfere, in the user function FN FN_A, (Cycle 2) at the earliest or later in another program organization unit.

He first sees that the S#ErrorFlg error bit is set and can then specifically evaluate S#ErrorNr/S#ErrorTyp accordingly.

After evaluation, it is sufficient to set S#ErrorFlg to 0, as by this way S#ErrorNr / S#ErrorTyp become invalid. S#ErrorNr, S#ErrorTyp do not have to be changed.

Name	Section	S#ErrorFlg	S#ErrorNr	S#ErrorTyp	Comment
PR TEST	(1)	0	0	0	
FB FB_X/x1	(1)	0	0	0	
FN FN_A	(1)	0	0	0	
fn FN_B		1	23	-1	Place of error
FN FN_A	(2a)	1	23	-1	Error detected and eliminated
FN FN_A	(2b)	0	23	-1	S#ErrorFlg reset
FB FB_X/x1	(2)	0	0	0	Error eliminated, no copy
fn FN_B		0	0	0	
FB FB_X/x1	(3)	0	0	0	
PR TEST	(2)	1	23	-1	
FN FN_A	(1)	0	0	0	
fn FN_B		0	0	0	
FN FN_A	(2)	0	0	0	
PR TEST	(3)	1	23	-1	

Fig. 15-8: Error evaluation in FN FN_A, (section 2a), reset S#ErrorFlg

15.3 Error Management in Case of Multiple Errors

Theoretically, it is possible to think of a second error occurring in a different program section and having an assignment which is different from or equal to S#ErrorNr and S#ErrorTyp, before the first error has been detected and eliminated. This error is also moved to the left in the import tree.

The following applies:

S#ErrorNr and S#ErrorTyp are not overwritten before S#ErrorFlg is reset.

The information on the second error thus is in wait position.

15.4 Error Management in User Files

As the three variables are automatically declared for newly generated user files and the error mechanism for standard and firmware files is known, the user can use this mechanism also for his concerns in own user files.

Note: To avoid mistakes, only error types (S#ErrorTyp) with positive number may be used.

15.5 S#ErrorTyp

Übersicht Overview of possible errors and their initiators

If TRUE is applied to S#ErrorFlg, an error occurred during the execution of an operation, a function or a function block.

S#ErrorTyp indicates the initiator:

- -1 ... -350 Errors in Functions and Function Blocks
- starting with -10000. Errors in Operations and IL Instructions
- starting with -11000. Sequential Function Chart Errors (SFC)

The error itself is characterized in more detail by the error number S#ErrorNr.

15.6 Errors in Functions and Function Blocks

Explanation:

S#ErrorTyp, indicates the fb/fn that initiated the error.

fn – firmware function

*fn – standard function

fb – firmware function block

*fb – standard function block

S#ErrorTyp	Type	Name	Comment
-1	fn	M_FKT	Polling of M help functions with indication of the help function number
-2	fn	M_FKT_Q	Acknowledgement of M help functions with indication of the help function number
-3	fn	S_FKT	Polling of S help functions with indication of the help function number
-4	fn	S_FKT_Q	Acknowledgement of S help functions with indication of the help function number
-5	fn	T_FKT	Polling of T help functions with indication of the help function number
-6	fn	T_FKT_Q	Acknowledgement of T help functions with indication of the help function number
-7	fn	Q_FKT	Polling of Q help functions with indication of the help function number
-8	fn	Q_FKT_Q	Acknowledgement of Q help functions with indication of the help function number
-9	fn	EVENT	Polling of events
-10	fn	EV_ST	Value transmission to events
-11	fn	EV_SET	Conditional setting of events
-12	fn	EV_RES	Conditional resetting of events
-13	fn	MSG_WR	Diagnosis output, message number directly defined
-14	fn	MSG_RD	Read-in of CNC message numbers
-15	fn	MRF	Request for Magazine reference run
-16	fn	MRF_Q	Acknowledgement of Magazine reference run
-17	fn	MMV	Request for Magazine on new position
-18	fn	MMV_Q	Acknowledgement of Magazine on new position
-19	fn	TCH	Request for General tool change

S#ErrorTyp	Type	Name	Comment
-20	fn	TCH_Q	Acknowledgement of General tool change
-21	fn	TMS	Request for Tool change magazine / spindle
-22	fn	TMS_Q	Acknowledgement of Tool change magazine / spindle
-23	fn	TSM	Request for Tool change spindle / magazine
-24	fn	TSM_Q	Acknowledgement of Tool change spindle / magazine
-25	fn	XMS	Initialization of Tool transfer magazine / spindle
-26	fn	XMS_PA	Tool transfer magazine / spindle allowed
-27	fn	XMS_NA	Tool transfer magazine / spindle not allowed
-28	fn	XMS_Q	Acknowledgement of Tool transfer magazine / spindle
-29	fn	XSM	Initialization of Tool transfer spindle / magazine
-30	fn	XSM_PA	Tool transfer spindle / magazine allowed
-31	fn	XSM_NA	Tool transfer spindle / magazine not allowed
-32	fn	XSM_Q	Acknowledgement of Tool transfer spindle / magazine
-33	fn	XMG	Initialization of Tool transfer magazine / gripper
-34	fn	XMG_PA	Tool transfer magazine / gripper allowed
-35	fn	XMG_NA	Tool transfer magazine / gripper not allowed
-36	fn	XMG_Q	Acknowledgement of Tool transfer magazine / gripper
-37	fn	XSG	Initialization of Tool transfer spindle / gripper
-38	fn	XSG_PA	Tool transfer spindle / gripper allowed
-39	fn	XSG_NA	Tool transfer spindle / gripper not allowed
-40	fn	XSG_Q	Acknowledgement of Tool transfer spindle / gripper
-41	fn	XGS	Initialization of Tool transfer gripper / spindle
-42	fn	XGS_PA	Tool transfer gripper / spindle allowed
-43	fn	XGS_NA	Tool transfer gripper / spindle not allowed
-44	fn	XGS_Q	Acknowledgement of Tool transfer gripper / spindle
-45	fn	XGM	Initialization of Tool transfer gripper / magazine
-46	fn	XGM_PA	Tool transfer gripper / magazine allowed
-47	fn	XGM_NA	Tool transfer gripper / magazine not allowed
-48	fn	XGM_Q	Acknowledgement of Tool transfer gripper / magazine
-49	*fn	GRAY_TO_BYTE	Type conversion of graycode -> BYTE
-50	*fn	BYTE_TO_GRAY	Type conversion of BYTE -> graycode
-51	*fn	BYTE_BCD_TO_INT	Type conversion of BCD code, byte, 2 digits-> INTEGER
-52	*fn	WORD_BCD_TO_INT	Type conversion of BCD code, word, 4 digits -> INTEGER
-53	*fn	BYTE_TO_INT	Type conversion of BYTE -> INTEGER
-54	*fn	WORD_TO_INT	Type conversion of WORD -> INTEGER
-55	*fn	INT_TO_BYTE	Type conversion of integer number -> BYTE
-56	*fn	INT_TO_WORD	Type conversion of integer number -> word
-57	*fn	INT_TO_BCD_WORD	Type conversion of integer number -> 4 digit BCD-coded word
-58	*fn	USINT_TO_INT	Type conversion of UNSIGNED SHORT INTEGER -> INTEGER
-59	*fn	INT_TO_USINT	Type conversion of INTEGER -> UNSIGNED SHORT INTEGER
-60	*fn	USINT_TO_BYTE	Type conversion of UNSIGNED SHORT INTEGER -> BYTE
-61	*fn	BYTE_TO_USINT	Type conversion of BYTE -> UNSIGNED SHORT INTEGER

S#ErrorTyp	Type	Name	Comment
-62	*fn	CONCAT_BYTE	Attachment of low byte to high byte
-63	*fn	CONCAT_WORD	Attachment of low word to high word
-64	*fn	HIGH_BYTE	Taking the high byte from the word
-65	*fn	LOW_BYTE	Taking the low byte from the word
-66	*fn	HIGH_WORD	Taking the high word from DWORD
-67	*fn	LOW_WORD	Taking the low word from DWORD
-68	*fn	SIGN_INT	Sign of an integer number
-69	*fn	ABS_INT	Absolute value of an integer number
-70	*fn	SHL_BYTE	Move BYTE by n digits to the left
-71	*fn	SHL_WORD	Move WORD by n digits to the left
-72	*fn	SHR_BYTE	Move BYTE by n digits to the right
-73	*fn	SHR_WORD	Move WORD by n digits to the right
-74	*fn	ROL_BYTE	Rotate BYTE by n digits to the left
-75	*fn	ROL_WORD	Rotate WORD by n digits to the left
-76	*fn	ROR_BYTE	Rotate BYTE by n digits to the right
-77	*fn	ROR_WORD	Rotate WORD by n digits to the right
-78	*fb	SR	FLIP_FLOP, dominating setting
-79	*fb	RS	FLIP_FLOP, dominating resetting
-80	*fb	R_TRIG	Identification of a rising edge
-81	*fb	F_TRIG	Identification of a falling edge
-82	*fb	CTUD_USINT_INDR	Up-down counter, value range UNSIGNED SHORT INTEGER
-83	*fb	CTUD_UINT_INDR	Up-down counter, value range UNSIGNED INTEGER
-84	*fb	CTUD_INT_INDR	Up-down counter, value range INTEGER
-85	*fb	TP	Timer pulse
-86	*fb	TON	On-delay timer function block
-87	*fb	TOFF	Off-delay timer function block
-88	fb	SC_WRITE	Function no longer supported
-89	fb	SC_READ	Function no longer supported
-90	fb	VAR_WR	Function no longer supported
-91	fb	VAR_RD	Function no longer supported
-92	fb	SEL_MEM	Selection of the NC program memory
-93	fb	ACT_MEM	Polling of the active NC program memory
-94	fn	XMS_CA	Cancel tool transfer from magazine to spindle
-95	fn	XSM_CA	Cancel tool transfer from spindle to magazine
-96	fn	XMG_CA	Cancel tool transfer from magazine to gripper
-97	fn	XSG_CA	Cancel tool transfer from spindle to gripper
-98	fn	XGS_CA	Cancel tool transfer from gripper to spindle
-99	fn	XGM_CA	Cancel tool transfer from gripper to magazine
-100	fn	MHP	Function no longer supported
-101	fn	MHP_Q	Function no longer supported
-102	fn	GRP	Function no longer supported
-103	fn	GRP_Q	Function no longer supported

S#ErrorTyp	Type	Name	Comment
-104	fn	REL	Function no longer supported
-105	fn	REL_Q	Function no longer supported
-106	fb	OPEN_COM	Initialization of a general data channel
-107	fb	CLOS_COM	Close data transmission of a general data channel
-108	fb	OPEN_SOT	Function no longer supported
-109	fb	CLOS_SOT	Function no longer supported
-110	fb	WR_BYTE	Write a byte to the transmit buffer
-111	fb	RD_BYTE	Read a byte to general transmission channel
-112	fb	CTRL_COM	Request status of a serial interface
-113	fn	MAG_ACT	Polling of selected magazine axis for combined spindle / turret axis
-114	fn	MAG_Q	Acknowledgement of selected magazine axis for combined spindle / revolving axis
-115	fn	SPDL_ACT	Polling of selected spindle for combined spindle / turret axis
-116	fn	SPDL_Q	Acknowledgement of selected spindle for combined spindle / turret axis
-117	fn	M_ALL	Polling of M help functions without indication of the help function number
-118	fn	M_ALL_Q	Acknowledgement of M help functions without indication of the help function number
-119	fn	S_ALL	Polling of S help functions without indication of the help function number
-120	fn	S_ALL_Q	Acknowledgement of S help functions without indication of the help function number
-121	fn	T_ALL	Polling of T help functions without indication of the help function number
-122	fn	T_ALL_Q	Acknowledgement of T help functions without indication of the help function number
-123	fn	Q_ALL	Polling of Q help functions without indication of the help function number
-124	fn	Q_ALL_Q	Acknowledgement of Q help functions without indication of the help function number
-125	fb	USERBOF	Function no longer supported
-126	fn	M_NR	Reading of the M help function number
-127	fn	S_NR	Reading of the S help function number
-128	fn	Q_NR	Reading of the Q help function number
-129	fn	XFER_CHK	Deactivate check of the tool transfer
-130	fn	MSG_WR_N	Message output with additional information as number
-131	fn	MSG_WR_A	Message output with additional information as axis identification
-132	fb	AXD_WR	Writing of demand data
-133	fb	AXD_RD	Reading of demand data
-134	fn	SPMOD	Request for preselection of spindle mode for rotary-axis-capable main spindle
-135	fn	SPMOD_Q	Acknowledgement of preselection of spindle mode for rotary-axis-capable main spindle
-136	fn	ROTMOD	Request for preselection of rotary axis mode for rotary-axis-capable main spindle
-137	fn	ROTMOD_Q	Acknowledgement of preselection of rotary axis mode for rotary-axis-capable main spindle
-138	*fn	CHAR_TO_BYTE	Type conversion of CHAR -> BYTE
-139	*fn	BYTE_TO_CHAR	Type conversion of BYTE -> CHAR

S#ErrorTyp	Type	Name	Comment
-140	*fn	INT_TO_STRING	Type conversion of INTEGER -> STRING
-141	*fn	STRING_TO_INT	Type conversion of STRING -> INTEGER
-142	*fn	LEN	Length of a STRING
-143	*fn	LEFT	Leftmost L_ character of a STRING
-144	*fn	RIGHT	Rightmost L_ character of a STRING
-145	*fn	MID	L_ character of a STRING, from the p^{th} character
-146	*fn	CONCAT_S	Combination of two STRINGS
-147	*fn	INSERT	Insert of a STRING after the L^{th} character
-148	*fn	DELETE	Delete L_ character of a STRING from p^{th} character
-149	*fn	REPLACE	Replace L_ character of a STRING from p^{th} character
-150	*fn	FIND	Find character string IN2_ in IN1_
-151	fb	GUI_SK	Function no longer supported
-152	*fn	DINT_TO_DWORD	Type conversion of DOUBLE INTEGER -> DOUBLE WORD
-153	*fn	DWORD_TO_DINT	Type conversion of DOUBLE WORD -> DOUBLE INTEGER
-154	*fn	DINT_TO_INT	Type conversion of DOUBLE INTEGER -> INTEGER
-155	*fn	INT_TO_DINT	Type conversion of INTEGER -> DOUBLE INTEGER
-156	*fn	DINT_TO_TIME	Type conversion of DOUBLE INTEGER -> Time
-157	*fn	TIME_TO_DINT	Type conversion of Time -> DOUBLE INTEGER
-158	fn	HNDWHEEL	Transmission of handwheel position
-159	*fn	SHL_DWORD	Move DOUBLE WORD by n digits to the left
-160	*fn	SHR_DWORD	Move DOUBLE WORD by n digits to the right
-161	*fn	ROL_DWORD	Rotate DOUBLE WORD by n digits to the left
-162	*fn	ROR_DWORD	Rotate DOUBLE WORD by n digits to the right
-163	fb	RLVAR_WR	Function no longer supported
-164	fb	RLVAR_RD	Function no longer supported
-165	*fn	DINT_TO_REAL	Type conversion of DOUBLE INTEGER -> REAL
-166	*fn	REAL_TO_DINT	Type conversion of REAL -> DOUBLE INTEGER
-167	*fn	STRING_TO_REAL	Type conversion of STRING -> REAL
-168	*fn	REAL_TO_STRING	Type conversion of REAL -> STRING
-169	fb	TLD_WR	Writing to tool data
-170	fb	TLD_RD	Reading of tool data
-171	*fn	DINT_TO_UDINT	Type conversion of DOUBLE INTEGER -> UNSIGNED DOUBLE INTEGER
-172	*fn	UDINT_TO_DINT	Type conversion of UNSIGNED DOUBLE INTEGER -> DOUBLE INTEGER
-173	fb	DATE_RD	Reading of the date
-174	fb	TOD_RD	Reading of the time
-175	fb	OTD_WR	Writing to zero point data
-176	fb	OTD_RD	Reading of zero point data
-177	fb	MTD_WR	Writing to machine data
-178	fb	MTD_RD	Reading of machine data
-179	fb	NETIO_RD	Reading of realtime bits
-180	fn	T_NR	Reading of T help function number for process (PROC)

S#ErrorTyp	Type	Name	Comment
-181	fn	E_FKT	Reading of E help functions (EDGE) for process (PROC)
-182	fn	E_FKT_Q	Acknowledgement of E help functions (EDGE) for process (PROC)
-183	fn	E_ALL	Polling of any E help function for process (PROC)
-184	fn	E_ALL_Q	Acknowledgement of any E help function for process (PROC)
-185	fn	E_NR	Reading of E help function number for process (PROC)
-186	fb	TLBD_WR	Writing of basic tool data
-187	fb	TLED_WR	Writing of tool tip data
-188	fb	TL_ENABLE	Enable of tool data
-189	fb	TLBD_RD	Reading of basic tool data
-190	fb	TLED_RD	Reading of tool tip data
-191	fb	TL_RESET	Reset tool
-192	fb	TL_DELETE	Delete tool
-193	fb	TL_MOVE	Move tool
-195	*fb	BOOL_BYTE	Conversion of 8-bit -> byte
-196	*fb	BYTE_BOOL	Conversion of Byte -> 8-bit
-197	*fb	BOOL_WORD	Conversion of 16-bit -> word
-198	*fb	WORD_BOOL	Conversion of word -> 16-bit
-199	*fb	BOOL_DW	Conversion of 32-bit -> doubleword
-200	*fb	DW_BOOL	Conversion of doubleword -> 32-bit
-201	fb	FLASH	Pulse generator
-202	fb	TOGGLE	Toggling of a bit
-203	fb	RD_STR	Reading a STRING via the serial interface
-204	fb	WR_STR	Writing a STRING to the transmitter buffer
-205	fn	TIME_DAY	Conversion of TIME_ to the numerical value of day
-206	fn	TIME_HOUR	Conversion of TIME_ to the numerical value of hour
-207	fn	TIME_MIN	Conversion of TIME_ to the numerical value of minute
-208	fn	TIME_SEC	Conversion of TIME_ to the numerical value of second
-209	fn	TIME_MS	Conversion of TIME_ to the numerical value of milliseconds
-210	fn	MAKETIME	Conversion of the FN inputs day ’D’, hour ‘H’, minute ‘M’, second ‘S’, and millisecond ‘MS’ to a time value
-211	fn	COL_CTRL	Switch on and off the one-dimensional approach monitoring
-212	fn	COL_CTRL_S	Request the status of the approach monitoring
-213	fb	TLG_RD	Read the group status information of the PLC user program's tool group
-214	fb	TLG_WR	Modification of the group status data by the PLC user
-215	fb	IB_GROFF	Function no longer supported
-216	fb	MODBUS	Function no longer supported
-217	fn	BT_START	Starts BT Bus
-218	fn	BT_STOP	Stops BT Bus
-219	fn	BT_STATUS	Status information on BT Bus process data exchange
-220			
-221	fb	DCD_RD	Reading of D-corrections
-222	fb	DCD_WR	Writing of D-corrections

S#ErrorTyp	Type	Name	Comment
-223	fb	NCVAR_RD	Reading of NC variables
-224	fb	NCVAR_WR	Writing of NC variables
-225	fb	GUI_SK16	Enable of (16) machine function keys GUI / menu9
-226	fn	REV_SYNC	Synchronous swiveling of the revolver in the NC set
-227	fb	CLR_COM	Clearing the receiver and transmitter buffers of a serial interface
-228	*fn	SINT_TO_INT	Conversion of SINT number into INT number
-229	*fn	INT_TO_SINT	Conversion of INT number into SINT number
-230	*fn	SINT_TO_BYTE	Conversion of SINT number to BYTE
-231	*fn	BYTE_TO_SINT	Conversion of BYTE to SINT number
-232	*fn	UINT_TO_INT	Type conversion of UNSIGNED INTEGER -> INTEGER
-233	*fn	INT_TO_UINT	Type conversion of INTEGER UNSIGNED -> INTEGER
-234	*fn	UINT_TO_WORD	Type conversion of UNSIGNED INTEGER -> WORD
-235	*fn	WORD_TO_UINT	Type conversion of WORD -> UNSIGNED INTEGER
-236	*fn	REAL_TO_DWORD	Type conversion of REAL -> DWORD
-237	*fn	DWORD_TO_REAL	Type conversion of DWORD -> REAL
-238	Fb	BTXX	Communication between PLC and HMI operating panels of BTV04, BTV05 and BTC06 via a serial interface
-239	fb	DPM_SLDIAG	Single diagnosis of a PROFIBUS slave
-240	fn	VLT_MEAS	In connection with the analog module RMC12.2-2E-1A, it is possible to measure voltages of up to ±10 V.
-241	fn	SAVE_IO	Safety function for projected axis
-242	fb	DPM_STATE	Status information on the PROFIBUS master:
-243	fn	DPM_STOP	Stopping the bus communication
-244	fn	DPM_START	Starting the bus communication
-245	fn	DPM_EXCHG	Status information on PROFIBUS process data exchange
-246	fn	AMP_MEAS	In connection with the analog module RMC12.2-2E-1A, it is possible to measure currents of up to ±20 mA.
-247	fn	RES_MEAS	In connection with the analog module RMC12.2-2E-1A, it is possible to measure resistances of up to 2000 Ω.
-248	fn	TMP1MEAS	In connection with the analog module RMC12.2-2E-1A, it is possible to measure temperature ranging from -100 °C to +850 °C.
-249	fn	AN_OUT	In connection with the analog module RMC12.2-2E-1A, it is possible to provide voltages of up to ±10 V and currents of up to +20 mA at the analog output.
-250	fn	DIAG_WORD	Diagnosis functions (hidden to the user)
-251	fn	DIAG_UINT	Diagnosis functions (hidden to the user)
-252	fn	DIAG_INT	Diagnosis functions (hidden to the user)
-253			
-254	Fn	BTXX_2	Communication block for manual device BTC06 with 64 IO.
-255	fn	RT_DATA	Function block for quick access to NC signal values
-256	fn	DIAG_BYTE	Diagnosis functions (hidden to the user)
-257	fn	DIAG_CHAR	Diagnosis functions (hidden to the user)
-258	fn	DIAG_SINT	Diagnosis functions (hidden to the user)
-259	fn	DIAG_USINT	Diagnosis functions (hidden to the user)
-260	fn	DIAG_BOOL32	Diagnosis functions (hidden to the user)

S#ErrorTyp	Type	Name	Comment
-261	fn	DIAG_DWORD	Diagnosis functions (hidden to the user)
-262	fn	DIAG_BOOL0	Diagnosis functions (hidden to the user)
-263	fn	DIAG_BOOL4	Diagnosis functions (hidden to the user)
-264	fn	DIAG_BOOL8	Diagnosis functions (hidden to the user)
-265	fn	DIAG_BOOL16	Diagnosis functions (hidden to the user)
-266	fn	DIAG_DINT	Diagnosis functions (hidden to the user)
-267	fn	DIAG_UDINT	Diagnosis functions (hidden to the user)
-268	fn	DIAG_REAL	Diagnosis functions (hidden to the user)
-269	fn	DIAG_TIME	Diagnosis functions (hidden to the user)
-270	fn	SQRT_REAL	Root
-271	fn	LN_REAL	Natural logarithm LN
-272	fn	LOG_REAL	Common logarithm LOG
-273	fn	EXP_REAL	Exponential function
-274	fn	SIN_REAL	Sinusoidal function
-275	fn	COS_REAL	Cosinoidal function
-276	fn	TAN_REAL	Tangential function
-277	fn	ASIN_REAL	Arc sinusoidal function
-278	fn	ACOS_REAL	Arc cosinoidal function
-279	fn	ATAN_REAL	Arc tangential function
-280	fb	MC_INITIALIZATION	Initialization of the DP-RAM interface MC-PLC
-281	fb	MC_CHANGE_ PHASE	Writing of the SERCOS communication phase (phase switchover)
-282	fb	MC_RD_PARA METER	Reading of an MC single parameter
-283	fb	MC_WR_PARA METER	Writing of an MC single parameter
-284	fb	MC_WR_LISTDATA	Writing of an MC list parameter
-285	fn	MC_DIAGNOSIS	Reading of an MC-SIS diagnosis
-286	fb	MC_RD_LISTDATA	Reading of an MC list parameter
-287	fb	MC_RD_DATASTA TUS	Reading of data state of an MC parameter
-288	fb	MC_ABORT_TRANS MISSION	Abortion of an MC parameter transmission
-289	fb	MC_RW_PTR_TLG	MC communication block
-290	fb	MC_RD_PHASE	Reading of SERCOS communication phase
-291	fb	MC_RD_ATTRIBUTE	Reading of attribute of an MC parameter
-292	fb	MC_RD_NAME	Reading of an MC parameter name
-293	fb	MC_RD_UNIT	Reading of an MC parameter unit
-294	fb	MC_RD_MIN_VALUE	Reading of minimum value of an MC parameter
-295	fb	MC_RD_MAX_VALUE	Reading of maximum value of an MC parameter
-296	fb	MC_RD_ELEMENT	Reading of an MC parameter element
-297	fb	MC_WR_ELEMENT	Writing of an MC parameter element
-298	fn	MC_CONCAT_TO_ IDENT_NO	Creating an MC parameter identification number

S#ErrorTyp	Type	Name	Comment
-299	fn	MC_CONVERT_TO_ IDENT_NO	Converting of an MC parameter identification number
-300	fb	MC_TYP01_CAM_ TABLE	Calculation of a cam (cross cutter) for short formats (Typ 01 – consistency up to the speed)
-301	fb	MC_TYP02_CAM_ TABLE	Calculation of a cam (cross cutter) for short formats (Typ 02 – consistency up to the acceleration)
-302	fb	MC_TYP03_CAM_ TABLE	Calculation of a cam (cross cutter) for long formats (Typ 03 – consistency up to the acceleration)
-303	fb	CALC_LINEAR_Y	calculates linearized for a default X value of a predetermined X-Y value table the respective Y value.
-304	fb	PID_CONTROL	Depending on the wiring of the input variables FB provides "P", "PI", "PD", "I" and "PID" functionalities.
-305	fb	AVERAGE_REAL	calculates the floating average value form maximum 64 REAL values.
-306	fb	AVERAGE_DINT	calculates the floating average value from maximum 1023 DINT values.
-307	fb	MC_RD_ARRAY	Reading of the operating data of a list parameter (4096 bytes, array)
-308	fb	MC_WR_ARRAY	Writing of the operating data of a list parameter (4096 bytes, array)
-309	fb	MC_RW_ARRAY_ TLG	Transmission of any telegram (263 bytes, array) as e.g. subsequently added SIS services or other transmission protocols
-310	fb	PT2_FILTER	serves for low pass filtering of a signal.
-311	fb	MC_TYP04_CAM_TA BLE	Calculation of a cam for pilgrim step mode (TYP04)
-312	fb	MC_TYP05_CAM_TA BLE	Calculation of a cam for a feed movement (TYP05)
-320	fb	CTUD_INT	UP / DOWN Counter INT corresponding IEC
-321	fb	CTUD_UINT	UP / DOWN Counter UINT corresponding IEC
-322	fb	CTUD_USINT	UP / DOWN Counter USINT corresponding IEC
-323	fn	NC_ENABLE	Synchronization of AXD and NC Initialization
-324	fb	PCP_INITIATE	Make a connection to a PCP-Slave
-325	fb	PCP_ABORT	Open a connection
-326	fb	PCP_READ	Read Object values
-327	fb	PCP_WRITE	Modify device parameters
-328	fb	PCP_IDENTIFY	Reading "Type designation plate"
-329	fb	PCP_GET_OD	Read several object discriptions
-330	fn	IB_STATE	Determines the status of the 1st INTERBUS
-331	fn	ASIM_START	ASI-Bus, starts IO data exchange
-332	fn	ASIM_STOP	ASI-Bus, stopps IO data exchange
-333	fn	ASIM_STATE_CH1	ASI-Bus, diagnosis of channel 1
-334	fn	ASIM_RESET	ASI-Bus reset
-335	fn	ASIM_STATE_CH2	ASI-Bus, diagnosis of channel 2
-336	fb	ASIM_SLDIAG	ASI-Bus, diagnosis of slaves
-337	fn	IB_STATE2	Determines the status of the 2nd INTERBUS

Fig. 15-9: Errors in functions and function blocks

15.7 Errors in Operations and IL Instructions

Explanation:

S#ErrorTyp indicates the operation or IL instruction which was the error initiator.

S#ErrorTyp	Comment
	Error in addition: ADD / ADD(
-10000	USINT
-10001	UINT
-10002	UDINT
-10003	ULINT
-10004	SINT
-10005	INT
-10006	DINT
-10007	LINT
-10008	REAL
-10009	LREAL
	Error in subtraction SUB / SUB(
-10010	USINT
-10011	UINT
-10012	UDINT
-10013	ULINT
-10014	SINT
-10015	INT
-10016	DINT
-10017	LINT
-10018	REAL
-10019	LREAL
	Error in multiplication MUL / MUL(
-10020	USINT
-10021	UINT
-10022	UDINT
-10023	ULINT
-10024	SINT
-10025	INT
-10026	DINT
-10027	LINT
-10028	REAL
-10029	LREAL

S#ErrorTyp	Comment
	Error in division DIV / DIV(
-10030	USINT
-10031	UINT
-10032	UDINT
-10033	ULINT
-10034	SINT
-10035	INT
-10036	DINT
-10037	LINT
-10038	REAL
-10039	LREAL
	Error in modulo division MOD / MOD(
-10040	USINT
-10041	UINT
-10042	UDINT
-10043	ULINT
-10044	SINT
-10045	INT
-10046	DINT
-10047	LINT
	Subscripting beyond type limit
-10050	USINT/SINT subscript
-10051	UINT/INT subscript
-10052	UDINT/DINT subscript
-10053	ULINT/LINT subscript
	Comparison not executable, type REAL
-10060	GT, greater than
-10061	GE, greater than or equal to
-10062	EQ, equal
-10063	LE, less than or equal to
-10064	LT, less than
-10065	NE, not equal to
	Pointer error
-10070	Pointer error, access beyond original data
-10071	Pointer error, attempt to write to an input variable
-10072	Pointer error, access to NIL pointer
	Time error
-10080	Addition (ADD), greater than 99d10h5m34s590ms
-10081	Subtraction (SUB), less than 0ms

Fig. 15-10: Errors in operation and IL instructions

15.8 Errors with REAL Operations in Borderline Cases

Explanation:

NaN - Not a Number	Result of a non-executable operation
oo - Infinite	Result of a range overflow
Number	Any number except zero, oo and NaN

The second lines indicates S#ErrorTyp/S#ErrorNr or no error.

ADD	**Number**	**Zero**	**oo**	**- oo**	**NaN**
Number +	Number	Number	invalid	invalid	invalid
	No error	No error	-10008/2	-10008/3	-10008/8
Zero +	Number	Zero	invalid	invalid	invalid
	No error	No error	-10008/2	-10008/3	-10008/8
oo +	invalid	invalid	invalid	invalid	invalid
	-10008/2	-10008/2	-10008/2	-10008/8	-10008/8
NaN +	invalid	invalid	invalid	invalid	invalid
	-10008/8	-10008/8	-10008/8	-10008/8	-10008/8
SUB	**Number**	**Zero**	**oo**	**- oo**	**NaN**
Number -	Number	Number	invalid	invalid	invalid
	No error	No error	-10018/3	-10018/2	-10018/8
Zero -	Number	Zero	invalid	invalid	invalid
	No error	No error	-10018/3	-10018/2	-10018/8
oo -	invalid	invalid	invalid	invalid	invalid
	-10018/2	-10018/2	-10018/8	-10018/2	-10018/8
NaN -	invalid	invalid	invalid	invalid	invalid
	-10018/8	-10018/8	-10018/8	-10018/8	-10018/8
MUL	**Number**	**Zero**	**oo**	**- oo**	**NaN**
Number *	Number	Number	invalid	invalid	invalid
	No error	No error	-10028/2	-10028/3	-10028/8
Zero *	Number	Zero	invalid	invalid	invalid
	No error	No error	-10028/8	-10008/8	-10028/8
oo *	invalid	invalid	invalid	invalid	invalid
	-10028/2	-10028/2	-10028/2	-10028/3	-10028/8
NaN *	invalid	invalid	invalid	invalid	invalid
	-10028/8	-10028/8	-10028/8	-10028/8	-10028/8

DIV	Number	Zero	oo	- oo	NaN
Number /	Number	invalid	invalid	invalid	invalid
	No error	-10038/8	-10038/8	-10038/8	-10038/8
Zero /	Zero	invalid	invalid	- Zero	invalid
	No error	-10038/8	-10038/8r	-10038/8	-10038/8
oo /	invalid	invalid	invalid	invalid	invalid
	-10038/2	-10038/2	-10038/8	-10038/8	-10038/8
NaN /	invalid	invalid	invalid	invalid	invalid
	-10038/8	-10038/8	-10038/8	-10038/8	-10038/8

Fig. 15-11: Errors with REAL-Operations in Borderline Cases

15.9 Sequential Function Chart Errors (SFC)

Errors, which can occur in connection with the execution of a sequence are grouped by errors which can be influenced by the user, such as repeated activation of an action, and errors which cannot be influenced by the user (general processing errors). The following table is an overview of both error groups:

S#ErrorTyp	Cause	Error description
-11000	General processing error	Cannot be influenced by the user
-11001	Multiple active connection to an action qualifier input "L","D","SD","DS","SL"	Further active connection(s) to action qualifier input "L" ignored
-11002	-"-	Further active connection(s) to qualifier input "D" ignored
-11003	-"-	Further active connection(s) to action qualifier input "SD" ignored
-11004	-"-	Further active connection(s) to action qualifier input "DS" ignored
-11005	-"-	Further active connection(s) to action qualifier input "SL" ignored
-11006	Several new active connections to different action qualifier inputs while input "L","D","SD","DS"or "SL" is already active	Active connection(s) to action qualifier input "D","SD","DS","SL" ignored because input "L" was active before
-11007	-"-	Active connection(s) to action qualifier input "L","SD","DS","SL" ignored because input "L" was active before
-11008	-"-	Active connection(s) to action qualifier input "D","L","DS","SL" ignored because input "SD" was active before
-11009	-"-	Active connection(s) to action qualifier input "D","SD","L","SL" ignored because input "DS" was active before
-11010	-"-	Active connection(s) to action qualifier input "D","SD","DS" ignored because input "SL" was active before
-11011	Several active connections to different action qualifier inputs	Input "L" got the priority of several active connections to action qualifier inputs ("D", "SD","DS","SL" were ignored)
-11012	-"-	Input "D" got the priority of several active connections to qualifier inputs ("L", "SD","DS","SL" were ignored)
-11013	-"-	Input "SD" got the priority of several active connections to action qualifier inputs ("D", "L","DS","SL" were ignored)
-11014	-"-	Input "L" got the priority of several active connections to action qualifier inputs ("DS", "SD","L","SL" were ignored)

S#ErrorTyp	Cause	Error description
-11015	Active connections to a stored and time-relayed action qualifier input while the action was already active	Active connection to an action qualifier input "SL" ignored because the action was already activated with time-stored delay ("SD")
-11016	-"-	Active connection to an action qualifier input "SD" ignored because the action was already activated with time-stored delay ("SL")
-11017	A sequence calls itself, directly or indirectly (recursion)	Sequence processing canceled
-11018 to		
-11050	General processing error	Cannot be influenced by the user
-11101	No active step contained in the sequence	Sequence structure cannot be restarted automatically
-11102	Not enough active steps contained in the sequence	Sequence structure cannot be restarted automatically
-11103	Too many active steps contained in the sequence	Sequence structure cannot be restarted automatically

Fig. 15-12: Sequential function chart errors (SFC)

15.10 S#ErrorNr

0 - No error

1 - Invalid input parameter

The operation, function, function block is not executed. Feed back of unreasonable results possible

2 - Range exceeded

Invalid result.

3 - Range fallen below

Invalid result.

4 - Conversion error

The input parameter cannot be converted correctly. Conversion is done with an internally modified input parameter. Feed back of unreasonable results possible

5 - Division by zero

Invalid result.

6 - Internal transmission error

An error occurred during an internal data request from / to the CNC.

7 - Subscript error, range exceeded

The operation is not executed.

8 - Operation not defined

9 - Pointer error, invalid address

10 - Error during activation of action blocks

233 - General SYNAX error

234 - Memory not available

235 - Addressed PC104 module not available

Unable to serve this bus connection, the addressed PC104 module is not available.

236 - Process data channel overflow

More than eight TLD, OTD, MTD, NC_VAR, TLED, TLBD, DCD programmed in parallel.

237 - Too many accesses to variables

More than 100 NC variables have been programmed.

238 - Interface not open

A serial interface, which is not yet open, is accessed by WR_STRING or RD_STRING.

239 - STRING overflow processing

When using STRING functions, a STRING with more than 255 characters occurred.

240 - Invalid input parameter DEVICE

A negative device number or an excessive DEVICE number was transmitted during parameterization of the serial interface.

241 - Invalid input parameter SERNR

A negative number or an excessive number for the serial interface SERNR was transmitted during parameterization of the serial interface.

242 - Invalid input parameter BAUD

A negative number or an excessive number for the baud rate BAUD was transmitted during parameterization of the serial interface.

243 - Invalid input parameter DATA

A negative number or an excessive number for the number of data bits DATA was transmitted during parameterization of the serial interface.

244 - Invalid input parameter PARITY

A negative number or an excessive number for the evaluation of the PARITY bit was transmitted during parameterization of the serial interface.

245 - Invalid input parameter STOP

A negative number or an excessive number for the number of STOP bits was transmitted during parameterization of the serial interface.

246 - Invalid input parameter PROTOKOL

A negative number or an excessive number for the type of serial interface PROTOKOL was transmitted during parameterization of the serial interface.

247 - Invalid input parameter HANDSH

A negative number or an excessive number for the type of handshake HANDSH was transmitted during parameterization of the serial interface.

248 - Interface not available

249 - All COM interfaces already open

250 - Not used any longer

251 - Not used any longer

252 - General interface error

Parity, frame, overrun

253 - Transmitter buffer overflow

254 - Receiver buffer overflow

255 - Timeout acknowledgement telegram

16 Glossary

Absolute time

The combination of time of day and date information.

Access path

The association of a symbolic name with a variable for the purpose of open communication.

Action

A Boolean variable, or a collection of operations to be performed, together with an associated control structure.

Action block

A graphical language element which utilizes a Boolean input variable to determine the value of a Boolean output variable or the enabling condition for an action, according to a predetermined control structure.

Address constant

Constant A#xxxx which contains the address information for Firmware data types.

Address of

Operator P# to establish the address of a variable; is needed at runtime to determine the basic address of a POINTER.

ANY_BIT

(generic data type) A combination of several data types as a group type, contains LWORD, DWORD, WORD, BYTE, BOOL.

ANY_DATE

(generic data type) A combination of several data types as a group type, contains DATE_AND_TIME, DATE, TIME_OF_DAY.

ANY_ELEMENTARY

(generic data type) A combination of several data types as a group type, contains

TIME,

ANY_BIT (LWORD, DWORD, WORD, BYTE, BOOL)

ANY_DATE (DATE_AND_TIME, DATE, TIME_OF_DAY),

ANY_INT (LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT),

ANY_REAL (LREAL, REAL),

ANY_STRING (STRING, CHAR, WSTRING, WCHAR).

ANY_INT

(generic data type) A combination of several data types as a group type, contains LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT.

Rexroth Indramat

ANY_MAGNITUDE

(generic data type) A combination of several data types as a group type, contains

TIME,

ANY_INT (LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT) und

ANY_REAL (LREAL, REAL).

ANY_NUM

(generic data type) A combination of several data types as a group type, contains ANY_INT (LINT, DINT, INT, SINT, ULINT, UDINT, UINT, USINT) and ANY_REAL (LREAL, REAL).

ANY_REAL

(generic data type) A combination of several data types as a group type, contains LREAL, REAL.

ANY_STRING

(generic data type) A combination of several data types as a group type, contains STRING, CHAR, WSTRING, WCHAR.

ARRAY

An aggregate that consists of data objects, with identical attributes, each of which may be uniquely referenced by subscripting (ISO).

Assignment

A mechanism to give a value to a variable or to an aggregate (ISO).

Bit String

A data element consisting of one or more bits.

BOOL

Elementary data type, seize: 1 Bit

Standard initial value: FALSE

Range of values: FALSE / TRUE and 0/1 and 2#0, 2#1 respectively

BYTE

Elementary data type, seize: 8 Bit

Standard initial value: 16#0

Range of values: 16#00 ...16#FF

Call

A language construct for invoking the execution of a function or function block.

CHAR

Character, elementary data type, seize: 8 Bit

Standard initial value: '' (empty)

Range of values: 16#00 ...16#FF

Character string

An aggregate that consists of an ordered sequence of characters.

Comment

A language construct for the inclusion of text in a program and having no impact on the execution of the program (ISO).

Compile

To translate a program organization or a data type specification into its machine language equivalent or an intermediate form.

Configuration

A language element which corresponding to a programmable controller system as defined in IEC 1131-1.

Counter function block

A function block which accumulates a value for the number of changes sensed at one or more specified inputs / input parameters.

Data type

A set of values together with a set of permitted operations (ISO).

Date and time

The date within the year and the time of day represented according to ISO 8601.

Declaration

The mechanism for establishing the definition of a language element. A declaration normally involves attaching an identifier to the language element, and allocating attributes such as data types and algorithms to it.

DINT

(double integer) Elementary data type, seize: 32 Bit

Standard initial value: 0

Range of values: -2147483648...2147483647

Direct representation

A means of representing a variable in a programmable controller program from which a manufacturer-specified correspondence to a physical or logical location (see logical location) may be determined directly.

DWORD

(double word) Elementary data type, seize: 32 Bit

Standard initial value: 16#0

Range of values: 16#00000000 ...16#FFFF FFFF

Error

S#ErrorFlg

Type: BOOL

S#ErrorFlg= FALSE, up to this time there was no error (standard); S#ErrorNr/S#ErrorTyp are unimportant.

S#ErrorFlg= TRUE, at least one error occurred which is further specified by S#ErrorNr, S#Error type.

S#ErrorNr

Type USINT, standard: 0,

S#ErrorNr> 0, detailed information about the error.

S#ErrorTyp

Type INT, standard: 0, detailed information about the causer,

S#ErrorTyp< 0, standard FN / FB, Firmware FN / FB or operations,

S#ErrorTyp> 0, reserved for user files.

Evaluation

The process of establishing a value for an expression or function, or for the outputs of a network or function block, during program execution.

Execution control element

A language element which controls the flow of program execution.

Falling edge

The change from 1 to 0 of a Boolean variable.

Firmware-

In connection with a function block, function, data type.

Completion of the minimum equipment of standard elements required by the standard; can be used, but not be modified by the user.

Focussed

Focussed file: file that is edited just now.

Function (procedure)

A program organization unit with 1...n input variables and internal variables which, when executed, yields exactly one data element and possibly additional output variables (which may be multi-valued, e.g. an array or structure), and whose invocation can be used in textual languages as an operand in an expression. Additionally, further outputs can be operated as output parameters (new in IEC 61131-3 2nd Edition).

Type name of the function:

Name with which the function is saved.

The name of the function is identical with the name of the main output of the function.

The type of the function corresponds to the type of the main output of the function.

Function block

Program organization unit, with 1...n input parameters, internal variables and 1...m output parameters.

Type name of the FB:

Name with which the function block is saved. See also function block type.

Application name of a FB / fb / Fb:

Name of the concrete application of the FB / fb in IEC 61131-3 also called instance, copy, case or assignment name (see also function block instance).

Function block diagram

A network in which the nodes are function block instances, graphically represented functions (procedures), variables, literals, and labels.

Function block instance

An instance of a function block type.

Function block type

A programmable controller programming language element consisting of:

- the definition of a data structure partitioned into input, output an internal variables;
- a set of operations to be performed upon the elements of the data structure when an instance of the function block type is invoked.

Generic data type

A combination of several data types as a group type (e.g. ANY_BIT contains LWORD, DWORD, WORD, BYTE, BOOL).

Global variable

A variable whose scope is global.

The programming system has a data pool on resource level. Each program and each function block in the resource can be reached by VAR_EXTERNAL.

Global scope

Scope of a declaration applying to all program organization units within a resource or configuration.

Identifier

A combination of letters, numbers, and underline characters which begins with a letter or underline and which names a language element.

Initial step

Within a procedure the first step as starting step is always very important. It is called initial step of the procedure. With the initial step the procedure starts and stops.

Initial value

The value assigned to a variable at system start-up.

Input variable (input)

A variable which is used to supply an argument to a program organization unit.

Instance

Generally: an individual named copy of a data structure.

Here: connected with a function block type or a program type which is preserved from on call of the belonging function to the next.

Instance name

An identifier associated with a specific instance.

Instantiation

The creation of an instance.

INT

(integer) Elementary data type, seize:16 Bit

Standard initial value: 0

Range of values: -32768...32767

Integer literal

Literal, which directly represents a value of type SINT, INT, DINT, LINT and USINT, UINT, UDINT respectively, or ULINT.

Invocation

The process of initiating the execution of the operations specified in a program organization unit.

Keyword

A lexical unit that characterizes a language element, e.g. "TRUE".

Label

A language construction naming an instruction, network, or group of networks, and including an identifier.

Language element

Any item identified by a symbol on the left-hand side of a production rule in the formal specification given in annex B IEC 1131, 2nd Edition.

LINT

(long integer) Elementary data type, seize: 64 Bit

Standard initial value: 0

Range of values: $-2^{63}...2^{63}-1$

Literal

A lexical unit that directly represents a value (ISO).

Loaded

Loaded file: file that was loaded in the main memory of the PC and over which at least one editor window is opened.

One of the windows can be active at this time.

Local menu

(PopUp menu) Menu that allows in every editor the access to every other editor.

Call by pressing <Shift>+<F10>.

Local scope

The scope of a declaration or label applying only to the program organization unit in which the declaration or label appears.

Logical location

The location of a hierarchically addressed variable in a schema which may or may not bear any relation to the physical structure of the programmable controller's inputs, outputs, and memory.

LWORD

(long word) Elementary data type, seize: 64 Bit

Standard initial value: 16#0

Range of values: 16#00000000 00000000..16#FFFFFFFF FFFFFFFF

Main File

File that can be preset in the menu „Compiler / Select main file“ and from which it is possible to compile, archive, save etc., matter if the file is loaded and / or active.

Network

An arrangement of nodes and interconnecting branches.

NIL-POINTER

Pointer without assigned address. The access to a variable, i.e. the memory, is not possible. The control is effected during the runtime of the program.

Off-delay / on-delay timer function block

A function block which delays the falling / rising edge of a Boolean input by a specific duration.

Operand

A language element on which an operation is performed.

Operation modes in the programming and commissioning system

Indicating mode, editing mode, status indication, online change.

Operator

A symbol that represents the action to be performed in an operation.

Output variable (output)

A variable which is used to return the result(s) of the evaluation of a program organization unit.

Overloaded

With respect to an operation or function, capable of operating on data of different types, e.g. ADD for INT or REAL.

Pointer

A pointer variable contains the address of a dynamic variable of a certain basis type. There are only typed pointers with length control during the access on the memory.

It is possible to assign a value to a pointer variable by the means of a P# operator (address of...):

The standard initial value is NIL.

PopUp-Menü

(Local menu) Menu that allows in every editor the access to further editor functions.

Call by pressing <Shift>+<F10>.

Power flow

The symbolic flow of electrical power in a ladder diagram, used to denote the progression of a logic solving algorithm.

Program

The program is the smallest software unit which can be loaded and started. A program can use function blocks and functions.

Program (verb)

To design, write and test user programs.

Program organization unit

A function, function block or program.
NOTE – This term may refer to either a type or an instance.

REAL

(real number) Elementary data type, seize: 32 Bit

Standard initial value: 0.0

Range of values:
-3.402823E38 ...-1.175495E-38 und +1.175495E-38 ... +3.402823E38

Concerning fixed-point numbers the number of the allowed characters is 7 Digits.

Real literal

A literal representing data of type REAL or LREAL.

Resource

A language element corresponding to a ”signal processing function” and its “main machine interface” and “sensor and actuator interface functions”, if any, as defined in IEC 1131-1.

Retentive data

Data stored in such a way that its value remains unchanged after a power down / power up sequence.

Return

A language construction within a program organization unit designating an end to the execution sequences in the unit.

Rising edge

The change from 0 to 1 of a Boolean variable.

Scope

That portion of a language element within which a declaration or label applies.

Semantics

The relationships between the symbolic elements of a programming language and their meaning, interpretation and use.

Single data element

A data element consisting of a single value.

Single-element variable

A variable which represents a single data element.

SINT

(short integer) Elementary data type, seize: 8 Bit

Standard initial value: 0

Range of values: -128..127

Special ...

In connection with function, function block and data type.

Protected user files or INDRAMAT files which can be used, but not be modified by the client.

Standard ...

In connection with function, function block and data type.

Minimum equipment of elements required by the standard; can be used, but not be modified by the client.

Step

A situation in which the behavior of a program organization unit with respect to its inputs and outputs follows a set of rules defined by the associated actions of the step.

STRING

Character string, elementary data type, 256 Byte are reserved for them.

Standard initial value: ” (empty),

Byte 1...255 can contain useable text,

Byte 0 contains the length, initial value: 16#00!

STRING[xx], character string with limited length, (xx+1) Byte are reserved for them.

Standard initial value: ” (empty),

Byte 1...xx can contain useable text,

Byte 0 contains the length, initial value: 16#00!

Structured data type

An aggregate data type which has been declared using a STRUCT or FUNCTION_BLOCK declaration.

Subscripting

A mechanism for referencing an array element by means of an array reference and one or more expressions that, when evaluated, denote the position of the element.

Symbolic representation

The use of identifiers to name variables.

Task

An execution control element providing for periodic or triggered execution of a group of associated program organization units.

Temporary flag

Flags allow to take temporary results out of the ladder diagram network. The result of the branch situated in front of the temporary flag is taken over.

TIME

(time) Elementary data type, seize: 32 Bit

Standard initial value: T#0s

Range of values: 0ms...23d23h59m59s999ms

Time literal

A literal representing data of type TIME, DATE, TIME_OF_DAY, or DATE_AND_TIME.

Transition

The condition whereby control passes from one or more predecessor steps to one or more successor steps along a directed link.

UDINT

(unsigned double integer) Elementary data type, seize: 32 Bit

Standard initial value: 0

Range of values: 0..4294967295

UINT

(unsigned integer) Elementary data type, seize: 16 Bit

Standard initial value: 0

Range of values: 0..65535

ULINT

(unsigned long integer) Elementary data type, seize: 64 Bit

Standard initial value: 0

Range of values: 0...2^{64}-1

Unsigned integer

An integer literal not containing a leading plus (+) or minus (-) sign.

User

In connection with program, function, function block, data type; written and governed by the user.

User administration

Component of the programming system to give access rights to every concrete user.

The user is assigned to a user group and receives in this way their rights.

Group	Description
Administrator	All commands, assignment of rights
WinPCL specialist	All commands
PLC programmer	All public rights
Service	No commands to change files
Observer (guest)	No commands to change files

USINT

(unsigned short integer) Elementary data type, seize: 8 Bit

Standard initial value: 0

Range of value: 0..255

Wired OR

A construction for achieving the Boolean OR function in the LD language by connecting together the right ends of horizontal connectives with vertical connectives.

WORD

Elementary data type, seize: 16 Bit

Standard initial value: 16#0

Range of values: 16#0000 ...16#FFFF

Work file

File that is edited in the programming and commissioning system:

user program, user function block, user function, user data type.

Abbreviations

AQ

Action qualifier

AT

Action time, completion for the AQ – action qualifier L, D, DS, SD, SL for action blocks, constants or variables of type TIME

CNC

Computerized Numerical Control

FB

Function block

Special: user written function block, user FB

fb

Special: Firmware / standard function block

Fb

Special: special function block, no possibility to edit

FBD

Function block diagram

FN

Function

Special: user written function, user FN

fn

Special: Firmware function, standard function

Fn

Special: special function, no possibility to edit

GUI

Graphical user interface

IBS

Interbus, field bus

IL

Instruction list

LD

Ladder diagram

MAP file

File containing the address information of the project in the PLC for GUI and the status display.

MUI

User interface for the control

PC

Personal Computer

PLC

Programmable controller

POU

Program organization unit, i.e. a program, function block or function

PR

Program; special: user written program, user program

RE

Resource; special: user written resource file to establish and globally enable variables, to establish tasks and to assign program instances to this tasks.

SFC

Sequential function chart

TY

Data type,

Special: user written data type, user TY

ty

Special: Firmware data type, standard data type

Ty

Special: special data type, no possibility to edit

17 List of Figures

Rexroth
Bosch Group

18 Index

%

?

^

1

2

3

4

5

6

7

8

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

V

W

X

19 Service & Support

19.1 Helpdesk

Unser Kundendienst-Helpdesk im Hauptwerk Lohr am Main steht Ihnen mit Rat und Tat zur Seite. Sie erreichen uns

Our service helpdesk at our headquarters in Lohr am Main, Germany can assist you in all kinds of inquiries. Contact us

- telefonisch - by phone: **49 (0) 9352 40 50 60**
  über Service Call Entry Center Mo-Fr 07:00-18:00
  - via Service Call Entry Center Mo-Fr 7:00 am - 6:00 pm
- per Fax - by fax: **+49 (0) 9352 40 49 41**
- per e-Mail - by e-mail: **service.svc@boschrexroth.de**

19.2 Service-Hotline

Außerhalb der Helpdesk-Zeiten ist der Service direkt ansprechbar unter

After helpdesk hours, contact our service department directly at

+49 (0) 171 333 88 26

oder - or **+49 (0) 172 660 04 06**

19.3 Internet

Unter **www.boschrexroth.com** finden Sie ergänzende Hinweise zu Service, Reparatur und Training sowie die **aktuellen** Adressen *) unserer auf den folgenden Seiten aufgeführten Vertriebs- und Servicebüros.

☐ Verkaufsniederlassungen
☐ Niederlassungen mit Kundendienst

Außerhalb Deutschlands nehmen Sie bitte zuerst Kontakt mit unserem für Sie nächstgelegenen Ansprechpartner auf.

*) Die Angaben in der vorliegenden Dokumentation können seit Drucklegung überholt sein.

At **www.boschrexroth.com** you may find additional notes about service, repairs and training in the Internet, as well as the **actual** addresses *) of our sales- and service facilities figuring on the following pages.

☐ sales agencies
☐ offices providing service

Please contact our sales / service office in your area first.

*) Data in the present documentation may have become obsolete since printing.

19.4 Vor der Kontaktaufnahme... - Before contacting us...

Wir können Ihnen schnell und effizient helfen wenn Sie folgende Informationen bereithalten:

1. detaillierte Beschreibung der Störung und der Umstände.
2. Angaben auf dem Typenschild der betreffenden Produkte, insbesondere Typenschlüssel und Seriennummern.
3. Tel.-/Faxnummern und e-Mail-Adresse, unter denen Sie für Rückfragen zu erreichen sind.

For quick and efficient help, please have the following information ready:

1. Detailed description of the failure and circumstances.
2. Information on the type plate of the affected products, especially type codes and serial numbers.
3. Your phone/fax numbers and e-mail address, so we can contact you in case of questions.

19.5 Kundenbetreuungsstellen - Sales & Service Facilities

Deutschland – Germany

vom Ausland: (0) nach Landeskennziffer weglassen!
from abroad: don't dial (0) after country code!

Vertriebsgebiet Mitte Germany Centre Rexroth Indramat GmbH Bgm.-Dr.-Nebel-Str. 2 / Postf. 1357 97816 Lohr am Main / 97803 Lohr **Kompetenz-Zentrum Europa** Tel.: +49 (0)9352 40-0 Fax: +49 (0)9352 40-4885	**SERVICE** **CALL ENTRY CENTER** **MO – FR** **von 07:00 - 18:00 Uhr** **from 7 am – 6 pm** **Tel. +49 (0) 9352 40 50 60** service.svc@boschrexroth.de	**SERVICE** **HOTLINE** **MO – FR** **von 17:00 - 07:00 Uhr** **from 5 pm - 7 am** **+ SA / SO** **Tel.: +49 (0)172 660 04 06** **oder / or** **Tel.: +49 (0)171 333 88 26**	**SERVICE** **ERSATZTEILE / SPARES** verlängerte Ansprechzeit - extended office time - ♦ nur an Werktagen - only on working days - ♦ von 07:00 - 18:00 Uhr - from 7 am - 6 pm - **Tel. +49 (0) 9352 40 42 22**
Vertriebsgebiet Süd Germany South Bosch Rexroth AG Landshuter Allee 8-10 80637 München Tel.: +49 (0)89 127 14-0 Fax: +49 (0)89 127 14-490	Vertriebsgebiet West Germany West Bosch Rexroth AG Regionalzentrum West Borsigstrasse 15 40880 Ratingen Tel.: +49 (0)2102 409-0 Fax: +49 (0)2102 409-406 +49 (0)2102 409-430	Gebiet Südwest Germany South-West Bosch Rexroth AG Service-Regionalzentrum Süd-West Siemensstr.1 70736 Fellbach Tel.: +49 (0)711 51046–0 Fax: +49 (0)711 51046–248	
Vertriebsgebiet Nord Germany North Bosch Rexroth AG Walsroder Str. 93 30853 Langenhagen Tel.: +49 (0) 511 72 66 57-0 Service: +49 (0) 511 72 66 57-256 Fax: +49 (0) 511 72 66 57-93 Service: +49 (0) 511 72 66 57-783	Vertriebsgebiet Mitte Germany Centre Bosch Rexroth AG Regionalzentrum Mitte Waldecker Straße 13 64546 Mörfelden-Walldorf Tel.: +49 (0) 61 05 702-3 Fax: +49 (0) 61 05 702-444	Vertriebsgebiet Ost Germany East Bosch Rexroth AG Beckerstraße 31 09120 Chemnitz Tel.: +49 (0)371 35 55-0 Fax: +49 (0)371 35 55-333	Vertriebsgebiet Ost Germany East Bosch Rexroth AG Regionalzentrum Ost Walter-Köhn-Str. 4d 04356 Leipzig Tel.: +49 (0)341 25 61-0 Fax: +49 (0)341 25 61-111

Europa (West) - Europe (West)

vom Ausland: (0) nach Landeskennziffer weglassen, Italien: 0 nach Landeskennziffer mitwählen
from abroad: don't dial (0) after country code, Italy: dial 0 after country code

Austria - Österreich Bosch Rexroth GmbH Electric Drives & Controls Stachegasse 13 1120 Wien Tel.: +43 (0)1 985 25 40 Fax: +43 (0)1 985 25 40-93	Austria – Österreich Bosch Rexroth GmbH Electric Drives & Controls Industriepark 18 4061 Pasching Tel.: +43 (0)7221 605-0 Fax: +43 (0)7221 605-21	Belgium - Belgien Bosch Rexroth AG Electric Drives & Controls Industrielaan 8 1740 Ternat Tel.: +32 (0)2 5830719 - service: +32 (0)2 5830717 Fax: +32 (0)2 5830731 service@boschrexroth.be	Denmark - Dänemark BEC A/S Zinkvej 6 8900 Randers Tel.: +45 (0)87 11 90 60 Fax: +45 (0)87 11 90 61
Great Britain – Großbritannien Bosch Rexroth Ltd. Electric Drives & Controls Broadway Lane, South Cerney Cirencester, Glos GL7 5UH Tel.: +44 (0)1285 863000 Fax: +44 (0)1285 863030 sales@boschrexroth.co.uk service@boschrexroth.co.uk	Finland - Finnland Bosch Rexroth Oy Electric Drives & Controls Ansatie 6 017 40 Vantaa Tel.: +358 (0)9 84 91-11 Fax: +358 (0)9 84 91-13 60	France - Frankreich Bosch Rexroth SAS Electric Drives & Controls Avenue de la Trentaine (BP. 74) 77503 Chelles Cedex Tel.: +33 (0)164 72-70 00 Fax: +33 (0)164 72-63 00 **Hotline:** +33 (0)608 33 43 28	France - Frankreich Bosch Rexroth SAS Electric Drives & Controls ZI de Thibaud, 20 bd. Thibaud (BP. 1751) 31084 Toulouse Tel.: +33 (0)5 61 43 61 87 Fax: +33 (0)5 61 43 94 12
France – Frankreich Bosch Rexroth SAS Electric Drives & Controls 91, Bd. Irène Joliot-Curie 69634 Vénissieux – Cedex Tel.: +33 (0)4 78 78 53 65 Fax: +33 (0)4 78 78 53 62	Italy - Italien Bosch Rexroth S.p.A. Via G. Di Vittoria, 1 20063 Cernusco S/N.MI Tel.: +39 02 92 365 1 +39 02 92 365 326 Fax: +39 02 92 365 500 +39 02 92 365 516378	Italy - Italien Bosch Rexroth S.p.A. Via Paolo Veronesi, 250 10148 Torino Tel.: +39 011 224 88 11 Fax: +39 011 224 88 30	Italy - Italien Bosch Rexroth S.p.A. Via del Progresso, 16 (Zona Ind.) 35020 Padova Tel.: +39 049 8 70 13 70 Fax: +39 049 8 70 13 77
Italy - Italien Bosch Rexroth S.p.A. Via Mascia, 1 80053 Castellamare di Stabia NA Tel.: +39 081 8 71 57 00 Fax: +39 081 8 71 68 85	Italy - Italien Bosch Rexroth S.p.A. Via Isonzo, 61 40033 Casalecchio di Reno (Bo) Tel.: +39 051 29 86 430 Fax: +39 051 29 86 490	Netherlands - Niederlande/Holland Bosch Rexroth Services B.V. Technical Services Kruisbroeksestraat 1 (P.O. Box 32) 5281 RV Boxtel Tel.: +31 (0) 411 65 16 40 +31 (0) 411 65 17 27 Fax: +31 (0) 411 67 78 14 +31 (0) 411 68 28 60 services@boschrexroth.nl	Netherlands – Niederlande/Holland Bosch Rexroth B.V. Kruisbroeksestraat 1 (P.O. Box 32) 5281 RV Boxtel Tel.: +31 (0) 411 65 19 51 Fax: +31 (0) 411 65 14 83 www.boschrexroth.nl
Norway - Norwegen Bosch Rexroth AS Electric Drives & Controls Berghagan 1 or: Box 3007 1405 Ski-Langhus 1402 Ski Tel.: +47 (0)64 86 41 00 Fax: +47 (0)64 86 90 62 jul.ruud@rexroth.no	Spain - Spanien Bosch Rexroth S.A. Electric Drives & Controls Centro Industrial Santiga Obradors s/n 08130 Santa Perpetua de Mogoda Barcelona Tel.: +34 9 37 47 94 00 Fax: +34 9 37 47 94 01	Spain – Spanien Goimendi S.A. Electric Drives & Controls Parque Empresarial Zuatzu C/ Francisco Grandmontagne no.2 20018 San Sebastian Tel.: +34 9 43 31 84 21 - service: +34 9 43 31 84 56 Fax: +34 9 43 31 84 27 - service: +34 9 43 31 84 60 sat.indramat@goimendi.es	Sweden - Schweden Bosch Rexroth AB Electric Drives & Controls - Varuvägen 7 (Service: Konsumentvägen 4, Älfsjö) 125 81 Stockholm Tel.: +46 (0)8 727 92 00 Fax: +46 (0)8 647 32 77
Sweden - Schweden Bosch Rexroth AB Electric Drives & Controls Ekvändan 7 254 67 Helsingborg Tel.: +46 (0) 42 38 88 -50 Fax: +46 (0) 42 38 88 -74	Switzerland West - Schweiz West Bosch Rexroth Suisse SA Electric Drives & Controls Rue du village 1 1020 Renens Tel.: +41 (0)21 632 84 20 Fax: +41 (0)21 632 84 21	Switzerland East - Schweiz Ost Bosch Rexroth Schweiz AG Electric Drives & Controls Hemrietstrasse 2 8863 Buttikon Tel. +41 (0) 55 46 46 111 Fax +41 (0) 55 46 46 222	

Europa (Ost) - Europe (East)

vom Ausland: (0) nach Landeskennziffer weglassen
from abroad: don't dial (0) after country code

Czech Republic - Tschechien Bosch -Rexroth, spol.s.r.o. Hviezdoslavova 5 627 00 Brno Tel.: +420 (0)5 48 126 358 Fax: +420 (0)5 48 126 112	Czech Republic - Tschechien DEL a.s. Strojírenská 38 591 01 Zdar nad Sázavou Tel.: +420 566 64 3144 Fax: +420 566 62 1657	Hungary - Ungarn Bosch Rexroth Kft. Angol utca 34 1149 Budapest Tel.: +36 (1) 422 3200 Fax: +36 (1) 422 3201	Poland – Polen Bosch Rexroth Sp.zo.o. ul. Staszica 1 05-800 Pruszków Tel.: +48 22 738 18 00 – service: +48 22 738 18 46 Fax: +48 22 758 87 35 – service: +48 22 738 18 42
Poland – Polen Bosch Rexroth Sp.zo.o. Biuro Poznan ul. Dabrowskiego 81/85 60-529 Poznan Tel.: +48 061 847 64 62 /-63 Fax: +48 061 847 64 02	Romania - Rumänien East Electric S.R.L. Bdul Basarabia no.250, sector 3 73429 Bucuresti Tel./Fax:: +40 (0)21 255 35 07 +40 (0)21 255 77 13 Fax: +40 (0)21 725 61 21 eastel@rdsnet.ro	Romania - Rumänien Bosch Rexroth Sp.zo.o. Str. Drobety nr. 4-10, app. 14 70258 Bucuresti, Sector 2 Tel.: +40 (0)1 210 48 25 +40 (0)1 210 29 50 Fax: +40 (0)1 210 29 52	Russia - Russland Bosch Rexroth OOO Wjatskaja ul. 27/15 127015 Moskau Tel.: +7-095-785 74 78 +7-095 785 74 79 Fax: +7 095 785 74 77 laura.kanina@boschrexroth.ru
Russia - Russland ELMIS 10, Internationalnaya 246640 Gomel, Belarus Tel.: +375/ 232 53 42 70 +375/ 232 53 21 69 Fax: +375/ 232 53 37 69 elmis_ltd@yahoo.com	Turkey - Türkei Bosch Rexroth Otomasyon San & Tic. A..S. Fevzi Cakmak Cad No. 3 34295 Sefaköy - Istanbul Tel.: +90 212 413 34-00 Fax: +90 212 413 34-17	Slowenia - Slowenien DOMEL Otoki 21 64 228 Zelezniki Tel.: +386 5 5117 152 Fax: +386 5 5117 225 brane.ozebek@domel.si	

Africa, Asia, Australia – incl. Pacific Rim

Australia - Australien
AIMS - Australian Industrial Machinery Services Pty. Ltd.
28 Westside Drive
Laverton North Vic 3026
Melbourne
Tel.: +61 3 93 59 0228
Fax: +61 3 93 59 0286
Hotline: +61 4 19 369 195
terryobrien@aimservices.com.au

Australia - Australien
Bosch Rexroth Pty. Ltd.
No. 7, Endeavour Way
Braeside Victoria, 31 95
Melbourne
Tel.: +61 3 95 80 39 33
Fax: +61 3 95 80 17 33
mel@rexroth.com.au

China
Shanghai Bosch Rexroth Hydraulics & Automation Ltd.
Waigaoqiao, Free Trade Zone
No.122, Fu Te Dong Yi Road
Shanghai 200131 - P.R.China
Tel.: +86 21 58 66 30 30
Fax: +86 21 58 66 55 23
richard.yang_sh@boschrexroth.com.cn
gf.zhu_sh@boschrexroth.com.cn

China
Shanghai Bosch Rexroth Hydraulics & Automation Ltd.
4/f, Marine Tower
No.1, Pudong Avenue
Shanghai 200120 - P.R.China
Tel.: +86 21 68 86 15 88
Fax: +86 21 58 40 65 77

China
Bosch Rexroth China Ltd.
15/F China World Trade Center
1, Jianguomenwai Avenue
Beijing 100004, P.R.China
Tel.: +86 10 65 05 03 80
Fax: +86 10 65 05 03 79

China
Bosch Rexroth China Ltd.
Guangzhou Repres. Office
Room 1014-1016, Metro Plaza,
Tian He District, 183 Tian He Bei Rd
Guangzhou 510075, P.R.China
Tel.: +86 20 8755-0030
+86 20 8755-0011
Fax: +86 20 8755-2387

China
Bosch Rexroth (China) Ltd.
A-5F., 123 Lian Shan Street
Sha He Kou District
Dalian 116 023, P.R.China
Tel.: +86 411 46 78 930
Fax: +86 411 46 78 932

China
Melchers GmbH
BRC-SE, Tightening & Press-fit
13 Floor Est Ocean Centre
No.588 Yanan Rd. East
65 Yanan Rd. West
Shanghai 200001
Tel.: +86 21 6352 8848
Fax: +86 21 6351 3138

Hongkong
Bosch Rexroth (China) Ltd.
6th Floor,
Yeung Yiu Chung No.6 Ind Bldg.
19 Cheung Shun Street
Cheung Sha Wan,
Kowloon, Hongkong
Tel.: +852 22 62 51 00
Fax: +852 27 41 33 44
alexis.siu@boschrexroth.com.hk

India - Indien
Bosch Rexroth (India) Ltd.
Electric Drives & Controls
Plot. No.96, Phase III
Peenya Industrial Area
Bangalore – 560058
Tel.: +91 80 51 17 0-211...-218
Fax: +91 80 83 94 345
+91 80 83 97 374
mohanvelu.t@boschrexroth.co.in

India - Indien
Bosch Rexroth (India) Ltd.
Electric Drives & Controls
Advance House, II Floor
Ark Industrial Compound
Narol Naka, Makwana Road
Andheri (East), Mumbai - 400 059
Tel.: +91 22 28 56 32 90
+91 22 28 56 33 18
Fax: +91 22 28 56 32 93
singh.op@boschrexroth.co.in

India - Indien
Bosch Rexroth (India) Ltd.
S-10, Green Park Extension
New Delhi – 110016
Tel.: +91 11 26 56 65 25
+91 11 26 56 65 27
Fax: +91 11 26 56 68 87
koul.rp@boschrexroth.co.in

Indonesia - Indonesien
PT. Bosch Rexroth
Building # 202, Cilandak
Commercial Estate
Jl. Cilandak KKO, Jakarta 12560
Tel.: +62 21 7891169 (5 lines)
Fax: +62 21 7891170 - 71

Japan
Bosch Rexroth Automation Corp.
Service Center Japan
Yutakagaoka 1810, Meito-ku,
NAGOYA 465-0035, Japan
Tel.: +81 52 777 88 41
+81 52 777 88 53
+81 52 777 88 79
Fax: +81 52 777 89 01

Japan
Bosch Rexroth Automation Corp.
Electric Drives & Controls
1F, I.R. Building
Nakamachidai 4-26-44, Tsuzuki-ku
YOKOHAMA 224-0041, Japan
Tel.: +81 45 942 72 10
Fax: +81 45 942 03 41

Korea
Bosch Rexroth-Korea Ltd.
Electric Drives and Controls
Bongwoo Bldg. 7FL, 31-7, 1Ga
Jangchoong-dong, Jung-gu
Seoul, 100-391
Tel.: +82 234 061 813
Fax: +82 222 641 295

Korea
Bosch Rexroth-Korea Ltd.
1515-14 Dadae-Dong, Saha-Ku
Electric Drives & Controls
Pusan Metropolitan City, 604-050
Tel.: +82 51 26 00 741
Fax: +82 51 26 00 747
gyhan@rexrothkorea.co.kr

Malaysia
Bosch Rexroth Sdn.Bhd.
11, Jalan U8/82, Seksyen U8
40150 Shah Alam
Selangor, Malaysia
Tel.: +60 3 78 44 80 00
Fax: +60 3 78 45 48 00
hockhwa@hotmail.com
rexroth1@tm.net.my

Singapore - Singapur
Bosch Rexroth Pte Ltd
15D Tuas Road
Singapore 638520
Tel.: +65 68 61 87 33
Fax: +65 68 61 18 25
sanjay.nemade
@boschrexroth.com.sg

South Africa - Südafrika
TECTRA Automation (Pty) Ltd.
71 Watt Street, Meadowdale
Edenvale 1609
Tel.: +27 11 971 94 00
Fax: +27 11 971 94 40
Hotline: +27 82 903 29 23
georgv@tectra.co.za

Taiwan
Bosch Rexroth Co., Ltd.
Taichung Branch
1F., No. 29, Fu-Ann 5th Street,
Xi-Tun Area, Taichung City
Taiwan, R.O.C.
Tel : +886 - 4 -23580400
Fax: +886 - 4 -23580402
charlie.chen@boschrexroth.com.tw
jim.lin@boschrexroth.com.tw
david.lai@boschrexroth.com.tw

Thailand
NC Advance Technology Co. Ltd.
59/76 Moo 9
Ramintra road 34
Tharang, Bangkhen,
Bangkok 10230
Tel.: +66 2 943 70 62
+66 2 943 71 21
Fax: +66 2 509 23 62
sonkawin@hotmail.com

Nordamerika – North America

USA Headquarters - Hauptniederlassung Bosch Rexroth Corporation Electric Drives & Controls 5150 Prairie Stone Parkway Hoffman Estates, IL 60192-3707 Tel.: +1 847 6 45 36 00 Fax: +1 847 6 45 62 01 servicebrc@boschrexroth-us.com repairbrc@boschrexroth-us.com	USA Central Region - Mitte Bosch Rexroth Corporation Electric Drives & Controls Central Region Technical Center 1701 Harmon Road Auburn Hills, MI 48326 Tel.: +1 248 3 93 33 30 Fax: +1 248 3 93 29 06	USA Southeast Region - Südwest Bosch Rexroth Corporation Electric Drives & Controls Southeastern Technical Center 3625 Swiftwater Park Drive Suwanee, Georgia 30124 Tel.: +1 770 9 32 32 00 Fax: +1 770 9 32 19 03	USA SERVICE-HOTLINE - 7 days x 24hrs - **+1-800-REX-ROTH** **+1-800-739-7684**
USA East Region – Ost Bosch Rexroth Corporation Electric Drives & Controls Charlotte Regional Sales Office 14001 South Lakes Drive Charlotte, North Carolina 28273 Tel.: +1 704 5 83 97 62 +1 704 5 83 14 86	USA Northeast Region – Nordost Bosch Rexroth Corporation Electric Drives & Controls Northeastern Technical Center 99 Rainbow Road East Granby, Connecticut 06026 Tel.: +1 860 8 44 83 77 Fax: +1 860 8 44 85 95	USA West Region – West Bosch Rexroth Corporation 7901 Stoneridge Drive, Suite 220 Pleasant Hill, California 94588 Tel.: +1 925 227 10 84 Fax: +1 925 227 10 81	
Canada East - Kanada Ost Bosch Rexroth Canada Corporation Burlington Division 3426 Mainway Drive Burlington, Ontario Canada L7M 1A8 Tel.: +1 905 335 55 11 Fax: +1 905 335-41 84 michael.moro@boschrexroth.ca	Canada West - Kanada West Bosch Rexroth Canada Corporation 5345 Goring St. Burnaby, British Columbia Canada V7J 1R1 Tel. +1 604 205-5777 Fax +1 604 205-6944 david.gunby@boschrexroth.ca	Mexico Bosch Rexroth Mexico S.A. de C.V. Calle Neptuno 72 Unidad Ind. Vallejo 07700 Mexico, D.F. Tel.: +52 55 57 54 17 11 Fax: +52 55 57 54 50 73 mariofelipe.hernandez@boschrexroth.com.mx	Mexico Bosch Rexroth S.A. de C.V. Calle Argentina No 3913 Fracc. las Torres 64930 Monterrey, N.L. Tel.: +52 81 83 65 22 53 +52 81 83 65 89 11 +52 81 83 49 80 91 Fax: +52 81 83 65 52 80 mario.quiroga@boschrexroth.com.mx

Südamerika – South America

Argentina - Argentinien Bosch Rexroth S.A.I.C. "The Drive & Control Company" Acassusso 48 41/47 1605 Munro Provincia de Buenos Aires Tel.: +54 11 4756 01 40 Fax: +54 11 4756 01 36 victor.jabif@boschrexroth.com.ar	Argentina - Argentinien NAKASE Servicio Tecnico CNC Calle 49, No. 5764/66 B1653AOX Villa Balester Provincia de Buenos Aires Tel.: +54 11 4768 36 43 Fax: +54 11 4768 24 13 nakase@usa.net nakase@nakase.com gerencia@nakase.com (Service)	Brazil - Brasilien Bosch Rexroth Ltda. Av. Tégula, 888 Ponte Alta, Atibaia SP CEP 12942-440 Tel.: +55 11 4414 56 92 +55 11 4414 56 84 Fax sales: +55 11 4414 57 07 Fax serv.: +55 11 4414 56 86 alexandre.wittwer@rexroth.com.br	Brazil - Brasilien Bosch Rexroth Ltda. R. Dr.Humberto Pinheiro Vieira, 100 Distrito Industrial [Caixa Postal 1273] 89220-390 Joinville - SC Tel./Fax: +55 47 473 58 33 Mobil: +55 47 9974 6645 prochnow@zaz.com.br
Columbia - Kolumbien Reflutec de Colombia Ltda. Calle 37 No. 22-31 Santafé de Bogotá, D.C. Colombia Tel.: +57 1 368 82 67 +57 1 368 02 59 Fax: +57 1 268 97 37 reflutec@neutel.com.co reflutec@007mundo.com			

Rexroth
Bosch Group

Bosch Rexroth AG
Electric Drives and Controls
Bgm.-Dr.-Nebel-Str. 2
97816 Lohr a. Main, Germany
info@boschrexroth.de
www.boschrexroth.de

298446

Printed in Germany
DOK-CONTRL-WINPCL*06VRS-AW01-EN-P