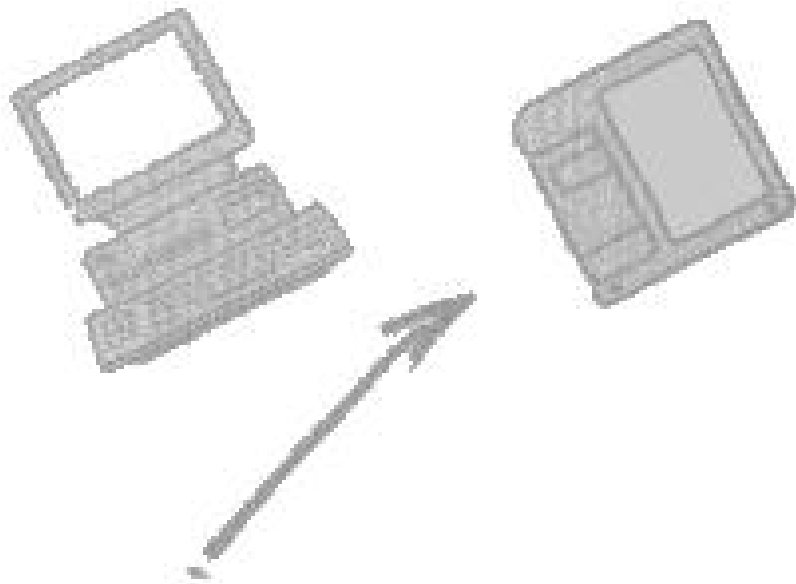


**HITACHI**

# Pro-H Manual



# Contents

<b>INTRODUCTION</b> .....	<b>1-1</b>
Pro-H - a highly efficient, powerful and complete tool .....	1-1
What kind of documentation do you get?.....	1-2
Symbols and textual conventions.....	1-3
<b>PRO-H AND IEC 61131-3</b> .....	<b>2-1</b>
What is IEC 61131-3?.....	2-1
Configuration elements.....	2-2
Configurations .....	2-2
Resources .....	2-2
Tasks .....	2-2
Configurations elements.....	2-3
POUs, programs, function blocks and functions.....	2-4
Program organization units - POUs .....	2-4
POUs .....	2-6
Instantiation.....	2-6
Declaration and instruction part of a POU .....	2-7
Variables and data types.....	2-8
Projects.....	2-10
Libraries.....	2-10
Programming languages and SFC.....	2-11

**GETTING STARTED ..... 3-1**

- System requirements ..... 3-1
  - Hardware requirements ..... 3-1
  - Software requirements ..... 3-1
- Installing the program ..... 3-2
- Calling the program..... 3-2
- Using mouse and keyboard ..... 3-3
- User interface ..... 3-4
  - Menu ..... 3-5
  - Toolbars ..... 3-7
  - Defining keyboard shortcuts with the Shortcut Manager ..... 3-9
  - Main screen and workspace ..... 3-13
  - Message window ..... 3-14
  - Cross reference window ..... 3-15
  - Status bar ..... 3-19
- Using help ..... 3-20
- Editors ..... 3-21
  - The project manager - a powerful tool for program organization ..... 3-21
  - The graphic editor - easy programming in SFC, FBD and LD..... 3-22
  - The text editors - easy programming in IL and ST ..... 3-23
  - The pagelayout editor - creating pagelayouts for printing ..... 3-23
- The Edit Wizard ..... 3-24
- Overview window for graphical worksheets ..... 3-27
- Saving changes while editing ..... 3-28
- Exiting worksheets ..... 3-28
- Exiting the program ..... 3-29

**EDITING THE PROJECT STRUCTURE ..... 4-1**

- Creating a new project ..... 4-1
- Changing the properties of existing POUs ..... 4-3
- Inserting new POUs ..... 4-5
- Inserting worksheets ..... 4-6
- Announcing libraries ..... 4-8
- Deleting worksheets, POUs or libraries..... 4-9
- Saving the existing project and zipping the project files ..... 4-9

**LITERALS, DATA TYPES AND VARIABLES ..... 5-1**

- Literals ..... 5-1
  - Numeric literals..... 5-1
  - Character string literals ..... 5-2
  - Duration literals ..... 5-2
- Introduction to the IEC data types ..... 5-2
- Elementary data types ..... 5-3
- Generic data types..... 5-4
- User defined data types ..... 5-4
- Array data types..... 5-5
  - Declaring arrays ..... 5-5
  - Programming example ..... 5-6
  - Multi-dimensional arrays..... 5-7
  - Initializing arrays..... 5-7
- Structured data types..... 5-8
  - Programming example ..... 5-8
  - Arrays of structures ..... 5-8
  - Structures with arrays..... 5-9
  - Initializing structures..... 5-9
- String data types ..... 5-10
  - Programming example ..... 5-10
- Calling the text editor with the data type worksheet ..... 5-10
- Editing type declarations using the Edit Wizard ..... 5-11
- Symbolic, located variables and directly represented variables..... 5-13
- Global and local variables ..... 5-14
- Retentive variables..... 5-14
- Initializing variables ..... 5-15
- Variable declaration keywords ..... 5-15
- Declaring variables ..... 5-18
- Instantiation ..... 5-20

<b>EDITING IN ST</b> .....	<b>6-1</b>
Calling the text editor with a ST worksheet .....	6-1
Introduction to ST .....	6-2
Inserting and editing assignment statements .....	6-3
Inserting and editing further statements .....	6-4
Inserting statements using the Edit Wizard .....	6-5
Inserting variables .....	6-7
Calling functions or function blocks using the Edit Wizard .....	6-10
<b>EDITING IN IL</b> .....	<b>7-1</b>
Calling the text editor with an IL worksheet .....	7-1
Instructions, operators, modifiers and operands .....	7-2
Inserting instructions using the Edit Wizard .....	7-4
Inserting variables .....	7-6
Using jumps and labels .....	7-9
Calling functions or function blocks using the Edit Wizard .....	7-10
<b>EDITING IN FBD</b> .....	<b>8-1</b>
Calling the graphic editor with a FBD worksheet .....	8-1
Introduction to FBD .....	8-2
Inserting functions and function blocks using the Edit Wizard .....	8-3
Changing the properties of functions and function blocks .....	8-5
Replacing functions and function blocks .....	8-6
Inserting variables .....	8-6
Connecting objects .....	8-9
Negation of inputs and outputs .....	8-13
Duplicating inputs of functions .....	8-14
<b>EDITING IN LD</b> .....	<b>9-1</b>
Calling the graphic editor with a LD worksheet .....	9-1
LD networks, contacts, coils and power rails .....	9-2
Inserting contacts and coils .....	9-4
Inserting serial contacts and coils .....	9-5
Inserting parallel contacts or coils .....	9-6
Using the LD branch edit mode .....	9-7
Changing the properties of contacts and coils .....	9-8
Inserting variables .....	9-11
Calling functions or function blocks using the Edit Wizard .....	9-11

**EDITING IN SFC ..... 10-1**

- Calling the graphic editor with a SFC worksheet ..... 10-1
- Introduction to SFC ..... 10-2
- Inserting a first SFC network..... 10-3
- Inserting more steps and transitions ..... 10-4
- Changing an initial step into a normal step or vice versa..... 10-6
- Inserting alternative branches ..... 10-7
- Inserting simultaneous branches..... 10-9
- Using the SFC branch edit mode ..... 10-11
- Inserting variables for actions ..... 10-12
- Inserting variables for transitions ..... 10-14
- Calling functions..... 10-18
- Action and transition details ..... 10-18

**COMPILING, DOWNLOADING AND DEBUGGING ..... 11-1**

- Inserting configurations, resources and tasks ..... 11-1
- Associating programs to tasks ..... 11-4
- Compiling a project ..... 11-5
- Compiling a project using 'Make' ..... 11-6
- Patching POUs ..... 11-8
- Downloading the project..... 11-10
- Calling worksheets in online mode ..... 11-12
- Switching between online and offline mode ..... 11-15
- Switching to address status and powerflow..... 11-16
- Forcing and overwriting variables..... 11-17
- Setting and resetting breakpoints..... 11-19
- Debugging with set breakpoints ..... 11-20
- Using the watch window..... 11-22
- Debugging user defined data types using the watch window ..... 11-24

<b>PRINTING YOUR PROJECT WITH A CUSTOMIZED PAGELAYOUT .....</b>	<b>12-1</b>
Printing the project .....	12-1
Controlling the print process using the dialog 'Print Project' .....	12-1
Defining a pagelayout as default pagelayout .....	12-3
Using the pagelayout editor .....	12-5
Creating a new pagelayout .....	12-5
Defining the source area .....	12-6
Inserting elements in your pagelayout .....	12-8
Editing environment items .....	12-9
Using preview .....	12-10
<b>EDITING SPECIFIC INLINE CODE .....</b>	<b>A1-1</b>
General information .....	A1-1
Which information do you get in this appendix? .....	A1-2
Editing specific inline code in IL .....	A1-3
Calling the text editor with an IL worksheet .....	A1-3
Specific code blocks .....	A1-3
Inserting specific code blocks using the Edit Wizard .....	A1-4
Inserting and declaring variables of specific code statements .....	A1-4
Editing specific inline code in LD .....	A1-6
Calling the graphic editor with a LD worksheet .....	A1-6
Arithmetic box with specific inline code .....	A1-7
Inserting an arithmetic box connected to an existing LD network .....	A1-8
Inserting an arithmetic box as a single object .....	A1-10
Connecting an arithmetic box to a LD network .....	A1-11
Online monitoring .....	A1-14
Compiling and downloading the project .....	A1-14
Calling worksheet in online mode and displaying online values in IL and LD work sheet .....	A1-14
<b>COMPILING, DOWNLOADING, DEBUGGING AND UPLOADING .....</b>	<b>A2-1</b>
Inserting configurations, resources and tasks .....	A2-1
Associating programs to tasks .....	A2-9
Compiling a project .....	A2-10
Compiling a project using 'Make' .....	A2-11
Patching POUs .....	A2-13
Downloading the project .....	A2-15
Calling worksheets in online mode .....	A2-17
Switching between online and offline mode .....	A2-20

Overwriting variables.....	A2-21
Setting and resetting breakpoints.....	A2-22
Debugging with set breakpoints .....	A2-22
Using the watch window.....	A2-22
Uploading the project .....	A2-25
<b>DIRECTLY REPRESENTED VARIABLES SUPPORTED BY HIDIC PLCS .....</b>	<b>A3-1</b>
<b>IEC 61131-3 COMPLIANCE LIST FOR PRO-H 1.0 KERNEL AND PLC-SIMULATION BASED ON PROCONOS 3.0 AND HIDIC PLC-ADAPTATION .....</b>	<b>A4-1</b>

**LIST OF FIGURES**

**INDEX**



# Introduction

**This chapter provides information about...**

- Pro-H
- the documentation for Pro-H
- conventions used in this manual

# Introduction

---

## Pro-H - a highly efficient, powerful and complete tool

Pro-H is a standard programming system for IEC designed PLCs and traditional PLCs. It is based on the standard IEC 61131-3 and includes the full range of IEC features.

Pro-H offers powerful features for the different developing phases of a PLC application:

- \* Edit
- \* Compile
- \* Debug
- \* Print

The Pro-H system is based on a modern 32 bit windows technology, providing comfortable handling using zooming, scrolling, customizable toolbars, drag & drop operations, a shortcut manager and dockable windows.

Pro-H allows especially handling of several configurations and resources within one project, including libraries and disposes of a powerful debug system. Projects are displayed and can be edited using a comfortable project tree editor in order to make the complexity of the IEC 61131-3 structure as simple and transparent as possible. Owing to the project tree editor easy inserting and editing of POU's, Data Types, Libraries and configuration elements is possible.

Pro-H consists of a PLC independent kernel for programming in the various IEC programming languages, such as the textual languages ST and IL as well as the graphical languages FBD, LD and SFC. Each editor provides an Edit Wizard for fast and easy inserting pre-edited keywords, statements, operators, functions and function blocks. The Edit Wizard can also be used to declare variables and data types.

The independent kernel is completed with specific parts adapted to the different PLCs.

The new easy Online handling and the 32 bit simulation offers fast powerflow debug functionality and a real time multitasking test environment.

A comfortable tool for project documentation is implemented for printing the project documentation alternatively in a time-saving optimized way (using less paper) or with a stylish customized page layout.

---

## What kind of documentation do you get?

The documentation is divided into several parts. For an understanding of all parts we are assuming knowledge about using MS-Windows.

The program manual provides all background information for a better understanding of the concepts of the PLC programming system and of the operations to be done. All steps from starting the program, editing worksheets up to exiting are described with several examples and figures. The manual should be used by users wishing to get a complete overview about how to realize a PLC program.

The context-sensitive Help which can be called by pressing F1 provides detailed and reference information for all program parts. The context-sensitive Help consists of two or more parts. A general part describes the general, non-specific parts. The specific part describes all objects, dialogs and operations which differ from PLC to PLC. It should be used by experienced users having a concrete problem and searching for detailed information. Context-sensitive Help is also available for functions and function blocks, which can be inserted using the Edit Wizard.



Please refer also to your hardware documentation for PLC specific information.

---

## Symbols and textual conventions

The following symbols are used in this manual:

- \* is used for enumeration.
- is used for an operation which has to be done.
- is used for an operation which is optional.



is used for a sequence of operations to be done with the mouse.

In the procedures described in this manual the instructions 'click' and 'double click' relate to the left mouse button. If the right mouse button is meant (e.g. to open an object context menu) this is explicitly mentioned.



is used for a sequence of operations to be done with the keyboard.



Notes are used to provide important information.



The book symbol is used to introduce references to other documents or chapters of this manual.

The following textual conventions have been set up for this manual:

- ' commas are used for names of icons, menu items or proper names of objects e.g. menu item 'Cut'; function block 'Level'.
- <ALT> brackets are used for the name of keys on your keyboard and for words you have to enter.
- <ALT> + <F4> is used if you have to press two keys at the same time.
- editor name* Italic letters are used as place holders for proper names.

# Pro-H and IEC 61131-3

**This chapter provides information about...**

- purpose and contents of IEC 61131-3
- configuration elements
- POUs, programs, function blocks and functions
- variables and data types
- projects
- libraries
- programming languages and SFC

# Pro-H and IEC 61131-3

---

## What is IEC 61131-3?

The standard IEC 61131 has been established to standardize the multiple languages, sets of instructions and different concepts existing in the field of automation systems. The great variety of PLC concepts has led to an incompatibility between the different PLC platforms and manufacturers. The result was a great effort to be made for training, hard- and software investments.

IEC 61131 standardizes the programming languages, the interfaces between PLC and programming system, the sets of instructions and the handling and structuring of projects. The advantage of using IEC 61131 conform PLCs and programming systems is a portability of all platforms and the use of same concepts reducing costs for automation systems.

The standard consists of several parts and technical reports. The third part of the standard is dedicated to programming languages.

Obviously this standard has a great influence on the concept, structure, features and the handling of a programming system such as Pro-H and the way to program the system.

The main changes that have come with IEC 61131-3 are:

- \* Declaration of variables is similar to the variable declaration in higher programming languages.
- \* Declaration of data types is possible.
- \* Global and local data can be differentiated.
- \* Programming means symbolic programming.

For a better understanding of Pro-H and an easier programming some IEC basics and their realization in Pro-H are described in the following sections.

---

# Configuration elements

An IEC 61131-3 conform PLC programming system reflects the hardware structure with the configuration elements.

These configuration elements are basically:

- \* Configurations
- \* Resources
- \* Tasks

## Configurations

A configuration can be compared to a programmable controller system, e.g. a rack. In a configuration one or several resources can be defined.

## Resources

A resource can be compared to a CPU which can be inserted in the rack. In a resource global variables can be declared, which are only valid within this resource. In a resource one or several tasks can be executed.

## Tasks

Tasks determine the time scheduling of the programs associated with them. This means that programs have to be associated to tasks. The settings of the task determine the time scheduling.

IEC 61131-3 describes different time scheduling which lead to three different task types:

- \* **Cyclic tasks** are activated in a certain time interval and the program is executed periodically.
- \* **Error tasks** will be activated if an error occurs in a different task.
- \* **Event or interrupt tasks** are activated if a certain event has happened.

Each task has a certain priority. In so called preemptive scheduling systems, an active task with low priority is interrupted immediately, when a task with higher priority becomes active due to a certain event. In systems with non-preemptive scheduling, task interruptions by tasks with higher priority are not possible.



The supported task types depend on the used PLC.

## Configurations elements

Configuration elements are represented graphically in the project tree. They are grouped together in the subtree 'Physical Hardware'.

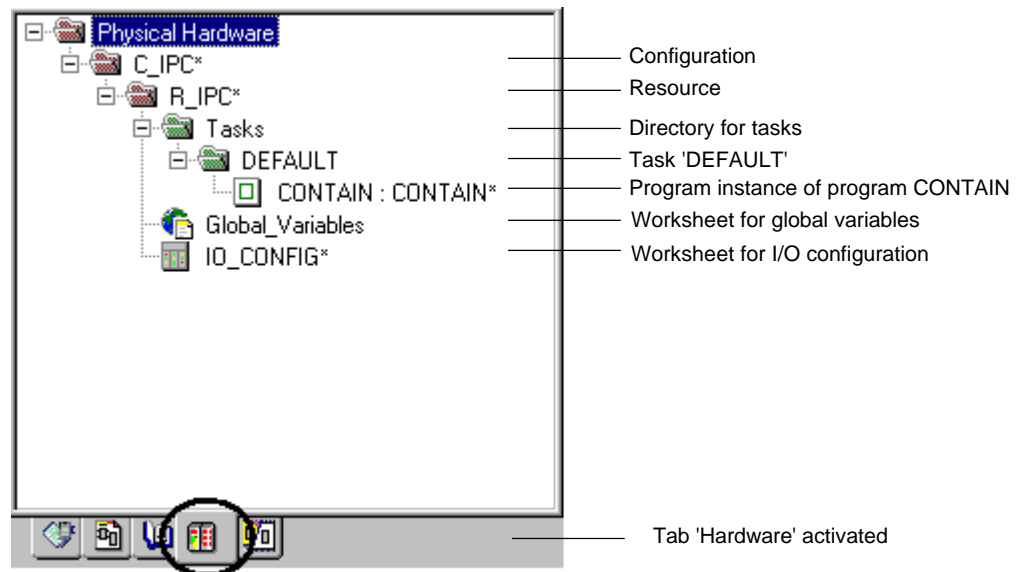


Figure 2-1: An example of configuration elements

The programming system reflects the structure of configuration elements in the subtree 'Physical Hardware' which may differ from PLC to PLC. In figure 2-1 the subtree 'Physical Hardware' with the configuration elements for ProConOS is shown.

In general one or several configurations can be used. In every configuration one or several resources can be declared. Several tasks with their associated programs can be used within one resource.



---

# POUs, programs, function blocks and functions

## Program organization units - POUs

Program organization units or POUs are the language elements of a PLC program. They are small, independent software units containing the program code. The name of a POU should be unique within the project.

In IEC 61131-3 three types of POUs are distinguished referring to their different use:

- \* Functions
- \* Function blocks
- \* Programs

## Functions

Functions are POUs with multiple input parameters and exactly one output parameter. Calling a function with the same values returns always the same result. Return values can be single data types. Within a function it is possible to call another function but not a function block or a program. Recursive calls are not allowed.

IEC 61131-3 lists different types of standard functions:

- \* Type conversion functions, such as INT\_TO\_REAL
- \* Numerical functions, such as ABS and LOG
- \* Standard arithmetic functions, such as ADD and MUL
- \* Bit-string functions, such as AND and SHL
- \* Selection and comparison functions, such as SEL and GE
- \* Character string functions, such as RIGHT and INSERT
- \* Functions of time data types, such as SUB with the data type TIME

## Function blocks

Function blocks are POUs with multiple input/output parameters and internal memory. The value returned by a function block depends on the value of its internal memory. Within a function block it is possible to call another function block or functions. Recursive calls are not allowed.

IEC 61131-3 lists different types of standard function blocks:

- \* Bistable elements, such as SR and RS
- \* Edge detection function blocks, such as R\_TRIG and F\_TRIG
- \* Counters, such as CTU and CTD
- \* Timer function blocks, such as TON and TOF

**Programs**

Programs are POU's which contain a logical combination of functions and function blocks according to the needs of the controller process. The behavior and the use of programs are similar to function blocks. Programs can have an internal memory. Programs must be associated to tasks.

Within a program it is possible to call functions and function blocks. Recursive calls are not allowed.

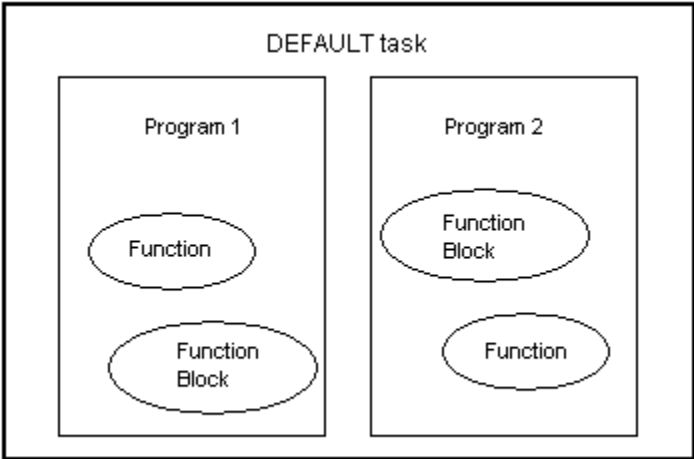


Figure 2-2: Diagram of a default task with two programs

## POUs

Programs, function blocks and functions can be edited in the project tree. You can either display the complete project tree or only the subtree 'Data Types' and 'Logical POUs' by clicking on the tab 'POUs' as shown in the following figure.

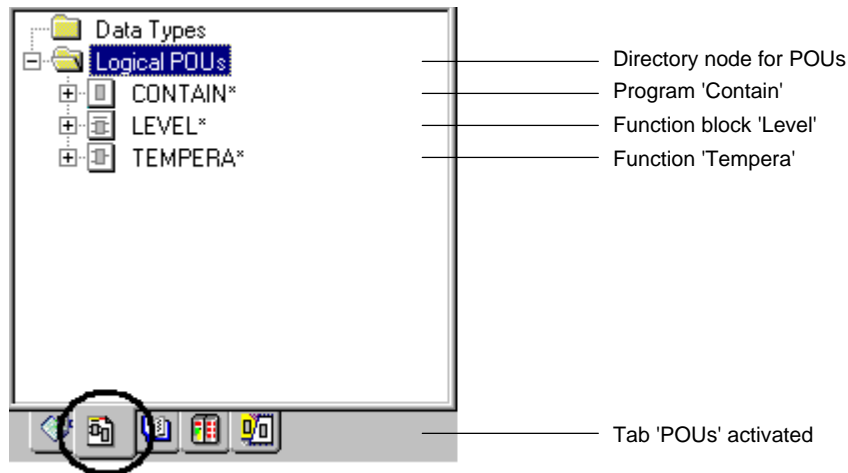


Figure 2-3: Subtree 'Logical POUs'

## Instantiation

For reusing function block definitions IEC 61131-3 provides the possibility of instantiation. This means that the function block code body is defined once and that its internal memory is allocated to different instances, different memory regions. Each instance has an associated identifier (called instance name) and contains the input and output parameter and the internal memory of the function block. A function block can be instantiated in another function block or in a program. The instance name of a function block has to be declared in the VAR declaration of the program or function block where it is going to be used. Programs can be instantiated within resources.

Instances are also displayed in the project tree window. The related subtree is made visible by clicking with the left mouse button on the tab 'Instances' as shown in the following figure.

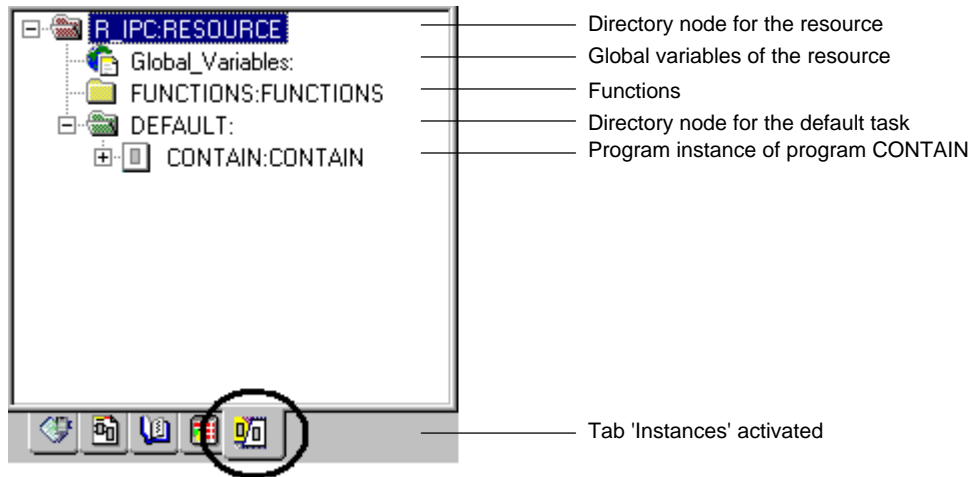


Figure 2-4: Project tree with the instances within the resource 'R\_IPC'

## Declaration and instruction part of a POU

Every POU consists of two different parts: The declaration part and the code body part.

In the **declaration part** all necessary variables are declared.

The **instruction or code body part** of a POU is the part in which the instructions are programmed in the desired programming language.

A POU consists of three types of worksheets. These three worksheets are represented graphically by icons:

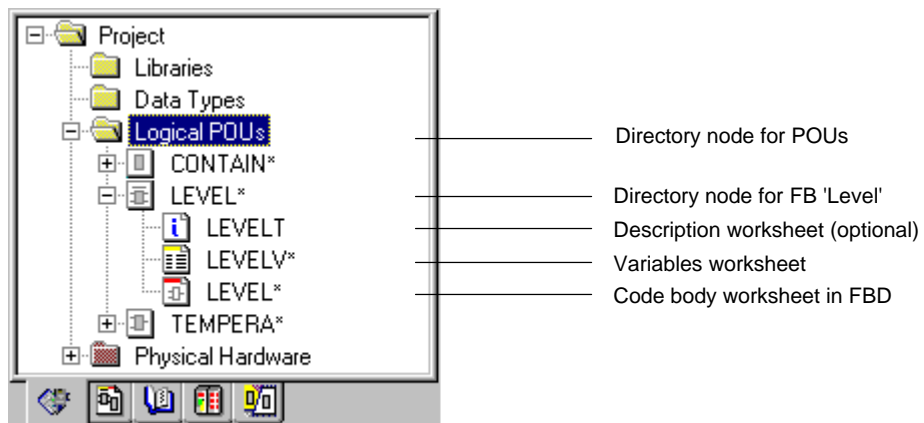


Figure 2-5: Worksheets of a function block in FBD

In the description worksheet annotations can be added for documentation purposes. In the variable worksheet all variables are going to be edited. The code body worksheet contains the instructions.

In the case of a SFC POU you have two more icons: the directory nodes for the action and transition worksheets.

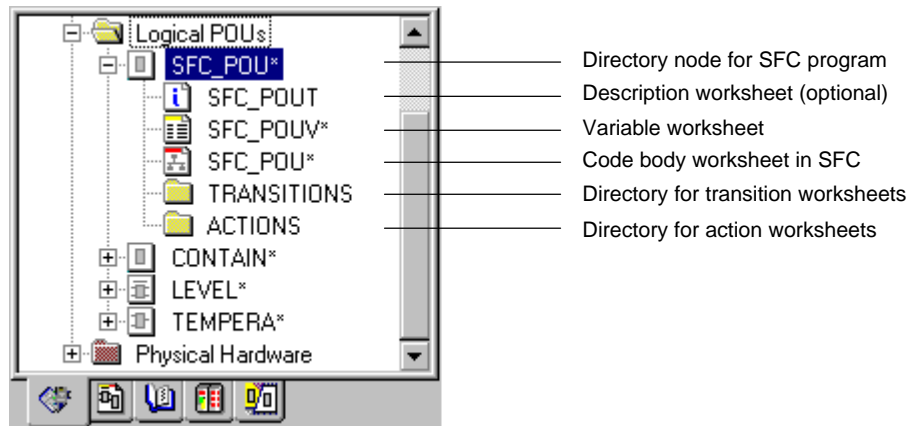


Figure 2-6: Icons of a SFC POU

## Variables and data types

In IEC 61131-3 programming systems, **variables** are used instead of direct addressing of memory regions in former systems. Variables are assigned automatically to a memory region while compiling. IEC 61131-3 distinguishes different types of variable declarations e.g. VAR or VAR\_INPUT. For PLC inputs and outputs direct addressing is possible using the keyword AT.

Variables with their properties are declared in the variable worksheet of the POU.

**Data types** determine what kind of value the variable can have. Data types define the initial value, range of possible values and the number of bits.

IEC 61131-3 distinguishes three kinds of data types:

- \* Elementary data types
- \* Generic data types
- \* User defined data types

Elementary data types are data types whose range of possible values and number of bits is defined in IEC 61131-3. Elementary data types are e.g. BYTE, WORD or BOOL.

Generic data types are data types which include groups of elementary data types. They are called e.g. ANY\_BIT or ANY\_INT. ANY\_INT includes e.g. the elementary data types INT, SINT, DINT, UINT, USINT and UDINT. Generic data types are necessary to define what kind of elementary data types can be connected to inputs or outputs of functions. If a function can be connected with ANY\_INT it means that variables of the data types INT, SINT, DINT, UINT, USINT and UDINT can be connected.

User defined data types are data types which can be declared by the user. They have to be defined with a TYPE ... END\_TYPE declaration. User defined data types can be structures or arrays.

User defined data types are declared in the data type worksheet in the subtree 'Data Types'. You can either display the complete project tree or only the subtree 'Data Types' and 'Logical POU's' by clicking on the tab 'POUs' as shown in the following figure.

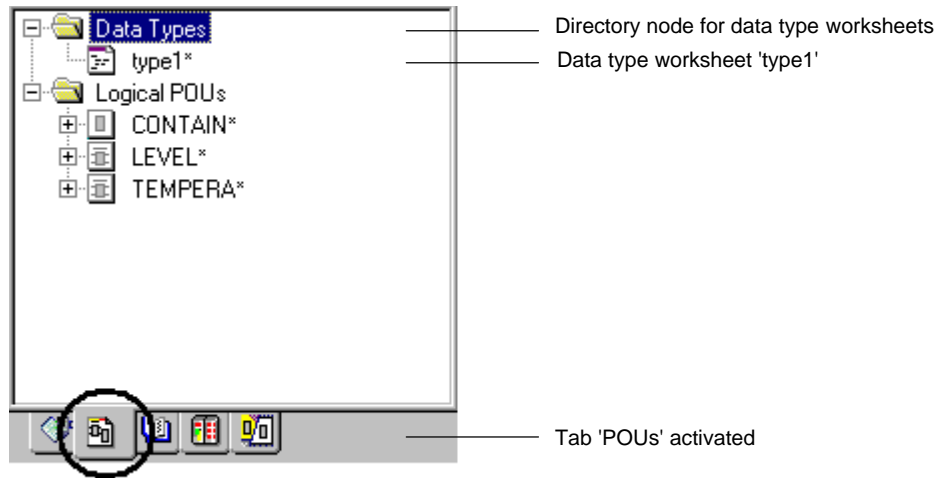


Figure 2-7: Subtree 'Data Types'



Variables and user defined data types and their declarations are described in the chapter 'Declaring variables and user defined data types' of this manual.

---

## Projects

An IEC 61131-3 project contains all necessary elements of an automation system. It consists of libraries, data types, POUs and the set of configuration elements (such as resources, tasks and programs) which are represented in the subtree 'Physical Hardware'. A project is represented in the project tree.

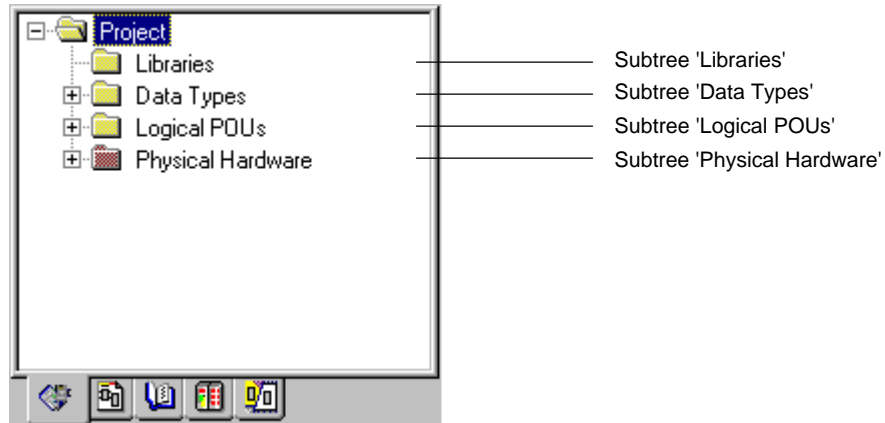


Figure 2-8: The project with its subtrees

---

## Libraries

Libraries are projects which have been announced as libraries. You can reuse the programs, function blocks, functions and the user defined data types of the library in the project you are editing.

**Firmware libraries** are libraries containing POUs prepared by your PLC manufacturer. The file extension for firmware libraries is \*.fwl.

**User libraries** are other projects you have done before and whose POUs you want to reuse. The file extension for user libraries is \*.pwt.

Libraries have an own subtree in the project tree. You can either display the complete project tree or only the subtree 'Libraries' by clicking on the tab 'Libraries' as shown in the following figure.

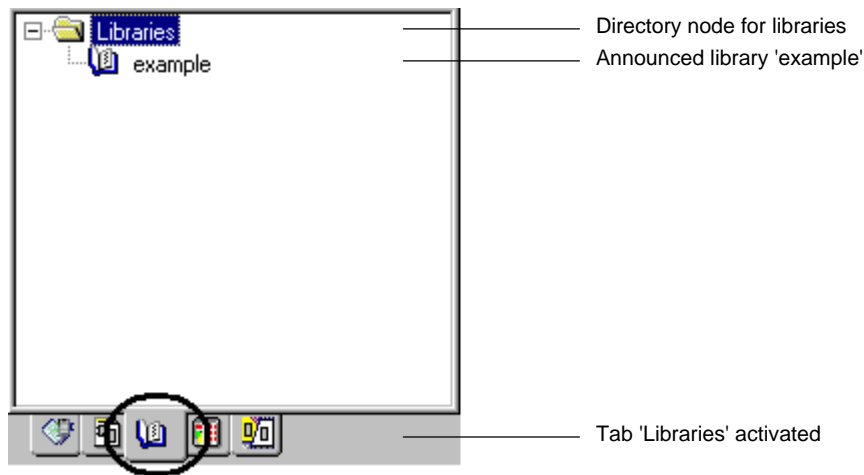


Figure 2-9: Subtree 'Library'

The subtree 'Library' consists of two or more icons. The first icon is a directory node. The icons within this directory node represent the announced libraries. In figure 2-9 you can see the announced user library 'example'.

---

## Programming languages and SFC

IEC 61131-3 defines the syntax, the representation and the available language elements of 4 programming languages.

The programming languages can be differentiated into 2 textual languages and 2 graphical languages:

- \* The **textual languages** are Structured Text (ST) and Instruction List (IL).
- \* The **graphical languages** are Function Block Diagram (FBD) and Ladder Diagram (LD).

For structuring the internal organization of programs and function blocks SFC or Sequential Function Chart elements are defined in IEC 61131-3.



SFC and the 4 programming languages are described in the corresponding chapters of this manual.



# Getting started

**This chapter provides information about...**

- system requirements
- installing the program
- calling the program
- using mouse and keyboard
- the user interface
- using the help system
- editors
- the Edit Wizard
- overview window for graphical worksheets
- saving changes while editing
- exiting worksheets
- exiting the program

# Getting started

---

## System requirements

### Hardware requirements

To run the PLC programming system, the following workstation requirements must at least be fulfilled:

Device	Minimum	Recommended
IBM compatible PC with Pentium processor	133 MHz	200 MHz
System RAM	32 MB	64 MB
Hard disk	60 MB free memory space	
CD ROM drive or floppy disk drive		
VGA Monitor Color settings Resolution	256 colors 800 x 600	True color 1024 x 768
RS232 interface	optional	
Mouse	recommended	

### Software requirements

To run the PLC programming system, the following software requirements must at least be fulfilled:

- Microsoft Windows® 95, 98 or Microsoft Windows® NT
- DCOM95 (Windows® 95 user only)
- The installation of Microsoft® smartdrive cache is recommended

---

## Installing the program

Using the installation program you can perform all necessary steps which are required to install the software. In order to start the installation program you have to perform the following steps:

- Insert the CD ROM disk into your CD ROM drive.
- Open the Windows 'Start' menu and choose 'Run'. The 'Run' dialog appears.
- Type `d:\setup.exe` (where `d` is the appropriate CD ROM drive indicator) and press `<↵>`.
- The installation includes several dialogs where you have to do the corresponding entries.

After a successful installation, you will find the program group in the Windows® program menu.



Pro-H requires DCOM95 and may notify you that DCOM95 does not exist on your PC when starting up. This may happen if your operating system is Windows®95. In this case, put the Pro-H CD-ROM into your CD-ROM drive again, and just execute 'DCOM95.exe' on the CD-ROM.

---

## Calling the program

To call the program open the Windows 'Start' menu, choose the menu 'Program' and select 'Pro-H'.

The program will be opened with the last project you have used. If you start the program for the first time it will be opened without any project.

To open an existing project, you have to perform the steps, which are described in the example of the following section 'Using mouse and keyboard'.

---

## Using mouse and keyboard

The program supports full use of the mouse or the keyboard. For beginners it may be easier to start working with the mouse because it does not make necessary to learn the keyboard shortcuts. In rough industrial environments the keyboard may be more appropriate.

This manual explains both: the use of mouse and keyboard. In the next sections the general use of mouse and keyboard for the menu and toolbars is described.

The following is an example, how the usage of mouse and keyboard is described in this manual:



### Opening an existing project using the mouse

- Click on the icon 'Open Project / Unzip Project' in the toolbar. The dialog 'Open project' appears.
- In the list box 'File type' select the desired file type.
- Browse to the corresponding project folder and locate the desired files (.mwt file for project file or .zwt file for zipped project files).
- Double click on the desired file name. The corresponding project is opened. If you choose a zipped project, the unzipping process is started automatically.



### Opening an existing project using keyboard shortcuts

- Press <CTRL> + <O>. The dialog 'Open project' appears.
- In the list box 'File type' select the desired file type.
- Browse to the corresponding project folder by typing the folder names into the field 'file name' and pressing <↓>. Locate the desired files (.mwt file for project file or .zwt file for zipped project files).
- Open the desired project by typing the corresponding file name and pressing <↓>.

The usage of mouse and keyboard in the different editors is described in the following chapters.

# User interface

The program user interface consists basically of six parts: Menu, toolbars, main screen, status bar, message window and cross reference window.

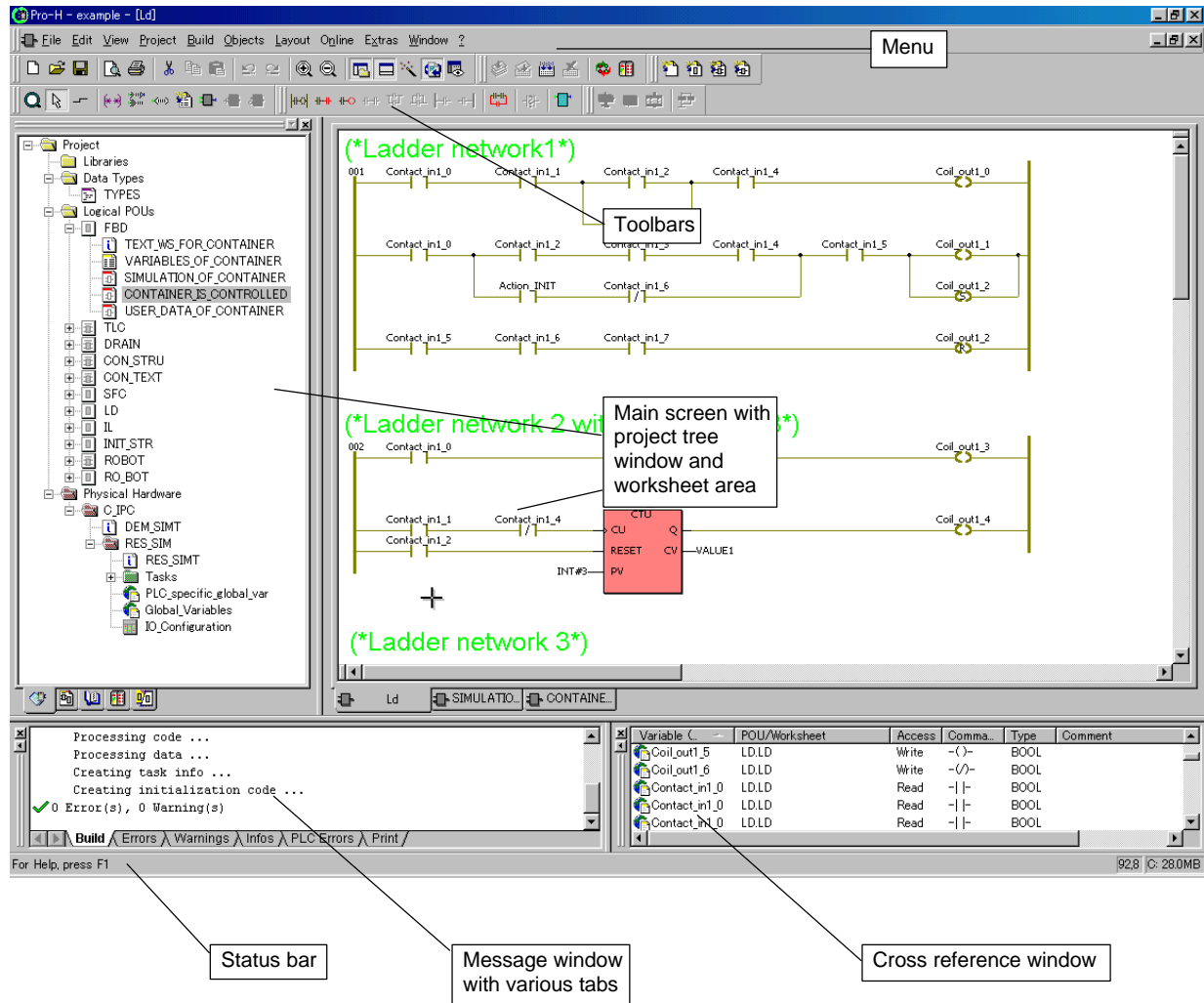


Figure 3-1: Program user interface with sample project 'example'

## Menu

The menu is located below the title bar. It contains several submenus.



The menu items of these submenus change according to the program part or editor you are working with.

- \* The submenu 'File' can be used to handle, save and zip/unzip projects. It also contains commands for printing, print setup and print preview.
- \* The submenu 'Edit' contains all commands which are necessary for editing such as marking, choosing different working modes or cutting and pasting. Additionally it provides functionality for searching and replacing text strings used to edit textual worksheets (e. g. description worksheets, variable worksheets or structured text worksheets).
- \* The submenu 'View' can be used to hide or show the different windows of the user interface (project tree window, message window, cross reference window, watch window, Edit Wizard) and the status bar.
- \* The submenu 'Project' can be used to insert new objects (such as data type worksheets and announced libraries), POUs and configurations.
- \* The submenu 'Build' consists of different commands for starting the compilation after editing.
- \* The submenu 'Objects' is available if you are using an editor. The menu item 'Variable' can be used to insert a new variable into the variable list of the current POU. When editing a graphical worksheet, the submenu provides additional menu items to insert and edit graphical objects, such as connectors, jumps, contacts, coils, etc. Depending on the graphical language you are using, some items may be grayed (i.e. inactive).
- \* The submenu 'Layout' is available if you are using the graphic editor. It contains several designing utilities. You can display e.g. page borders or a grid for better organizing the content of your worksheets. Furthermore you can zoom into and out of the worksheet, modify the worksheet size, the autoscroll speed and the object size. This submenu also disposes of some features for the online layout.
- \* The submenu 'Online' offers you commands for debugging a project, calling the Resource Control and activating the powerflow. In addition the command 'Online Layout' allows to set the appearance of graphical worksheets in online mode.
- \* The submenu 'Extras' can be used to call the dialogs 'Shortcut Keys' and 'Options' as well as other optional tools, such as the pagelayout editor. The dialog 'Shortcut Keys' (also known as Shortcut Manager) allows you to define your own keyboard shortcuts or customize the default shortcuts. The dialog 'Options' provides the facility to customize the menus, toolbars, text editors and text colors.
- \* The submenu 'Window' can be used to arrange the windows and symbols on your screen and to close all open windows in one step.
- \* The submenu 'Help' contains all commands for calling help.

The following procedures illustrate, how to call a menu item using the mouse and the keyboard.



### Calling the menu item 'New Project...' with the mouse

- Click on the submenu 'File'. The submenu is opened and you can see the menu items.
- Select the menu item 'New Project...' with a left mouse click. The dialog 'New Project' appears.



### Calling the menu item 'New Project...' using the keyboard

- Press <ALT> + <F>. The submenu is opened and you can see the menu items.
- Press <w> as it is the underlined character of the menu item 'NewProject...'. The dialog 'New Project' appears.



All submenus or menu items and dialog fields and boxes can be called pressing the underlined character of the corresponding word.

Using shortcuts is the easiest way of calling a menu item with the keyboard. For that reason the above mentioned method how to open a submenu and to choose an item is described only one time in this manual. In the following procedures the usage of shortcuts is described.



### Calling the menu item 'New Project...' using the keyboard shortcut

- Press <CTRL> + <N>. This is the default shortcut for creating a new project. The dialog 'New Project' appears.



Default shortcuts are already associated to the most important menu items. If not, you can open the dialog 'Shortcut Keys' (Shortcut Manager) and assign the corresponding menu item to your own shortcut.

## Toolbars

The toolbars are located below the menu bar. By default all available toolbars are visible, providing two lines of different icons.

If you place the mouse cursor on any icon (without clicking it), a short description text, the so called tooltip appears. These tooltips display the name of the current icon. Additionally the status bar displays the function of the desired icon. The following figure shows an example:

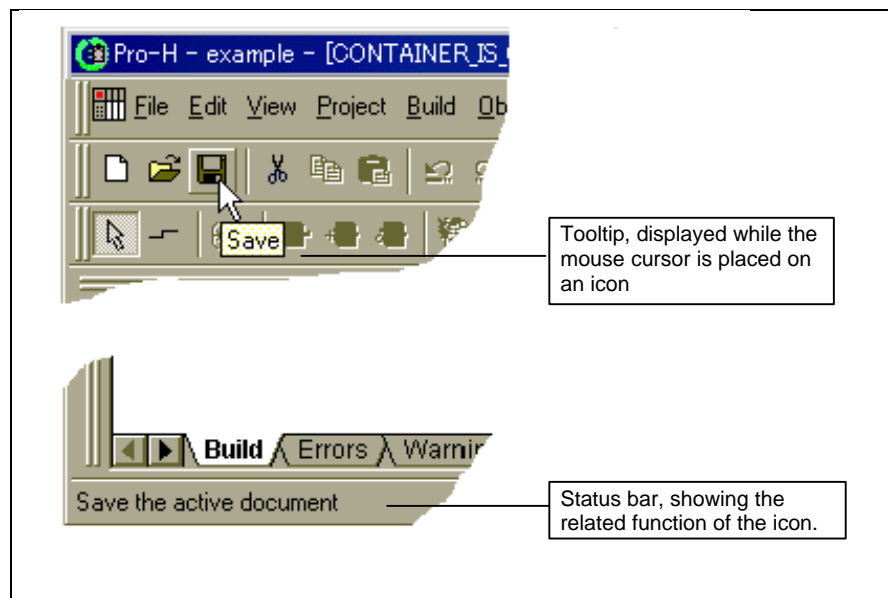


Figure 3-2: Sample tooltip (icon 'Save') and the corresponding description displayed in the status bar



If the tooltips are not shown, open the dialog 'Options' by selecting the menu item 'Options' in the submenu 'Extras'. Activate the checkbox 'Show Tooltips' on the page 'Toolbars' and press 'OK' to confirm the dialog.

The toolbars have been implemented for realizing quickly often used operations with the mouse. In those cases one mouse click on a toolbar icon leads to the same result as doing several steps without the toolbar. You can adapt the toolbars corresponding to your needs. For this purpose you have to call the dialog 'Options' by selecting the menu item 'Options' in the submenu 'Extras'. In order to hide one certain toolbar, deselect the corresponding checkbox on the page 'Toolbars'.

Two different toolbar parts can be distinguished: General toolbars and specific toolbars.

- \* General toolbars contain icons which are available everywhere in the program.
- \* Specific toolbars contain icons which can be used only in specific editors. All icons are visible but the icons which cannot be used in a specific editor are grayed. You can hide each toolbar by using the dialog 'Options' as mentioned above.





You can detach every toolbar from the other toolbar by double clicking on the gray toolbar background. The toolbar is then displayed in a window, which can be resized and moved to any position on your screen. To reinsert the toolbar window, just double click on the blue toolbar window title bar.

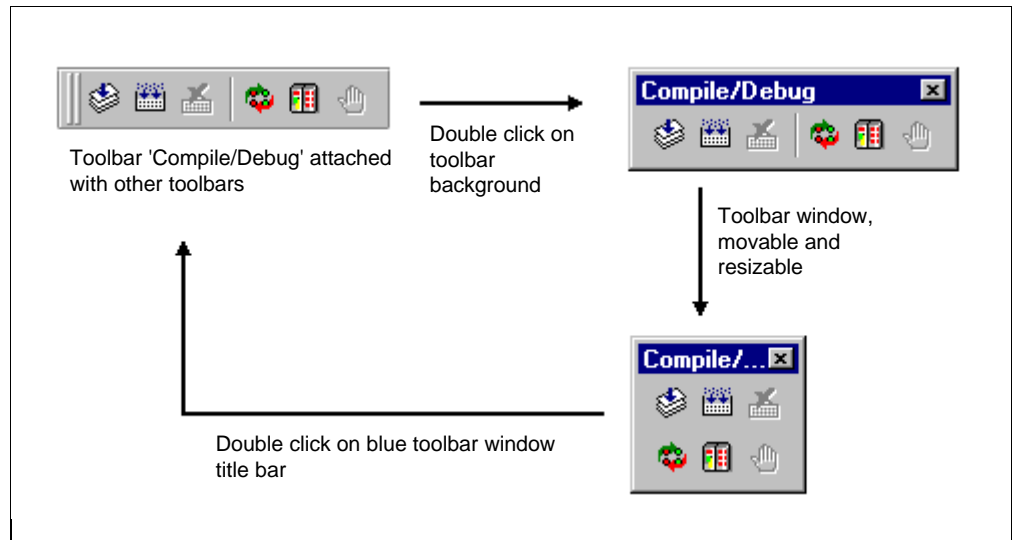


Figure 3-3: Example of a detached toolbar window



### Using the toolbar for creating a new project

- Click on the icon 'New Project'.  
The dialog 'New Project' appears, to select a template for the new project.



## Defining keyboard shortcuts with the Shortcut Manager

As already mentioned in the section 'Menu' in this chapter, you can select certain menu items easier and faster by using keyboard shortcuts. A keyboard shortcut performs the same operation as the menu item to which it is assigned by simply pressing only one key or a key combination. In your PLC programming system several keyboard shortcuts can be used. The most important menu items (i. e. operations) are already associated to shortcuts by default. Assigned shortcuts are shown beneath the corresponding menu item in the submenu.

Using the Shortcut Manager, you can add new shortcuts for a specific menu item or modify existing shortcuts. For this purpose call the Shortcut Manager as follows.



Many keyboard shortcuts are defined by default. Keep in mind, that this manual describes those default shortcuts. After modifying the default setting, some descriptions may not match your actual setting.



### Calling the Shortcut Manager for adding/modifying shortcuts

- Click on the submenu 'Extras' to see the menu items.
- Select the menu item 'Shortcuts'. The dialog 'Shortcut Keys' appears.

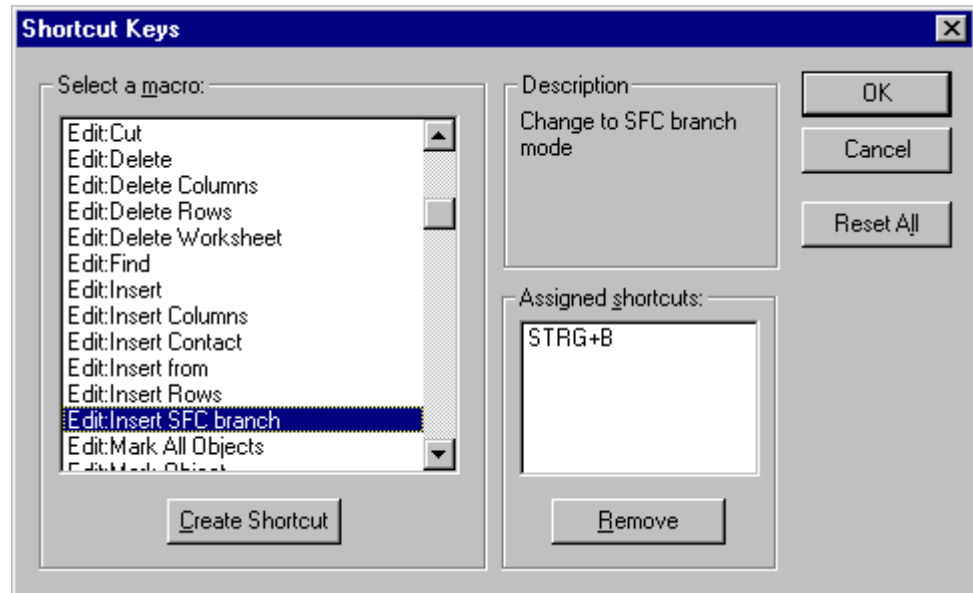


Figure 3-4: The 'Shortcut Manager' for adding/modifying keyboard shortcuts

The dialog shown in figure 3-4 is divided into three areas: The selection list 'Select a macro' contains a list of operations, which can be assigned to a shortcut. Most of these entries are available as menu items in the submenus. The field 'Description' displays a short text, describing the functionality of the selected macro (if available). If a shortcut is already assigned to the selected macro, this shortcut is displayed in the field 'Assigned shortcuts'.



## Adding/modifying a keyboard shortcut

- Click on the desired operation/menu item in the list 'Select a macro'. The entry is highlighted. If available a short description is shown in the field 'Description'.  
If a shortcut is already assigned, it is shown in the field 'Assigned shortcuts'.
- To **modify** a shortcut, you have to remove the current shortcut and assign a new shortcut (refer to the steps below).
- To **remove** a certain shortcut, select it with a left mouse click and press 'Remove'.
- To **add** a new shortcut, perform the steps described in the following procedure. In this example, it is assumed, that you want to assign an additional shortcut to the menu item 'Stop Compile' in the submenu 'Build'.
- To **reset all** shortcuts to their default settings, click on the button 'Reset All'. All shortcuts you have created will be overwritten.



## Assigning the shortcut <ALT> + <F6> to the macro 'Build:Stop Compile'

For the following description it is assumed, that the dialog 'Shortcut Keys' is already opened.

- Double click on the desired macro 'Build:Stop Compile' in the field 'Select a macro'. The dialog 'Assign Shortcut' appears.



Figure 3-5: Dialog 'Assign Shortcut' before pressing the desired shortcut keys

- Press the desired shortcut keys (in our example <ALT> and <F6>). They are shown in the text field. Below the text field the Shortcut Manager shows, whether the desired keys are already assigned to another macro.

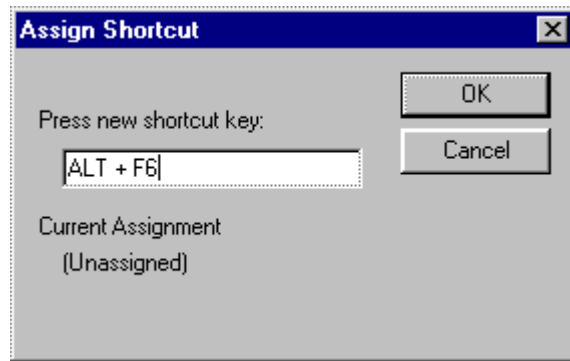


Figure 3-6: Dialog 'Assign Shortcut' after pressing the desired shortcut keys

- Confirm the new shortcut by clicking on 'OK'.



If the desired keys are already assigned to another macro and you confirm your current selection, the shortcut keys will be reassigned without a warning message.

The dialog 'Assign Shortcuts' is closed and the new assigned shortcut is shown in the dialog 'Shortcut Keys'.

The following list contains the default shortcuts in alphabetical order, assigned by the software manufacturer:

This operation/menu item is assigned to ...	... this default shortcut(s)
Activate: Cross References	<ALT> + <4>
Activate: Edit Wizard	<ALT> + <3>
Activate: Output	<ALT> + <2>
Activate: Watch Window	<ALT> + <5>
Activate: Workspace	<ALT> + <1>
Build: Compile Worksheet	<SHIFT> + <F9>
Build: Go to next Error/Tag	<CTRL> + <F12>
Build: Make	<F9>
Build: Patch POU	<ALT> + <F9>
Build: Rebuild Project	<CTRL> + <F9>
Edit: Connect Objects	<CTRL> + <L>
Edit: Copy	<CTRL> + <C> <CTRL> + <INS>
Edit: Cut	<SHIFT> + <DEL> <CTRL> + <X>
Edit: Delete	<DEL>
Edit: Find	<CTRL> + <F>
Edit: Insert	<INS>
Edit: Insert LD Branch	<CTRL> + <T>
Edit: Insert SFC branch	<CTRL> + <B>
Edit: Mark All Objects	<CTRL> + <A>

<b>This operation/menu item is assigned to ...</b>	<b>... this default shortcut(s)</b>
Edit: Mark Object	<CTRL> + <M>
Edit: Paste	<CTRL> + <V> <SHIFT> + <INS>
Edit: Replace	<CTRL> + <H>
Edit: Undo	<CTRL> + <Z>
File: New Project	<CTRL> + <N>
File: Open Project / Unzip Project	<CTRL> + <O>
File: Print	<CTRL> + <P>
File: Save	<CTRL> + <S>
Help Contents	<SHIFT> + <F1>
Help, context sensitive	<F1>
Layout: Default Size	<CTRL> + <#>
Layout: Overview window	<ALT> + <6>
Layout: Previous View	<CTRL> + <<>
Layout: Zoom in	<CTRL> + <+>
Layout: Zoom out	<CTRL> + <->
Objects: Action block	<ALT> + <F8>
Objects: Contact below	<CTRL> + <F7>
Objects: Contact left	<ALT> + <F7>
Objects: Contact network	<F6>
Objects: Contact right	<F7>
Objects: Duplicate FP	<CTRL> + <F5>
Objects: Function/Function block	<ALT> + <F5>
Objects: Simultaneous / Alternative Branches (divergence)	<CTRL> + <F8>
Objects: Step/Transition	<F8>
Objects: Toggle contact/coil properties	<SHIFT> + <F7>
Objects: Toggle FP negation	<SHIFT> + <F5>
Objects: Variable	<F5>
Online: Debug	<F10>
Online: Resource Control	<CTRL> + <F10>
View: Cross References Window	<ALT> + <F2>
View: Edit Wizard	<SHIFT> + <F2>
View: Message Window	<CTRL> + <F2>
View: Project Tree Window	<F2>

Figure 3-7: Default shortcuts in alphabetical order

## Main screen and workspace

The main screen (see figure 3-1) is divided into two parts: The project tree window and the workspace. The workspace contains the opened worksheets. You can open a worksheet by double clicking on the corresponding worksheet icon in the project tree.

If several worksheets are opened only one worksheet is visible. A sheet tab is assigned to every opened worksheet as shown in the following figure. You can select (activate) a particular worksheet by clicking on the corresponding tab or by pressing <CTRL> + <TAB>.

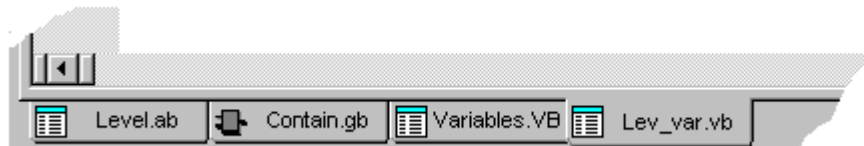


Figure 3-8: Worksheet tabs in the workspace



If the worksheet tabs are not visible, open the dialog 'Options' by selecting the menu item 'Options' in the submenu 'Extras'. Click on the tab 'General' and activate the checkbox 'Workbook style'. Confirm your selection with 'OK'.

It is also possible to arrange several worksheet windows in a desired combination. For this purpose choose the menu items 'Cascade' and 'Tile' in the submenu 'Window'. This way you can easily prepare the workspace for different working phases.

You can maximize the workspace, which means, that the project tree is not displayed and the workspace is enlarged to the whole main screen width. This could be useful for displaying large networks in graphical editors.



### Displaying/Hiding the project tree window with the mouse

- Click on the icon 'Project Tree'. Depending on the previous state, the project tree window is now visible or hidden.



### Displaying/Hiding the project tree window with the keyboard

- Press <F2>. Depending on the previous state, the project tree window is now visible or hidden.

## Message window

The message window is located below the main screen. It contains several pages, which are activated by clicking on the corresponding tab.

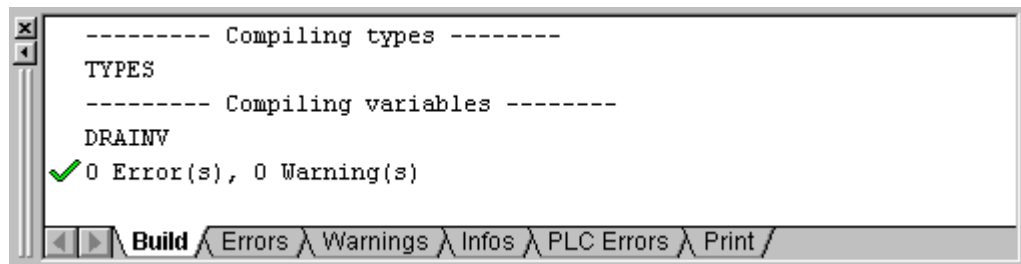


Figure 3-9: Message window with its different page tabs

The message window allows to display different steps during compiling, the compiler errors and warnings and some other information.

One of the main benefits of the message window is the possibility of direct accessing the worksheets in which the compiler detected errors. For this purpose you just have to double click on the corresponding error.

You can toggle between hidden and visible message window.



### Displaying/hiding the message window with the mouse

- Click on the icon 'Messages'. Depending on the previous state, the message window is now visible or hidden.



### Displaying/hiding the message window with the keyboard

- Press <CTRL> + <F2>. Depending on the previous state, the message window is now visible or hidden.

The message window is a dockable window. This means, that you can detach the window from the desktop by double clicking on the gray window border. It is then displayed as a usual window, i.e. you can change the size and move the window to any position on the screen. To reattach it into the desk, just double click into the blue window title bar. The handling is similar as for the toolbars.

In order to enlarge the message window, it is possible to float the window into the main window, i. e. the workspace. The result of this operation is, that the message window is displayed in the workspace like a worksheet, having a sheet tab at it's border.



### Floating the message window in the workspace

- Place the mouse cursor into the message window.
- Click with the right mouse button on the window background to open the context menu.
- Select the menu item 'Float In Main Window'.  
The message window is opened in the same way as a worksheet.
- To reverse this operation, open again the context menu and deselect the menu item 'Float In Main Window'.

## Cross reference window

The cross reference list contains all external variables, local variables and function blocks, which are used within the current project. It is a helpful tool for debugging and fault isolation.



Every POU contains its own local data. That means if you open a particular worksheet, the local variables in the cross reference list are updated.

To use the cross reference list in an existing or a new project, you have to build the cross references in order to display the required information in the cross reference window.

In addition the program allows to toggle between hidden and visible cross reference window.



### Building the cross reference list with the mouse

- If the cross reference window is not visible, click on the icon 'Cross References'.  
The cross reference window is now visible.
- Place the mouse cursor in the cross reference window.
- Click with the right mouse button on the window background to open the context menu.
- Select the menu item 'Build Cross References'.  
The cross reference list is created automatically as shown in the following figure.



It is also possible to call the menu item 'Build Cross References' in the submenu 'Build' or in the editor context menu.





## Displaying/hiding the cross reference window with the keyboard

- Press <ALT> + <F2>. Depending on the previous state, the cross reference window is now visible or hidden.

Variable (LEF)	POU/Worksheet	Access	Command	I/O Address	Type	Comment	Line/Position(X/Y)
%IX0.2	POE\NL\LAB	Read	LD	%IX0.2			1
%IX0.3	POE\NL\LAB	Read	AND	%IX0.3			2
Action_init	POE\FBD\SIMULATIO...	Read			BOOL	Action I...	29/10
Action_INIT	POE\NL\LAB	Read	OR		BOOL	Action I...	3
Action_INIT	POE\NL\LAB	Write	ST		BOOL	Action I...	16

Figure 3-10: Cross reference window

For each entry the following information are available:

- \* 'POU/Worksheet': POU name in which the variable/FB is declared and the particular worksheet name where it is implemented.
- \* 'Access': The access of the variable is 'Read' (Load) or 'Write' (Stored).
- \* 'Command': Command in which the variable is used (only applicable for IL and LD code body worksheets).
- \* 'I/O Address': Physical PLC address.
- \* 'Type': The associated data type.
- \* 'Comment': User-defined comment.
- \* 'Line/Position(X/Y)': Line number in textual worksheets or element position in graphical worksheets.



To open the corresponding worksheet in which a particular variable is used, just double click on the required line in the cross reference window. The worksheet is opened automatically and the variable is marked. Furthermore a variable is marked in the cross reference window, if you select the variable in an editor.

You can filter the cross reference list, to show only a particular subset of variables or function blocks.



## Filtering the cross reference list

- Move the mouse cursor into the cross reference window.
- Click with the right mouse button on the window background to open the context menu.
- Choose the menu item 'Filter...'.  
The dialog 'Cross Reference Filter' appears.

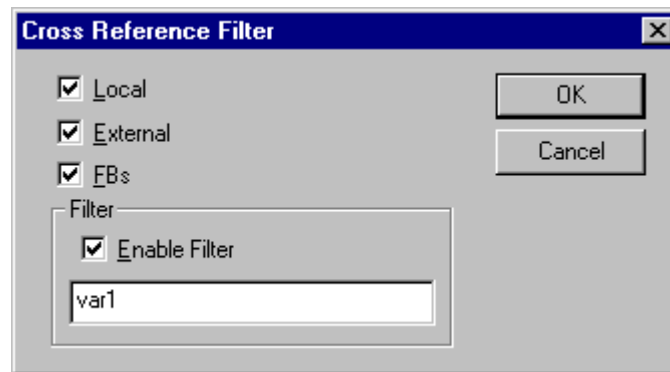


Figure 3-11: Dialog 'Cross Reference Filter', used to filter the displayed entries in the cross reference list

- Choose the desired filter settings by selecting one or several checkboxes. The filtered elements will be shown in the cross reference list and the column 'Variable' will display the current filter settings in the column head. The checkboxes 'Local', 'External' and 'FBs' are activated by default.  
To filter the list for a specific variable or FB name, mark the checkbox 'Enable Filter' and type the name into the text field. Only elements matching this name will be displayed.
- Using **wildcards** in the dialog 'Cross Reference Filter':  
In the text field you can use the wildcard symbol '\*'. A wild card serves as a placeholder for any character and is used in conjunction with other characters.

Example: Let us assume, that you want to filter the cross reference list in a way, that it displays the variables var1, var2 up to var10. For this purpose you can use the wildcard in the following way:

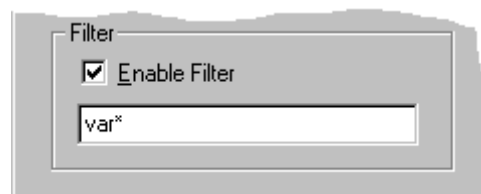


Figure 3-12: Using the wildcard symbol in the dialog 'Cross Reference Filter'

It is possible to sort the list entries alphabetically. Each column can be used as sort criterion. The entries can be sorted in an ascending and descending order. To sort the list, perform the following steps:



## Sorting the cross reference list

- Click on the column name, which is intended to be the sort criterion. It is marked by an arrow, which indicates the sort order:

Variable ▲

Variable ▼

Ascending or descending sort order.

If you have specified a filter string this button would appear as follows:

Variable (LEF var1) ▲

This figure corresponds to the described example where the checkboxes 'Local', 'External' and 'FBs' were activated and the string 'var1' was used.

- Click once more on the same column name to reverse the sort order.

Like the output window the cross reference window is dockable. This means, that you can detach the window from the desktop by double clicking in the gray window border. It is then displayed as a usual window, i.e. you can change the size and move the window to any position on the screen. To reattach it into the desk, just double click into the blue window title bar. The handling is similar as for the toolbars.



## Floating the cross reference window in the workspace

- Place the mouse cursor into the cross reference window.
- Click with the right mouse button on the window background to open the context menu.
- Select the menu item 'Float In Main Window'.  
The cross reference window is opened in the same way as a worksheet.
- To reverse this operation, open again the context menu and deselect the menu item 'Float In Main Window'.

## Status bar

The status bar displays different messages while you are working with the program.

The left part of the status bar provides information about operations you have done or displays system messages. If your mouse pointer is placed on an icon or a menu item (without selecting it) the status bar displays a short description concerning the icon or menu item under the cursor.

The fields on the right show the cursor position in the editor: When using a graphical editor, x-y-coordinates are shown, in text editors the fields display the current row and column. Beneath the cursor position the free hard disk space is displayed. If there is not enough disk space available, this field appears red.

The different states of the programming system are represented in the status bar by colors. The following colors are used for the different states:

- \* gray: offline
- \* green: online
- \* red: timeout.

It is possible to hide the status bar by selecting the menu item 'Status Bar' in the submenu 'View'.

---

## Using help

The context-sensitive help system provides topics for each of the program parts. Help is separated into two or more parts: A general help containing general help topics and one or more specific help files with PLC specific topics.

Both, the general and the specific help are arranged in a hierarchical structure using three types of topics:

- \* **Main topics:** These help topics describe the general handling of editors. Main topics are called pressing F1 while an editor is opened.
- \* **Object topics.** These help topics give background information about an object that has been used in an editor. Object topics are called by marking the object in the worksheet and pressing <SHIFT> + <F1>.
- \* **Dialog topics:** These help topics explain the meaning and the usage of the dialog fields. Dialog topics are called by clicking the button 'Help' in the dialog.

All these topics can be called context-sensitive. But it is also possible to call the help contents or help index for getting an overview and then choosing the topics to read. For this purpose you have to select the menu item 'Help Topics' in the submenu 'Help'.

In the Edit Wizard you can call specific help to each function and function block, by performing the following steps: Click with the right mouse button on the specific function or function block in the Edit Wizard selection area. The context menu appears. Select the menu item 'Help on FU/FB'. The related help topics will be displayed.

The help topics for IEC 61131-3 functions and function blocks can also be called choosing the menu item 'Help on IEC firmware' in the submenu 'Help' of the menu 'Pro-H 1.x' in the Windows start menu.



### Searching a topic in the help system

- Select the menu item 'Help topics' in the submenu '?'. The general help contents appears displaying the contained topic titles with the hierarchical structure.
- If you are searching for an entire topic, click on the tab 'Index' and enter the desired topic title.
- If you are searching for a specific word within the help information, click on the tab 'Search'. Follow the instructions on your screen to create a list of all words that are contained in the help file. When the search list has been created, just type the word you are looking for and then select one of the offered topics.
- Further information to a specific topic can be called by clicking on the hypertext jumps at the end of the topic non scrolling region. These jumps to other related topics appear as green underlined phrases.

---

# Editors

## The project manager - a powerful tool for program organization

The project manager is a comfortable and powerful tool for program organization and project management. It includes the project tree editor and the instance tree.

The project tree editor allows you to edit the structure of your project. You can edit your project within four subtrees. In the subtree 'Library' libraries can be announced. In the subtrees 'Data Types' and 'Logical POU's' new POU's and worksheets can be added for editing data type, variable and code body worksheets. In the subtree 'Physical Hardware' all configuration elements can be inserted, global variables can be declared and programs can be associated to tasks.

The instance tree displays the project structure in the PLC and is used to open the worksheets in online mode.

It is possible to display the complete project tree or only particular subtrees. For this purpose the project tree window provides several tabs at its bottom. They can be used to switch between the various views.

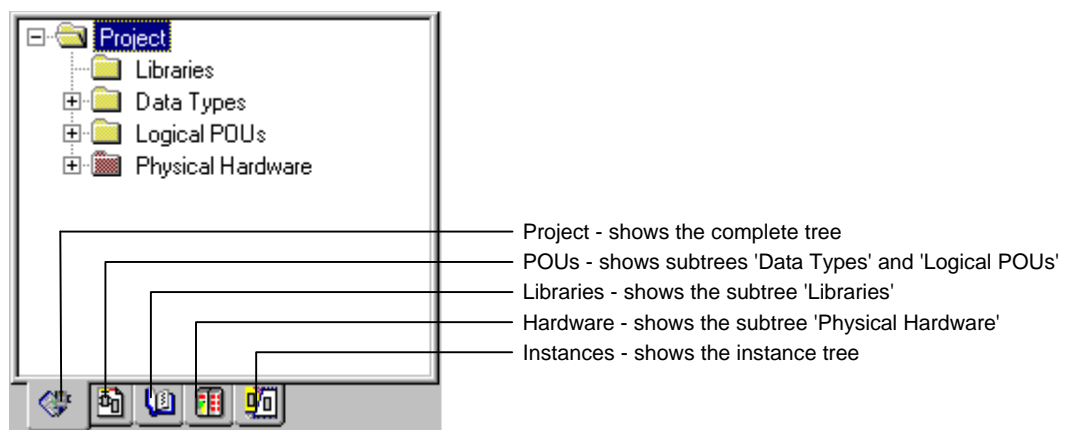


Figure 3-13: Project tree window with sheet tabs

## The graphic editor - easy programming in SFC, FBD and LD

The graphic editor is one of the editors which have been implemented for programming graphical code body worksheets of the POU's. The system supports three graphical programming languages: SFC, FBD and LD.

The following features facilitate editing in graphical languages:

- \* An Edit Wizard provides full edit functionality for inserting and exchanging functions and function blocks
- \* All graphical editors provide simple keyboard operation for inserting and scrolling (Cursor keys /CTRL Cursor keys for Object mode and SHIFT/SHIFT CTR Cursor keys for Mouse cursor mode).
- \* Duplication of function inputs can be done directly via keyboard, toolbar and menu.
- \* Negation of Inputs, Outputs, Contacts and Coils can be done directly via keyboard, toolbar and menu.
- \* Easy auto routing for standard editing cases is possible.
- \* Items can be inserted directly on a line or to the inputs or outputs of already existing items (only in FBD editor).
- \* Splitter and overview windows are available.
- \* Freestyle editing allows to arrange items smoothly wherever you want.
- \* Double click on user functions and function blocks opens the contents of an user POU.

It is possible to mix the graphical programming languages. The system checks all user entries to detect and avoid invalid connections (e.g. connection between two outputs). While inserting new graphical elements the layout of the already existing network is adapted automatically.

## **The text editor - easy programming in IL and ST**

The text editor is used for programming textual code body worksheets of the POU's. The system supports two textual programming languages: IL and ST.

The text editor is also used to edit data type worksheets, variable worksheets and I/O configuration worksheets.

The handling of the text editor is similar to the handling of a normal ASCII-editor. The following features facilitate editing in IL and ST:

- \* An Edit Wizard provides full edit functionality for inserting pre-edited Data types, Operands, Keywords, functions and function blocks.
- \* Full drag and drop functionality is provided.
- \* Optional line numbers can be displayed.
- \* Syntax highlighting is possible.
- \* Multiple Undo/Redo.
- \* Multiple Zooms.
- \* Different views via splitter window are possible.

## **The pagelayout editor - creating pagelayouts for printing**

The pagelayout editor provides an easy way to create pagelayouts for printing the project documentation.

A pagelayout represents a template which is used to print the contents of a code body worksheet. It defines an area in which the content of the worksheet is going to be printed.



# The Edit Wizard

The Edit Wizard is a useful tool, which facilitates the insertion and replacement of

- \* keywords and statements (in ST),
- \* operators (in IL),
- \* function and function blocks (all languages).

The Edit Wizard for the various languages and worksheets is shown in the following figure.

In general, the Edit Wizard is used as follows: Locate the code body position in your worksheet, where a new keyword, statement, function or function block has to be inserted and place the text cursor or insertion mark at this position. Then determine the desired keyword, statement, function or function block in the Edit Wizard selection area and insert it into the code body with a left mouse double click.

Especially in text editors the usage of the Edit Wizard provides the following advantages:

- \* It prevents from entering syntactical faults, such as forgotten semicolons, selection or iteration statements without end statement, etc. This is done by inserting pre-edited statements, functions or function blocks, i. e. the statement structure is already prepared with place holders. The variables and values are inserted as comments, which the user simply has to overwrite.
- \* It is not necessary, that the user knows the syntax of all different statement types, such as functions or function blocks.

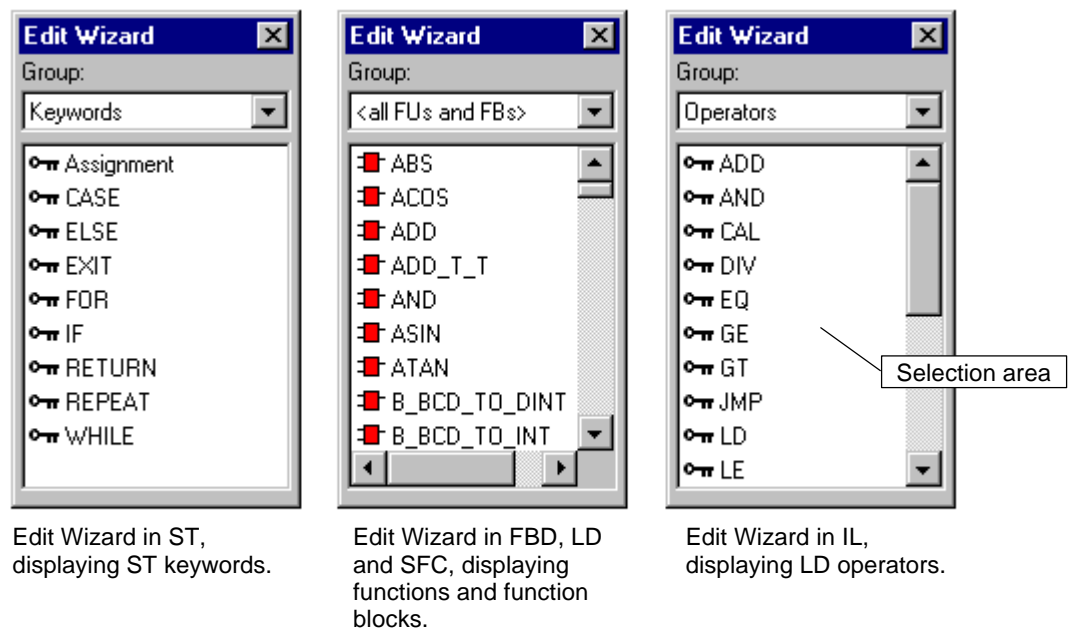


Figure 3-14: Edit Wizard for the different languages

The Edit Wizard can also be used to insert variable definitions in the various variable worksheets as well as data types in a data type worksheet.

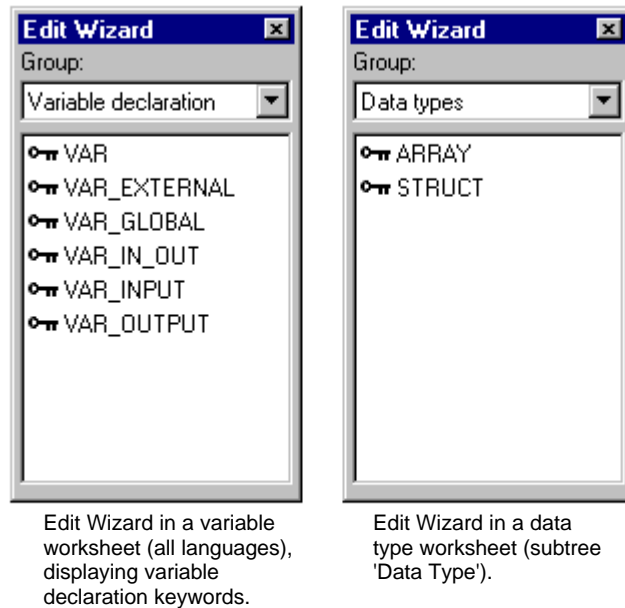






Figure 3-15: Edit Wizard for variables and data type worksheets

The dockable Edit Wizard window is divided into two areas: The list box 'Group' and the selection area. Depending on the selected group the selection area displays the elements which can be inserted into the code body.

The list box 'Group' contains the available groups of code body elements for the corresponding language. The selection area displays the available elements contained in the selected group.

Depending on the editor, the following groups are provided in the list box:

- \* 'Keywords', represented by the key symbol:  (only in ST).
- \* 'Operators', represented by the key symbol:  (only in IL).
- \* '<all FUs and FBs>', where functions are represented by the symbol , and FBs are represented by the symbol .
- \* The color of a symbol points to the origin of the related FU or FB: FIRMWARE FUs and FBs are displayed in red. LIBRARY FUs and FBs are displayed in blue. USER FUs and FBs are displayed in green.
- \* 'Function blocks'. The selection area only offers function blocks. The symbols and their colors are described above.
- \* 'Functions'. The selection area only offers functions. The symbols and their colors are described above.
- \* 'PLC specific'. The selection area offers elements only available for a specific PLC.

- \* 'PROCESSOR specific'. The selection area offers elements only available for a specific processor.
- \* 'String FUs'. The selection area only offers string functions. The symbols and their colors are described above.
- \* '*project name*'. One group is designated with the name of the project file. This group only contains the user defined functions and function blocks, which have already been created within the actual project. This is why only green symbols are available.
- \* 'Type conv. FUs'. This group contains type converting functions, such as 'Bool to Integer' or 'Bool to Word'.
- \* '*Libraries name*'. Each announced library is represented as an own group.

If the Edit Wizard is not visible in the main screen area, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar.



### Calling the Edit Wizard with the keyboard

- Press the shortcut <SHIFT> + <F2>.



The usage of the Edit Wizard is described in detail for each editor in the particular editor description chapter.

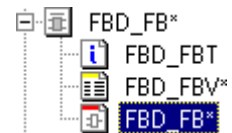
# Overview window for graphical worksheets

When using the graphic editor you can call the overview window to get an overview of the complete contents of the worksheet. This simplifies navigation in your worksheet.



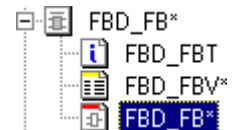
## Calling the overview window with the mouse

- Open a graphical worksheet in the graphic editor by double clicking on the desired worksheet icon in the project tree.
- Click on the submenu 'Layout'.
- Select the menu item 'Overview window'.  
The overview window appears.



## Calling the overview window with its shortcut

- Open a graphical worksheet in the graphic editor. For this purpose mark the desired worksheet icon in the project tree and press <↵>.
- Press <ALT> + <6>.  
The overview window appears.



The overview window shows the entire contents of the current graphical worksheet. You can use the overview window to move the worksheet area which is visible in the editor.



## Navigating in graphical worksheets using the overview window

- Click on the drawing in the overview window. A hand symbol is added to the cursor and the visible area is represented as a rectangle.
- Keep the left mouse button pressed and drag the rectangle to the area you want to be visible in the editor. When releasing the mouse button, the area which is covered by the rectangle is visible in the editor window.

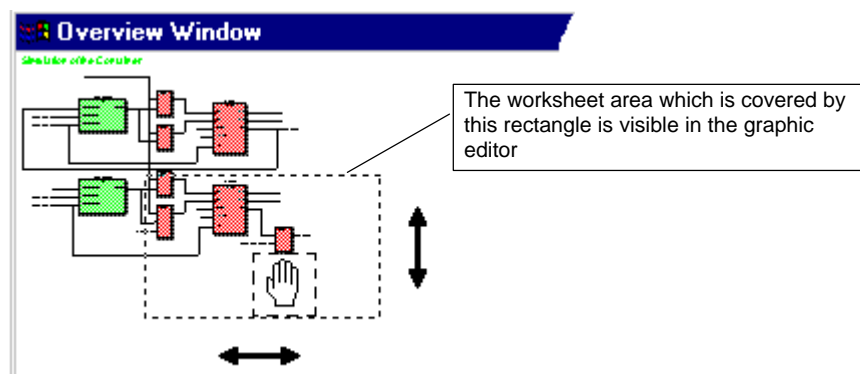


Figure 3-16: Navigating in a graphical worksheet using the overview window

---

## Saving changes while editing

While you are editing you should regularly save the changes you have done. In case of a power failure or other events you risk loss of data if you do not save your work. You can save at any time the changes of the current worksheet.



The project tree is saved automatically.



### Saving the changes of the current worksheet with the mouse

- Click on the icon 'Save' in the toolbar.  
The worksheet is saved.



### Saving the changes of the current worksheet with the keyboard

- Press <CTRL> + <S>.  
The worksheet is saved.



When saving and closing an edited worksheet, the worksheet will be compiled automatically.

---

## Exiting worksheets

If you have finished editing the worksheet you can close the worksheet. A dialog appears where you can save your changes or not.



For saving and exiting all opened worksheets choose the menu item 'Close all' in the submenu 'Window'.



### Exiting the worksheet with the mouse

- Click on the submenu 'File'.
- Select the menu item 'Close'.  
The dialog 'Pro-H' appears.



### Exiting the worksheet with the keyboard

- Press <CTRL> + <F4>.  
The dialog 'Pro-H' appears.

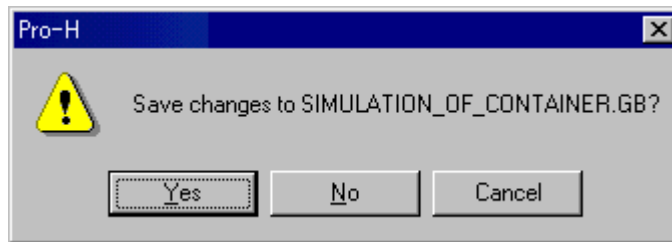


Figure 3-17: Dialog 'Pro-H'



### Using the dialog 'Pro-H'

- Click 'Yes' to exit the worksheet with saving the changes.
- Click 'No' to exit the worksheet without saving the changes.
- Click 'Cancel' to get back to the worksheet without exiting.

---

## Exiting the program

If you exit the program, it does not matter whether one or several editors are still open or all windows are already closed. If you have not saved the changes you have done, a dialog appears and you can either save the changes or close the corresponding windows without saving any changes.



### Exiting the program with the mouse

- Choose the menu item 'Exit' in the submenu 'File'.  
The program is closed.



### Exiting the program using the keyboard

- Press <ALT> + <F4>.  
The program is closed.



To exit the program you can also double click on the icon of the system menu in the upper left window corner or click on the icon 'Close' in the upper right corner of the program window.

# Editing the project structure

**This chapter provides information about...**

- creating a new project
- changing properties of POUs
- inserting POUs
- inserting worksheets
- announcing libraries
- deleting worksheets, POUs or libraries
- saving/zipping the project files

# Editing the project structure

---

## Creating a new project

The first step you have to do after calling the program is creating a new project or opening an existing project. This section describes the steps, which are necessary to create a new project.

While creating a new project the program copies the template you have chosen to a project named 'Untitled'. The template consists of POUs, worksheets or configuration elements necessary for the PLC type.

You can edit the project 'Untitled' and save it afterwards under the name you want to use.

Every project is represented with its own project tree in the project manager. It is recommended to leave the project manager open the whole time while you are working with the program, because it gives you a better orientation at what level you are working and what is left to do. You can either edit the whole structure of the project tree and the single worksheets or the POUs and worksheets immediately after inserting them.



### Creating a new project with the mouse

- Click on the icon 'New Project' in the toolbar. The dialog 'New Project' appears.



### Creating a new project with the keyboard

- Press <CTRL> + <N>.  
The dialog 'New Project' appears.



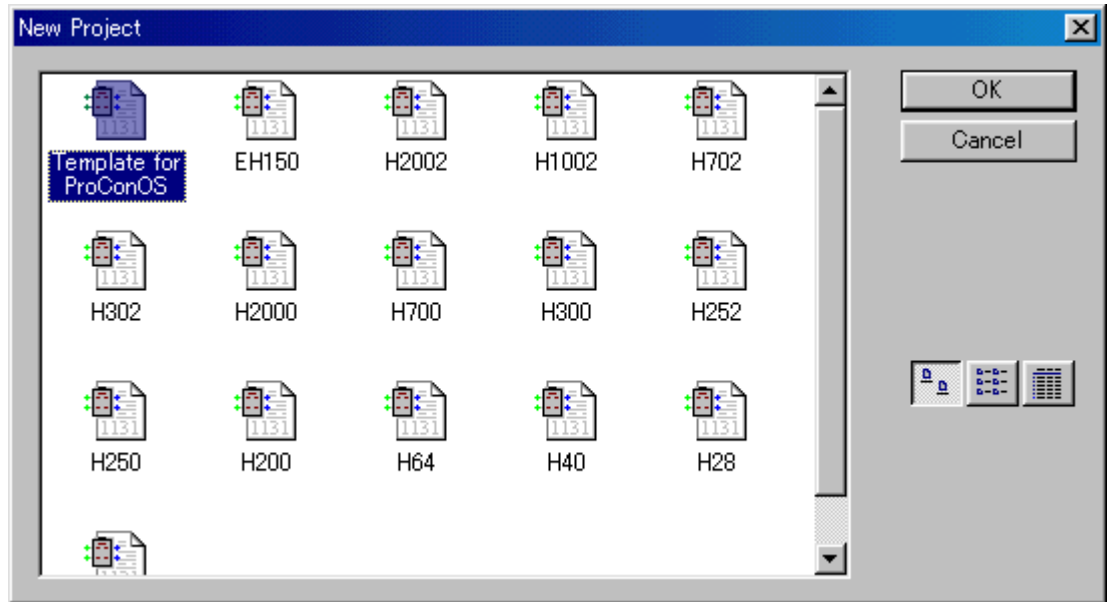


Figure 4-1: Dialog 'New Project' containing the available project templates



### Using the dialog 'New Project'

- Choose the template for your PLC type with a left mouse click.
- Confirm the dialog.  
The new project with the name 'Untitled' is created.

Your project tree should look like the following figure now:

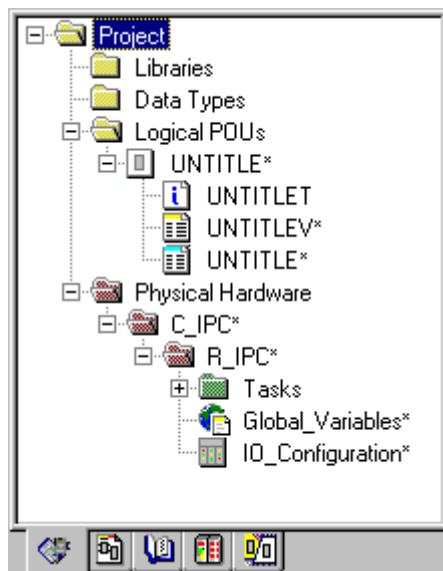


Figure 4-2: Project 'Untitled' with program 'Untitle' and its worksheets

The project 'Untitled' includes automatically one POU: the program 'Untitle'.  
The program has 3 worksheets:

- \* The description worksheet 'UntitleT' for the POU documentation (optional)
- \* The variable worksheet 'UntitleV' for the declaration of variables and function block instances
- \* The code body worksheet 'Untitle' for the code body definition



Some properties of this program can be changed. The default language for the first autoinserted program is IL, thus the language cannot be changed. If a program in another language is required, you have to insert a new POU.

---

## Changing the properties of existing POUs

Let us assume that you want to use the program 'Untitle' just with a different name. In this case you have to change the properties of this program.

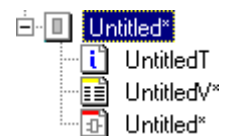


Everywhere in the program you can change the properties of existing objects choosing 'Object Properties...' in the context menu of the object. In order to open the context menu of an object, mark the object and click the right mouse button. The context menu appears. The menu items always relate to the marked object.



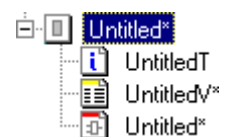
### Changing the properties of existing POUs with the mouse

- Choose 'Properties...' in the context menu of the icon '*Program name*'.  
The dialog 'Properties' appears.  
Or
- Click on the icon '*Program name*' in the project tree, to mark it.
- Click on the icon 'Change Node Properties'.  
The dialog 'Properties' appears.



### Changing the properties of existing POUs with the keyboard

- Press < > or < > to mark the icon '*Program name*'.
- Press <ALT> + <↓>.  
The dialog 'Properties' appears.



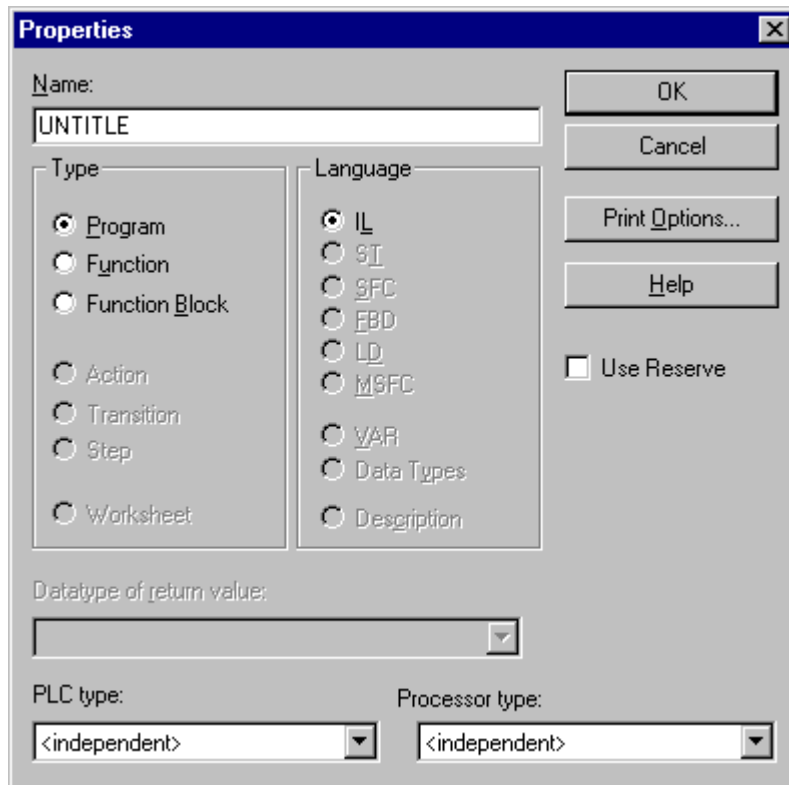


Figure 4-3: Dialog 'Properties' for changing the properties of existing POU



### Using the dialog 'Properties' for changing the properties of existing POU

- If desired, enter a new name for the POU.
- If required, change the POU type. The preselected language cannot be changed for this POU.
- Confirm the dialog.



You can also edit the name of a particular worksheet in the dialog 'Properties'. To perform this, mark the desired worksheet icon in the project tree and click on the icon 'Change Node Properties'. The dialog 'Properties' appears as shown above. Enter the new name for the worksheet in the field 'Name' and confirm the dialog.



If you have changed the name of a program, you also have to change the name of the instantiated program in the task. Otherwise an error occurs when compiling the project, because the system is unable to find the renamed POU. The steps to be performed for associating a program to a task are described in section 'Associating programs to tasks' in the chapter 'Compiling, downloading and debugging'.

# Inserting new POU's

The next steps you would certainly like to do is inserting new POU's in different programming languages.



## Inserting a POU with the mouse

- To insert a new program click on the icon 'Add program'. The dialog 'Insert' appears.
- To insert a new function block click on the icon 'Add function block'. The dialog 'Insert' appears.
- To insert a new function click on the icon 'Add function'. The dialog 'Insert' appears.



## Inserting a POU with the keyboard

- Press < > or < > to mark the icon 'Logical POU'.
- Press <INS>. The dialog 'Insert' appears.

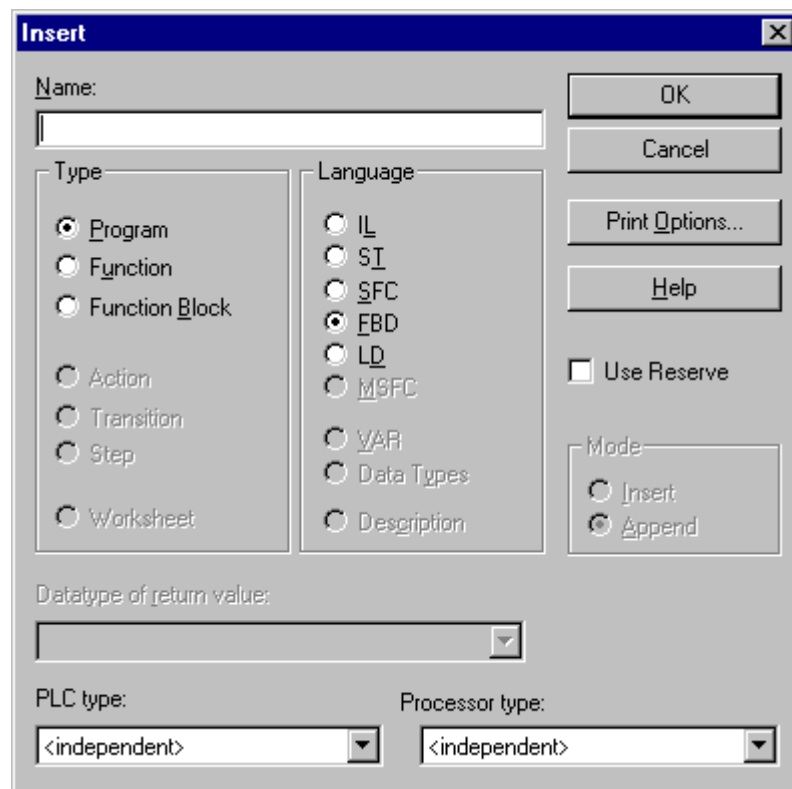


Figure 4-4: Dialog 'Insert' for inserting a new POU



## Using the dialog 'Insert' while inserting a new POU

- Enter a name for your new POU.
- Choose the POU type.
- Choose the programming language. The default language for a new inserted POU is FBD.
- Confirm the dialog.  
The new POU with its worksheets is inserted in the project tree.



Some programming languages may be grayed according to the number of the available editors.

The new worksheets are marked with an asterisk in the project tree. These asterisks mean that the worksheets have been inserted or changed but not yet compiled.

---

## Inserting worksheets

It is also possible to insert new worksheets in POUs. This feature is necessary if you have big code bodies and you want to split them into several pieces for better orientation. If you want to insert new worksheets in POUs the language of the new worksheet is determined by the POU language. This means all worksheets of a POU have the same language e.g. FBD. The language of a new inserted worksheet is set automatically.



### Inserting a code body worksheet in a POU with the mouse

- Click on the code body worksheet icon where the new worksheet has to be inserted in the project tree.
- Choose the menu item 'Insert' from the context menu.  
**Or:**  
Click on the icon 'Add Worksheet'.  
In both cases, the dialog 'Insert' appears.
- Enter a name for the new worksheet in the field 'Name'.
- Determine, whether the new worksheet should be inserted after ('Append') or before ('Insert') the marked worksheet.
- Confirm the dialog 'Insert'.  
The new code body worksheet is inserted in the project tree.





## Inserting a variable worksheet in a POU with the mouse

- Click on the variable worksheet icon where the new worksheet has to be inserted in the project tree.
- Choose the menu item 'Insert' from the context menu.
- Or:**  
Click on the icon 'Add Worksheet'.  
In both cases, the dialog 'Insert' appears.
- Click on the radio button 'VAR' in the 'Language' area of the dialog.
- Enter a name for the new worksheet in the field 'Name'.
- Determine, whether the new worksheet should be inserted after ('Append') or before ('Insert') the marked worksheet.
- Confirm the dialog 'Insert'. The new variable worksheet is inserted in the project tree.



The steps to insert a new description worksheet are similar to the steps described for code body or variable worksheets. The only difference is, that you have to mark an existing description worksheet, before calling the dialog 'Insert'.

The steps to be done for inserting new data type worksheets in the subtree 'Data Types' are also similar to the steps described for inserting new code body worksheets.



## Inserting a data type worksheet with the mouse

- Click on the icon 'Data Types' in the project tree.
- Choose the menu item 'Insert' from the context menu.
- Or:**  
Click on the icon 'Add Worksheet'.
- Choose a name and confirm the dialog. The new data type worksheet is inserted in the subtree 'Data Types'.



Inserting of worksheets (i. e. all types of worksheets) can also be done using the keyboard.



## Inserting a worksheet with the keyboard

- Press < > or < > to mark the desired icon. Which icon you have to mark, depends on the worksheet type you want to insert (refer to the preceding descriptions).
- Press <INS>. The dialog 'Insert' appears.
- Fill in and confirm the dialog 'Insert'.



---

# Announcing libraries

Having already finished one project, you can reuse these POU's and worksheets in a new project. This feature makes it unnecessary to define code bodies which already exist. To reuse POU's and worksheets of an existing project you have to announce this project as an user library of your new project.

It is also possible to use firmware functions and function blocks which are prepared by your PLC manufacturer. Firmware functions and function blocks have to be announced as firmware libraries.

You can use the programs, function blocks and functions of a library but you can not view or edit them.

In the following section it is described how to announce an user defined library. For announcing firmware libraries just choose a different file extension.



## Announcing a library with the mouse

- Click on the icon 'Libraries' to mark it.
- Choose the menu item 'Insert' from the context menu.  
**Or:**  
Click on the icon 'Add Worksheet'.
- In both cases, the dialog 'Insert library' appears.
- Choose the project you want to announce as a library.
- Click on the button 'OK' to confirm the dialog.



## Announcing a library with the keyboard

- Press < > or < > to mark the icon 'Libraries'.
- Press <INS>.  
The dialog 'Announce Library' appears.
- Choose the project you want to announce as a library.
- Press <↵> to confirm the dialog.



Your project tree should look like the following figure now:

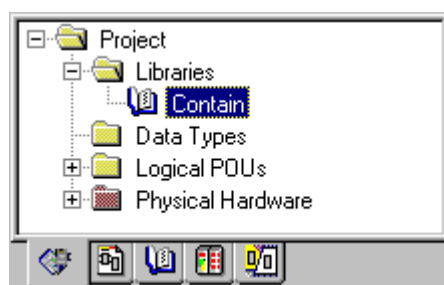


Figure 4-5: Project with announced library 'Contain'

---

## Deleting worksheets, POU's or libraries

It is possible to delete worksheets, whole POU's or libraries. In every case, you first have to mark the object, which has to be deleted.

For the following description let us assume, that you want to delete the POU 'Untitled'.

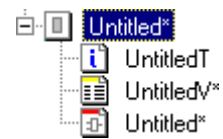


When deleting a whole POU, keep in mind, that all worksheets contained in this POU are deleted too.



### Deleting the POU 'Untitled'

- Click on the POU icon to be deleted.  
Or:  
Press < > or < > to mark the POU to be deleted.
- Press <DEL>.  
A message box appears in which you can confirm or abort the deletion.
- Confirm the dialog.  
The POU is deleted.



Having confirmed the message box there is no way of restoring the data. Use this feature very carefully!

---

## Saving the existing project and zipping the project files

The program allows to save the existing project under a new file name and to zip all project files into one archive file. The zip archive contains all files that are necessary for your project. This will include the project file '*projectname.mwt*', the files with the code bodies and variable declarations and some internal files.



It is recommended to zip your files regularly, e.g. once a day, and to save your archive on a floppy disk to make sure that no loss of data occurs.

The following procedures describe the necessary steps to be performed for saving your existing project under a new name and zipping the project into an archive file. In both cases the dialog 'Save project as' is used.





### **Saving the existing project under a new name with the mouse**

- Click on the submenu 'File'.
- Select the menu item 'Save Project As/Zip Project as...'.  
The dialog 'Save project as' appears.
- Ensure that the entry 'Project Files (\*.mwt)' is selected in the list box 'File type' .
- Enter the new name in the field 'File name'. Use the extension 'mwt'.
- Choose a new path if you want and confirm the dialog.



### **Saving the existing project under a new name with the keyboard**

- Press <ALT> + <F>. The submenu 'File' is opened.
- Press < > or < > to mark the menu item 'Save Project As/Zip Project as...'.  
The dialog 'Save project as' appears.
- Press <↵>.
- Ensure that the entry 'Project Files (\*.mwt)' is selected in the list box 'File type' .
- Enter the new name in the field 'File name'.
- Choose a new path if you want and confirm the dialog by pressing <↵>.



### **Zippping the project files with the mouse**

- Click on the submenu 'File'.
- Select the menu item 'Save Project As/Zip Project as...'.  
The dialog 'Save project as' appears.
- In the list box 'File type' select the entry 'Zipped Project Files (\*.zwt)'.  
The checkboxes 'Zip user libraries', 'Zip frontend code' and 'Zip pagelayouts' are now available.
- Enter a name for the zip file in the field 'File name'. Use the extension 'zwt' (for example contain.zwt).
- Activate the checkbox 'Zip user libraries' if you want to zip the library too.
- Activate the checkbox 'Zip frontend code' if you want to zip the compilation files too.
- Activate the checkbox 'Zip pagelayouts' if you want to zip pagelayouts and bitmaps too.

- Press 'Zip' to start the zipping process.  
The progress of the zipping process is displayed in the status bar.
- Confirm the message box after zipping.



Detailed information about the dialog 'Save project as' can be found in the program help. The procedure how to unzip project files is also described in the help.

# Literals, data types and variables

**This chapter provides information about...**

- literals
- data types
- editing user defined data types
- ways how to declare variables
- types of variables and keywords
- instantiation

# Literals, data types and variables

---

## Literals

Literals can be used to enter external representation of data. Literals are necessary for the representation of numeric, character strings and time data. Every time you have to enter values you have to use literals.

### Numeric literals

The numeric literals which can be used are shown in the following table:

Type	Examples
Integer literals	-12 0 123_456 +986
Real literals	-12.0 0.0 0.4560 3.14159_26
Real literals with exponents	-1.34E-12 -1.34e-12 1.0E+6
Base 2 literals	INT#2#1111_1111
Base 8 literals	INT#8#377
Base 16 literals	INT#16#FF SINT#16#ff
Boolean FALSE and TRUE	FALSE TRUE
Boolean 0 and 1	0, 1

Figure 5-1: Numeric literals



Literals which are used in variable worksheets, literals of data type INT or BOOL can be used without keyword as it is shown in the following examples:

For INT#16#ff you can use 16#ff  
For BOOL#FALSE you can use FALSE.



In variable declarations you can use  
"var1 : DINT :=10"  
but in the code body you have to use "LD DINT#10".

## Character string literals

A character string literal is a sequence of zero or more characters included in two single quote characters.

The character string literals which can be used are shown in the following table:

Type	Examples
Empty string	"
String with a blank	' '
Not empty string	'this is a text'

Figure 5-2: Character string literals

## Duration literals

Duration data can be represented in hours, minutes, seconds, milliseconds and in combination.

The duration literals which can be used are shown in the following table:

Type	Examples
Short prefix	T#14ms t#14ms t#12m18s3.5ms T#25h_15m t#25h_15m
Long prefix	TIME#14ms time#14ms TIME#25h_15m time#25h_15m

Figure 5-3: Duration literals

---

## Introduction to the IEC data types

Data types define the bitsize, the value range and the initial value of a variable. Elementary and user defined data types can be used. Generic data types are basically used for describing the available data types of inputs and outputs of functions.

---

## Elementary data types

The value ranges and the bitsize of elementary data types are described in IEC 61131-3.

Elementary data types are shown in the following table:

Data type	Description	Size	Range	Default initial value
BOOL	Boolean	1	0 up to 1	0
SINT	Short integer	8	-128 up to 127	0
INT	Integer	16	-32768 up to 32767	0
DINT	Double integer	32	-2.147.483.648 up to +2147.483.647	0
USINT	Unsigned short integer	8	0 up to 255	0
UINT	Unsigned integer	16	0 up to 65535	0
UDINT	Unsigned double integer	32	0 up to 4.294.967.295	0
REAL	Real numbers	32	$1.18 \times 10^{-38}$ up to $3.40 \times 10^{38}$	0.0
TIME	Duration	32	about 24 days	t#0s
BYTE	Bit string of length 8	8	00 up to FF	0
WORD	Bit string of length 16	16	0000 up to FFFF	0
DWORD	Bit string of length 32	32	00000000 up to FFFFFFFF	0

Figure 5-4: Elementary data types



The use of data types also depends on your hardware. Please refer to your hardware documentation for information about the supported data types.



The data type STRING is also an elementary data type but does not belong to the above mentioned group. Its format in the memory depends on your PLC type.

The data type STRING has the following structure:

Byte 0-1      offset to maximum length (0 corresponds to 80)  
Byte 2-3      current length  
Byte 4-83     characters  
Byte 84       zero terminator

---

## Generic data types

Generic data types are data types which include hierarchical groups of elementary data types. ANY\_INT includes the elementary data types DINT, INT, SINT, UDINT, UINT and USINT. If a function can be connected with ANY\_INT it means that variables of the data types DINT, INT, SINT, UDINT, UINT and USINT can be handled.

Generic data types are shown in the following table:

ANY	
ANY_NUM	
ANY_REAL	
REAL	
ANY_INT	
DINT, INT, SINT	
UDINT, UINT, USINT	
ANY_BIT	
DWORD, WORD, BYTE, BOOL	
STRING	
TIME	

Figure 5-5: Generic data types



The use of data types also depends on your hardware. Please refer to your hardware documentation for restrictions concerning data types.

---

## User defined data types

User defined data types can be declared to create new data types in addition to the elementary data types defined in IEC 61131-3. This is useful because new data types with different properties can be created for an easier and quicker programming. User defined data types are also called derived data types.

Three types of user defined data types can be declared:

- \* Array data types
- \* Structured data types
- \* String data types.



Please refer to your PLC documentation for information about available user defined data types and restrictions. It is possible that the PLC does not support all described possibilities of using user defined data types!

---

# Array data types

## Declaring arrays

Array data types include several elements of the same data type. The particular elements within one array are identified and accessed using an array index. An array can be used to declare several elements of the same type with only one line in the type declaration.

```
TYPE
  graph      :      ARRAY[1..23]  OF INT;
END_TYPE
```

Figure 5-6: Type declaration of an array data type

In the example the array 'graph' contains 23 elements of the data type 'INT'. All elements of an array are stored continuously one after another in the PLC memory.



In order to declare an array, you first have to declare the array in the type declaration of the data type worksheet. Subsequently you can declare a variable in the variable worksheet of the POU (refer to the following programming example).



## Programming example

An array should be typically used for data describing the same subject. Let us imagine that a process changes an input value every 3 seconds. It is necessary to store each of these input values to compare them with set points. All input values are of the same data type. In this case it is useful to declare an array because in the code body declaration the two values can be easily compared using an iteration statement (e.g. a FOR loop). The single components of the array can be accessed using the array index.

```
Type declaration:
TYPE
  graph      :      ARRAY[1..23]  OF INT;
  set_point  :      ARRAY[1..23]  OF INT;
END_TYPE

Variable declaration:
VAR
  input      :      graph;  (* incoming values of the machine *)
  values     :      set_point;  (* values to compare with *)
  i          :      INT :=1; (* variable for array index *)
  run        :      BOOL :=TRUE;
  ERROR      :      BOOL;
  timer      :      FB_TIMER;  (* declare FB instance *)
END_VAR

Code body declaration in ST:
timer (pt:=t#3s;in:=run);
IF timer.Q THEN  (* provide input values to array 'graph' *)
  input[i] := %IW0;  (* assign input value to array *)
  run := 0;  (* edge detection to start timer again *)
  i := i+1;  (* higher array index *)
ELSE
  run :=1;  (* count up *)
END_IF;
IF i = 23
  FOR i:=1 TO 23 BY 1 DO
    IF input[i] <> values[i] THEN  (* comparing array
                                  'graph' to 'set point' *)
      ERROR := TRUE;
    END_IF;
  END_FOR;
  i := 1;
END_IF;
```

Figure 5-7: Programming example of an array data type

## Multi-dimensional arrays

If multi-dimensional arrays are needed, arrays of arrays can be used. An example for an array of an array is shown in the following figure:

```
TYPE
  graph      :      ARRAY [1..10] OF INT;
  my array   :      ARRAY [1..3] OF graph;
END_TYPE
```

Figure 5-8: Type declaration of an array of array

In an array of an array a single element is accessed using two indexes as shown in the following figure:

```
Variable declaration:
VAR
  var1      :      my_array;
  var2      :      INT;
END_VAR

Code body declaration in ST:
var2 := var1[1][3];
```

Figure 5-9: Accessing elements of an array of array

## Initializing arrays

Arrays can be initialized, i. e. to each element contained in the array, a value can be assigned. As already mentioned above, each single element is accessed using its index. The initialization of an array can be done while editing the code body declaration.

```
Variable declaration:
VAR
  graph      :      ARRAY [1..10] OF INT;
END_VAR

Code body declaration in ST:
graph[1]:=7;
graph[2]:=1092;
.
.
.
graph[10]:=13;
```

Figure 5-10: Initializing elements of an array

It is not necessary to initialize all elements of an array. If no initial value is used, the array element is initialized with the default initial value when starting the program execution.

---

# Structured data types

## Declaring structures

Structured data types include several elements of the same or of different data types.

```
TYPE
  machine:
  STRUCT
      x_pos  :    REAL;
      y_pos  :    REAL;
      depth  :    INT;
      rpm    :    INT;
  END_STRUCT;
END_TYPE
```

Figure 5-11: Declaration of a structured data type

In the example the structure 'machine' consists of 4 components. All components describe the characteristics of the machine.

## Programming example

Structures should be used if data describing the same objects have to be declared. For example a drilling machine drills several holes in one work piece. All holes have a position in x and y position on the work piece, a drilling depth and different revolutions per minutes to drill with. The concrete values for the holes are different but the variables which are needed are always the same. In this case it is useful to declare a structure consisting of three components for the position, drilling depth and revolutions per minute. For each hole different values can be assigned to the components. The function block for the drilling process is just working on the same variable being a structure.

## Arrays of structures

It is possible to use a structure within arrays as it is shown in the following example:

```
TYPE
  machine:
  STRUCT
      x_pos  :    REAL;
      y_pos  :    REAL;
      depth  :    INT;
      rpm    :    INT;
  END_STRUCT;
  my array  :    ARRAY[1..10]  OF machine;
END_TYPE
```

Figure 5-12: Declaration of an array of structure

An application example for arrays of structures could be a transfer line with several drilling machines. Via the array index the concrete drilling machine can be accessed and via the structure components the different values for drilling can be assigned.

## Structures with arrays

It is possible to use arrays in structures as it is shown in the following example:

```
TYPE
  graph      :      ARRAY[1..10]  of INT;
  drive:
  STRUCT
      rpm      :      INT;
      inputs   :      IN_BOOL;
      performance :      graph;
  END_STRUCT;
END_TYPE
```

Figure 5-13: Declaration of a structure of array

## Initializing structures

Structures can be initialized by assigning the values to the components while editing the code body declaration. An example is shown in the following figure:

```
Variable declaration:
VAR
  var1      :      machine;
  first     :      BOOL :=TRUE;
END_VAR

Code body declaration in ST:
IF first THEN
  var1.x-pos := REAL#1.3E+2
  var1.rpm := 3000;
...
  first := FALSE;
END_IF
...
```

Figure 5-14: Assigning values to components of a structure

---

# String data types

## Declaring strings

User defined string data types are strings with a variable number of characters. Declaring a user defined string the length is set in parentheses behind the data type.

## Programming example

In the following example the string length is 10 characters:

```
TYPE
  STRING10   :   STRING(10)
END_TYPE
```

Figure 5-15: Declaration of a string data type

---

# Calling the text editor with the data type worksheet

The first step before editing user defined data types is to call the text editor with the data type worksheet. This step is done using the project tree. Let us assume for the following description that you want to edit the data type worksheet 'Type'. Therefore you have to insert first the data type worksheet 'Type' in the project tree as it is described in the chapter 'Editing the project structure'.



### Calling the data type editor with the mouse

- Double click on the icon 'Data type worksheet'.  
The text editor with the data type worksheet appears.



### Calling the data type editor with the keyboard

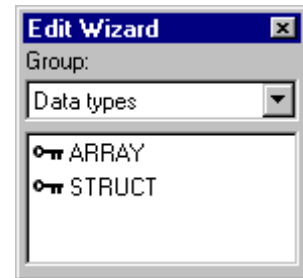
- Press < > or < > to mark the icon 'Data type worksheet'.
- Press <↵>.  
The text editor with the data type worksheet appears.



## Editing type declarations using the Edit Wizard

Type declaration of arrays and structures can be done using the Edit Wizard.

After calling the Edit Wizard in a data type worksheet, by pressing <SHIFT> + <F2>, the Edit Wizard is opened. The list box 'Group' contains only one entry: The group 'Data types'.



The selection area contains two keywords: ARRAY and STRUCT.

You can insert these keywords into your data type worksheet. The Edit Wizard inserts pre-edited constructions, where you just have to replace the green displayed comments by your actual values and variables.

Figure 5-16: Edit Wizard in a data type worksheet

The following figure shows both, an array and a structure, each inserted using the Edit Wizard. For each keyword, a new declaration block is inserted.

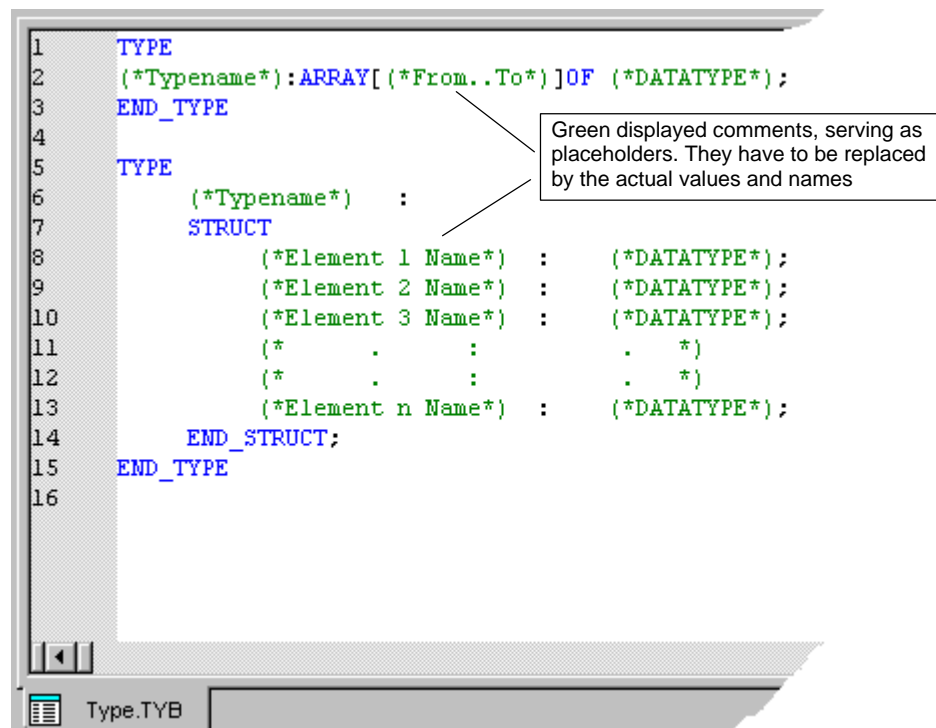


Figure 5-17: Data type worksheet 'Type' with pre-edited keywords, inserted using the Edit Wizard



General information about the Edit Wizard can be found in chapter 'Getting started'.



### Inserting a structure using the Edit Wizard with the mouse

- Locate the position, where the new structure is to be inserted in the data type worksheet. Keep in mind, that every 'Wizard inserted' array or structure is included in an own declaration block.  
Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- In the list of available keywords in the Edit Wizard double click on the keyword 'STRUCT'. It is automatically inserted at the text cursor position. The actual variables and values are replaced by comments (green text, enclosed by parentheses and asterisks) as shown in figure 5-17.
- Replace the comments by the actual names and values used in your project.



### Inserting a structure using the Edit Wizard with the keyboard

- Press < > or < > to move the text cursor to the position, where the new structure is to be inserted in the data type worksheet. Keep in mind, that every 'Wizard inserted' array or structure is included in an own declaration block.
- Press <↵> to insert a new line.
- Press <ALT> + <3> to set your cursor into the Edit Wizard selection area.
- Press < > or < > to mark the keyword 'STRUCT'.
- Press <↵> to insert the structure. It is automatically inserted at the text cursor position. The actual variables and values are replaced by comments (green text, enclosed by parentheses and asterisks) as shown in figure 5-17.
- Replace the comments by the actual names and values used in your project.

---

## Symbolic, located variables and directly represented variables

According to IEC 61131-3 variables are used for programming instead of direct addressing inputs, outputs or flags. Symbolic, located or directly represented variables can be declared.

A declaration of a symbolic variable consists of a variable name and a data type. A declaration of a located variable consists of a variable name, the variable location and a data type. A declaration of a directly represented variable consists of the variable location and a data type.



Directly represented variables can only be declared in the declaration of global variables or in programs.

In the following figure an example for each variable type is given:

```
VAR
  name           : data type;
  name           AT   %location : data type;
  name           AT   %location : data type;
END_VAR
```

Figure 5-18: Declaration of a symbolic, a located variable and a directly represented variable

The location of the variable consists of a location prefix and a size prefix. Location prefixes are I for inputs, Q for outputs and M for internal memory. Size prefixes are X for single bits, B for byte, W for word and D for double word.



Located and directly represented variables are stored at the declared logical address and it is up to the application programmer to check that no memory address is used twice.



---

## Global and local variables

The scope of each variable which is determined by the use of the variable keyword is limited either to a POU or to the whole project. Therefore two types can be distinguished:

- \* Local variables
- \* Global variables

If a variable can be used only within a POU it is called local variable. In those cases the variable keywords VAR, VAR\_INPUT and VAR\_OUTPUT must be used.

If a variable can be used within the whole project it is called global variable. It has to be declared as VAR\_GLOBAL in the global declaration and as VAR\_EXTERNAL in each POU where it is used.



It might be useful to declare all I/Os as global variables. In the global variable declaration they should be declared as located variables and in the VAR\_EXTERNAL declaration of the POU they should be declared as symbolic variables. The typing effort in case of address changes is minimized doing it this way.

---

## Retentive variables

Retentive variables are variables whose values are stored even if the power is switched off. In the case of a warm start the last value of the variable is going to be used.

Retentive variables are declared using the keyword RETAIN as it is shown in the following example:

```
VAR RETAIN
  var1      :      BOOL := TRUE;
END_VAR
```

Figure 5-19: Example for the declaration of a retentive variable

In this example the variable has got the initial value 'TRUE' which is the initial value for a cold start. In case of a warm start the current value of the variable is used.

The keyword RETAIN can be used in combination with the keywords VAR, VAR\_OUTPUT and VAR\_GLOBAL. It is not possible to declare retentive variables with the keywords VAR\_INPUT and VAR\_EXTERNAL.

---

## Initializing variables

According to IEC 61131-3 initial values can be assigned to variables. Initial values can be given to all kind of variables except in VAR\_EXTERNAL declarations. If the PLC is started the variable is processed using the initial value.

Initial values have to be inserted at the end of the declaration line of the variable using ':=' as it is shown in the following figure:

```
VAR
  name                :      data type      := initial value;
  name AT %location   :      data type      := initial value;
                    AT %location   :      data type      := initial value;
END_VAR
```

Figure 5-20: Declaration and initialization of a symbolic, a located variable and a directly represented variable

The initial value has to fit to the data type. It is not possible to use e.g. the data type BOOL and an initial value '5'. In this case the system displays an error message.

The initial value is optional. If no initial value is used, the variable is initialized with the default initial value of the data type or with the retained value in case of retentive variables.

---

## Variable declaration keywords

According to IEC 61131-3 different types of variable declarations exist. For each type a different keyword is used as you can see in the following table:

Keyword	Variable type / Explanation
VAR	for internal variables which can be used only within a POU for declaring the instances of function blocks can be used for the declaration of directly represented and located variables in programs can be used for the declaration of symbolic variables can be used with the keyword 'RETAIN' for declaring retentive variables  to be continued....

Keyword	Variable type / Explanation
VAR_INPUT	<p>for variables which are inputs to functions, function blocks and programs</p> <p>to give a value to the POU coming e.g. from another POU</p> <p>its value is only read within the POU</p> <p>can be used only for the declaration of symbolic variables</p>
VAR_OUTPUT	<p>for variables which are outputs to function blocks and programs</p> <p>supplies an output value for e.g. other POUs</p> <p>its value is written within the POU</p> <p>it is also allowed to read the value</p> <p>can be used only for the declaration of symbolic variables</p> <p>can be used with the keyword 'RETAIN' for declaring retentive variables</p>
VAR_IN_OUT	<p>address of the variable is passed by reference</p> <p>the variable can be read or written</p> <p>typically used for complex data types such as strings, arrays and structures.</p>
VAR_EXTERNAL	<p>for global variables in the POU</p> <p>its value is supplied by the declaration of VAR_GLOBAL</p> <p>cannot be initialized</p> <p>its value can be modified within the POU</p> <p>can be used only for the declaration of symbolic variables</p>
VAR_EXTERNAL_PG	<p>for global variables in the program</p> <p>its value is supplied by the declaration of VAR_GLOBAL_PG</p> <p>cannot be initialized</p> <p>its value can be modified within the program</p> <p>can be used only for the declaration of symbolic variables</p>
VAR_EXTERNAL_FB	<p>for global variables in the function block</p> <p>its value is supplied by the declaration of VAR_GLOBAL_FB</p> <p>cannot be initialized</p> <p>its value can be modified within the function block</p> <p>can be used only for the declaration of symbolic variables</p>

to be continued...

Keyword	Variable type / Explanation
VAR_GLOBAL	for global variables which can be used in all programs and function blocks of the project can be used for the declaration of directly represented, located and symbolic variables can be used with the keyword 'RETAIN' for declaring retentive variables
VAR_GLOBAL_PG	for global variables which can be used in all programs of the project can be used for the declaration of directly represented, located and symbolic variables can be used with the keyword 'RETAIN' for declaring retentive variables
VAR_GLOBAL_FB	for global variables which can be used in all function blocks of the project can be used for the declaration of directly represented, located and symbolic variables can be used with the keyword 'RETAIN' for declaring retentive variables
END_VAR	to finish a variable declaration block

Figure 5-21: Table of keywords for variable declaration blocks



Global variables have to be declared as VAR\_GLOBAL in the global variable declaration of the resource and as VAR\_EXTERNAL in the variable declaration of the POU.



The keywords VAR\_GLOBAL\_PG, VAR\_GLOBAL\_FB, VAR\_EXTERNAL\_PG and VAR\_EXTERNAL\_FB are IEC extensions.

---

## Declaring variables

You have two possibilities to declare variables:

- \* Declaring a variable while editing a code body
- \* Declaring variables in a variable worksheet using the text editor

The first method means inserting a variable in a code body worksheet which has not been declared before. In this case the dialog 'Variable' with its subdialog 'Automatic Variable Declaration' appears for the automatic declaration of the variable. Confirming this dialog the variable declaration is autoinserted in the variable worksheet and the new variable is inserted in the code body worksheet. This method is described in the corresponding chapters for the programming languages and SFC.



If the corresponding variable worksheet is opened (i. e. the worksheet to which the new variable declaration has to be autoinserted), it is closed automatically when the dialog 'Variable' appears.

The second method means declaring variables just typing the declarations in the variable worksheet. To perform this the Edit Wizard can be used to insert the variable declaration keywords (VAR, VAR\_INPUT, etc.).

After calling the Edit Wizard in a variable worksheet, by pressing <SHIFT> + <F2>, the Edit Wizard is opened. The list box 'Group' contains only one entry: The group 'Variable declaration'.

The selection area contains the available variable declaration keywords.

You can insert these keywords into your variable worksheet. The Edit Wizard inserts pre-edited constructions, where you just have to replace the green displayed comments by your actual variable declarations.



Figure 5-22: Edit Wizard in a variable worksheet

The following figure shows a VAR declaration inserted using the Edit Wizard. Note, that for each keyword a new declaration block is inserted.

```

1  VAR (*RETAIN*)
2      (*local Variable*) : (*Datatype*);
3      (*local Variable*) : (*Datatype*) :=(*initial value*);
4  END_VAR
5

```

Green displayed comments, serving as placeholders. They have to be replaced by the actual declarations.

Tlcv

Figure 5-23: Variable worksheet with pre-edited keyword 'VAR', inserted using the Edit Wizard



General information about the Edit Wizard can be found in chapter 'Getting started'.



### Inserting a VAR declaration using the Edit Wizard with the mouse

- Locate the position, where the VAR declaration is to be inserted in the variable worksheet. This position must not be within another declaration block.  
Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- In the list of available keywords in the Edit Wizard double click on the keyword 'VAR'. It is automatically inserted at the text cursor position.  
The actual variable declarations are replaced by comments (green text, enclosed by parentheses and asterisks) as shown in figure 5-23.
- Replace the comments by the actual declarations.



### Inserting a VAR declaration using the Edit Wizard with the keyboard

- Press < > or < > to move the text cursor to the position, where the VAR declaration is to be inserted in the variable worksheet. This position must not be within another declaration block.
- Press <↵> to insert a new line.
- Press <ALT> + <3> to set your cursor into the Edit Wizard selection area.
- Press < > or < > to mark the keyword 'VAR'.

- Press <↵> to insert the keyword. It is inserted automatically at the text cursor position. The actual variable declarations are replaced by comments (green text, enclosed by parentheses and asterisks) as shown in figure 5-23.
- Replace the comments by the actual declarations.

---

## Instantiation

IEC 61131-3 provides the possibility of instantiation. Instantiation means that a function block is defined once and can be used several times. As function blocks always have an internal memory it is necessary to store their values for each time the function block is used to a different memory region. This is done using instance names. The instance name is declared in the variable declaration of the POU where the function block is going to be used. In the following figure an example of a variable declaration for the function block 'FB\_exam' with two instances is shown:

```
VAR
    drive1      :      FB_exam;
    drive2      :      FB_exam;
END_VAR
```

Figure 5-24: Instantiation of a function block

The function block 'FB\_exam', whose code body has been defined somewhere in the project, has got two instances. The instance name of the first instance is 'drive1', of the second 'drive2'. In the corresponding code body worksheet you can use the function block 'FB\_exam' twice, entering in both cases the correct instance name.



Instance names are created automatically whenever a function block is inserted using the Edit Wizard.

Function blocks can be instantiated within programs or other function blocks. Functions can be called without instantiation because they do not have an internal memory.

The instance tree shows all instances used in your project as it is shown in the following example:

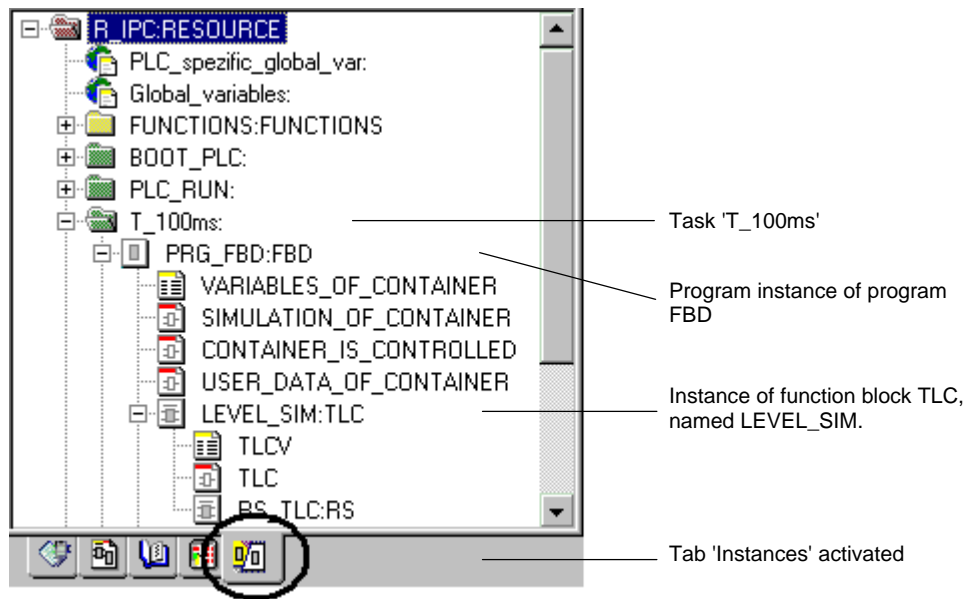


Figure 5-25: Example of an instance tree

In this example one program instance of the program 'FBD' is used in the task 'T\_100ms'. The function block 'TLC' has been instantiated in the program 'FBD'.

The instance tree is made visible by clicking with the left mouse button on the tab 'Instances' at the bottom of the project tree (refer to figure 5-25).



Program instances are created just associating a program to a task and entering an instance name in the corresponding dialog.



Associating programs to tasks is described in the chapter 'Compiling, downloading and debugging' of this manual.



# Editing in ST

**This chapter provides information about...**

- calling the text editor with a ST worksheet
- introduction to ST
- inserting and editing assignment statements
- inserting and editing further statements
- inserting statements using the Edit Wizard
- inserting variables
- calling functions or function blocks using the Edit Wizard

# Editing in ST

---

## Calling the text editor with a ST worksheet

Before editing a ST code body worksheet you have to call the text editor with the ST worksheet, using the project tree.



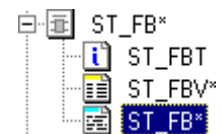
A general description of handling the project tree and browsing through POU's and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

As an example, let us assume that you want to edit the ST worksheet of a function block which is called ST\_FB. For that purpose, you first have to insert a function block with this name as it is described in the chapter 'Editing the project structure'. Keep in mind that the language of the worksheet is determined by the POU language. It is set by inserting the POU respectively the worksheet.



### Calling the text editor for the ST code body worksheet with the mouse

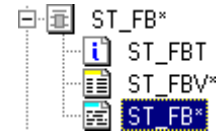
- If the desired worksheet icon is not visible in the project tree, open the corresponding subtree, containing the POU worksheets. For this purpose double click on the POU name (e.g. 'ST\_FB').
- Double click on the icon 'Worksheet in ST' of the function block 'ST\_FB'.  
The text editor with the ST worksheet appears.





## Calling the text editor for the ST code body worksheet with the keyboard

- If the desired worksheet icon is not visible in the project tree, open the 'ST\_FB' subtree, containing the POU worksheets as follows: Press < > or < > to highlight the function block 'ST\_FB'. Press <=> to open the function block subtree.
- Press < > or < > to mark the desired icon 'Worksheet in ST' of the function block 'ST\_FB'.
- Press <↵>.  
The text editor with the ST worksheet appears.



---

## Introduction to ST

A code body which is programmed in the textual language ST consists of statements and expressions.

Different types of statements can be used while editing.

An expression which is part of a statement returns one value for the execution of the statement. Expressions consist of operators and operands. The operators have to be applied to the operands in the way that the operator with the highest precedence is followed by the operators with the next lower precedence.

Statements and expressions can be entered by just typing them or using the Edit Wizard. The Edit Wizard simplifies editing in the text editors. In ST it contains a lot of standard keywords, functions and function blocks which can be inserted. When entering the statements by typing, it is recommended for a better orientation to start each statement in a new line and to use indents for statements and loops. Each statement has to end with a semicolon. When using the Edit Wizard this is done automatically. Comments can be inserted using asterisks and parentheses. Each line starts with the line number. The syntax highlighting represents the different elements by colors: keywords are blue; variables and instance names are black; comments are green.



The usage of the Edit Wizard is described in the section 'Editing statements using the Edit Wizard' in this chapter.

## Inserting and editing assignment statements

An assignment statement is a specific type of statement. It copies the value of the expression on the right to the variable on the left as it is shown in the following figure:

<i>variable name</i> :=        expression;
--

Figure 6-1: Structure of an assignment statement in ST

Assignment statements are inserted using the Edit Wizard or by typing them.

In assignment statements it is important that the variable on the left and the value of the expression on the right are of the same data type. If not, a type conversion must be used.

The expression on the right is composed of operands and operators. Operands can be literals, variables, function calls or other expressions. IEC 61131-3 provides a list of possible operators which can be used to connect the operands. These operators have a certain priority. The operator with the highest priority is evaluated first. The list of possible operators is shown in the following table. The operator with the highest priority is explained in the first line, the operator with the lowest in the last.

Operator	Example	Value of example	Meaning
( )	(2+3) * (4+5)	45	Parenthesization
**	3**4	81	Exponentiation
-	-10	-10	Negation
NOT	<b>NOT</b> TRUE	FALSE	Complement
*	10*3	30	Multiplication
/	6/2	3	Division
MOD	17 <b>MOD</b> 10	7	Modulo
+	2+3	5	Addition
-	4-2	2	Subtraction
<, >, <=, >=	4 > 12	FALSE	Comparison
=	T#26h = T#1d2h	TRUE	Equality
<>	8 <>16	TRUE	Inequality
&, AND	TRUE <b>&amp;</b> FALSE	FALSE	Boolean AND
XOR	TRUE <b>XOR</b> FALSE	TRUE	Boolean exclusive OR
OR	TRUE <b>OR</b> FALSE	TRUE	Boolean OR

Figure 6-2: Table of operators in ST

## Inserting and editing further statements

While editing in ST more statements can be used in addition to the assignment statements (such as selection statements, iteration statements or return statements). The keywords of the statements, examples for using the statements and their meanings are shown in the following table.



You can use the Edit Wizard for entering statements in ST. In this case their structure is pre-edited.

For entering a statement using the Edit Wizard, perform the steps described in the following section 'Inserting statements using the Edit Wizard'.

Keyword	Example	Meaning
IF	<b>IF</b> a < b <b>THEN</b> c:=1; <b>ELSIF</b> a=b <b>THEN</b> c:=2; <b>ELSE</b> c:=3; <b>END_IF</b> ;	<b>selection statement:</b> A group of statements is executed only, if the associated expression 'a<b' is TRUE. If the condition is FALSE, either no statement is executed or the group of statements following ELSE is executed.
CASE	<b>CASE</b> f <b>OF</b> 1: a:=3; 2..5: a:=4; 6: a:=2; b:=1; <b>ELSE</b> a:=0; <b>END_CASE</b> ;	<b>selection statement:</b> A group of statements is executed according to the value of the expression following the keyword CASE. The variable or expression 'f' must be of the data type INT.
FOR	<b>FOR</b> a:=1 <b>TO</b> 10 <b>BY</b> 3 <b>DO</b> f[a] :=b; <b>END_FOR</b> ;	<b>iteration statement:</b> A group of statements is executed repeatedly increasing the variable 'a' by '3', beginning at '1' and finishing at '10'. The starting point is indicated by the assigned value of the control variable 'a'. The final value is indicated following 'TO' and the increments are indicated following 'BY'. All values must be of the data type ANY_INT.  Note: If 'BY' is not indicated, the default value '1' is used. In this case, all values must be of the data type INT.

To be continued ....

Keyword	Example	Meaning
WHILE	<b>WHILE</b> b > 1 <b>DO</b> b:= b/2 <b>END_WHILE</b> ;	<b>iteration statement:</b> A group of statements is executed repeatedly, until the associated expression 'b>1' is FALSE. The condition of the statement is executed at the beginning of the loop. If the condition is FALSE, the loop is not executed.
REPEAT	<b>REPEAT</b> a := a*b; <b>UNTIL</b> a < 10000 <b>END_REPEAT</b> ;	<b>iteration statement:</b> A group of statements is executed repeatedly until the associated expression is TRUE. The condition of the statement is executed at the end of the loop. If the condition is FALSE, the loop is executed at least once.
RETURN	<b>RETURN</b> ;	<b>return statement:</b> The return statement exits the called function, function block or program and returns to the calling POU.
EXIT	<b>FOR</b> a:=1 <b>TO</b> 2 <b>DO</b> <b>IF</b> flag <b>THEN</b> <b>EXIT</b> ; <b>END_IF</b> <b>SUM</b> := <b>SUM</b> + a <b>END_FOR</b>	<b>exit statement:</b> The exit statement can be used to abort the execution of an iteration statement.

Figure 6-3: Table of statements in ST

## Inserting statements using the Edit Wizard

As already mentioned above, the most comfortable and fault preventing method to edit the code body (i. e. to insert statements) is to use the Edit Wizard.



A general description of the Edit Wizard can be found in the section 'The Edit Wizard' in chapter 'Getting started' in this manual.

In ST worksheets the Edit Wizard contains keywords, functions and function blocks, which can be easily inserted into the code body. Using the Edit Wizard provides the following advantages:

- \* It prevents from entering syntactical faults, such as missing semicolons, selection or iteration statements without END keywords, etc. This is done by inserting pre-edited statements, functions or function blocks, i. e. the statement structure is already completed by place holders. The specific variables and values are inserted as comments, which the user may simply overwrite.

- \* It is not necessary, that the user knows the syntax of all different statements, such as functions or function blocks.

If the Edit Wizard is not visible in the workspace, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



### Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

The following procedure describes the steps how to insert a CASE statement into the code body using the Edit Wizard with the mouse. It is assumed, that the Edit Wizard is visible. Otherwise, call the Edit Wizard as described above.



### Inserting a CASE statement using the Edit Wizard with the mouse

- Locate the code body position, where the new statement is to be inserted. Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Keywords'. The available keywords are displayed in the selection area of the Wizard.
- Locate the keyword 'CASE'.
- Double click on the CASE statement. It is inserted automatically at the text cursor position as shown below. The actual variables and values are replaced by comments (green text, enclosed by parentheses and asterisks).

```

CASE (*EXPRESSION (must return an INT value*) OF
(* VALUE a*): (*STATEMENTS*); (* VALUE can be a single value*)
(* VALUES b*): (*STATEMENTS*); (* or a set of VALUES *)
(* . : . *) (* for Example: *)
(* . : . *) (* 1 : ....; *)
(* VALUE x*): (*STATEMENTS*); (* 2..4: ....; *)
ELSE (*STATEMENTS*);
END_CASE;

```

Text cursor

Comments, used as place holders. They have to be overwritten by the user.

Comments, used for code body documentation.

Figure 6-1: Pre-edited CASE statement, inserted using the Edit Wizard

The next step is to overwrite the comments, which serve as place holders with the necessary variables and values.



### Editing the new inserted statement

- Mark the green colored place holders and replace them by statements and expressions (variables and values).

---

## Inserting variables

While editing in ST there are three possibilities for using variables:

- \* Inserting variables, which have already been declared in the variable worksheet of the POU.
- \* Inserting variables, which have not been declared before, and declaring them afterwards in the variable worksheet of the POU.
- \* Inserting a variable and declaring it while editing.

In the first two cases, the variable is inserted in the code body worksheet by just typing the variable name.

In the following section, the last case is described. Let us assume that you want to insert the variable 'T\_value' in your code body worksheet.



### Inserting a variable with the mouse

- Type the variable at the desired code body position.
- Mark it with a left mouse double click on the variable name.
- Click on the icon 'Variable' in the toolbar. The dialog 'Variable' appears. The *variable name* is displayed in the field 'Variable list of POU *POUname*'.

```
T_value := 100;
```

```
T_value := 100;
```



### Inserting a variable with the keyboard

- Type the variable at the desired code body position.
- Mark the variable name by holding the <SHIFT> key and pressing <→> repeatedly.
- Press <F5>. The dialog 'Variable' appears. The *variable name* is displayed in the field 'Variable list of POU *POUname*'.

```
T_value := 100;
```

```
T_value := 100;
```





It is also possible to open the dialog 'Variable' without having marked the variable to be declared. In this case the dialog is opened with the empty field 'Variable list of POU *POUname*'.

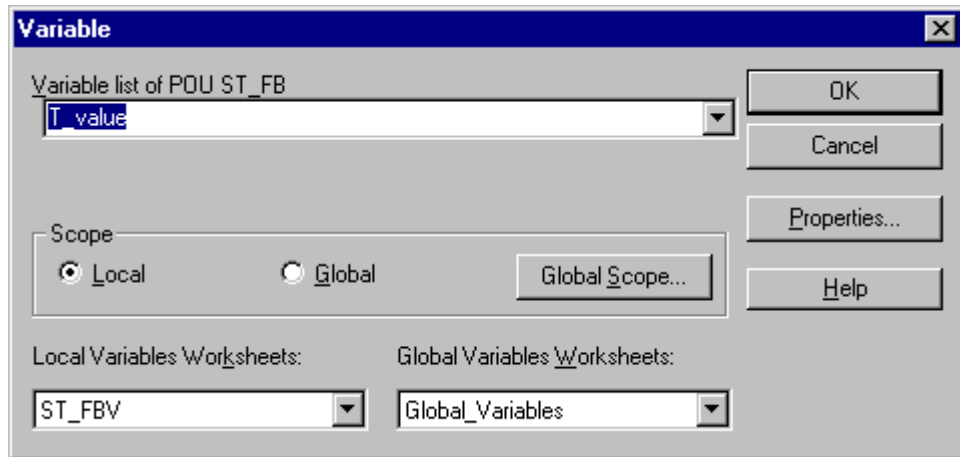


Figure 6-2: Dialog 'Variable'



### Using the dialog 'Variable'

- Enter a variable name if not already shown (refer to the note above).
- Confirm the dialog 'Variable'.  
The dialog 'Automatic Variables Declaration' appears.

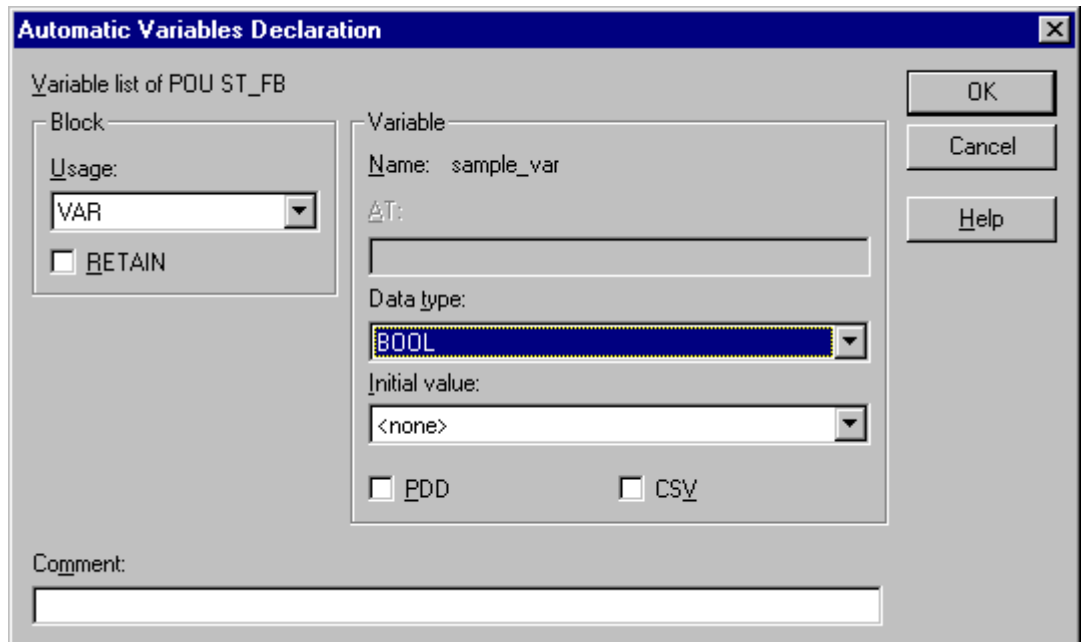


Figure 6-3: Dialog 'Automatic Variables Declaration'



## Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.
- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in an OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



When inserting variables, which have already been declared before, the name of the variable appears in the listbox in the dialog 'Variable'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC Server can be found in the 'OPC Server Manual'.

---

# Calling functions or function blocks using the Edit Wizard

**Functions** can be used in any expression e.g. within assignment statements as it is shown in the following figure:

```
variable name      :=      function name(invar1, invar2);
```

Figure 6-4: Function call in ST

In the example the value of the function on the right is copied to the variable on the left. Instead of the variables 'invar1' and 'invar2' also constants can be used. In this way, either standard functions or user defined functions can be called.

**Function blocks** can be called as a single statement using the instance name, as it is shown in the following figure:

```
instance(invar1:=1, invar2:=2);  
a:= instance.outvar1;
```

Figure 6-5: Function block call in ST

The order of the formal parameters does not have any importance. If parameters are missing, the initial value or the value of the last call is used. There is no difference in calling either standard function blocks or user defined function blocks.



Do not forget to declare the instance of the function block in the variable declaration of the POU, as it is described in the section 'Instantiation' in chapter 'Literals, variables and data types'.

The most comfortable and fault preventing method to insert function or function blocks into the code body is to use the Edit Wizard.

If the Edit Wizard is not visible in the workspace, perform the following steps:



## Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



## Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

The following procedures describe the steps how to insert a MAX function and a CTU function block into the code body using the Edit Wizard with the mouse. It is assumed, that the Edit Wizard is visible. Otherwise, call the Edit Wizard as described above.



### Inserting a MAX function using the Edit Wizard with the mouse

- Locate the code body position, where the new function is to be inserted. Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Functions'. The available functions are displayed in the selection area of the Wizard.
- Locate the function 'MAX'.
- Double click on the function 'MAX'. It is inserted automatically at the text cursor position as shown in the following figure. Replace the green comments (enclosed by parentheses and asterisks) with the necessary elements.



### Inserting a CTU function block using the Edit Wizard with the mouse

- Locate the code body position, where the new function block is to be inserted. Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Function blocks'. The available function blocks are displayed in the selection area of the Wizard.
- Double click on the function block 'CTU'. The dialog 'FB Instances' appears. The field 'FB Instances' displays the default instance name (e.g. for a CTU the name 'CTU\_n' is proposed, where *n* is the first available number which is free for this instance name). To define the name of the new FB you have the following possibilities:
  - \* Enter a new instance name in the field.
  - \* Accept the proposed name.
  - \* Select an already existing name in the list box 'FB Instances'.
- Press 'OK' to confirm the dialog. If you have entered a new instance name, the dialog 'Automatic FB Declaration' appears.
- Enter a comment if you want and press 'OK' to confirm the dialog. The function block 'CTU' is inserted automatically at the text cursor position as shown in the following figure . Replace the green comments (enclosed by parentheses and asterisks) with the necessary elements.

```
1 (* Result as ELEMENTARY *):=MAX(* IN1 as ELEMENTARY *),(* IN2 as ELEMENTARY *);
2
3 CTU_1(CU:=(* BOOL *),RESET:=(* BOOL *),PV:=(* INT *));
4 (* BOOL *):=CTU_1.Q;
5 (* INT *):=CTU_1.CV;
6
```

Pre-edited function 'MAX'

Pre-edited function block 'CTU', named CTU\_1

Green comments, which serve as placeholders and have to be overwritten by the actual values and names

Figure 6-6: Pre-edited MAX function and CTU function block, inserted using the Edit Wizard

# Editing in IL

**This chapter provides information about...**

- calling the text editor with an IL worksheet
- instructions, operators, modifiers and operands
- inserting instructions using the Edit Wizard
- inserting variables
- using jumps and labels
- calling functions or function blocks using the Edit Wizard

# Editing in IL

---

## Calling the text editor with an IL worksheet

The first step before editing an IL code body worksheet is to call the text editor with the IL worksheet, using the project tree.



A general description of handling the project tree and browsing through POU and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

As an example, let us assume that you want to edit the IL worksheet of a function block which is called IL\_FB. For that purpose, you first have to insert a function block with this name as it is described in the chapter 'Editing the project structure'. Keep in mind, that the programming language of the worksheet is determined by the POU language. It is set by inserting the POU respectively the worksheet.



### Calling the text editor for the IL code body worksheet with the mouse

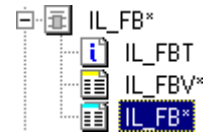
- If the desired worksheet icon is not visible in the project tree, open the corresponding subtree, containing the POU worksheets. For this purpose double click on the POU name (e.g. 'IL\_FB').
- Double click on the icon 'Worksheet in IL' of the function block 'IL\_FB'.  
The text editor with the IL worksheet appears.





## Calling the text editor for the IL code body worksheet with the keyboard

- If the desired worksheet icon is not visible in the project tree, open the 'IL\_FB' subtree, containing the POU worksheets as follows: Press < > or < > to highlight the function block 'IL\_FB'. Press <→> to open the function block subtree.
- Press < > or < > to mark the desired icon 'Worksheet in IL' of the function block 'IL\_FB'.
- Press <↵>. The text editor with the IL worksheet appears.



---

## Instructions, operators, modifiers and operands

An instruction list consists of several instructions. Each instruction starts at a new line and contains an operator. Modifier and operand are optional. A comment can be entered at the end of the line using parentheses and asterisks. A line number is displayed in front of each line. The syntax highlighting represents the different elements by colors: operators and their modifiers are blue; variables and operands are black; comments are green. An instruction list with three instructions is shown in the following example:

```
LD  %IX2.2      (* input value *)
ADD value      (* add value *)
ST  %QX2.2      (* result of addition *)
```

Figure 7-1: Example of an instruction list

In the example each instruction consists of an operator, an operand and a comment.

While editing in IL the following operators, modifiers and operands can be used:



Operator	Modifier	Operand	Description
LD	N	ANY	Set current result equal to operand
ST	N	ANY	Store current result to operand location
S		BOOL	Set Boolean operand to 1 if the current result is 1
R		BOOL	Set Boolean operand to 0 if the current result is 1
AND	N,(	ANY_BIT	Boolean AND
OR	N,(	ANY_BIT	Boolean OR
XOR	N,(	ANY_BIT	Boolean exclusive OR
ADD	(	ANY_NUM	Addition
SUB	(	ANY_NUM	Subtraction
MUL	(	ANY_NUM	Multiplication
DIV	(	ANY_NUM	Division
GT	(	ANY_NUM + ANY_BIT	Comparison: >
GE	(	ANY_NUM + ANY_BIT	Comparison: >=
EQ	(	ANY_NUM + ANY_BIT	Comparison: =
NE	(	ANY_NUM + ANY_BIT	Comparison: <>
LE	(	ANY_NUM + ANY_BIT	Comparison: <=
LT	(	ANY_NUM + ANY_BIT	Comparison: <
JMP	C, N	LABEL	Jump to label
CAL	C, N	NAME	Call function block
RET	C, N		Return to the calling function block or program
)			Evaluate deferred operation

Figure 7-2: Table of operators, modifiers and operands in IL

The meaning of the modifiers is described in the following table:

Modifier	Description
N	negated
(	process the included expression first
C	conditional

Figure 7-3: Table of the modifiers in IL and their meaning

## Inserting instructions using the Edit Wizard

Instructions can be inserted by just typing them or using the Edit Wizard. The Edit Wizard is a feature to simplify editing in the text editors. In IL it contains a lot of standard operators, functions and function blocks which can be inserted.



A general description of the Edit Wizard can be found in the section 'The Edit Wizard' in chapter 'Getting started' in this manual.

Using the Edit Wizard provides the following advantages:

- \* It prevents from entering syntactical faults, such as wrong instruction sequences. This is done by inserting pre-edited operators, functions or function blocks, i. e. the structure is already completed by placeholders. The specific variables and values are inserted as comments, which the user simply has to overwrite.
- \* It is not necessary, that the user knows the syntax of all different operations, such as functions or function blocks.

Example: After inserting the operator 'ADD' and the function block 'RS' using the Edit Wizard, the following instruction list is visible in your IL worksheet:

```

1  LD  (* IN1 as ANY_NUM *)
2  ADD (* IN2 as ANY_NUM *)
3  ST  (* Result as ANY_NUM *)
4
5  LD  (* BOOL *)
6  ST  RS_1.SET
7  LD  (* BOOL *)
8  ST  RS_1.RESET1
9  CAL RS_1
10 LD  RS_1.Q1
11 ST  (* BOOL *)
12

```

Pre-edited operator 'ADD'

Pre-edited function block 'RS', named RS\_1

Green comments, which serve as placeholders and have to be overwritten by the actual values and names

Figure 7-4: Operator 'ADD' and FB 'RS', both inserted using the Edit Wizard

If the Edit Wizard is not visible in the workspace, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



### Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

The following procedure describes the steps how to insert an ADD operator into the code body using the Edit Wizard with the mouse. It is assumed, that the Edit Wizard is visible. Otherwise, call the Edit Wizard as described above.



To insert functions or function blocks into the code body, the same procedure must be performed with the exception of selecting the appropriate function or function block group in the Edit Wizard. This procedure is described in detail in section 'Calling functions or function blocks using the Edit Wizard' in this chapter.



### Inserting an ADD operator using the Edit Wizard

- Locate the code body position, where the new instructions are to be inserted. Click the left mouse button to set a text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Operators'. The available operators are displayed in the selection area of the Wizard.
- Locate the operator 'ADD'.
- Double click on the operator. It is inserted automatically at the text cursor position as shown in figure 7-4. The actual variables and values are replaced by comments (green text, enclosed by parentheses and asterisks).

The next step is to overwrite the comments, which serve as place holders with the necessary variables and values.



### Editing the new inserted instructions

- Mark the green colored place holders and replace them by the actual operands (variables and values).

- Now you have to declare the variables which have been inserted with the operator. To perform this, double click on the variable to mark it. Then press <F5> to open the dialog 'Variable'. You can also change the names of the variables if necessary.

---

## Inserting variables

If you are editing IL instructions manually by just typing them you have two possibilities to insert variables. The first possibility is typing the variable name in the code body worksheet and declaring the variable using the variable editor or the variable dialog.

You can also insert variables by clicking on the icon 'Variable' in the toolbar and using the dialog 'Variable'. In a second example at the end of this section, we assume, that you want to insert a variable which has already been declared in the global variable declaration of your project.

In the first example let us assume, that you have inserted the operator 'ADD' using the Edit Wizard. The required variable 'invar' is not yet declared in the variable worksheet 'IL\_FBv'.



### Declaring a variable with the mouse

- Double click on the variable name to be declared.
- Click on the icon 'Variable' in the toolbar. The dialog 'Variable' appears. The field 'Variable list of POU *POUname*' displays the current *variable name*.

ADD invar



### Declaring a variable with the keyboard

- Mark the variable name by holding the <SHIFT> key and pressing <→> repeatedly.
- Press <F5>. The dialog 'Variable' appears. The field 'Variable list of POU *POUname*' displays the current *variable name*.

ADD invar



It is also possible to open the dialog 'Variable' without having marked the variable to be declared. In this case the dialog is opened and the field 'Variable list of POU *POUname*' is empty.

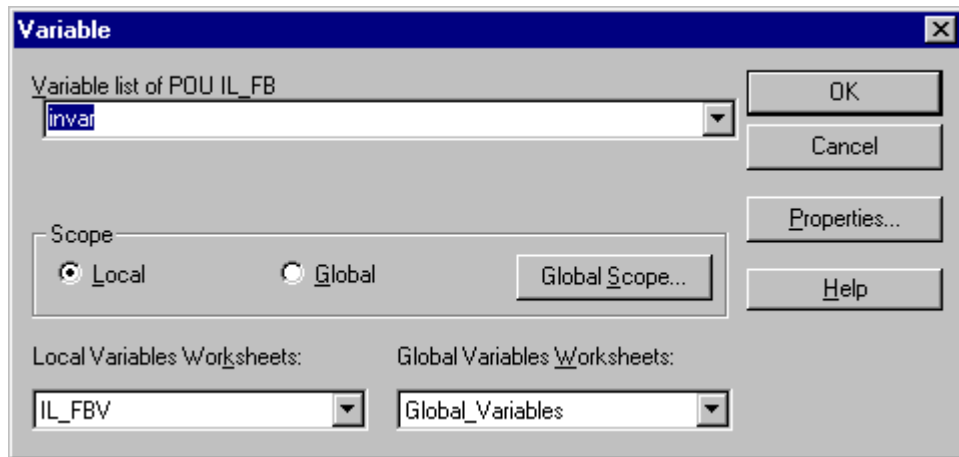


Figure 7-5: Dialog 'Variable', called with a variable marked



### Using the dialog 'Variable'

- Enter a variable name if not already shown (refer to the note above).
- Confirm the dialog 'Variable'.  
The dialog 'Automatic Variables Declaration' appears.

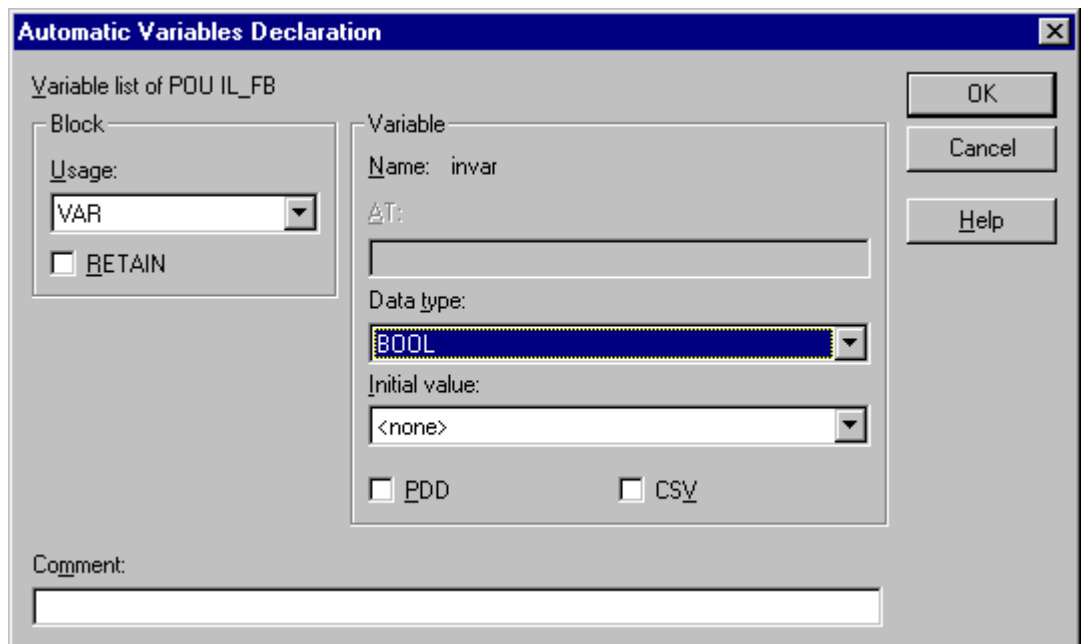


Figure 7-6: Dialog 'Automatic Variables Declaration'



### Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.

- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in an OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



When inserting variables, which have already been declared before, the name of the variable appears in the listbox of the dialog 'Variable'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC server can be found in the 'OPC Server Manual'.

In a second example let us assume, that you want to insert a variable which has already been declared in the global variable declaration of your project.



### Calling the dialog 'Variable' with the mouse

- Click on the icon 'Variable' in the toolbar.  
The empty dialog 'Variable' appears.



### Calling the dialog 'Variable' with the keyboard

- Press <F5>.  
The empty dialog 'Variable' appears.

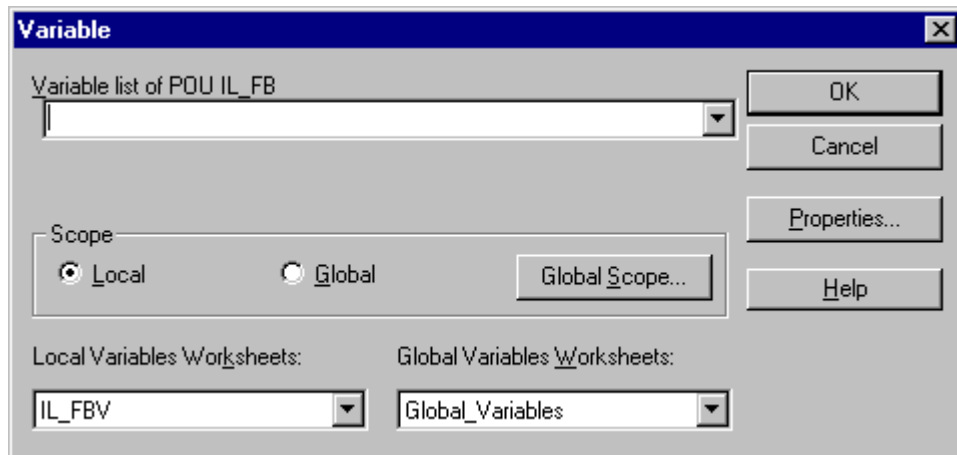


Figure 7-7: Dialog 'Variable'



### Using the dialog 'Variable' to insert a variable, which has already been declared in the global variable declaration of the project

- Activate the radio button 'Global'.
- Check if the correct resource is listed.
- Choose the desired variable in the list box 'Variable list of *resource name*'.
- Confirm the dialog.  
The variable is inserted in the code body worksheet and additionally in the variable declaration of the POU as VAR\_EXTERNAL since it is a global variable.

## Using jumps and labels

Jumps can be used to jump to a line of the instruction list. In these cases the operator 'JMP' and a label in front of the destination line is used as it is shown in the following example:

```

LD      value1
EQ      INT#100
JMPC   label
LD      value2
ADD     value3
ST      value4
label:  LD      %IX2.2

```

Figure 7-8: Example of a jump

In the example a jump to the label is executed if 'value1' is 100. If it is not 100 no jump is executed and 'value2' is loaded.

---

## Calling functions or function blocks using the Edit Wizard

**Functions** can be called in an instruction list by placing the function name in the operator field. In the following figure an example of calling a function with three input parameters and one output parameter is shown:

```
LD  inpar1
   function name      inpar2, inpar3
ST  var1
```

Figure 7-9: Example of calling a function

The first declared input parameter has to be loaded in the preceding line of the function call. In the example the first input parameter is 'inpar1'. All other input parameters have to be used in the second line as operands separated by commas. The result is stored to the variable 'var1' as it is shown in the last line of the figure.



This way either standard functions or user defined functions can be called.

**Function blocks** can be called using the operator CAL and the name of the function block in the operand field. Calling a function block needs more typing effort than calling a function. For the next example let us imagine a function block with two input parameters 'inpar1' and 'inpar2' and two output parameters 'outpar1' and 'outpar2'. The function block is called 'FB\_exam'. Its instance name is '*instance*'.

For this example the function block call should look like the following figure:

```
LD  var1
ST  instance.inpar1
LD  var2
ST  instance.inpar2
CAL FB_exam
LD  instance.outpar1
ST  var3
LD  instance.outpar2
ST  var4
```

Figure 7-10: Example of calling a function block

The function block call consists of three parts: input parameter introduction, the proper call with the operator CAL and storing the output parameters.



Do not forget to declare the instances of function blocks in the variable declaration of the POU as it is described in the chapter 'Literals, data types and variables'.



The most comfortable and fault preventing method to insert functions or function blocks into the code body is to use the Edit Wizard.

If the Edit Wizard is not visible in the workspace, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



### Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

The following procedures describe the steps how to insert a MAX function and an CTU function block into the code body using the Edit Wizard with the mouse. It is assumed, that the Edit Wizard is visible. Otherwise, call the Edit Wizard as described above.



### Inserting a MAX function using the Edit Wizard with the mouse

- Locate the code body position, where the new function is to be inserted. Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Functions'. The available functions are displayed in the selection area of the Wizard.
- Locate the function 'MAX'.
- Double click on the function 'MAX'. It is inserted automatically at the text cursor position as shown in the following figure. Replace the green comments (enclosed by parentheses and asterisks) with the necessary elements.



### Inserting a CTU function block using the Edit Wizard with the mouse

- Locate the code body position, where the new function block is to be inserted. Click the left mouse button to position the text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Function blocks'. The available function blocks are displayed in the selection area of the Wizard.

- Double click on the function block 'CTU'. The dialog 'FB Instances' appears. The field 'FB Instances' displays the default instance name (e.g. for a CTU the name 'CTU\_n' is proposed, where *n* is the first available number which is free for this instance name). To define the name of the new FB you have the following possibilities:
  - \* Enter a new instance name in the field.
  - \* Accept the proposed name.
  - \* Select an already existing name in the list box 'FB Instances'.
- Press 'OK' to confirm the dialog. If you have entered a new instance name, the dialog 'Automatic FB Declaration' appears.
- Enter a comment if you want and press 'OK' to confirm the dialog. The function block 'CTU' is inserted automatically at the text cursor position as shown in the following figure. Replace the green comments (enclosed by parentheses and asterisks) with the necessary elements.

```

1  LD  (* IN1 as ELEMENTARY *)
2  MAX (* IN2 as ELEMENTARY *)
3  ST  (* Result as ELEMENTARY *)
4
5  LD  (* BOOL *)
6  ST  CTU_1.CU
7  LD  (* BOOL *)
8  ST  CTU_1.RESET
9  LD  (* INT *)
10 ST  CTU_1.PV
11 CAL CTU_1
12 LD  CTU_1.Q
13 ST  (* BOOL *)
14 LD  CTU_1.CV
15 ST  (* INT *)

```

Pre-edited function 'MAX'

Pre-edited function block 'CTU', named CTU\_1

Green comments, which serve as placeholders and have to be overwritten by the actual values and names

Figure 7-11: Pre-edited MAX function and CTU function block, inserted using the Edit Wizard

# Editing in FBD

**This chapter provides information about...**

- calling the graphic editor with a FBD worksheet
- introduction to FBD
- inserting functions and function blocks using the Edit Wizard
- changing the properties of functions and function blocks
- replacing functions and function blocks
- inserting variables
- connecting objects
- negation of inputs and outputs
- duplicating inputs of functions

# Editing in FBD

---

## Calling the graphic editor with a FBD worksheet

Before editing a FBD code body worksheet you have to call the graphic editor with the FBD worksheet, using the project tree.



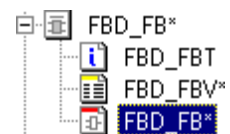
A general description of handling the project tree and browsing through POU and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

As an example, let us assume that you want to edit the FBD worksheet of a function block which is called FBD\_FB. For that purpose you first have to insert a function block with this name as it is described in the chapter 'Editing the project structure'. Keep in mind that the language of the worksheet is determined by the POU language. It is set by inserting the POU respectively the worksheet.



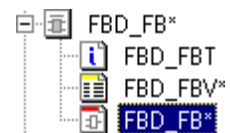
### Calling the graphic editor for the FBD code body worksheet with the mouse

- If the desired worksheet icon is not visible in the project tree, open the corresponding subtree, containing the POU worksheets. For this purpose double click on the POU name (e.g. 'FBD\_FB').
- Double click on the icon 'Worksheet in FBD' of the function block 'FBD\_FB'. The graphic editor with the FBD worksheet appears.



### Calling the graphic editor for the FBD code body worksheet with the keyboard

- If the desired worksheet icon is not visible in the project tree, open the 'FBD\_FB' subtree, containing the POU worksheets as follows: Press < > or < > to highlight the function block 'FBD\_FB'. Press <=> to open the function block branch.



- Press < > or < > to mark the icon 'Worksheet in FBD' of the function block 'FBD\_FB'.
- Press <↵>. The graphic editor with the FBD worksheet appears.

---

## Introduction to FBD

A code body which is programmed in the graphical language FBD (Functional Block Diagram) consists of functions, function blocks or variables which are connected by lines. A line can also be connected to a line. The set of connected objects is called a FBD network. In FBD networks it is not possible to connect outputs with outputs.

In FBD code bodies, comments can be inserted using asterisks and parentheses.

Functions and function blocks can be inserted using the Edit Wizard and edited using the dialog 'Function/Function Block'. Both procedures are described in this chapter. The Edit Wizard is a feature to simplify editing in the graphical editor. It contains all functions and function blocks which can be inserted.

The graphical programming language FBD provides many additional features, which facilitate the creation of a program, FB or function code body, e.g:

- \* The graphic editor provides simple keyboard operations for insertion and scrolling (Cursor keys /CTRL Cursor keys for object mode and SHIFT/SHIFT CTR Cursor keys for Mouse cursor mode).
- \* Duplication of inputs can be done directly via keyboard, toolbar and menu.
- \* Negation of Inputs, Outputs, Contacts and Coils can be done directly via keyboard, toolbar and menu.
- \* Easy auto routing for standard editing cases.
- \* Items can be inserted directly on a line or can be connected to the inputs or outputs of present items.
- \* Splitter and overview windows are available.
- \* Freestyle editing allows to arrange items wherever you want.
- \* Double clicking on user functions and function blocks opens the contents of the corresponding POU.

---

# Inserting functions and function blocks using the Edit Wizard



A general description of the Edit Wizard can be found in the section 'The Edit Wizard' in chapter 'Getting started' in this manual.

In FBD worksheets the Edit Wizard contains functions and function blocks, which can be easily inserted into the graphical language code body.

If the Edit Wizard is not visible in the workspace, perform the following steps:



## Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



## Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>.  
The Edit Wizard window appears.

The following procedure describes the steps how to insert a CTU function block into the FBD worksheet using the Edit Wizard with the mouse. It is assumed that the Edit Wizard is visible. Otherwise, call the Edit Wizard as described above.



## Inserting the CTU function block using the Edit Wizard

- Click into the worksheet to set an insertion mark.
- Open the Edit Wizard list box 'Group' and select the group 'Function blocks'. The available function blocks are displayed in the selection area of the Wizard.
- Double click on the function block 'CTU'. The dialog 'FB Instances' appears. The field 'FB Instances' displays the default instance name (e.g. for a CTU the name 'CTU\_1' is proposed, where *n* is the first available number which is free for this instance name).  
To define the name of the new FB you have the following possibilities:
  - \* Enter a new instance name in the field.
  - \* Accept the proposed name.
  - \* Select an already existing name in the list box 'FB Instances'.
- Press 'OK' to confirm the dialog. If you have changed the instance name, the dialog 'Automatic FB Declaration' appears. In this case enter a comment if you want and press 'OK'.  
The function block 'CTU' is automatically inserted into the worksheet and the declaration of the function block instance is autoinserted into the variable declaration of the POU.

With the next steps, the properties of the inserted function/function block are edited and the element is connected to the FBD network.



The procedures how to change the properties of functions and function blocks are described in the following section 'Changing the properties of functions and function blocks'. For a description concerning the connection of objects refer to the section 'Connecting objects' in this chapter.

---

# Changing the properties of functions and function blocks

You can change the properties of any function or function block inserted into the FBD worksheet by calling the dialog 'Function/Function Block'.



## Calling the dialog 'Function/Function Block' for an existing function/function block

- In the FBD worksheet click on the function or function block to be edited. The marked object changes its color.
- Click with the right mouse button on the marked function or function block to open the context menu for this object.
- Select the menu item 'Object properties'. The dialog 'Function/Function Block' appears.

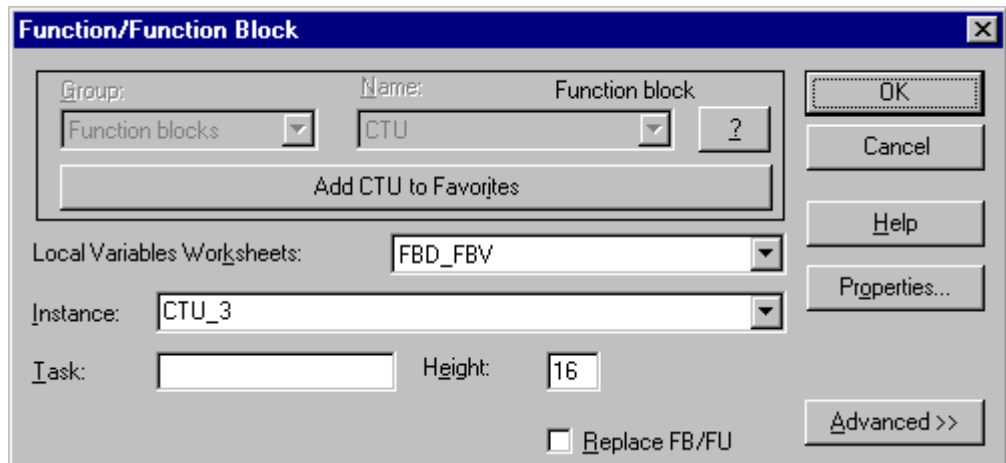


Figure 8-1: Dialog 'Function/Function Block'



## Using the dialog 'Function/Function Block'

- The list boxes 'Group' and 'Name' are grayed and inactive because the object is already created.
- Change the instance name if you want.
- Confirm the dialog.  
If you have changed the instance name of a function block, the dialog 'Automatic FB Declaration' appears. Enter a comment if you want and press 'OK' to confirm the dialog. The declaration of the function block instance is autoinserted into the variable declaration of the POU.



---

## Replacing functions and function blocks

To replace a specific function or function block by another, you can either use the Edit Wizard or the dialog 'Function/Function Block'.



### Replacing an object using the Edit Wizard

- Click on the specific object to be replaced. The marked object changes its color.
- Replace the marked object by double clicking on a function/function block in the selection area of the Edit Wizard as described in section 'Inserting functions and function blocks using the Edit Wizard'.
- Edit, if necessary, the new object as described in section 'Changing the properties of functions and function blocks'.



### Replacing an object using the dialog 'Function/Function block'

- Click on the specific object to be replaced. The marked object changes its color.
- Click with the right mouse button on the marked object to open the context menu. Choose 'Object properties'. The dialog 'Function/Function Block' appears with the list boxes 'Group' and 'Name' grayed.
- Click on the check box 'Replace FB/FU'. The list boxes 'Group' and 'Name' become active.
- Define the new object to be inserted by selecting the required entries in the list boxes 'Group' and 'Name'. An instance name is proposed by the system.
- Continue editing the new object as described in section 'Changing the properties of functions and function blocks'.

---

## Inserting variables

For inserting variables in a FBD worksheet you have two possibilities:

- \* Inserting a variable which has already been declared as described in the section 'Declaring variables' in the chapter 'Literals, data types and variables'.
- \* Inserting and declaring a new variable.

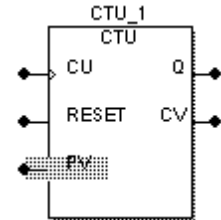
Variables can be inserted anywhere in the FBD worksheet or directly connected to an input/output (formal parameter) of functions or function blocks.

For the next steps let us assume, that you want to insert a variable directly connected to the formal parameter 'PV' of the function block 'CTU'. This variable has not yet been declared.



### Inserting a variable at the formal parameter 'PV' with the mouse

- Click on the formal parameter 'PV' to mark it. The marked input changes its color.



- Click on the icon 'Variable' in the toolbar. The dialog 'Variable' appears.



### Inserting a variable at the formal parameter 'PV' with the keyboard

- Press the cursor keys to mark the formal parameter 'PV'.
- Press <F5>. The dialog 'Variable' appears.

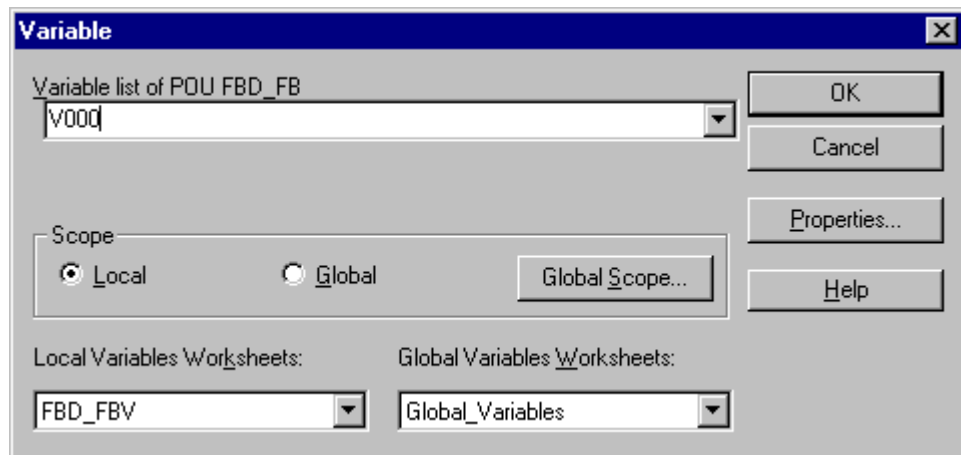


Figure 8-2: Dialog 'Variable'



### Using the dialog 'Variable'

- Enter the variable name 'VAR1'.
- Confirm the dialog. The dialog 'Automatic Variables Declaration' appears.

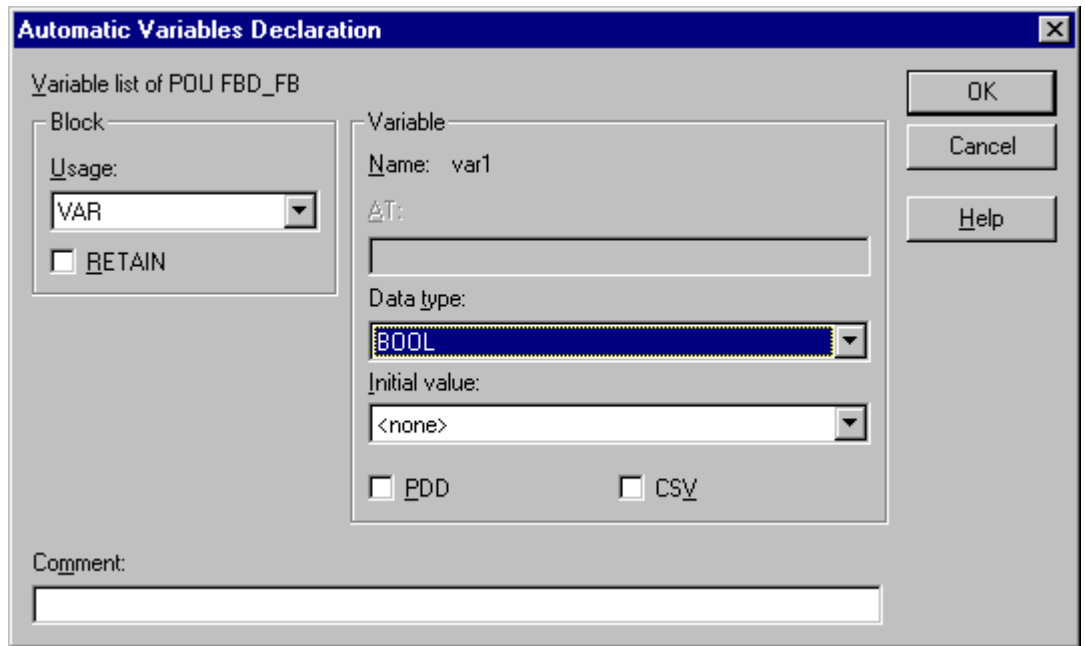


Figure 8-3: Dialog 'Automatic Variables Declaration'



### Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.
- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in an OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



When inserting variables, which have already been declared before, the name of the variable appears in the list box of the dialog 'Variable'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC Server can be found in the 'OPC Server Manual'.

---

## Connecting objects

If you have inserted functions, function blocks or variables into your worksheet, you have to connect them to create a legal FBD network. To connect objects in a FBD network you have the following possibilities:

- \* connecting objects using the connection mode
- \* connecting objects by drag & drop with the mouse
- \* connecting objects while inserting a new object

The connection mode can be used to connect all objects which have free connection points. Connection points are represented as green and blue dots. It is also possible to connect a new line to an existing one using the connection mode.



Objects can be linked only using a horizontal line. So your line should start at a connection point of an object and move away from the object in a horizontal way.

While connecting inputs and outputs of functions and function blocks the program provides an auto routing feature. Auto routing means that the program will determine the routing of the connection line between two free connection points automatically. In this case there is no need to mark any corner.



Auto routing is only available if you connect an input to an output or vice versa. If you create a feedback or the connection starts at a connection line, auto routing is not possible.

For the following procedures let us assume that you want to connect the function block 'CTU' with the function 'ADD'.



### Connecting a function block output to a function input using the connection mode with auto routing

- Click on the icon 'Connect objects' in the toolbar. A symbol for a connection is added to the cursor. During the connection mode, the icon appears 'pressed'.
- Click on the output of the function block 'CTU', which you want to connect.



- Move the mouse to the input of the function 'ADD' you want to connect. The connection line is displayed red. If the program recognizes the desired connection point, the line is routed automatically and the color of the line changes to green.
- Click on the input.  
The new connection line is auto routed and inserted automatically.



### Connecting two objects using the connection mode without auto routing

- Click on the icon 'Connect objects' in the toolbar. A symbol for a connection is added to the cursor. During the connection mode, the icon appears 'pressed'.
- Click at the point where you want to start drawing the line.
- Move the mouse to the object you want to connect.
- Click with the left mouse button to mark a corner if you want.
- Click at the point where you want the line to end.



### Connecting two objects by drag & drop with the mouse

- Click on the function 'ADD' and keep the mouse button pressed.
- Move the mouse towards the function block 'CTU' so that the connection points overlap.
- Release the mouse button.  
The connection is set.
- If required, move either the function or the function block to a vacant position.

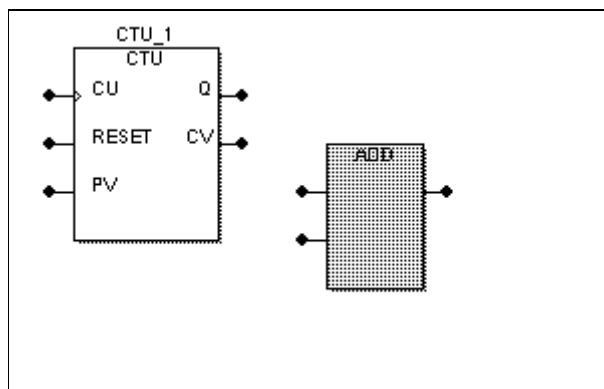


Figure 8-4: Function and function block before establishing the connection

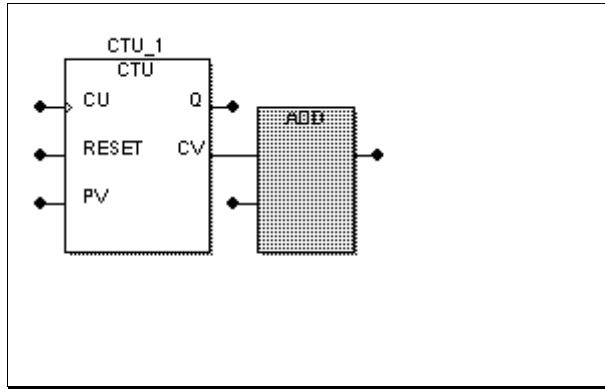


Figure 8-5: The connection is set

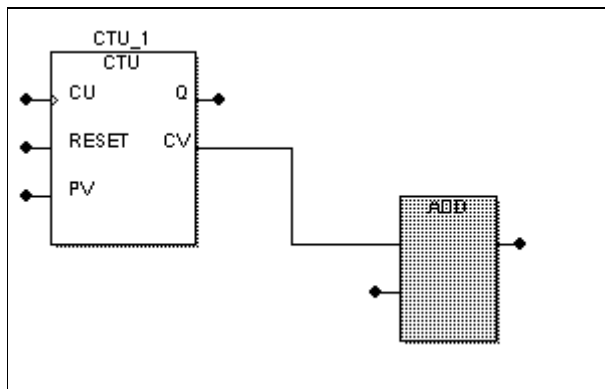


Figure 8-6: The function is moved to a vacant position. The connection is routed automatically.

You can also connect a function or function block to another function or function block by marking an output parameter and then inserting the object as described in the following procedure.



### Connecting an object to a new function/function block while inserting

- Click on the output 'CV' of the function block 'CTU', before the function 'ADD' is inserted. The marked output changes its color.
- Insert the new function 'ADD' using the Edit Wizard as described in the section 'Inserting functions and function blocks using the Edit Wizard'. The connection between the highlighted output 'CV' and the first input of the 'ADD' function is established automatically.
- If required, move either the function or the function block to a vacant position.

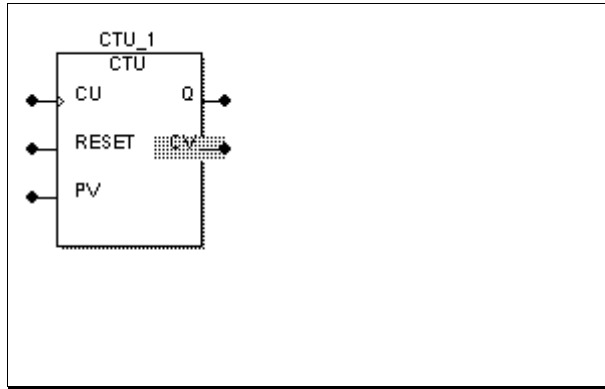


Figure 8-7: The output 'CV' of the FB 'CTU' is defined to be one end of the connection before the 'ADD' function is inserted

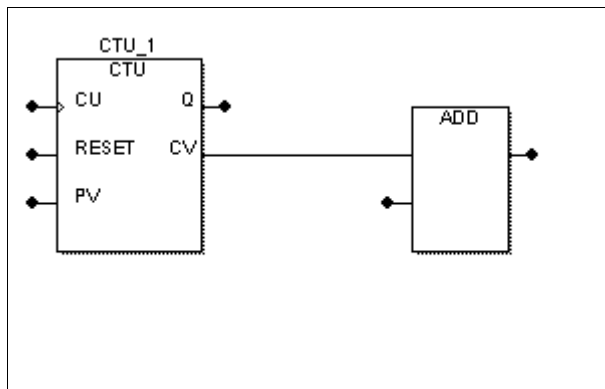


Figure 8-8: The connection is established automatically when the new 'ADD' function has been inserted

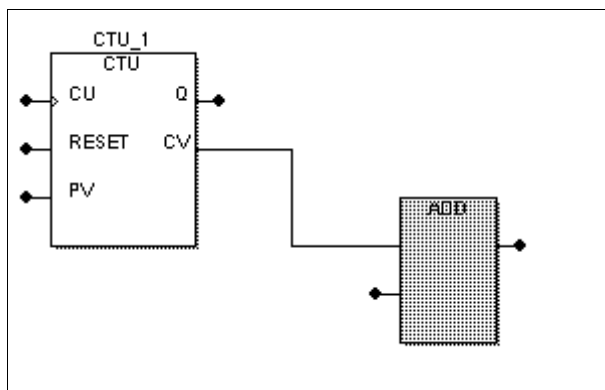


Figure 8-9: The function is moved to a vacant position. The connection is routed automatically.

---

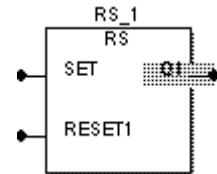
## Negation of inputs and outputs

Using the graphic editor, it is easy to negate and to toggle the negation of inputs or outputs of functions and function blocks. There are several possibilities to negate a FP (Formal Parameter). They are described in this section using the output Q1 of an RS function block as an example.



### Negating a formal parameter 'Q1' using the mouse

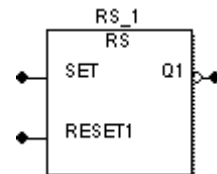
- Click on the formal parameter 'Q1' to mark it. The marked output changes its color.



- Click on the icon 'Toggle negation of FP' in the toolbar.

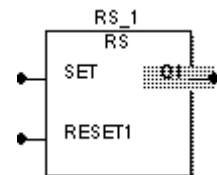


The output is shown with a circle as negation symbol. To toggle the negation, just perform the described steps again for the same output.



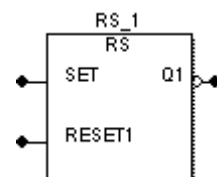
### Negating a formal parameter 'Q1' with the keyboard

- Press the cursor keys to select the formal parameter 'Q1'. The marked output changes its color.



- Press the shortcut <SHIFT> + <F5>.

The output is shown with a circle as negation symbol. To toggle the negation, just perform the described steps again for the same output.





---

## Duplicating inputs of functions

The program allows to duplicate inputs of **extensible** functions. Using this feature you can add as many inputs to an extensible function as required for your application.



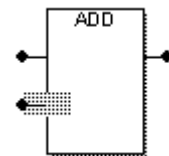
It is only possible to duplicate the last input of a function.

For the following example let us assume that you want to duplicate the input of the extensible function 'ADD'.

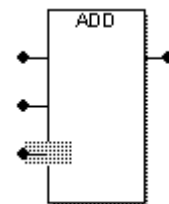


### Duplicating an input of the function 'ADD' with the mouse

- Click on the last input of the function. The marked input changes its color.
- Click on the icon 'Duplicate FP' in the toolbar.

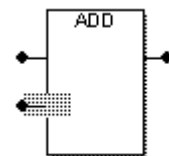


The new input is added below the previously selected input.

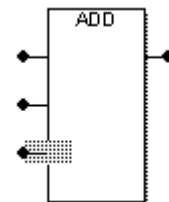


### Duplicating an input of the function 'ADD' with the keyboard

- Press the cursor keys to select the last input. The marked input changes its color.
- Press <CTRL> + <F5>.



The new input is added below the previously selected input.



# Editing in LD

**This chapter provides information about...**

- calling the graphic editor with a LD worksheet
- networks, contacts, coils and power rails
- inserting contacts and coils
- inserting serial contacts and coils
- inserting parallel contacts and coils
- using the LD branch edit mode
- changing the properties of contacts and coils
- inserting variables
- calling functions or function blocks using the Edit Wizard

# Editing in LD

---

## Calling the graphic editor with a LD worksheet

The first step before editing a LD code body worksheet is to call the graphic editor with the LD worksheet, using the project tree.



A general description of handling the project tree and browsing through POU's and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

As an example, let us assume that you want to edit the LD worksheet of a function block which is called 'LD\_FB'. For that purpose you first have to insert a function block with this name as it is described in the chapter 'Editing the project structure'. Keep in mind, that the programming language of the worksheet is determined by the POU language. It is set by inserting the POU respectively the worksheet.



### Calling the graphic editor for the LD code body worksheet with the mouse

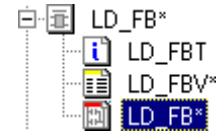
- If the desired worksheet icon is not visible in the project tree, open the corresponding subtree, containing the POU worksheets. For this purpose double click on the POU name (e.g. 'LD\_FB').
- Double click on the icon 'Worksheet in LD' of the function block 'LD\_FB'.  
The graphic editor with the LD worksheet appears.





## Calling the graphic editor for the LD code body worksheet with the keyboard

- If the desired worksheet icon is not visible in the project tree, open the 'LD\_FB' subtree, containing the POU worksheets as follows: Press < > or < > to highlight the function block 'LD\_FB'. Press <=> to open the function block subtree.
- Press < > or < > to mark the icon 'Worksheet in LD' of the function block 'LD\_FB'.
- Press <↵>. The graphic editor with the LD worksheet appears.



## LD networks, contacts, coils and power rails

A code body programmed in the graphic language LD is composed of contacts and coils. According to IEC 61131-3 different types of contacts and coils can be used:

Symbol	Name	Description
--     --	Normally open contact	The Boolean value is copied from the left to the right if the state of the associated variable is ON.
--   /   --	Normally closed contact	The Boolean value is copied from the left to the right if the state of the associated variable is OFF.
-- ( ) --	Coil	The Boolean value is copied from the left to the right and to the associated variable.
-- ( / ) --	Negated coil	The Boolean value is copied from the left to the right. The negated Boolean value is copied to the associated variable.
-- ( S ) --	SET coil	The Boolean value is copied from the left to the right. The associated variable is set if the left link is TRUE.
-- ( R ) --	RESET coil	The Boolean value is copied from the left to the right. The associated variable is reset if the left link is TRUE.

Figure 9-1: Table of contacts and coils in LD

Contacts and coils are connected by lines and are bound on the left and on the right with power rails. The state of the left power rail is considered ON all the time. The right power rail is optional.

In addition to the serial connections of contacts and coils parallel branches can be created. Parallel branches are also called wired-ORs.

The set of connected objects is called a LD network. Every LD network shall contain at least one contact, one coil and a left power rail.

Variables in LD used with contacts and coils are always Boolean variables. While inserting contacts or coils the variable name can be entered. The variable name is displayed above the contact or coil in the worksheet.

In LD code bodies comments can be inserted using asterisks and parentheses.

The following figure shows an example for a simple LD network.

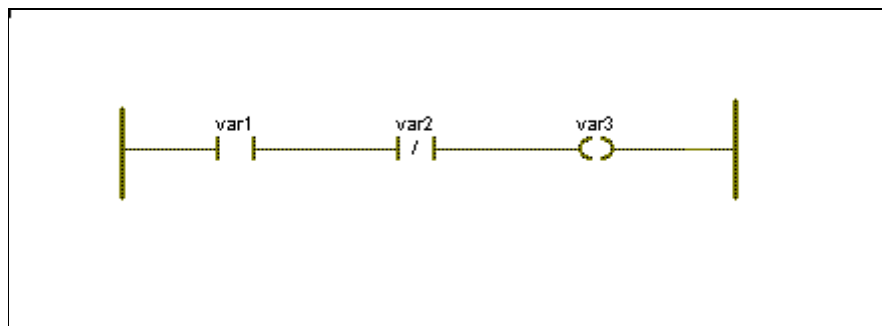


Figure 9-2: Example of a LD network

The example network consists of a left and right power rail which limits the LD network to the left and to the right. In the middle you can see two contacts and one coil connected with horizontal connection lines. The first contact, called the normally open contact, passes the incoming value from the left to the right if the value of variable 'var1' is TRUE. The second contact passes the incoming value if the value of the variable 'var2' is FALSE. The coil stores the incoming value to the linked variable 'var3'.



For compiling your worksheet it is necessary to declare the variables as described in chapter 'Literals, data types and variables'.



In LD code body worksheets it is possible to insert FBD elements manually or using the Edit Wizard. The required steps are described in the section 'Calling functions or function blocks using the Edit Wizard' in this chapter.

---

## Inserting contacts and coils

For the next steps let us assume that you have already inserted a LD function block called 'LD\_FB' as it is described in the chapter 'Editing the project structure'. Assuming furthermore that you want to insert a LD network with four contacts in a first step.



### Inserting a first LD network with the mouse

- Click into the worksheet to set an insertion mark.
- Click on the icon 'Contact network' in the toolbar.  
A first LD network with one contact and one coil is inserted.



### Inserting a first LD network with the keyboard

- Press <SPACE> to set an insertion mark.
- Press <F6>.  
A first LD network with one contact and one coil is inserted.

Your screen should look like the following figure:

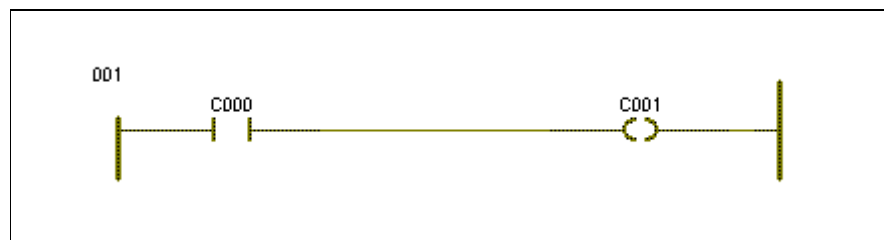


Figure 9-3: First LD network inserted

---

## Inserting serial contacts and coils

For inserting more contacts and coils you can use the icons 'Contact left', 'Contact right' and 'Coil right' in the toolbar or the corresponding menu items or keyboard shortcuts. For the next steps let us assume that you want to insert one more serial contact in your LD network.



### Inserting more serial contacts with the mouse

- Click on the contact 'C000' to mark it.
- Click on the icon 'Add contact right' in the toolbar.  
The new contact 'C002' is inserted to the right of the marked contact.



### Inserting more serial contacts with the keyboard

- Press the cursor keys to mark the contact 'C000'.
- Press <F7>.  
The new contact 'C002' is inserted to the right of the marked contact.

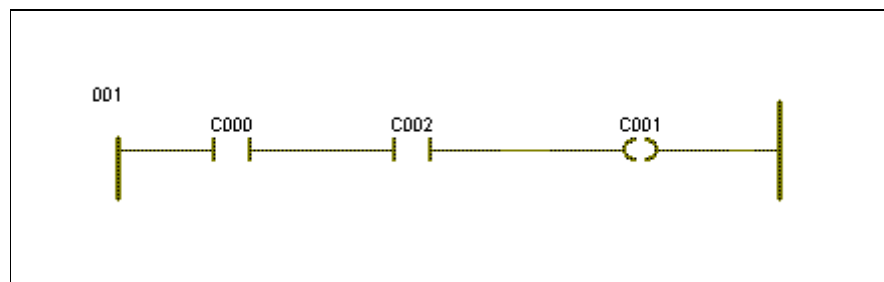


Figure 9-4: First LD network with inserted new contact 'C002'

---

## Inserting parallel contacts or coils

Up to now only serial LD networks have been considered. In LD it is also possible to edit parallel branches or the so called Wired-ORs. For the next steps let us assume that you want to insert a branch parallel to contact 'C002' in the LD network described in the previous section.



### Inserting a parallel branch with the mouse

- Click on the contact 'C002' to mark it.
- Click on the icon 'Add contact/coil below' to insert a parallel branch below contact 'C002'.



**Or:**

- Click on the icon 'Add contact/coil above' to insert a parallel branch above contact 'C002'.



In both cases the new contact 'C003' is inserted.



### Inserting a parallel branch with the keyboard

- Press the cursor keys to mark the contact 'C002'.
- Press <CTRL> + <F7> to insert a contact below contact 'C002'.  
The new contact 'C003' is inserted.

Your screen should look like the following figure now:

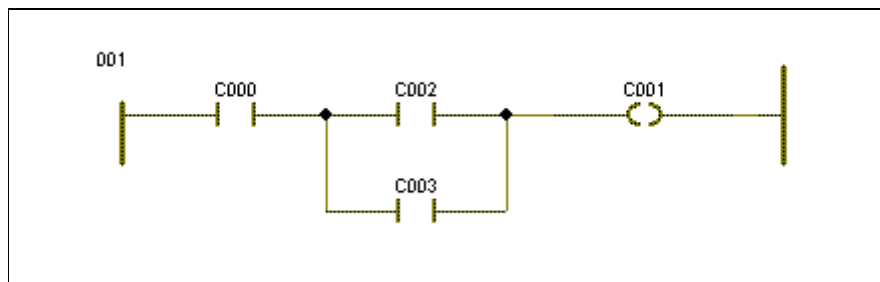


Figure 9-5: LD network with a parallel branch



## Using the LD branch edit mode

For inserting new branches the LD branch edit mode is provided. The LD branch edit mode allows to create complex branch structures rapidly. It is used in existing LD networks. Before using the LD branch edit mode you first have to insert a network with at least one contact or coil.

For the next description let us assume that you want to insert another parallel contact.



### Inserting a parallel branch with the mouse

- Click on the icon 'Insert LD branch' in the toolbar. A symbol for a LD branch is added to the cursor. While the LD branch edit mode is active, the icon appears 'pressed' in the toolbar.
- Click on a link between two objects where you want to start your parallel branch (see step 1. in the figure below).
- Move the mouse up or downwards to a free position (see step 2. in the figure below).
- Click the left mouse button at the position where you want to place the new object.
- Move the mouse to the desired end of the parallel branch (see step 3. in the figure below).
- Click the left mouse button to set the connection.

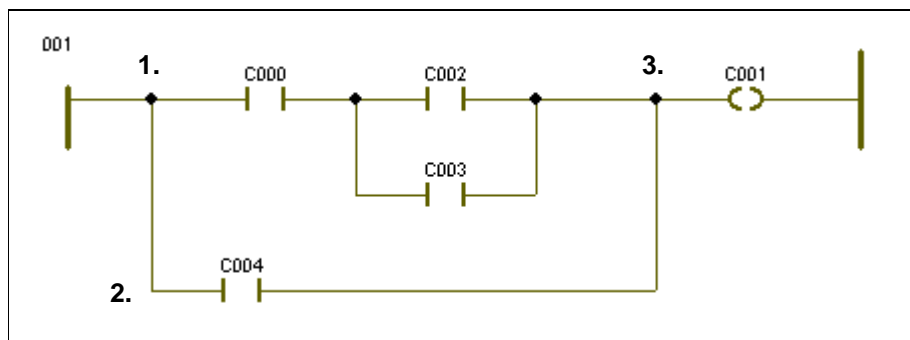


Figure 9-6: LD network with a parallel branch, inserted in LD branch edit mode

---

## Changing the properties of contacts and coils

Having inserted all objects in your LD worksheet you may change the properties of the elements. You probably want to change normally open contacts into normally closed contacts or coils into SET coils etc. For changing the types of contacts and coils several icons in the toolbar can be used. You get a short description text to the corresponding function of any icon, if you position the mouse cursor on the particular icon (without clicking it). Additionally the status bar displays the function of any icon.

For the next steps let us assume that you want to change one contact into a normally closed contact.



### Changing the properties of a contact with the mouse

- Click on a contact to mark it.
- Click on the icon 'Normally closed contact' in the toolbar.  
The contact type is changed.



### Changing the properties of a contact with the keyboard

- Press the cursor keys to mark a contact.
- Press <SHIFT> + <F7> to toggle the contact properties.  
The contact type is changed.



When toggling the properties of a coil, the various coil types (coil, negated coil, set coil and reset coil) are toggled.



### Changing the properties of a contact using the dialog 'Contact/Coil'

- Double click on the contact to be changed.  
The dialog 'Contact/Coil' appears.

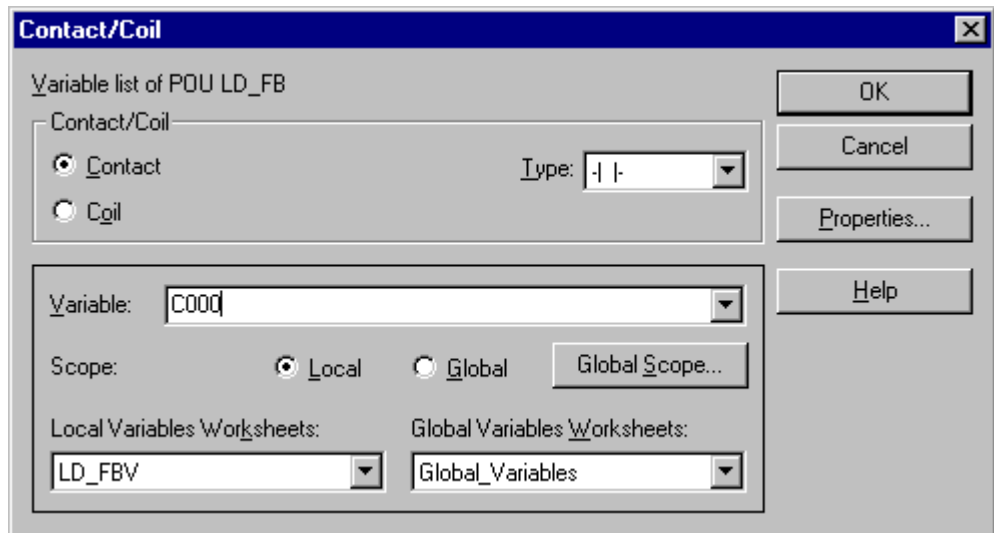


Figure 9-7: Dialog 'Contact/Coil'



### Using the dialog 'Contact/Coil'

- In the list box 'Type' choose the required contact type.
- Enter a name for the variable if you want.
- Confirm the dialog.  
The dialog 'Automatic Variables Declaration' appears.

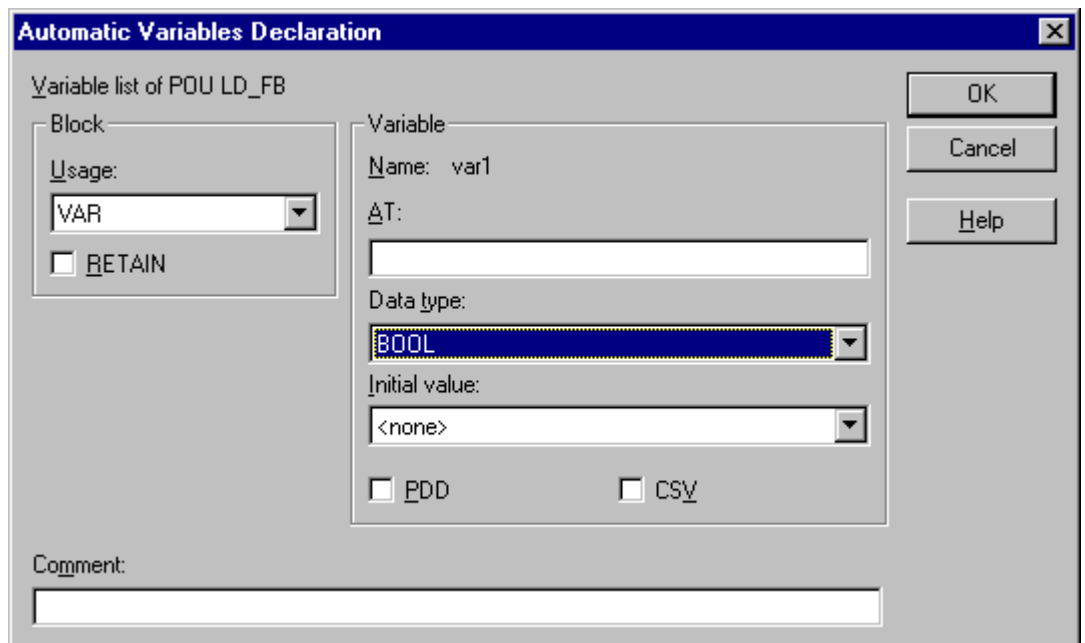


Figure 9-8: Dialog 'Automatic Variables Declaration'



## Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.
- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in an OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



When inserting variables, which have already been declared before, the name of the variable appears in the list box of the dialog 'Variable'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC server can be found in the 'OPC Server Manual'.

If you have done all steps of the programming example your screen should look like the following figure:

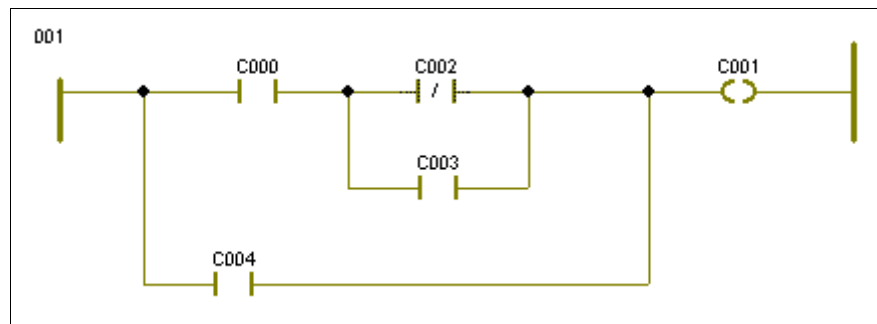


Figure 9-9: LD network with changed properties

---

## Inserting variables

While inserting contacts and coils the graphic editor uses a place holder for the name of the variable. These place holders start with the letter 'C' followed by a number. In most cases you don't want to use these place holders but your own variable names. In these cases you have to do the same steps as described in the section 'Changing the properties of contacts and coils' of this chapter.

The first step to do is calling the dialog 'Contact/Coil' by double clicking on the contact or coil. In this dialog you first have to choose if you want to use a local or global variable by activating the corresponding radio button. If the variable is already declared, you may just select it in the listbox. If it is not declared you can enter a name for the new variable. The dialog 'Automatic Variables Declaration' appears. In both cases the variable name is displayed in the LD worksheet after confirming the dialog. If necessary the variable declaration is inserted automatically in the variable worksheet of the POU.

---

## Calling functions or function blocks using the Edit Wizard

In a LD network also functions and function blocks can be called.



Functions and function blocks are represented as FBD elements, i.e. LD and FBD elements are mixed in LD code body worksheets.

You can insert a function or a function block anywhere in the worksheet and connect it later or you can insert it directly in an existing LD network. In this case the function or function block will be directly connected to the network.

The most comfortable way to insert a FBD function or function block into the LD worksheet is to use the Edit Wizard.



A general description of the Edit Wizard can be found in the section 'The Edit Wizard' in chapter 'Getting started' in this manual.

If the Edit Wizard is not visible in the workspace, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



### Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

As an example, let us assume that you want to insert the function block 'CTU' connected to an existing LD network consisting of a left and a right power rail, a contact and a coil.



### Inserting the function block CTU into an existing LD network using the Edit Wizard

- Click on a contact in the LD network to mark it.
- Open the Edit Wizard list box 'Group' and select the group 'Function blocks'. The available function blocks are displayed in the selection area of the Wizard.
- Double click on the function block 'CTU'. The dialog 'FB Instances' appears. The field 'FB Instances' displays the default instance name (e.g. for a CTU the name 'CTU\_n' is proposed, where *n* is the first available number which is free for this instance name). To define the name of the new FB you have the following possibilities:
  - \* Enter a new instance name in the field.
  - \* Accept the proposed name.
  - \* Select an already existing name in the list box 'FB Instances'.
- Press 'OK' to confirm the dialog. If you have entered a new instance name, the dialog 'Automatic FB Declaration' appears.
- Enter a comment if you want and press 'OK' to confirm the dialog. The function block 'CTU' is inserted automatically into the LD network.



### Inserting the function block CTU into an existing LD network with the keyboard

- Press the cursor keys to mark a contact in the LD network.
- Press <ALT> + <3> to activate the Edit Wizard.
- Press the <TAB> key to activate the list box 'Group' in the Edit Wizard. Then press the cursor keys to browse through the available functions and function blocks and select the group 'Function blocks'.
- Press the <TAB> key to activate the selection area and mark the function block 'CTU' with the cursor keys.

- Press <↵>. The dialog 'FB Instances' appears. The field 'FB Instances' displays the default instance name (e.g. for a CTU the name 'CTU\_n' is proposed, where *n* is the first available number which is free for this instance name).  
To define the name of the new FB you have the following possibilities:
  - \* Enter a new instance name in the field.
  - \* Accept the proposed name.
  - \* Select an already existing name in the list box 'FB Instances'.
- Press <↵> to confirm the dialog. If you have entered a new instance name, the dialog 'Automatic FB Declaration' appears.
- Enter a comment if you want and press <↵> to confirm the dialog. The function block 'CTU' is automatically inserted into the LD network.

Your screen should look like the following figure now:

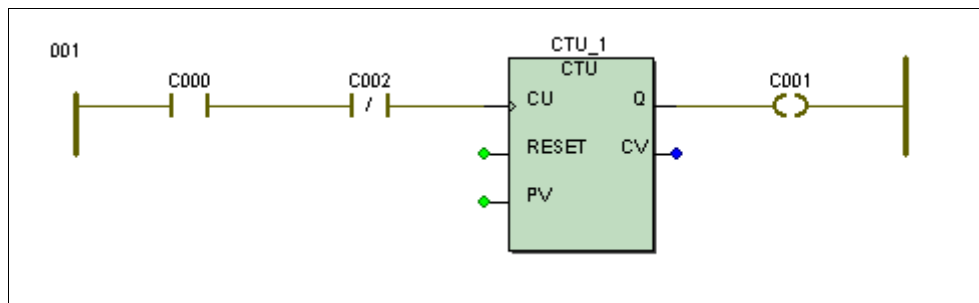


Figure 9-10: LD network with function block 'CTU'



The unconnected formal parameters of the function block can be connected to other contacts, coils or variables.



The steps to edit an inserted FBD function or function block are described in the section 'Changing the properties of functions and function blocks' in the chapter 'Editing in FBD'.

# Editing in SFC

**This chapter provides information about...**

- calling the graphic editor with a SFC worksheet
- SFC networks
- inserting SFC networks and branches
- inserting variables for actions and transitions
- calling functions
- action and transition details



# Editing in SFC

---

## Calling the graphic editor with a SFC worksheet

The first step before editing a SFC code body worksheet is to call the graphic editor with the SFC worksheet, using the project tree.



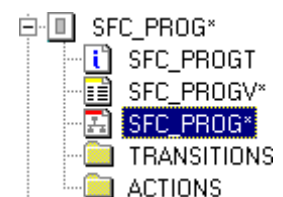
A general description of handling the project tree and browsing through POU's and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

As an example, let us assume that you want to edit the SFC worksheet of a program which is called SFC\_PROG. Therefore you first have to insert a program with this name as it is described in the chapter 'Editing the project structure'.



### Calling the graphic editor for the SFC code body worksheet with the mouse

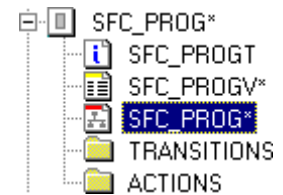
- If the desired worksheet icon is not visible in the project tree, open the corresponding subtree, containing the POU worksheets. For this purpose double click on the POU name (e.g. 'SFC\_PROG').
- Double click on the icon 'Worksheet in SFC' of the program 'SFC\_PROG'.  
The graphic editor with the SFC worksheet appears.





## Calling the graphic editor for the SFC code body worksheet with the keyboard

- If the desired worksheet icon is not visible in the project tree, open the 'SFC\_PROG' subtree, containing the POU worksheets as follows: Press < > or < > to highlight the program 'SFC\_PROG'. Press <→> to open the subtree.
- Press < > or < > to mark the icon 'Worksheet in SFC' of the program 'SFC\_PROG'.
- Press <↵>.  
The graphic editor with the SFC worksheet appears.



---

## Introduction to SFC

A code body programmed in SFC is composed of steps and transitions which are connected by directed links.

One or several action blocks can be associated to a step. While the step is active the associated action is executed according to the action qualifier. The action can be a boolean variable. It is also possible to define code to be executed in an additional code body worksheet i.e. detail. In this case the name of the code body worksheet has to be used as the name for the action. An action representing a detail appears in green, an action representing a variable appears in red.

A transition represents the condition when execution moves from one step to another. If a transition becomes TRUE the preceding step is executed once again and the succeeding step becomes active. The transition can be either a boolean variable or a directly connected boolean expression in FBD or LD. It is also possible to define code to be executed in an additional code body worksheet i.e. detail. In this case the name of the code body worksheet has to be used as the name for the transition. A transition representing a direct connection is displayed like a normal transition but with a green connection point.

A set of connected objects is called SFC network. A SFC network must always have one initial step which is the first step to be executed after a cold start or warm start. An initial step is represented by a rectangle with a double line. All steps are represented in blue.

Simultaneous or alternative branches can be inserted in the SFC network.

In SFC code bodies comments can be inserted using asterisks and parentheses. Additionally it is possible to insert FBD or LD elements into a SFC worksheet. For this purpose, you can use the Edit Wizard, as described in the chapters 'Editing in FBD' or 'Editing in LD'.

---

## Inserting a first SFC network

For the next descriptions let us start with a simple SFC network. In this first example you may insert a SFC network with one step, one transition and one action block.



### Inserting a SFC network with the mouse

- Click into the worksheet to set an insertion mark.
- Click on the icon 'Create step transition sequence'. A SFC network with one step and one transition is inserted.



### Inserting a SFC network with the keyboard

- Press <SPACE> to set an insertion mark.
- Press <F8>. A SFC network with one step and one transition is inserted.

Your worksheet should look like the following figure:

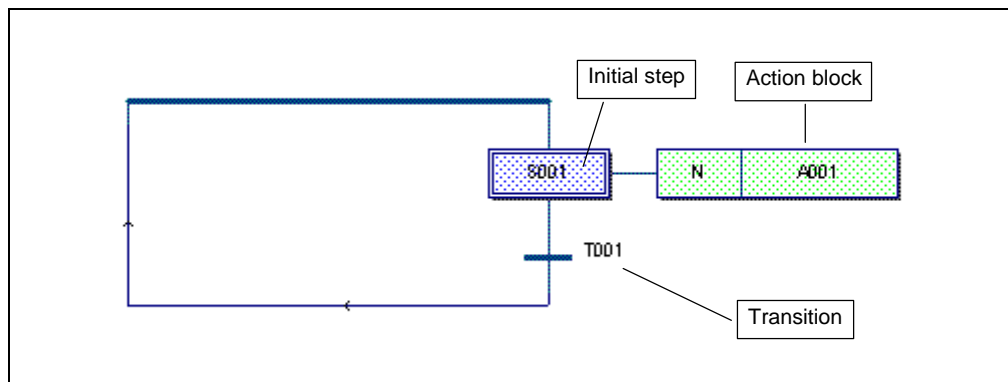


Figure 10-1: SFC network with one step and one transition

The figure shows a SFC network with one initial step (S001), the corresponding action block (A001) and one transition (T001).

---

## Inserting more steps and transitions

Let us assume that you want to insert three more steps and transitions into your SFC network.



To keep the legal network, steps and transitions are always inserted in pairs.



### Inserting more steps and transitions with the mouse

- Click on the step, the new transition and step will succeed.
- Click on the icon 'Insert step transition sequence'. Another transition and step is inserted between the marked step and the next transition.
- Repeat the steps for two additional step and transition pairs.



If you mark a **transition** instead of a step and you insert a new step-transition-sequence, the new step and transition is inserted below the marked transition.



### Inserting more steps and transitions with the keyboard

- Press the cursor keys to mark the step, the new transition and step will succeed.
- Press <F8>. Another transition and step is inserted after the marked step.
- Repeat the steps for two additional step and transition pairs.



If you mark a **transition** instead of a step and you insert a new step-transition-sequence, the new step and transition is inserted below the marked transition.

Your screen should look like the following figure now:



In the following sample network, three new step-transition-sequences were inserted by marking step S001 and then pressing the icon/shortcut three times. Thus, the new sequences are inserted **between** step S001 and transition T001.

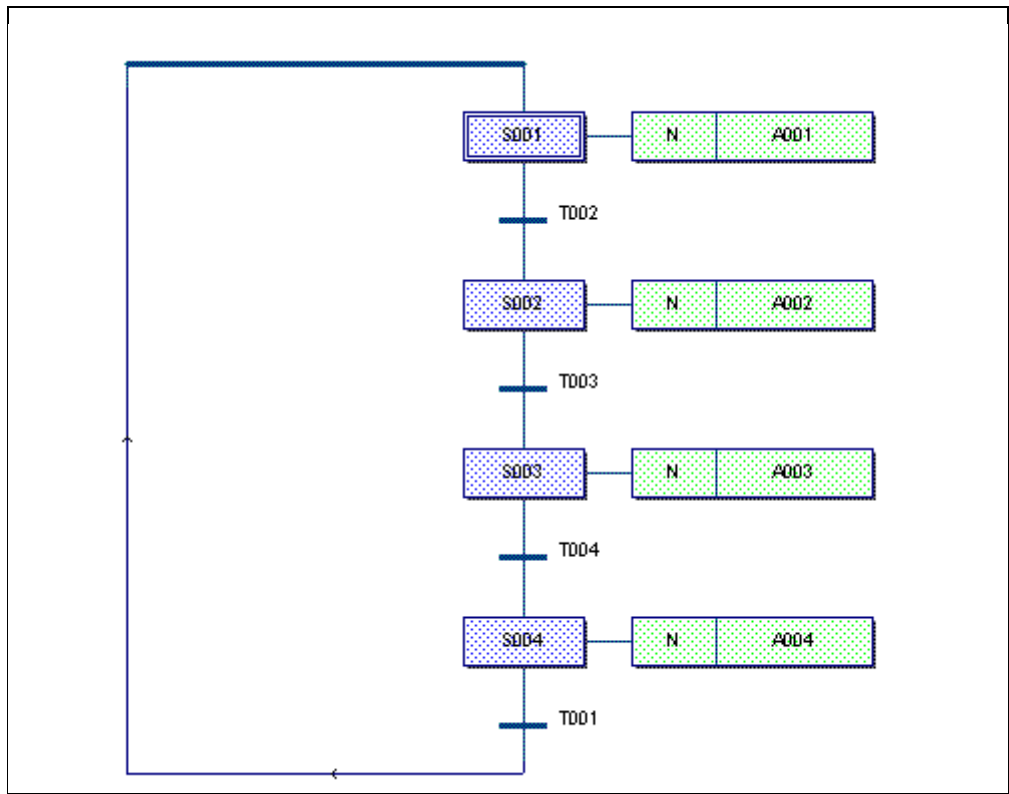


Figure 10-2: SFC network with four steps

---

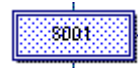
## Changing an initial step into a normal step or vice versa

Your first inserted step is an initial step. For the next sections let us assume that you want the second step to be the initial step. Therefore you have to change the first step into a normal step and the second step into an initial step. In the following section it is described how to change the initial step into a normal step.



### Changing an initial step into a normal step with the mouse

- Double click on the initial step.  
The dialog 'Step' appears.



### Changing an initial step into a normal step with the keyboard

- Press the cursor keys to move to the position of the initial step.
- Press <↵>.  
The dialog 'Step' appears.

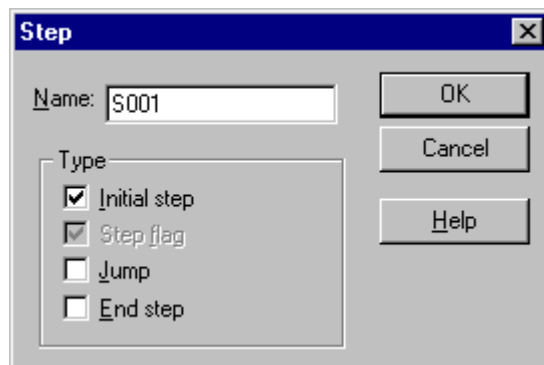
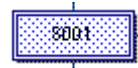


Figure 10-3: Dialog 'Step'



### Using the dialog 'Step'

- Deactivate the checkbox 'Initial step'.
- Confirm the dialog.

---

## Inserting alternative branches

Alternative branches mean that either one or another transition becomes true and only one of the branches is executed. For the next description let us assume that you want to insert an alternative branch following step S002.



### Inserting alternative branches with the mouse

- Click on step S002 to mark it.
- Click on the icon 'Insert Simultaneous/Alternative Branches' in the toolbar.  
The dialog 'Divergence' appears.



### Inserting alternative branches with the keyboard

- Press the cursor keys to move to the position of step S002.
- Press <CTRL> + <F8>.  
The dialog 'Divergence' appears.

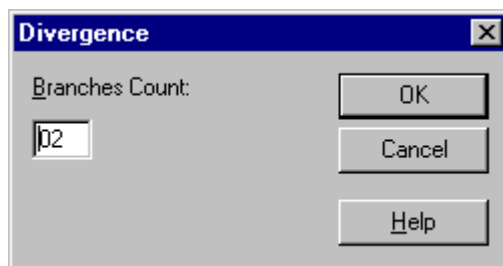


Figure 10-4: Dialog 'Divergence'



### Using the dialog 'Divergence'

- The value in the field 'Branch Count' determines, how many branches are inserted below the marked step. In our example, you can confirm the dialog with the default value.

Your screen should look like the following figure:

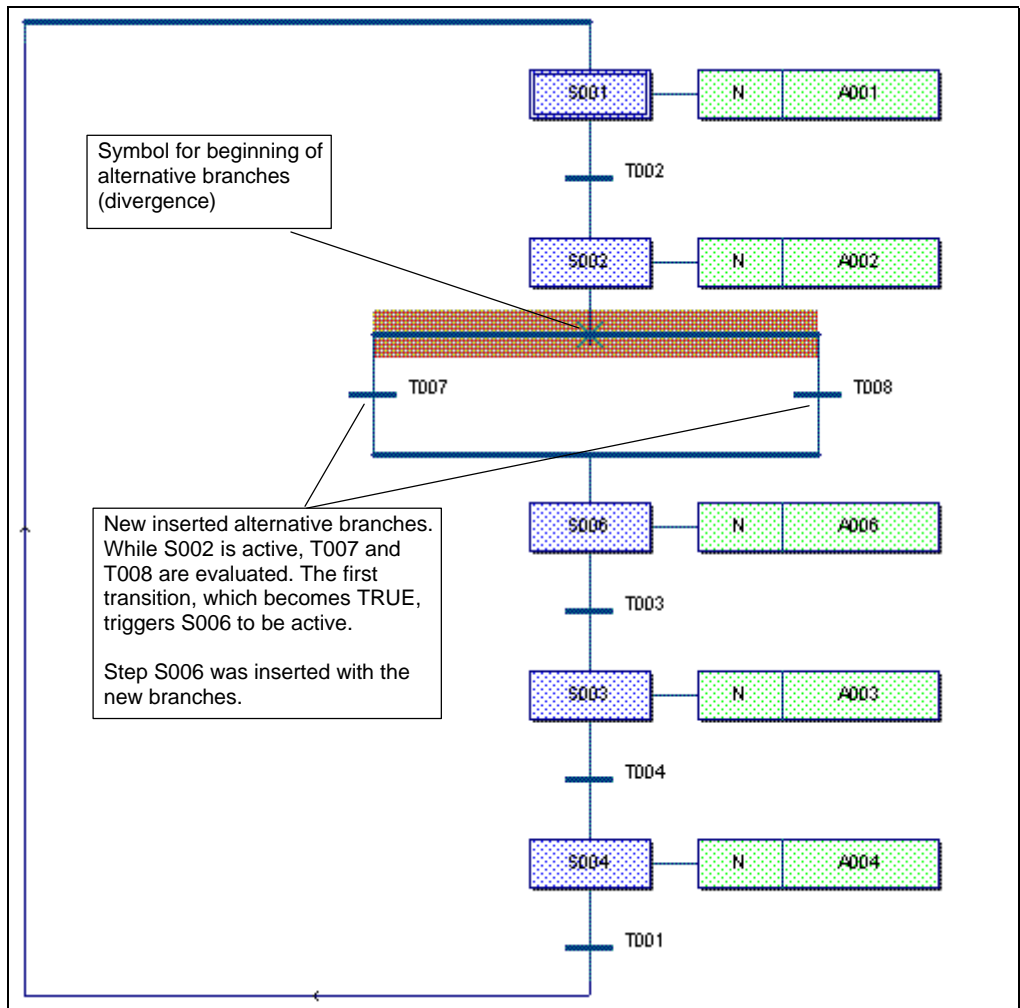


Figure 10-5: SFC network with 2 alternative branches



---

## Inserting simultaneous branches

Simultaneous branches are branches where steps and their action blocks are executed simultaneously. For the next description let us assume that you want to insert a simultaneous branch following transition T004.



### Inserting simultaneous branches with the mouse

- Click on transition T004 to mark it.
- Click on the icon 'Insert Simultaneous/Alternative Branches' in the toolbar.  
The dialog 'Divergence' appears.



### Inserting simultaneous branches with the keyboard

- Press the cursor keys to move to the position of transition T004.
- Press <CTRL> + <F8>.  
The dialog 'Divergence' appears.

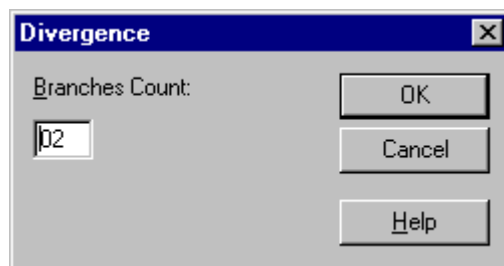


Figure 10-6: Dialog 'Divergence'



### Using the dialog 'Divergence'

- The value in the field 'Branch Count' determines, how many branches are inserted below the marked transition. In our example, you can confirm the dialog with the default value.

Your screen should look like the following figure:

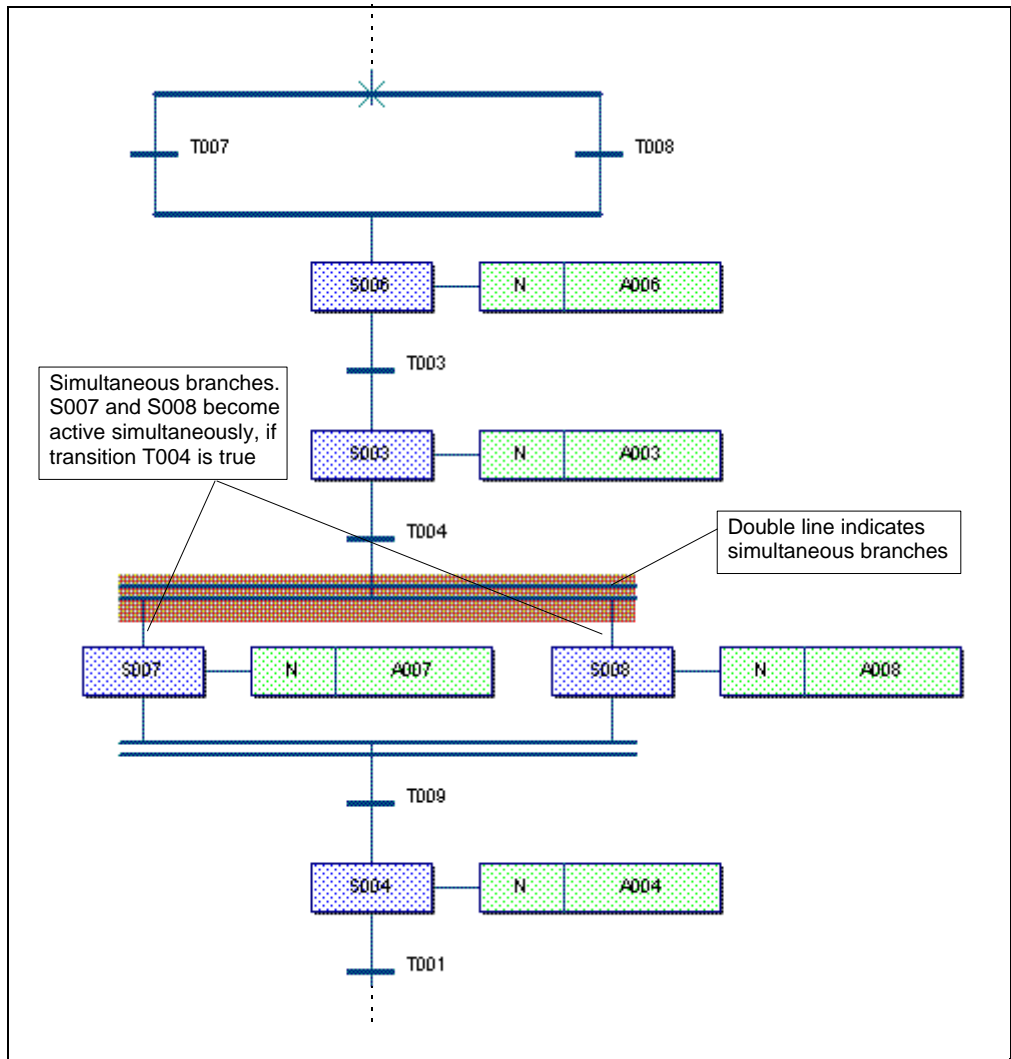


Figure 10-7: Part of SFC network with an alternative and a simultaneous branch

---

## Using the SFC branch edit mode

You can insert branches using the SFC branch edit mode. The SFC branch edit mode is a comfortable tool for inserting simultaneous or for alternative branches.



While using the SFC branch edit mode always click on an object and not on a line.



### Inserting an alternative branch with the mouse

- Click on the icon 'Insert SFC branch'.  
A symbol for a SFC branch is added to the cursor. During the SFC branch edit mode, the icon appears 'pressed'.
- Click with the left mouse button in the **lower** half of a step to insert an alternative branch.
- Move the cursor to a free position where you want to drop the transition symbol. Avoid collisions with other symbols. Click the left mouse button to place the symbol.
- Move the cursor to the step where you want to close the branch. Click the left mouse button at this point.



The SFC branch edit mode is terminated automatically, after the new branch is inserted. For inserting another branch, reselect the SFC branch edit mode by clicking on the icon again.



### Inserting a simultaneous branch with the mouse

- Click on the icon 'Insert SFC branch'.  
A symbol for a SFC branch is added to the cursor. During the SFC branch edit mode, the icon appears 'pressed'.
- Click with the left mouse button in the **upper** half of a step to insert a simultaneous branch.
- Move the cursor to a free position where you want to drop the transition symbol. Avoid collisions with other symbols. Click the left mouse button to place the symbol.
- Move the cursor to the transition where you want to close the branch. Click the left mouse button at this point.



The SFC branch edit mode is terminated automatically, after the new branch is inserted. For inserting another branch, reselect the SFC branch edit mode by clicking on the icon again.

---

## Inserting variables for actions

In SFC it is possible to connect boolean variables to actions. According to the action qualifier the value of the variable is set.



Please refer to your PLC documentation or to the context-sensitive Help for detailed information about the available action qualifiers.



### Connecting variables to action blocks with the mouse

- Click on the action block 'A004' with the right mouse button to open the context menu.
- Select the context menu item 'Object properties'.  
The dialog 'Action' appears.



### Connecting variables to action blocks with the keyboard

- Press the cursor keys to mark the action block 'A004'.
- Press <ALT> + <↓>.  
The dialog 'Action' appears.

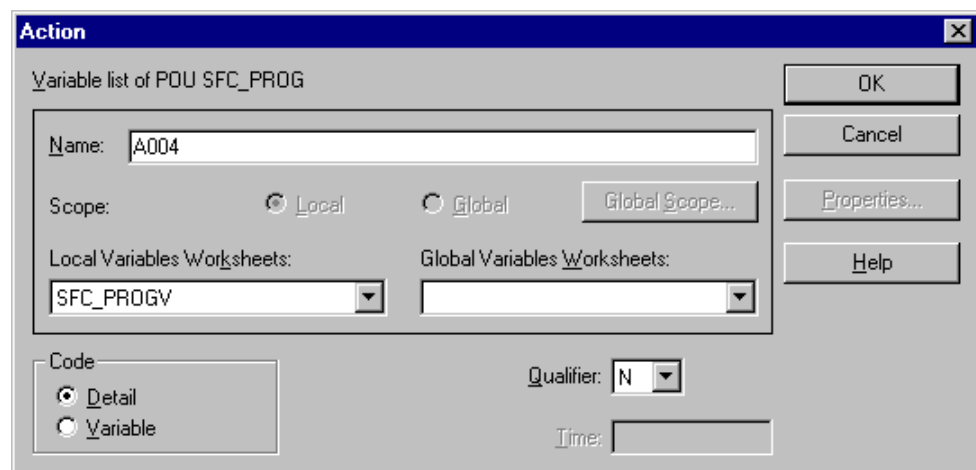


Figure 10-8: Dialog 'Action'



### Using the dialog 'Action'

- Enter the name of the variable in the field 'Name'. For this purpose, delete or overwrite the current entry (A004 in our example).
- Activate the radio button 'Variable'.
- Confirm the dialog.  
The dialog 'Automatic Variables Declaration' appears.

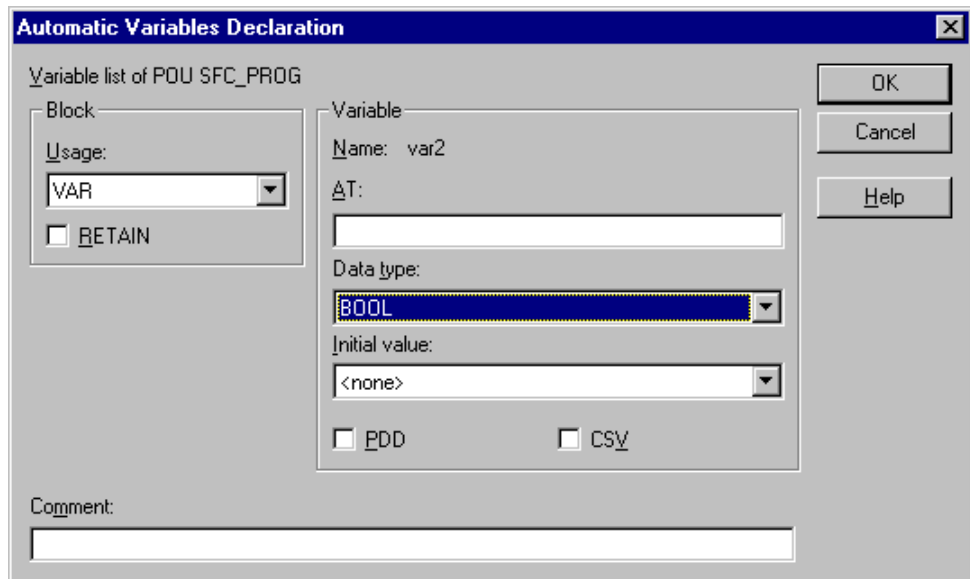


Figure 10-9: Dialog 'Automatic Variables Declaration'



### Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.
- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in a OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



While inserting variables which have already been declared before the name of the variable appears in the listbox of the dialog 'Action'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC server can be found in the 'OPC Server Manual'.

In the following figure you can see the action block with the variable name 'var2':

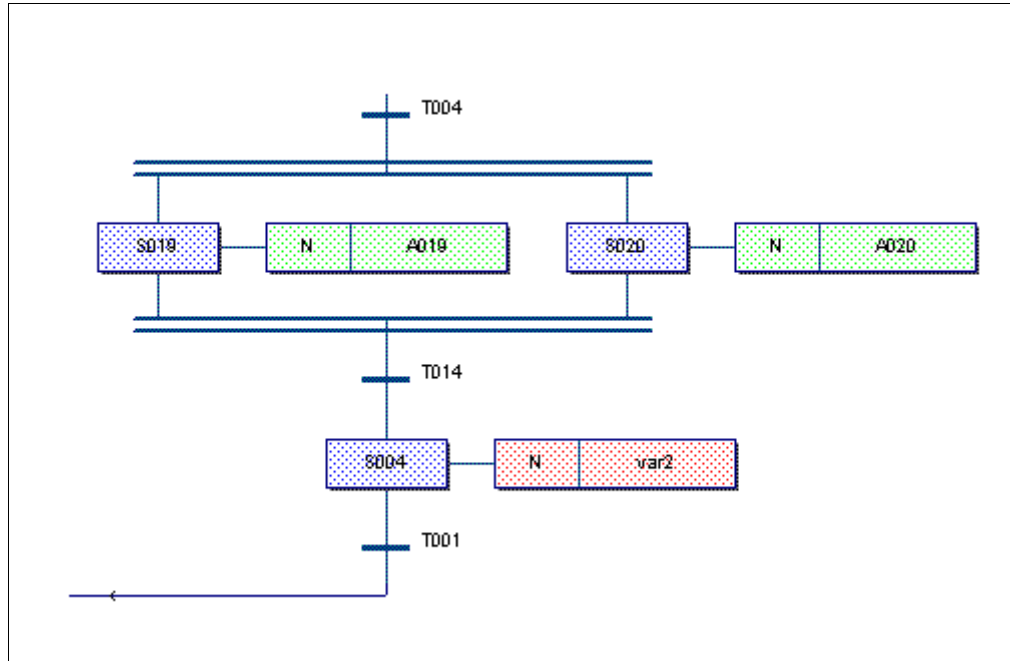


Figure 10-10: Action block with variable name

---

## Inserting variables for transitions

If you want to connect a variable to a transition you have to do two main steps. First changing the properties of the transition into a direct connection and then inserting and connecting the variable. For the next procedures let us assume that you want to connect a variable called 'var1' to transition T003.



### Changing the properties of the transition with the mouse

- Click on the transition 'T003' with the right mouse button to open the context menu.
- Select the context menu item 'Object properties'. The dialog 'Transition' appears.



## Changing the properties of the transition with the keyboard

- Press the cursor keys to mark the transition 'T003'.
- Press <ALT> + <↓>.  
The dialog 'Transition' appears.

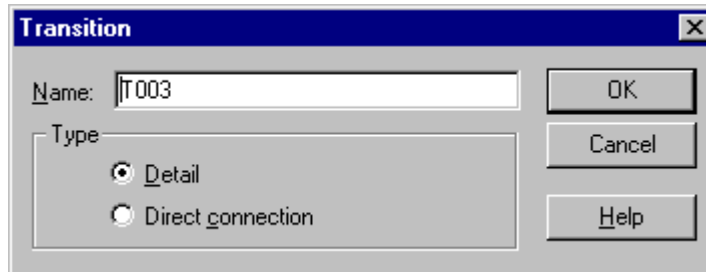


Figure 10-11: Dialog 'Transition'



## Using the dialog 'Transition'

- Activate the radio button 'Direct connection'.
- Confirm the dialog.  
The transition is shown with a green connection point.

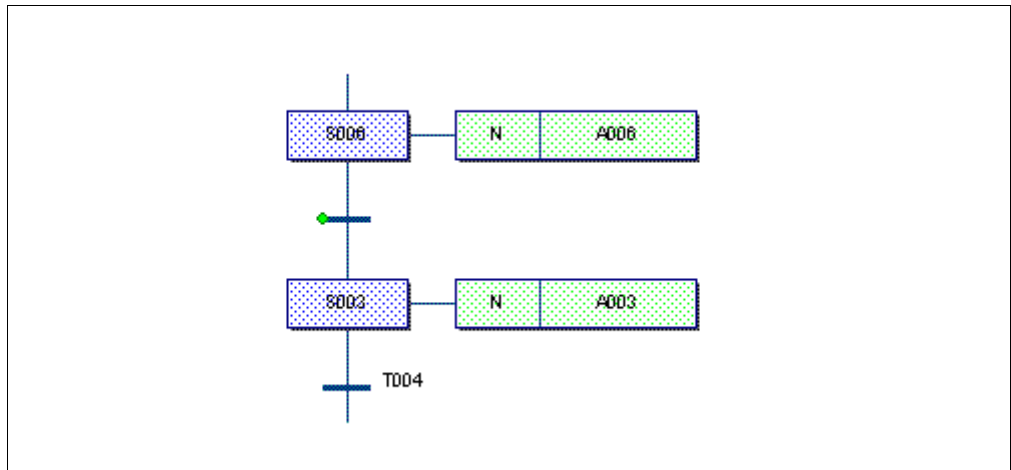


Figure 10-12: Transition, specified as direct connection with a green connection point



## Inserting the variable with the mouse

- Click on the transition to mark it.
- Click on the icon 'Variable'.  
The dialog 'Variable' appears.





## Inserting the variable with the keyboard

- Press the cursor keys to mark the transition.
- Press <F5>.  
The dialog 'Variable' appears.

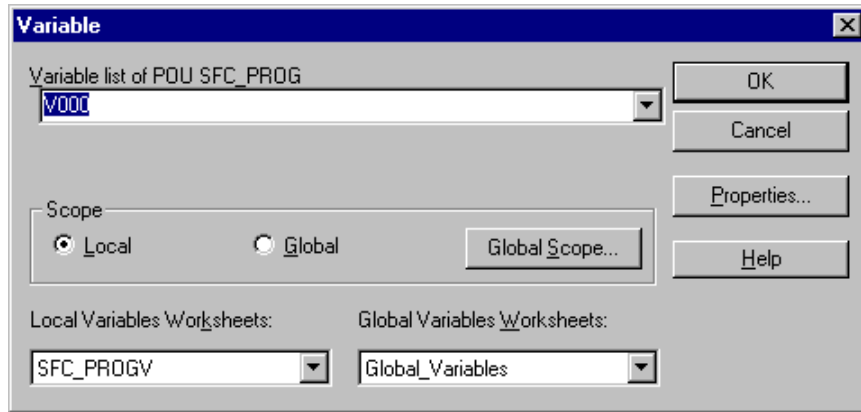


Figure 10-13: Dialog 'Variable'



## Using the dialog 'Variable'

- Enter a variable name.
- Confirm the dialog.  
The dialog 'Automatic Variables Declaration' appears.

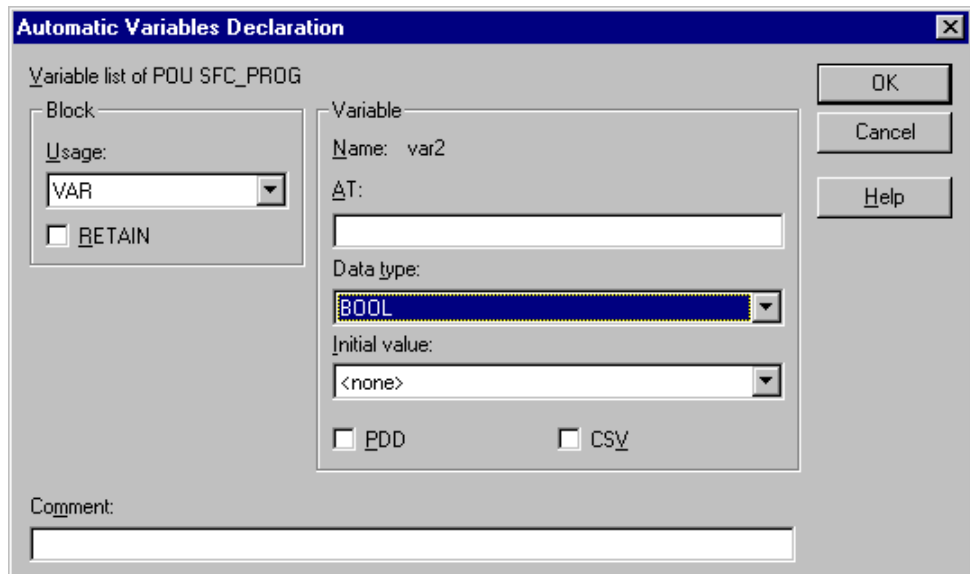


Figure 10-14: Dialog 'Automatic Variables Declaration'





## Using the dialog 'Automatic Variables Declaration'

- Choose a variable keyword in the list box 'Usage'.
- Enter a location in the field 'AT' if you want to declare a located variable (only possible in programs or for global variables).
- Choose the correct data type in the field 'Data type'.
- If required, enter an initial value.
- Mark the checkbox 'PDD' if the variable should be stored in the ProConOS PDD (Process Data Directory), i.e. is intended to be used with IEC 61131-5 communication function blocks.
- Mark the checkbox 'CSV' if the variable should be stored in the CSV file, i.e. is intended to be used with the OPC Server. The OPC Server processes only variables, which are declared in the CSV file, in order to be used in a OPC client process (e. g. a visualization).
- Enter a comment if you want.
- Confirm the dialog.  
The new variable is inserted in the code body worksheet and the declaration of the variable is autoinserted in the variable declaration of the POU.



While inserting variables which have already been declared before the name of the variable appears in the listbox of the dialog 'Variable'. Confirming the dialog, the variable is directly inserted in the code body worksheet. The dialog 'Automatic Variables Declaration' does not appear.



Detailed information about the OPC server can be found in the 'OPC Server Manual'.

In the following figure you can see a variable connected to a transition:

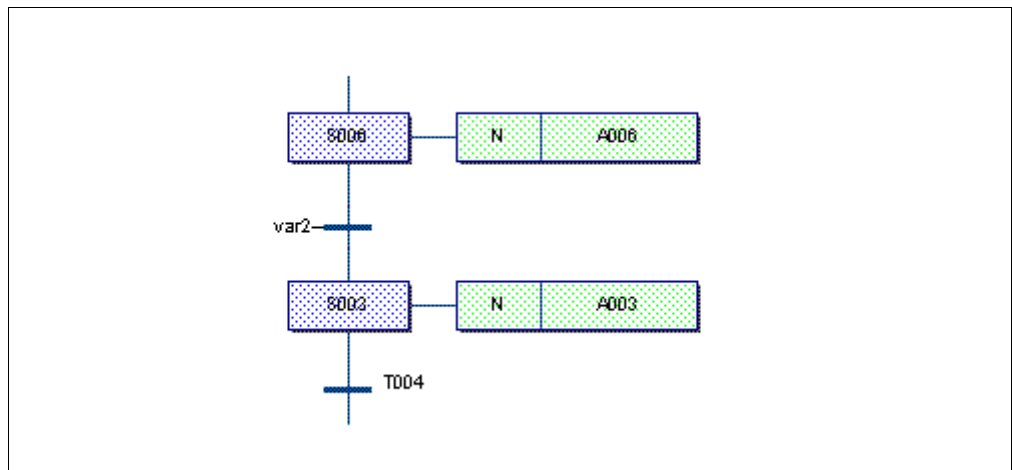


Figure 10-15: Variable connected to a transition

---

## Calling functions

Instead of variables as it has been described in the section above also functions can be connected to transitions with direct connections. In these cases you first have to insert the function and then connect it with the connection point of the transition.



The steps how to insert and connect functions using the Edit Wizard are described in the section 'Inserting functions and function blocks using the Edit Wizard' in the chapter 'Editing in FBD' of this manual.

It is also possible to insert LD networks and connect them to the transition.



The steps how to insert contacts and coils are described in the chapter 'Editing in LD' of this manual.

---

## Action and transition details

According to IEC 61131-3 it is possible to edit a code body for actions and transitions instead of connecting variables. These code bodies are edited in worksheets which you can find in the project tree below the directory nodes for actions and transitions. For the following steps let us assume that you have selected action A005 to be a detail.



### Creating an action detail with the mouse

- Double click on the action block A005 in the SFC network.  
The dialog 'Insert' appears.



### Creating an action detail with the keyboard

- Press the cursor keys to mark the action block A005 in the SFC network.
- Press <↓>.  
The dialog 'Insert' appears.

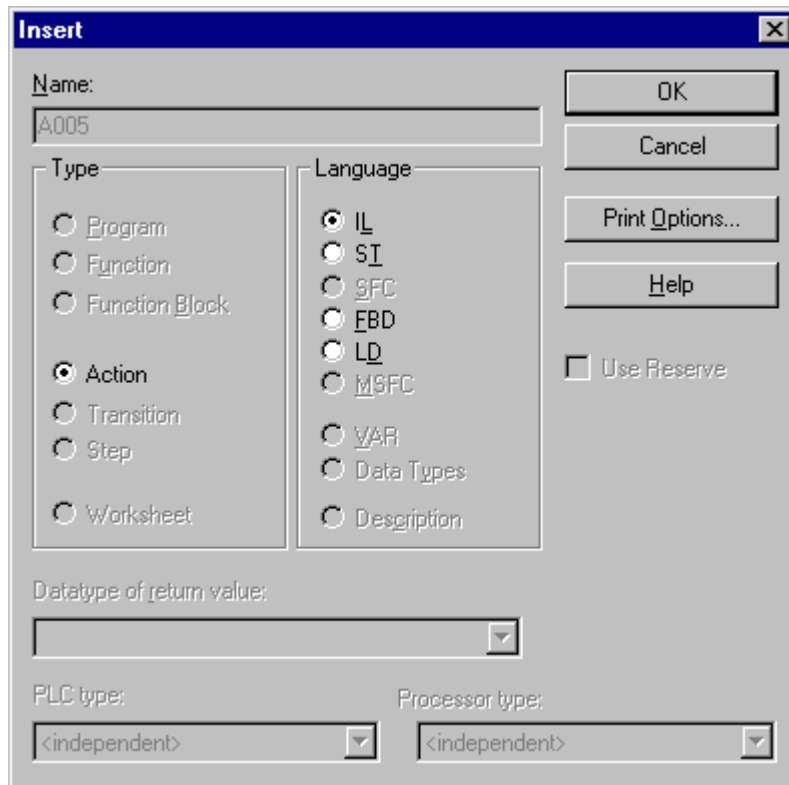


Figure 10-16: Dialog 'Insert'



### Using the dialog 'Insert'

- Choose a programming language.
- Confirm the dialog.  
The worksheet is inserted in the project tree and it is opened for immediate editing.
- Edit the code body for the action detail.



For transitions the same steps as for actions have to be done.

# Compiling, downloading and debugging

**This chapter provides information about...**

- inserting configurations, resources and tasks in the project tree
- associating programs to tasks
- compiling the project
- patching POUs
- downloading the project
- calling worksheets in online mode
- debug possibilities

# Compiling, downloading and debugging

---

## Inserting configurations, resources and tasks

The project tree normally has one configuration, one resource and one task if you create a new project. It is possible to insert either new configurations, resources or tasks using the project tree editor. The steps to be done are almost the same independently if you are inserting configurations, resources or tasks.



The PLC type of the configurations and the processor type of the resources depend on the connected PLC.

For the next steps let us assume that the project already contains a configuration and a resource and you want to insert a new resource called 'res\_2' for IPC as processor type.



### Inserting a new resource with the mouse

- Click with the right mouse button on the icon '*Resource name*' in the project tree to open the context menu.
- Select the menu item 'Insert'.  
The dialog 'Insert' appears.



### Inserting a new resource with the keyboard

- Press < > or < > to mark the icon '*Resource name*' in the project tree.
- Press <INS>.  
The dialog 'Insert' appears.



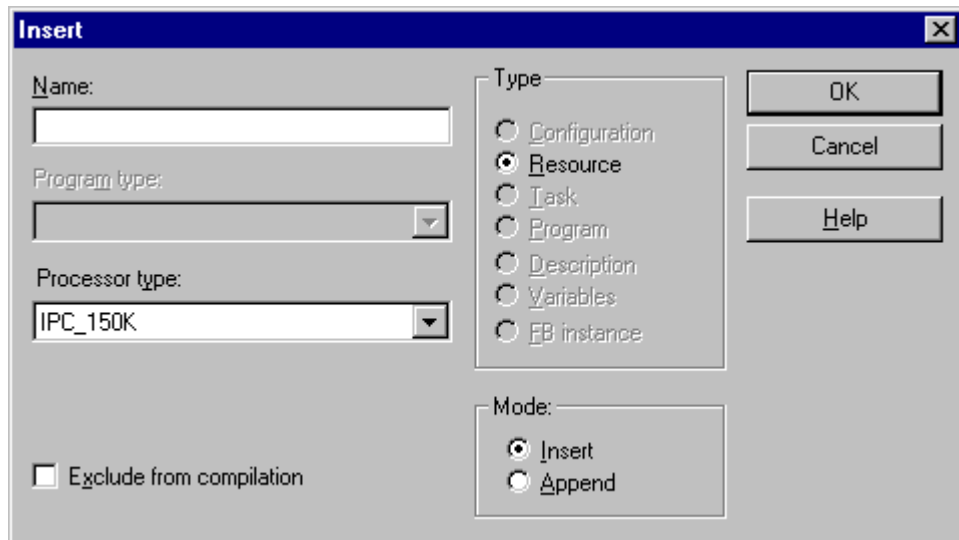


Figure 11-1: Dialog 'Insert'



### Using the dialog 'Insert'

- Enter 'res\_2' as the name of the resource and select the desired processor type.
- Confirm the dialog.
- Select the menu item 'PLC Settings...' in the context menu of the resource.  
The dialog 'Resource Settings...' appears.

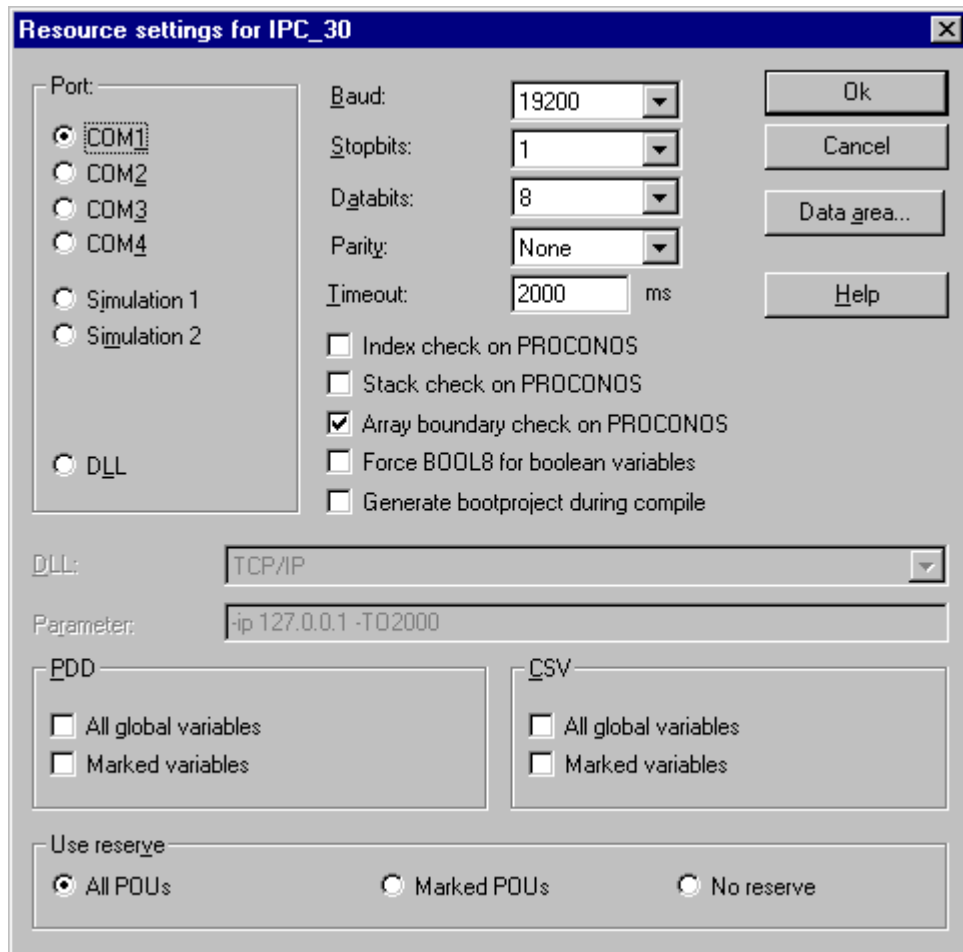


Figure 11-2: Dialog 'Resource Settings...'



### Using the dialog 'Resource Settings...'

- Activate the radio button 'Simulation 1' to send the PLC program to simulation 1 while downloading.
- If desired, change the properties of the resource.
- Activate the CSV checkboxes, if the resource variables are intended to be used with the OPC Server. To store all global variables in the CSV file, activate the checkbox 'All global variables'. If you want to store variables explicitly marked as CSV variables in the CSV file, activate the checkbox 'Marked variables'. The OPC Server processes these variables and transfers their actual values to an OPC client (e. g. a visualization).
- Confirm the dialog.  
The resource is inserted in the project tree.



The easiest way to insert new configurations or tasks is just marking the corresponding icon in the project tree and then repeating the same steps as described above.



Detailed information about the OPC Server are contained in the 'OPC Server manual'.

---

## Associating programs to tasks

If you have inserted a new resource you have to insert one or several tasks. For the next steps let us assume that you have already inserted the task 'DEFAULT' in 'res\_2' following the steps described in the previous section.

The next step to be done before compiling is to associate programs to tasks. This means deciding in which task a program is processed.



### Associating a program to a task with the mouse

- Click with the right mouse button on the task icon in the project tree to open the context menu.
- Select the menu item 'Insert'.  
The dialog 'Insert' appears



### Associating a program to a task with the keyboard

- Press < > or < > to mark the task icon in the project tree.
- Press <INS>.  
The dialog 'Insert' appears.

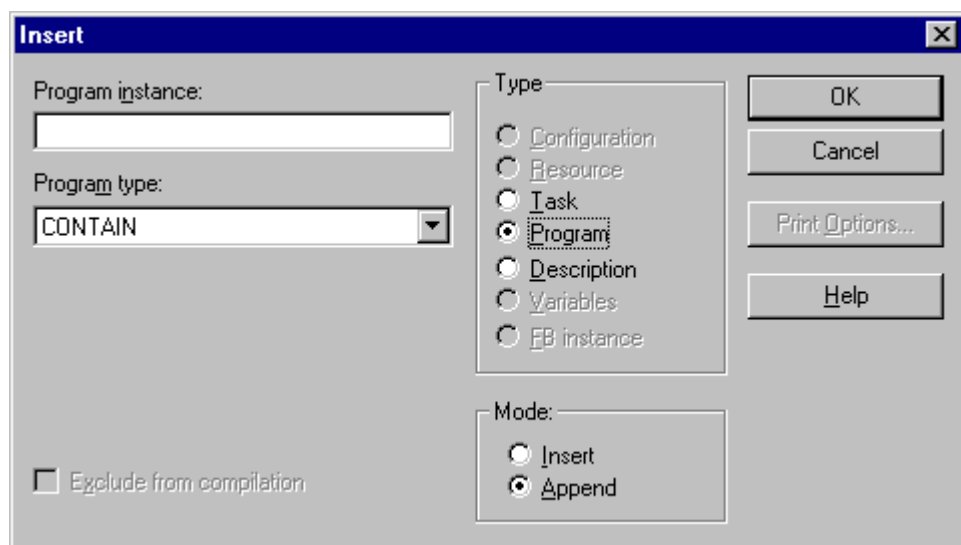


Figure 11-3: Dialog 'Insert'





## Using the dialog 'Insert'

- Activate the radio button 'Program' as shown in figure 11-3.
- Enter the instance name of the program in the field 'Program instance'.
- Choose the name of the program which is contained in the list box 'Program type'.
- Confirm the dialog.  
The icon of the program is inserted in the project tree.

---

## Compiling a project

Compiling means translating and transforming the contents of the worksheets in special code which can be executed by your PLC. The compilation process is performed in several steps. It starts with compiling (i.e. syntax checking) the different worksheets. During the second main step the compiled worksheets are linked together and an intermediate IEC code is generated. The last step generates the PLC code.

You have several possibilities for compiling either the whole project or only parts of it. In the following list the different possibilities are described, which are called using the menu items in the submenu 'Build' or the corresponding icons in the toolbar.

- \* 'Make' - This is the standard mode for compiling the project when you have finished editing. The menu item can be used to compile all worksheets which have been edited. These worksheets are marked with an asterisk in the project tree. After using 'Make' the PLC specific code is generated and the project is ready for downloading to the PLC.
- \* 'Patch POU' - This menu item is used to compile changes, which have been made e.g. after debugging a project. The changes are automatically downloaded to the PLC, so that you can view them immediately after switching into online mode.
- \* 'Compile worksheet' - This menu item is used to compile a single worksheet after editing it. Choosing this menu item means, that syntax errors within the current code body worksheet and the related variable worksheet are going to be detected by the compiler. All detected errors and warnings are displayed in the message window. By double clicking on an error or warning you can open the related worksheet, where the error was detected.  
When closing a worksheet it is compiled automatically.  
Using only the compiler no code is generated!
- \* 'Rebuild Project' - This menu item is used to compile the whole project for the first time after editing. It should only be used, if 'Make' generates compiler errors or you have unzipped your project without the front end code.

Using 'Rebuild Project' all worksheets are going to be compiled and linked. Detected errors and warnings are displayed in the message window. After the syntax checking the IEC code as well as the PLC specific code is generated automatically. The project is then ready for downloading to the PLC.

The compilation modes 'Make' and 'Patch POU' are described in the following sections.

---

## Compiling a project using 'Make'

This section describes the steps how to compile the changes you have made in the worksheets. The other possibilities are described in the context-sensitive Help.

Using the menu item/icon 'Make' the changed worksheets are compiled, linked and the changed PLC code is going to be generated. After successful execution the changed project is ready for downloading to the PLC.



Before starting the compilation, ensure that the message window is visible. This window displays the compilation process, any detected errors and warnings and additional information to the process. If the window is not visible, press <CTRL> + <F2>.



### Compiling the changes with the mouse

- Click on the icon 'Make' in the toolbar. The compilation process is displayed in the sheet 'Build' of the message window. Errors and warnings detected during compilation are logged in the corresponding sheets of the message window.



### Compiling the changes with the keyboard

- Press <F9>.  
The compilation process is displayed in the page 'Build' of the message window. Errors and warnings detected during compilation are logged in the corresponding sheets of the message window.

In most cases while compiling for a first time the different compilers detect programming errors, such as a variable name which has been used twice or typing errors. In those cases a message appears in the sheet 'Build' of the message window, announcing the number of detected errors and warnings.

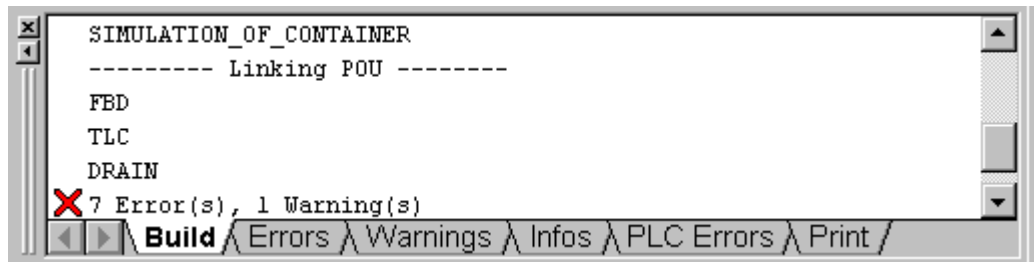


Figure 11-4: Announced errors after making a project

To display the detected errors, click on the tab 'Errors' in the message window. The error list is then shown in the message window.

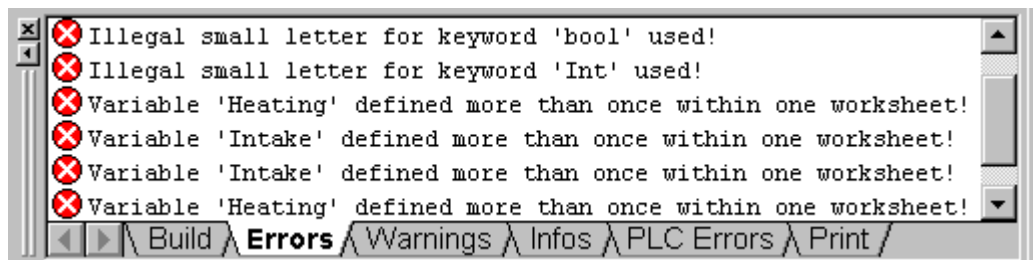


Figure 11-5: Error list, displayed in the message window

In order to display the list of warnings, just click on the tab 'Warnings'.

In most cases double clicking on a displayed error/warning will open directly the worksheet where the programming error/the reason for the warning is occurred. The corresponding line or the object is marked. You can also mark the error and press <F1> to get the corresponding help topic with information about the cause of the error and what steps have to be done now.



Having corrected the first error you can press <CTRL> + <F12> to go directly to the worksheet with the next error.

---

## Patching POU

To patch a POU means, that changes you have done while debugging a project are compiled, the corresponding code is generated and downloaded automatically to the PLC in one step.

If you have detected a programming error using the online mode and you have switched to the offline mode to remove the programming error you can use 'Patch POU' to compile these changes. The changes are downloaded automatically so that you can view them immediately having switched into online mode.

The following table lists the cases where Patch POU can be used.

Language	You can use Patch POU if ...
all	<ul style="list-style-type: none"><li>* new local variables have been inserted.</li><li>* you have inserted a new function call if the function has already been called in the POU before.</li><li>* you have inserted a function block call if the function block has already been instantiated in the POU before.</li><li>* new, empty lines has been inserted.</li><li>* comments have been changed or added.</li><li>* inserting global variables if they had already been declared as VAR_EXTERNAL in the POU before.</li><li>* you have deleted variables which are not used in *.csv.</li></ul>
IL	<ul style="list-style-type: none"><li>* you have changed or inserted IL operators.</li><li>* you have changed the nesting level.</li><li>* new local variables have been declared.</li></ul>
ST	<ul style="list-style-type: none"><li>* you have changed statements or expressions.</li></ul>
FBD	<ul style="list-style-type: none"><li>* existing networks have been changed.</li></ul>
LD	<ul style="list-style-type: none"><li>* existing networks have been changed.</li></ul>
SFC	<ul style="list-style-type: none"><li>* FBD networks, LD networks or variables connected to direct transitions have been changed.</li><li>* the time interval of time action qualifier has been changed.</li><li>* the variable name in action blocks has been changed.</li></ul>

Figure 11-6: Premises for the use of the 'Patch POU' operation



If you want to use 'Patch POU' for POU's where new variables have been added, you should activate the POU memory reserve for Patch POU. Therefore activate the checkboxes 'Use reserve' in the properties dialog of the POU and in the dialog 'Resource settings'. In the dialog 'Data Area' the reserve size can be set.

Patch POU cannot be used in the following cases:

- \* after changing string constants, user defined strings, variables, the physical hardware and the formal parameters of functions and function blocks (VAR\_INPUT, VAR\_OUTPUT and VAR\_IN\_OUT).
- \* after inserting new string constants, user defined strings, functions, function block instances, global variables, POUs, libraries and CASE statements without using the reserve.
- \* after deleting POUs or libraries.



The menu item 'Patch POU' is only available if you switch the worksheet in offline mode by clicking on the icon 'Debug on/off' in the toolbar.



### Patching POUs with the mouse

- Make sure that the worksheet is the active window.
- Make sure that the worksheet is in offline mode. In offline mode, the corresponding icon 'Debug on/off' appears not pressed (as shown beneath).
- Edit your worksheet and correct any detected programming errors.
- Click on the submenu 'Build' and select the menu item 'Patch POU'.  
The compilation process is started and its progress is displayed in the message window.  
Detected errors and warnings can be displayed in the corresponding sheets of the message window.



The steps to display error and warning messages and to open the corresponding worksheets are described in the previous section 'Compiling a project using Make'.

---

## Downloading the project

Having compiled your project using 'Make' or 'Rebuild Project' you have to download it to the simulation or PLC. If you have patched a POU the new generated code is downloaded automatically.



The target of the downloading process is set in the dialog 'Resource settings...'. Choose the menu item 'PLC Settings...' in the context menu of the resource to call the dialog. You can see to which simulation or port the PLC program is sent.



### Downloading the project with the mouse

- Click on the icon 'Show Control Dialog' in the toolbar. The control dialog appears directly if only one resource is available. If the project tree contains several resources, the dialog 'Select resource' appears. In this case choose the desired resource to which the project is to be downloaded and confirm the dialog to display the control dialog.



### Downloading the project with the keyboard

- Press <CTRL> + <F10>. The control dialog appears directly if only one resource is available. If the project tree contains several resources, the dialog 'Select resource' appears. In this case choose the desired resource to which the project is to be downloaded and press <↓> to display the control dialog.



Figure 11-7: Control dialog



### Using the control dialog

- Press the button 'Download'. The dialog 'Download' appears.

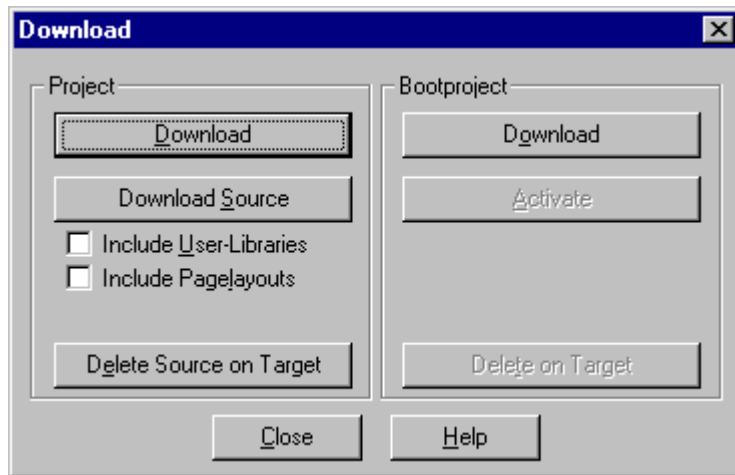


Figure 11-8: Dialog 'Download'



### Using the dialog 'Download'

- Press the button 'Download' in the 'Project' area.  
The full project including POU's and the configuration data is downloaded to the target.  
The PLC passes into the state 'STOP', which is displayed below the title bar in the control dialog.
- Press the button 'Cold' in the control dialog to start the program execution.




Please refer to the context-sensitive Help for detailed information about downloading.

---

## Calling worksheets in online mode

Having compiled and downloaded the project to the target it is possible to call the editors in online mode for debugging the worksheets. To call the worksheets in online mode you have the following possibilities:

- \* activate the icon 'Debug on/off' in the toolbar or press <F10>. By activating this icon or pressing <F10> all worksheets already opened are switched automatically to online mode. If you open a new worksheet from the project tree or instance tree, this is also called in online mode. The icon 'Debug on/off' appears pressed if online mode is switched on. Refer also to the following note. 
- \* choose the menu item 'Open instance' in the context menu of a POU in the project tree or in the context menu of an already opened worksheet. Refer also to the following note.





If already opened function block code bodies or program code bodies are instantiated several times and you want to debug these worksheets, i.e. calling in online mode by activating the icon 'Debug on/off', a warning message appears indicating that you have to use 'Open instance' to call these worksheets in online mode. In this case choose the menu item 'Open instance' in the context menu of the corresponding worksheets. Choosing this menu item the dialog 'Open Instance' appears where you have to choose the desired instance.

The dialog 'Open Instance' also appears if online mode is switched on and you want to open a function block code body or program code body from the project tree, which is instantiated several times.

For the next steps let us assume that you want debug a textual worksheet in online mode.



### Calling a textual worksheet in online mode with the mouse

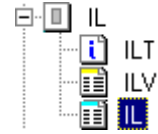
- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the worksheet. For this purpose double click on the POU name. 
- Double click on the worksheet icon (e.g. 'IL'). The worksheet is opened.
- Click on the icon 'Debug on/off'. All opened worksheets are switched to online mode. If online mode is switched on, the icon appears 'pressed'. 





## Calling a textual worksheet in online mode with the keyboard

- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the worksheet as follows: Press < > or < > to highlight the POU name and press < > to open the subtree.
- Press < > or < > to mark the worksheet icon (e.g. 'IL') and then press <↵>. The worksheet is opened.
- Press <F10>. All opened worksheets are switched to online mode. If online mode is switched on, the icon 'Debug on/off' appears 'pressed'.



```
1      FALSE LD    %IX0.2    (*direct variables*)
2      FALSE AND   %IX0.3
3      FALSE OR    Action_INIT
4      FALSE ST    IL_VAR
5
6      FALSE LD    Input_IX0_0
7      JMPC   MANUAL
8
9      (* Timer FB TON *)
10     TRUE  LD    Timer_start
11     TRUE  ST    TON_IL.IN
12     1.500 LD    PT_TON_IL
13     1.500 ST    TON_IL.PT
```

ll.ab

Figure 11-9: IL worksheet in online mode

In textual worksheets in online mode a gray line separates the worksheet into two parts. On the left the online values are displayed. On the right the code body is shown.



You can change the width of the columns by positioning the mouse pointer on the gray separation line (when the correct position is reached, the cursor changes to a double line), pressing and holding the left mouse button while moving the cursor to the left or to the right.

In the following figure an example for a graphical worksheet in online mode is shown:

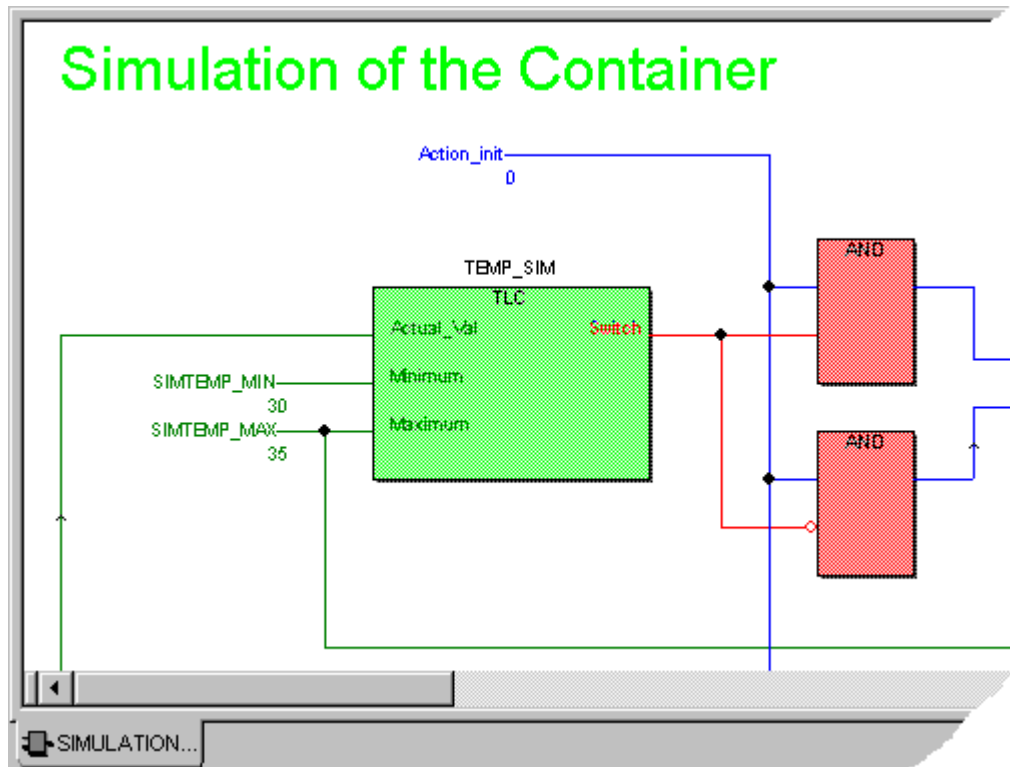


Figure 11-10: Graphical worksheet in online mode

The colors which are used in textual and in graphical worksheets have the following meanings:

Color	Meaning
Red	Boolean true
Blue	Boolean false
Green	integer values
Grey	a set breakpoint
Yellow	a reached breakpoint

Figure 11-11: Meaning of the colors in online mode



In graphical worksheets the way how the online values are displayed can be changed choosing the menu item 'Online Layout' in the submenu 'Online'.

---

## Switching between online and offline mode

If you have detected a programming error in any worksheet in online mode it is possible to switch to the offline mode to correct the programming error immediately. Having corrected the error the changes have to be compiled and sent to the target using 'Patch POU'.



### Switching between online and offline mode with the mouse

- Click on the icon 'Debug on/off' in the toolbar.  
All opened worksheets are switched automatically to offline mode. The icon appears not pressed if offline mode is switched on.
- Correct the programming errors.
- Choose the menu item 'Patch POU' in the submenu 'Build'. The changes you have done are compiled and sent to the target.
- Click on the icon 'Debug on/off' in the toolbar.  
All worksheets are switched automatically to online mode. The icon 'Debug on/off' appears pressed if online mode is switched on.



### Switching between online and offline mode with the keyboard

- Press <F10>.  
All opened worksheets are switched automatically to offline mode. The icon 'Debug on/off' appears not pressed if offline mode is switched on.
- Correct the programming errors.
- Press <ALT> + <F9>.  
The changes you have done are compiled and sent to the target.
- Press <F10>.  
All worksheets are switched automatically to online mode. The icon 'Debug on/off' appears pressed if online mode is switched on.



For information about calling function block code bodies and program code bodies in online mode, which are instantiated several times, refer to the previous section 'Calling worksheets in online mode'.

---

## Switching to address status and powerflow

In online mode you use normally the variable status in which you can check the behavior of the variables. In the variable status you get the values stored in the I/O image at the end of a working cycle. But you can also switch to the address status. The address status displays the current values of the accumulator corresponding to the current moment of the program execution. Having activated the address status you see also the powerflow display. Powerflow means that it is displayed which program parts are actually executed and which ones not. This is e.g. important for debugging worksheets with conditional jumps.



It is not possible to switch to the address status without powerflow. The address status and powerflow can only be used together.

If you have opened a function worksheet in online mode the address status with powerflow is always switched on.



### Switching to address status and powerflow with the mouse

- Click on the submenu 'Online'.  
The submenu is opened.
- Choose the menu item 'Powerflow'.  
The worksheet is switched to the address status with powerflow.

In the text editor in online mode powerflow is displayed with three different signs. The meaning of these signs is explained in the following table:

Powerflow sign	Meaning
Horizontal line	program part is not working
Vertical line	program part is working
Vertical, double line	program part is working in several POU's

Figure 11-12: Meaning of the powerflow signs in text worksheets in online mode

In the graphic editor in online mode powerflow is displayed with colored lines. The meaning of these lines is explained in the following table:

Powerflow sign	Meaning
Red line	True
Blue line	False
Small, horizontal line	program part is working
Small, vertical line	program part is not working
Small, vertical double line	program part is working in several POU's

Figure 11-13: Meaning of the powerflow signs in graphic worksheets in online mode

---

## Forcing and overwriting variables

In online mode variables can be forced or overwritten. Forcing and overwriting means assigning to a variable a new value. In the case of overwriting the new value of the variable is only used for one working cycle. Having finished this cycle the variable is processed normally. Forcing means using the new value for the variable until the forced variable is reset to its normal value by the user. The steps to do for forcing and overwriting are nearly the same.



Be very careful forcing or overwriting variables while your PLC is running. Forcing and overwriting variables mean that the PLC program is executed with the values of the forced or overwritten variable.

Only physical inputs and outputs can be forced.



### Forcing variables with the mouse

- Choose the menu item 'Online dialog...' in the context menu of the variable in your worksheet in online mode. The dialog 'Online Debug' appears.



### Forcing variables with the keyboard

- Press the cursor keys to go to the variable in your worksheet in online mode.
- Press <SHIFT> and keep it pressed. Press < > or < > to mark the variable.
- Press <↵>. The dialog 'Online Debug' appears.

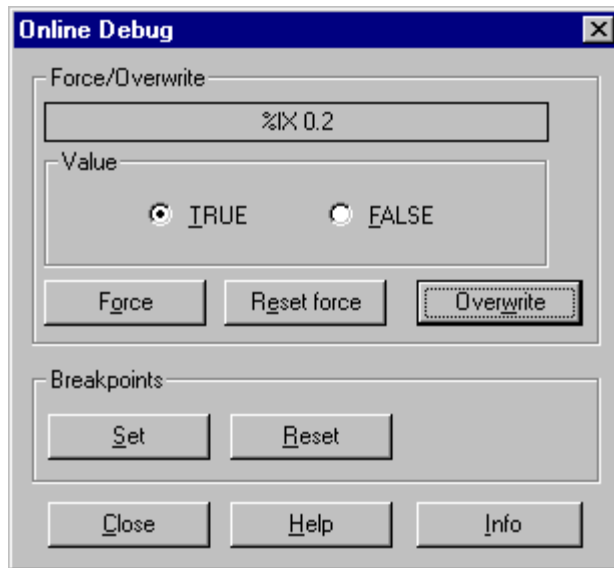


Figure 11-14: Dialog 'Online Debug'



### Using the dialog 'Online Debug'

- Activate the radio button 'TRUE' or 'FALSE'.  
If you force a non-Boolean variable the dialog includes an input field. In this case enter the desired value.
- Press the button 'Force'.
- Confirm the dialog.  
The variable is forced.

The programming system allows to reset a particular forced variable or all forced variables at the same time.



### Resetting forced variables

- Choose the menu item 'Online dialog...' in the context menu of the forced variable in your worksheet in online mode.  
The dialog 'Online Debug' appears as shown in figure 11-14.
- Press the button 'Reset force' to reset the marked variable.
- To reset all forced variables at the same time, press the button 'Info' in the dialog 'Online Debug'.  
The dialog 'Resource: *resource name*' appears.
- Activate the checkbox 'Reset forcelist' in the 'Force' area and confirm the dialog. All forced variables are reset at the same time.

---

## Setting and resetting breakpoints

In online mode breakpoints can be set or reset in code body worksheets. If a breakpoint is set the program execution halts at this point. The program execution continues at the breakpoint if the button 'Go' in the control dialog is pressed. To restart the program execution at the program beginning press the button 'Restart' in the control dialog.



Be very careful using breakpoints while your PLC is running. Breakpoints mean that the program execution is halted at the point where the breakpoint has been set.

The behavior of the I/O's when reaching a breakpoint depends on the PLC type.

Breakpoints cannot be set in function block instances. Setting a breakpoint in an instance means automatically that the program execution is halted each time when the function block is used.



### Setting a breakpoint with the mouse

- Double click on a line or an object in your code body worksheet in online mode.  
The dialog 'Online Debug' appears.



### Setting a breakpoint with the keyboard

- Press the cursor keys to go to the line or object in your code body worksheet in online mode.
- Press <↵>.  
The dialog 'Online Debug' appears.

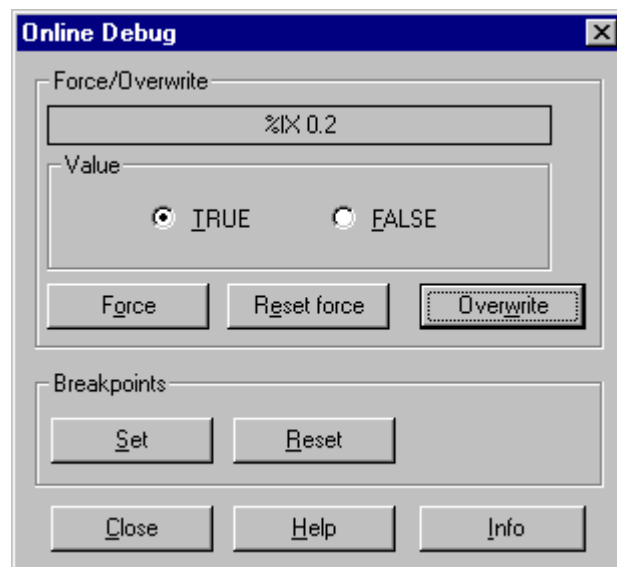


Figure 11-15: Dialog 'Online Debug'



## Using the dialog 'Online Debug'

- Press the button 'Set'.  
The breakpoint is set.



In any worksheets a set breakpoint is represented in gray, a reached breakpoint in yellow color.



If you have set any breakpoints the programming system provides several functions for debugging your PLC program. These functions are described in the following section 'Debugging with set breakpoints'.

The programming system allows to reset a particular breakpoint or all breakpoints at the same time.



## Resetting breakpoints

- Double click on the line or object in your graphical worksheet in online mode, for which a breakpoint has been set.  
The dialog 'Online Debug' appears as shown in figure 11-15.
- Press the button 'Reset' to reset the breakpoint for the marked object.
- To reset all breakpoints at the same time, press the button 'Info' in the dialog 'Online Debug'.  
The dialog 'Resource: *resource name*' appears.
- Activate the checkbox 'Reset breakpoints' in the 'Breakpoints' area and confirm the dialog. All breakpoints are reset at the same time.

---

## Debugging with set breakpoints

The programming system provides a step and trace function which can be used for debugging your PLC program. These functions allow to continue the program execution line by line after a breakpoint has been reached. Stepping means that if a function or a function block call is reached, the call is stepped over, i.e. the code body of the function or function block is not debugged. Tracing means that if a function or a function block call is reached, the function or function block code body is opened for debugging. Stepping and tracing is performed using the control dialog.



Stepping and tracing is only possible if a breakpoint has been set.

For the next steps let us assume that you have set a breakpoint in a ST code body worksheet in online mode.





## Stepping and tracing after a breakpoint is reached using the mouse

- If the control dialog is not visible, click on the icon 'Show Control Dialog' in the toolbar. The control dialog appears.



## Stepping and tracing after a breakpoint is reached using the keyboard

- If the control dialog is not visible, press <CTRL> + <F10>. The control dialog appears.



Figure 11-16: Control dialog if a breakpoint is set



## Using the control dialog

- Press the button 'Step' to go to the next line or object and to halt the program execution at this point. If a function or function block call is reached, the call is stepped over and the next line is highlighted.
- Press the button 'Trace' to go to the next line or object and to halt the program execution at this point. If a function or function block call is reached, the function or function block code body is opened for debugging.



The line or the object where the program execution is halted after pressing 'Step' or 'Trace' is highlighted in yellow color. In addition in textual worksheets a yellow arrow is used to highlight the line where the program execution is halted (as shown in the following figure).

The following figure displays an example for a ST worksheet in online mode with set breakpoint and halted program execution.

```

1      2  CASE MODUS OF
2      -8000 1: ROBOT_X:= ROBOT_X + 200;
3      -4000 ROBOT_Z := ROBOT_Z + ADD_ARM ;
4      2     MODUS:=1;
5      -8000 IF ROBOT_X >= ROBOT_RANGE_POS_1 THEN
6      2     MODUS:=2;
7      END_IF;
8      -8000 2: ROBOT_X:= ROBOT_X - 200;
9      -4000 ROBOT_Z := ROBOT_Z - ADD_ARM ;
10     2     MODUS:=2;
11     -8000 IF ROBOT_X <= ROBOT_RANGE_NEG_1 THEN
12     2     MODUS:=1;
13     END_IF;
14     END_CASE;
15     -8000 ROBOT_Y:= ROBOT_X;
16     325   COUNTER_1 := COUNTER_1+1;
17     325   IF COUNTER_1 >1000 THEN
18     325   COUNTER_1 :=0;
19     END_IF;

```

Figure 11-17: ST worksheet in online mode with set breakpoint

## Using the watch window

The watch window can be used to collect variables from different worksheets to get an appreciation about how these variables work together. If a variable is added once to the watch window, the associated worksheet must not be opened to monitor the current value. Furthermore the watch window is used to debug elements of user defined data types such as arrays and structures.



The procedures how to debug user defined data types using the watch window is described in the following section 'Debugging user defined data types using the watch window'.

In the watch window you can insert and delete variables. There is no limit for the number of variables inserted in the watch window. In addition you can use the watch window to force and overwrite variables and to set and reset breakpoints. This is performed by choosing the menu item 'Online dialog...' in the context menu of a variable.

For the next description let us assume that you want to insert a variable to the watch window. The first step to do is to call the watch window and then to write the variable into it.



## Calling the watch window with the mouse

- Click on the submenu 'View'.  
The submenu is opened.
- Choose the menu item 'Watch Window'.  
The watch window appears.



## Calling the watch window with the keyboard

- Press <ALT> + <V> to open the submenu 'View'.
- Press <A> to open the watch window.

The watch window is displayed as follows:

Variable	Value	Type	Instance
Level	750	INT	C_IPC.R_IPC.T_100ms.PRG_FBD.Level
Temp	34	INT	C_IPC.R_IPC.T_100ms.PRG_FBD.Temp
Intake_Q	FALSE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Intake_Q
Heater_Q	FALSE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Heater_Q
Drain_Q	TRUE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Drain_Q
Container States		Container_Struct_Array	C_IPC.R_IPC.T_100ms.PRG_FBD.Container_States

At the bottom of the window, there are tabs labeled 'Watch 1', 'Watch 2', 'Watch 3', and 'Watch 4'. 'Watch 1' is currently selected.

Figure 11-18: Watch window with several variables inserted

The watch window contains the following information:

- \* 'Variable': Displays the variable name. User defined data types such as arrays and structures are marked with a '+' (as for example 'Container States' in the above figure). To display the elements of these data types, click on the '+' sign. Normal variables are displayed such as 'Level' in the above example.
- \* 'Value': Displays the current value of the variable.
- \* 'Type': Displays the data type.
- \* 'Instance': Displays the instance path where the variable is used. The path always contains the configuration, resource, task, the associated program name and variable name.

The watch window allows to manage several pages where every page can be used independently. The individual pages are called by clicking on the sheet tabs at the bottom of the watch window.



## Inserting variables in the watch window with the mouse

- Mark the variable in the worksheet in online mode.
- Click the right mouse button to open the context menu.
- Choose the menu item 'Add to Watch Window'.  
The variable is inserted in the watch window.



If you want to delete variables in the watch window, mark the desired variable in the watch window and choose 'Delete' in the context menu of the variable.

---

## Debugging user defined data types using the watch window

For debugging user defined data types such as arrays and structures the watch window has to be used. The first step is to call the watch window and then to write the variable being an array or a structure into it. The second step is debugging the elements of the array or structure in the watch window.

For the next steps let us assume that you want to debug an array of structure. The first thing to do is to call the watch window and to insert the variable being the array of structure as it has been described in the previous section 'Using the watch window' of this chapter. Figure 11-18 shows how an array of structure (e.g. 'Container States') is displayed in the watch window. Having inserted the array of structure in the watch window the components of the array of structure must be displayed.

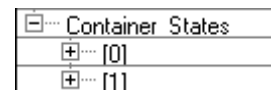


### Debugging an array of structure with the mouse

- In the watch window click on the '+' sign associated to the array of structure.



The components of the array of structure are displayed.



- Click on the '+' sign associated to a structure (e.g. [0]).

The components of the structure are displayed as shown in the following figure.

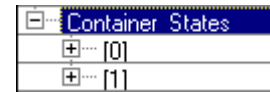


## Debugging an array of structure with the keyboard

- In the watch window press < > or < > to go to the line with the array of structure.



Press < > to display the components of the array of structure.



- Press < > to go to a structure (e.g. [0]) and press < > to display the components of the structure.

The components of the structure are displayed as shown in the following figure.

Variable	Value	Type	Instance
[-] Container States		Container_Struct_Array	C_IPC.R_IPC....
[-] [0]		Container_Struct	C_IPC.R_IPC....
[-] Drain	FALSE	BOOL	C_IPC.R_IPC....
[-] Heater	TRUE	BOOL	C_IPC.R_IPC....
[-] Intake	FALSE	BOOL	C_IPC.R_IPC....
[-] Level	650	INT	C_IPC.R_IPC....
[-] Temp	30	INT	C_IPC.R_IPC....
[-] State of container	Intake: CLOSE Heater: ON Drain: CLOSE	STRING	C_IPC.R_IPC....
[-] Values of container	Level: 650 Liter Temperature: 30 Grad	STRING	C_IPC.R_IPC....
[+] [1]		Container_Struct	C_IPC.R_IPC....

Watch 1 / Watch 2 / Watch 3 / Watch 4 /

Figure 11-19: Watch window with opened array of structure



Having displayed the components of an user defined data type you can use the watch window to force and overwrite variables and to set and reset breakpoints. This is performed by choosing the menu item 'Online dialog...' in the context menu of a variable in the watch window.



The procedures how to force and overwrite variables and to set and reset breakpoints are described in the sections 'Forcing and overwriting variables' and 'Setting and resetting breakpoints' in this chapter.

# Printing your project with a customized pagelayout

**This chapter provides information about...**

- printing your project
- using the pagelayout editor
- using preview

# Printing your project with a customized pagelayout

---

## Printing the project

The submenu 'File' contains the commands used to define the printer settings, to display the preview of the current page and to print the entire project or parts of it.

The program offers several possibilities to print your project documentation. To set the printing options, the program provides two dialogs:

- \* In the dialog 'Print Project' you can select the items to be printed and the used print mode.
- \* On the page 'Default Pagelayout' in the dialog 'Options' you can define the pagelayout which is used to print the data.

The above mentioned dialogs and the various print modes are described in the following sections.

## Controlling the print process using the dialog 'Print Project'

If you want to print only parts of your project, you first have to select the desired folder icon in the project tree (e.g. a POU icon) before calling the dialog 'Print Project'.



It is not possible to print libraries and worksheets in online mode.



### Selecting the desired folder icon (node) in the project tree

- Select the desired folders in the project tree, which you want to print. This could be for example a single POU or the whole subtree 'Logical POU's'. If you want to print the whole project, skip this step and proceed with the next step.



### Calling the dialog 'Print Project'

- Select the menu item 'Print Project' in the submenu 'File'. The dialog 'Print Project' appears as shown in the following figure.

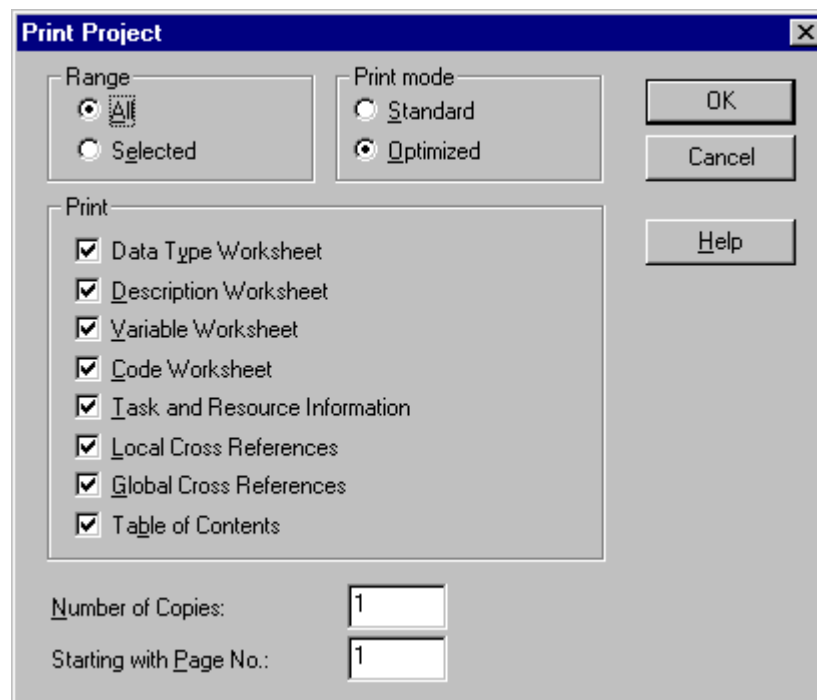


Figure 12-1: Dialog 'Print Project'



### Using the dialog 'Print Project'

- Select the range to be printed.
  - \* 'All' means the entire project, including all subtrees in the project tree.
  - \* 'Selected' means, that only the nodes which are marked in the project tree before calling the dialog 'Print Project' are printed.
- Select the items to be printed by activating/deactivating the corresponding checkboxes in the area 'Print'. Keep in mind, that each checkbox relates to the marked range ('All' or 'Selected'). Example: If you mark the checkboxes 'Description Worksheet', 'Variable Worksheet' and 'Code Worksheet' only those worksheets are going to be printed, which are within the selected range.



- Select the desired **print mode**:
  - \* **'Standard'** means, that each selected item is printed with the associated default pagelayout.  
In order to set the default pagelayout for the standard print mode call the dialog 'Options', open the page 'Default Pagelayout' and edit the fields 'Text' and 'Graphic'.
  - \* **'Optimized'** means that the selected items are printed with less paper as possible. All worksheets are printed one after the other. A new page is only used, if a new POU or a new step (in SFC) starts. While using optimized printing a table of contents and cross references can be printed.  
The default pagelayout which is used for optimized printing is set in the dialog 'Options' on the page 'Default Pagelayout' (list box 'Optimized Printing').



The steps to define a certain pagelayout as default pagelayout for the different print modes are described in the section 'Defining a pagelayout as default pagelayout' in this chapter.



The procedures how to create and edit a pagelayout are described in the section 'Editing a pagelayout' in this chapter.

- If desired, change the number of copies to be printed.
- Enter another page number for the first page to be printed.
- Confirm the dialog.  
The selected data are printed on the standard printer.

## Defining a pagelayout as default pagelayout

You can define a pagelayout as default pagelayout (e.g. a customized pagelayout, using the pagelayout editor). This default pagelayout is used automatically when printing your project or parts of it. The default pagelayout is set in the dialog 'Options'.



### Calling the page 'Default Pagelayouts' in the dialog 'Options'

- Select the menu item 'Options' in the submenu 'Extras'.  
The dialog 'Options' appears.
- Click on the tab 'Default Pagelayouts'.

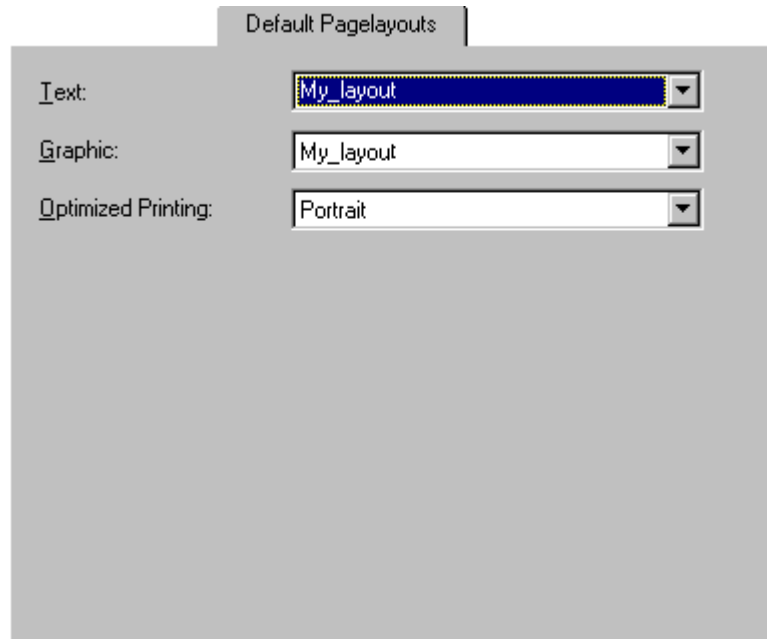


Figure 12-2: Page 'Default Pagelayouts' in the dialog 'Options'



### Using the dialog 'Default Pagelayouts'

The page provides three list boxes used to define different default pagelayouts. Each list box contains all pagelayouts which are stored in the corresponding pagelayout directory. This can either be predefined layouts (delivered with the program) or customized layouts, which you have created using the pagelayout editor.

In figure 12-2 the customized pagelayout 'My\_layout' is defined as default pagelayout for text and graphic in standard print mode. For the print mode 'Optimized' the predefined layout 'Portrait' is selected.

- Open each list box and select the desired pagelayout.
  - \* **'Text'**: Defines the pagelayout for textual worksheets, such as description worksheets, variable worksheets, textual code bodies, cross references, etc. The setting in the field 'Text' is only relevant when using the print mode 'Standard'.
  - \* **'Graphic'**: Defines the pagelayout for graphical worksheets, such as code bodies in FBD, LD or SFC. The setting in the field 'Graphic' is only relevant when using the print mode 'Standard'.
  - \* **'Optimized Printing'**: Defines the pagelayout for text and graphic when using the print mode 'Optimized'.
- Confirm the dialog.

---

## Using the pagelayout editor

The pagelayout editor is a tool which allows you to create new pagelayouts or edit existing pagelayouts.

After saving a new pagelayout, it is stored in the corresponding directory and can be selected as default pagelayout in the dialog 'Options'.

The first step before editing your own pagelayout is to call the pagelayout editor.



### Calling the pagelayout editor

- Choose the menu item 'Pagelayout Editor' in the submenu 'Extras'.  
The pagelayout editor is opened.



The pagelayout editor is normally opened with the default pagelayout.

## Creating a new pagelayout

To create a new pagelayout, perform the following steps:



### Creating a new pagelayout with the mouse

- Choose the menu item 'New' in the submenu 'File'.  
The pagelayout editor with the pagelayout 'untitled' is opened.



### Creating a new pagelayout with the keyboard

- Press <ALT> + <F>.  
The submenu 'File' is opened.
- Press <N>.  
The pagelayout editor with the pagelayout 'untitled' is opened.

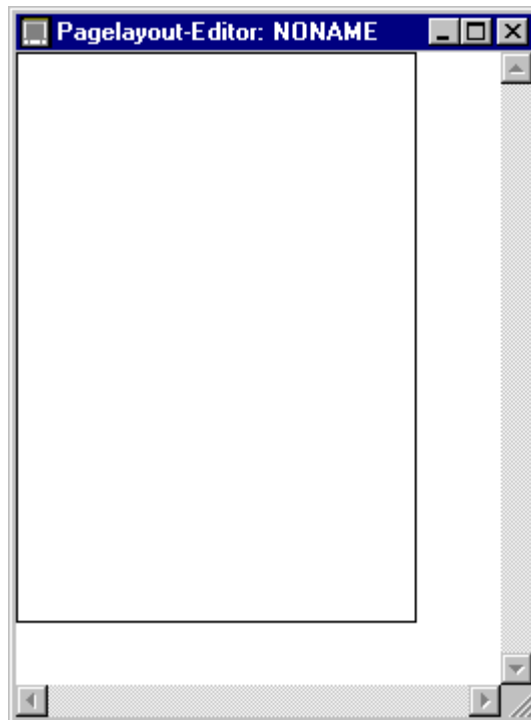


Figure 12-3: Pagelayout editor with a new pagelayout

## Defining the source area

Having created a new pagelayout you should first define the area where you want to place the contents of the worksheets. This area is called 'source area' and is represented with a red rectangle.



### Defining the source area with the mouse

- Click on the icon 'Source area' in the toolbar. A rectangle is added to the shape of the cursor.
- Place the mouse cursor to the position of one intended source area corner.
- Press the left mouse button and keep it pressed.
- Move the mouse drawing a rectangle.
- Release the mouse button. The source area is drawn.





## Defining the source area with the keyboard

- Press <S>.  
A rectangle is added to the shape of the cursor.
- Press the cursor keys to move to the position of one intended source area corner.
- Press <SPACE> and keep it pressed.
- Press the cursor keys to move to the position of the opposite corner of the rectangle.
- Release <SPACE>.  
The source area is drawn.

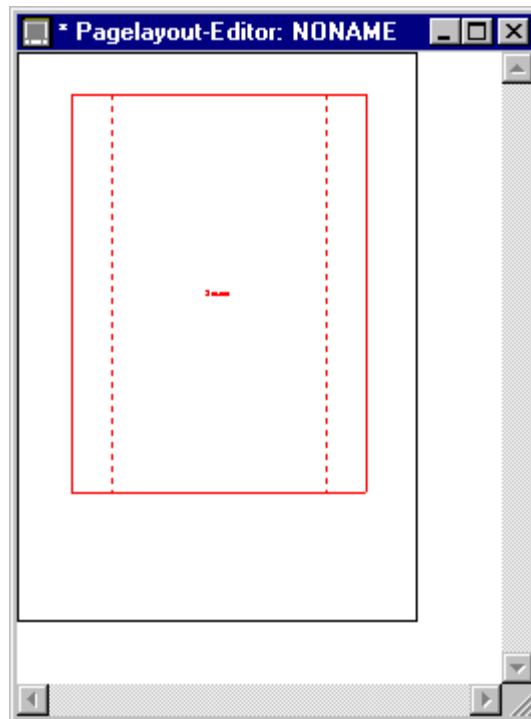


Figure -4



To move the new source to another position on the page, mark it with a left mouse click and keep the mouse button pressed.

mouse.



To change the source code area size after creating it, click on properties...'. The dialog 'Settings source area' appears. Using this dialog you can change the position, size and other source

## Inserting elements in your pagelayout

You can insert several elements in your pagelayout, such as rectangles, horizontal line at the bottom of the page.



### Inserting a line with the mouse

Click on the icon 'Line' in the toolbar.



- Press the left mouse button and keep it pressed.  
Move the mouse drawing a line.
- The line is drawn.

### Inserting a line with the keyboard

- A line is added to the shape of the cursor.
- 
- Press the cursor keys to move to the position where you want the line to
- Release <SPACE>.




The ways how rectangles, texts and bitmaps can be inserted in the pagelayout

## Editing environment items

Another kind of objects which can be inserted in a pagelayout are . Environment items are place holders for e.g. page numbers, the company name or other kind of texts. Several standard

### Editing environment items with the mouse

-  in the toolbar. A 'T' is added to the shape of the cursor.



Click in the editing field where you want to insert the environment item.



- Press <V>.
- Press the cursor keys to go to the position where you want to insert the
- Press <SPACE>.



- 
- 
- 

The place holder with the name of the item is inserted in the pagelayout.



existing items are explained in the context-sensitive Help.



To view the contents of the place holder you can use preview.

---

## Using preview

allows to get an appreciation of the appearance of your worksheet as it would be for printing. It helps you to organize the elements on the

Preview means that the content of the active window is shown with the with the default pagelayout.



Cross references are not displayed using preview.



- Make sure that the desired worksheet is the active window.

Choose the menu item 'Print Preview...' in the submenu 'File'.  
Preview is called and the active worksheet is displayed.



### Calling preview with the keyboard

Make sure that the desired worksheet is the active window.

- The submenu 'File' is opened.
- Preview is called and the active worksheet is displayed.



# APPENDIX 1

## Editing specific inline code

**This chapter provides information about...**

- editing specific inline code statements in IL
- editing specific inline code statements in LD
- online monitoring of worksheets containing specific inline code statements

# Editing specific inline code

---

## General information



This appendix contains only information relevant for editing specific inline code statements and for online monitoring of textual and graphical worksheets containing these statements. All procedures, which are generally applicable are described in detail in the preceding chapters of this manual.

Specific inline code statements can be used in the textual language IL and the graphical language LD. Specific means, that the statements are provided by a distinct PLC manufacturer and that they are only applicable in conjunction with this specific PLC. For compiling and linking these inline code statements the appropriate DLLs, delivered with the programming system are required.

In IL the specific statements are embedded in a code block, which uses the keywords `BEGIN_SPECIFIC` and `END_SPECIFIC` to switch between the IEC mode and the specific mode. In LD the programming system provides a so called arithmetic box, in which the user has to enter the specific code statements. This arithmetic box can be inserted already connected to an existing LD network or as a single object, which has to be connected to the LD network using the connection mode.

When editing specific code statements in LD in the arithmetic box, the programming system provides a helpful syntax check function, which allows to check the entered statements directly within in the arithmetic box. If the programming system detects any errors in the syntax the corresponding dialog box is extended automatically by a new field containing the error and warning messages detected by the check routines. The error handling is then equal to the error handling in the general programming system, i.e. by double clicking on a particular error message, the corresponding error is marked in the arithmetic box.

---

## Which information do you get in this appendix?

This appendix is divided into the following main sections:

- \* Editing specific inline code in IL - describes the necessary steps to edit specific inline code statements in IL worksheets. It contains information how to insert the specific keywords using the Edit Wizard and to embed these keywords in an existing instruction list.
- \* Editing specific inline code in LD - contains information about using the arithmetic box in LD networks, in which the specific code statements has to be entered. The chapter describes the procedures how to insert the arithmetic box as a single object or in an existing LD network. When inserted as a single object, the necessary steps are explained how to connect the arithmetic box to the LD network.
- \* Online monitoring - describes the online handling of IL and LD worksheets, which contain specific code statements and explains how the online values are displayed in the worksheets. Additionally the necessary steps are described for using the watch window to monitor the online values.



All general information and procedures, which are valid for both, the general programming and the specific code programming are described in the preceding chapters of this manual.

---

# Editing specific inline code in IL

## Calling the text editor with an IL worksheet

Before editing specific inline code in IL you have to call the text editor with the IL worksheet, using the project tree. To perform this, double click on the POU name in the project tree to open the corresponding subtree which contains the relevant IL worksheet. Having performed this, double click on the icon 'Worksheet in IL' in the project tree to open the worksheet.



For a detailed description how to call the text editor for the IL code body worksheet refer to section 'Calling the text editor with an IL worksheet' in chapter 'Editing in IL'.



A general description of handling the project tree and browsing through POUs and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

## Specific code blocks

In IL the keywords `BEGIN_SPECIFIC` and `END_SPECIFIC` are used to define the code blocks containing the specific code statements. These keywords indicate the switch between IEC mode and the specific mode. They can be inserted by just typing them or using the Edit Wizard.



For a description how to insert the specific keywords using the Edit Wizard refer to the following section 'Inserting specific code blocks using the Edit Wizard'.

Following the start keyword `BEGIN_SPECIFIC`, which introduces programming in the specific mode, the code statements has to be entered in the usual way, i.e. the complete text editor functionality can be used when entering the specific code. When editing specific code comments can also be inserted using asterisks and parentheses according to editing in IEC IL. Having entered the statements the keyword `END_SPECIFIC` is required to finish the specific code block and to switch back to the IEC mode. The following figure shows how a specific code block is defined in an IL worksheet.

```
BEGIN_SPECIFIC          (* introducing specific code *)
  R1 = WX10 == WX20
  R2 = WX10 < WX20
  R3 = WX20 < WX10
END_SPECIFIC           (* finishing specific code *)
```

Figure A1-1: Example of a specific code block in IL

In order to execute the statements in the specific code block, a condition must be loaded into the accumulator in the preceding IEC IL, which decides whether the following specific code is executed or not. If the preceding IEC IL operator returns the boolean value TRUE, the specific statements are executed otherwise the code block is ignored.

The following figure shows an example for an instruction list with embedded specific code statements. In this example the code statements are executed in any case, as the preceding operator 'LD' returns the boolean value TRUE.

```
LD    X2
AND(  X0
OR    R0
)
ANDN  R1
ST    R0
LD    TRUE

BEGIN_SPECIFIC      (* introducing specific code *)
  R1 = WX10 == WX20
  R2 = WX10 < WX20
  R3 = WX20 < WX10
END_SPECIFIC        (* finishing specific code *)
```

Figure A1-2: Example of a specific code block embedded in an IEC IL

## Inserting specific code blocks using the Edit Wizard

Specific inline code blocks can be inserted by just typing them or using the Edit Wizard. Using the Edit Wizard simplifies editing in the text editors and prevents from entering syntactical faults, such as wrong instruction sequences. By applying the Edit Wizard pre-edited operators are inserted, i.e. the structure is already completed by placeholders.



A general description of the Edit Wizard can be found in the section 'The Edit Wizard' in chapter 'Getting started' in this manual.

For the next steps let us assume that you want to insert a specific code block in an existing IL worksheet, which already contains an IEC instruction list using the Edit Wizard. When inserting the code block, note that the specific statements are only executed, if the preceding IEC IL operator returns the boolean value TRUE, as already described in the previous section.

If the Edit Wizard is not visible in the workspace, perform the following steps:



### Calling the Edit Wizard with the mouse

- Click on the icon 'Edit Wizard' in the toolbar. The Edit Wizard window appears.



### Calling the Edit Wizard with the keyboard

- Press <SHIFT> + <F2>. The Edit Wizard window appears.

If the Edit Wizard is visible, perform the following steps to insert the specific code block using the Edit Wizard with the mouse.



### Inserting the specific code block using the Edit Wizard

- Locate the code body position, where the new code block (including the keywords BEGIN\_SPECIFIC and END\_SPECIFIC) is to be inserted. Click the left mouse button to set a text cursor.
- Press <↵> to insert a new line.
- Open the Edit Wizard list box 'Group' and select the group 'Operators'. The available operators are displayed in the selection area of the Wizard.
- Locate the operator 'BEGIN\_SPECIFIC'.
- Double click on the operator. The keywords BEGIN\_SPECIFIC and END\_SPECIFIC are inserted automatically at the text cursor position.

The next step is to enter the specific code statements following the keyword BEGIN\_SPECIFIC.



### Entering the specific code statements

- Position the text cursor behind the keyword BEGIN\_SPECIFIC and press <↵> to insert a new line.
- Enter the specific code statements in the usual way using the complete text editor functionality. Note that new statements has to start always in a new line.

The following figure shows how the specific code block is embedded in the IEC instruction list.

```

1 LD X2
2 AND( X0
3 OR R0
4 )
5 ANDN R1
6 ST R0
7 LD TRUE
8 BEGIN_SPECIFIC (* introducing specific code *)
9 R1 = WX10 == WX20
10 R2 = WX10 < WX20
11 R3 = WX20 < WX10
12 END_SPECIFIC (* finishing specific code *)

```

IEC IL code  
Specific code statements

Figure A1-3: Instruction list including IEC IL code and specific code statements

## Inserting and declaring variables of specific code statements

While editing the specific code statements in IL you can insert and declare variables in the same way as already described for editing IEC IL code in the chapter 'Editing in LD'. You can either type the variable name in the code body worksheet and declare it afterwards in the variable worksheet of the POU or you can insert and declare a variable while editing the code body worksheet.

If you want to declare a variable of a specific code statement when editing the specific code, perform the following steps:

- call the dialog 'Variable' for the desired variable by marking the variable either with the mouse or the keyboard and clicking on the icon 'Variable' in the toolbar or pressing <F5>.
- If you have made the required settings in the dialog 'Variable', confirm the dialog to display the dialog 'Automatic Variables Declaration'.
- In the dialog 'Automatic Variables Declaration' define the different parameters for the new variable, for example the data type, an initial value etc.
- Having performed these settings, confirm the dialog 'Automatic Variables Declaration'. The new variable declaration is then inserted automatically in the variable declaration of the POU.



For detailed information about inserting variables while editing IEC code and using the dialogs 'Variable' and 'Automatic Variables Declaration' refer to section 'Inserting variables' in the chapter 'Editing in IL' in this manual.

---

# Editing specific inline code in LD

## Calling the graphic editor with a LD worksheet

Before editing specific inline code in LD you have to call the graphic editor with the LD worksheet, using the project tree. To perform this, double click on the POU name in the project tree to open the corresponding subtree which contains the relevant LD worksheet. Having performed this, double click on the icon 'Worksheet in LD' in the project tree to open the LD worksheet.



For a detailed description how to call the graphic editor for the LD code body worksheet refer to section 'Calling the graphic editor with a LD worksheet' in chapter 'Editing in LD'.



A general description of handling the project tree and browsing through POU's and worksheets is contained in the chapters 'Getting started' and 'Editing the project structure' in this manual.

## Arithmetic box with specific inline code

Editing the specific inline code in LD is done using a specific input dialog box, the so called arithmetic box. The statements according to the specific code will be entered directly in this arithmetic box. It includes a text editor functionality, which provides all standard functions of a conventional text editor, such as copying, cutting, pasting etc. IEC comments are also possible using parenthesis and asterisks. While editing in the arithmetic box a maximum of 19 code and comment lines are allowed. If this maximum number is exceeded the following code lines are cut.

An arithmetic box can be inserted already connected to an existing LD network or as a single object. Having inserted the arithmetic box as a single object, you can connect the box to an existing LD network using the connection mode.



An arithmetic box can only be inserted in an existing LD network next to the right power rail, i.e. there are no following objects possible on the right side of the arithmetic box. If you try to insert an arithmetic box for example between two LD objects (e.g. two contacts) a collision message is displayed.



The specific code entered in the arithmetic box is only executed, if the object left to the arithmetic box returns the boolean value TRUE otherwise the specific code is ignored.



A helpful tool when editing the statements in the arithmetic box is the available syntax check function. This function allows you to check the entered code directly within the arithmetic box. If there are no errors a box with the message 'Check successful!' appears, otherwise the corresponding dialog box is extended automatically by a new field containing the error and warning messages detected by the check routines. Double clicking on a particular message marks the related line automatically.

## Inserting an arithmetic box connected to an existing LD network

For the next steps let us assume that you have already inserted a LD network in a LD worksheet consisting of a left and right power rail and one contact as shown in the following figure and you want to insert an arithmetic box connected automatically to this LD network.

Note that the specific code entered in the arithmetic box is only executed, if the object left to the arithmetic box returns the boolean value TRUE otherwise the specific code is ignored. This means for the following example that contact 'C001' must return the boolean value TRUE in order to execute the statements in the arithmetic box.

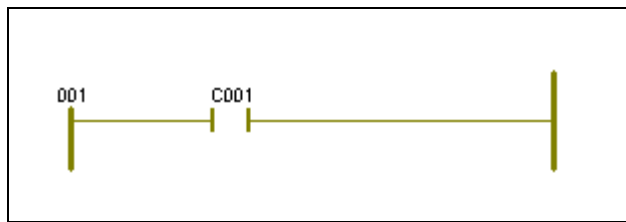


Figure A1-4: Simple LD network



For a detailed description how to insert a LD network and LD objects in an existing LD network please refer to chapter 'Editing in LD' in this manual.

As already mentioned in the section before, an arithmetic box can only be inserted next to the right power rail, i.e. there are no following objects possible on the right side of the arithmetic box. In our example this means that you can only insert the arithmetic box between contact 'C001' and the right power rail.



### Inserting an arithmetic box in an existing LD network with the mouse

- Click on the contact 'C001' to mark it.
- Click on the icon 'Create inline box' in the toolbar.  
The empty dialog 'Inline Code' appears.



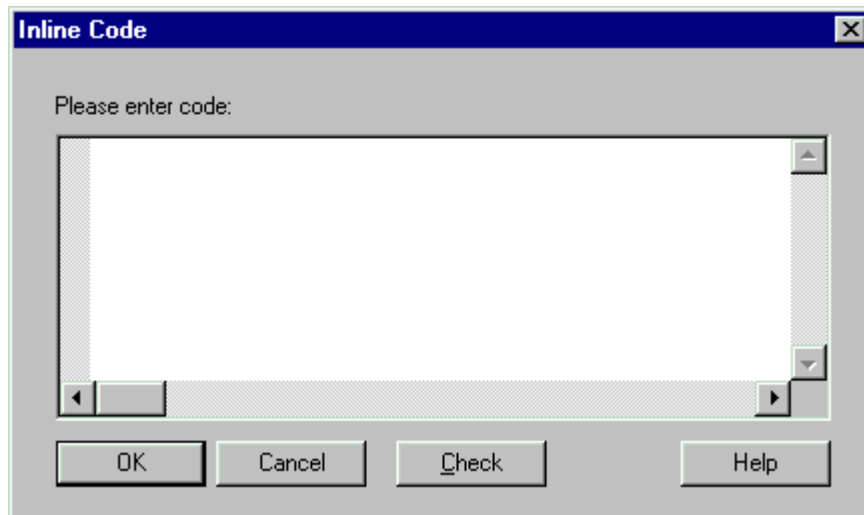


Figure A1-5: Dialog 'Inline Code'



### Using the dialog 'Inline Code'

- In the editing field enter the statements according to the specific code. IEC comments are also allowed using parenthesis and asterisks.



A maximum of 19 code and comment lines are possible in the dialog 'Inline Code'. If this maximum number is exceeded the following code lines are cut.

- When you have finished entering the specific code, press the button 'Check' to check the syntax of the entered code. If there are no syntax errors a box with the message 'Check successful!' appears, otherwise the dialog box is extended automatically by a new field containing the error and warning messages detected by the check routines. Double clicking on a particular message in this field marks the related line automatically in the arithmetic box.
- After editing and checking the specific statements confirm the dialog. The arithmetic box is now inserted behind contact 'C001' in the LD network as shown in the following figure.



The height of the arithmetic box is automatically extended in accordance with the amount of lines. Following objects underneath the edited arithmetic box are moved automatically.

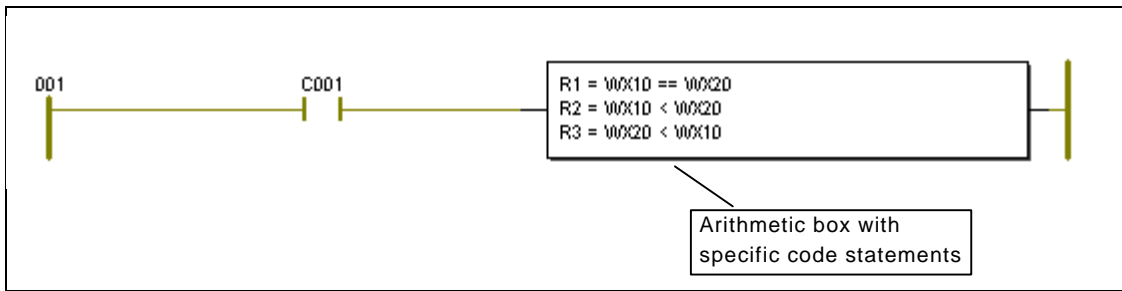


Figure A1-6: LD network with inserted arithmetic box

If you want to edit the contents in the dialog 'Inline Code' just double click on the arithmetic box in the LD network. The dialog is opened and you can edit the statements in the usual way.

## Inserting an arithmetic box as a single object

An arithmetic box can be inserted as a single object at any free position in the LD worksheet. This is done by setting an insertion mark and using the corresponding icon in the toolbar. Having inserted the dialog box, you have to connect it to an existing LD network (for this procedure refer to the following section 'Connecting an arithmetic box to a LD network'). For the next steps let us assume that you want to insert an arithmetic box in a LD worksheet.



### Inserting an arithmetic box as a single object with the mouse

- Locate the position where you want to place the arithmetic box. Click into the worksheet to set an insertion mark.
- Click on the icon 'Create inline box' in the toolbar. The empty dialog 'Inline Code' appears.

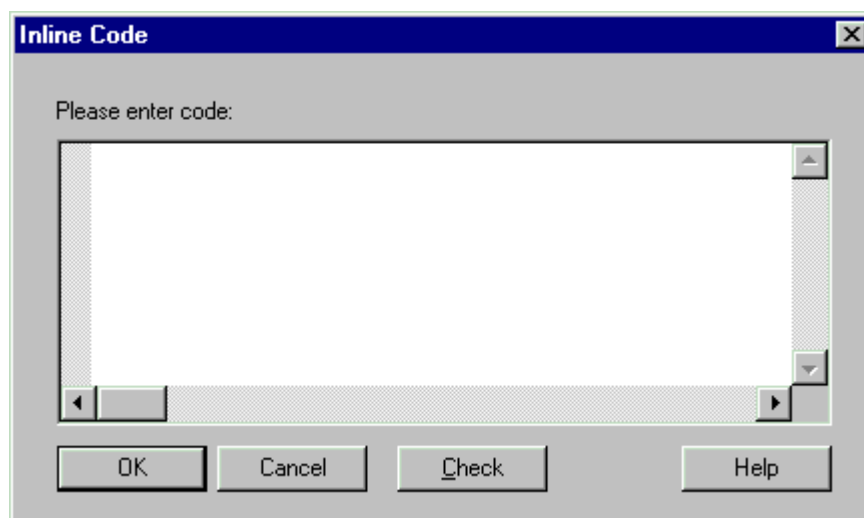


Figure A1-7: Dialog 'Inline Code'



## Using the dialog 'Inline Code'

- In the editing field enter the statements according to the specific code. IEC comments are also allowed using parenthesis and asterisks.



A maximum of 19 code and comment lines are possible in the dialog 'Inline Code'. If this maximum number is exceeded the following code lines are cut.

- When you have finished entering the specific code, press the button 'Check' to check the syntax of the entered code. If there are no syntax errors a box with the message 'Check successful!' appears, otherwise the dialog box is extended automatically by a new field containing the error and warning messages detected by the check routines. Double clicking on a particular message in this field marks the related line automatically in the arithmetic box.
- After editing and checking the specific statements confirm the dialog. The arithmetic box is now inserted in the LD worksheet.



The height of the arithmetic box is automatically extended in accordance with the amount of lines. Following objects underneath the edited arithmetic box are moved automatically.

## Connecting an arithmetic box to a LD network

If you have inserted an arithmetic box as a single object in a LD worksheet as described in the previous section, you have to connect it to the existing LD network in order to create a legal network. This connection is done using the connection mode.



When connecting the arithmetic box to the LD network, note that the specific code entered in the arithmetic box is only executed, if the object left to the arithmetic box returns the boolean value TRUE otherwise the specific code is ignored.

For the next step let us assume that your LD worksheet contains a simple LD network consisting of one left and right power rail and one contact and you want to connect the already inserted arithmetic box to this network as shown in the following figure.

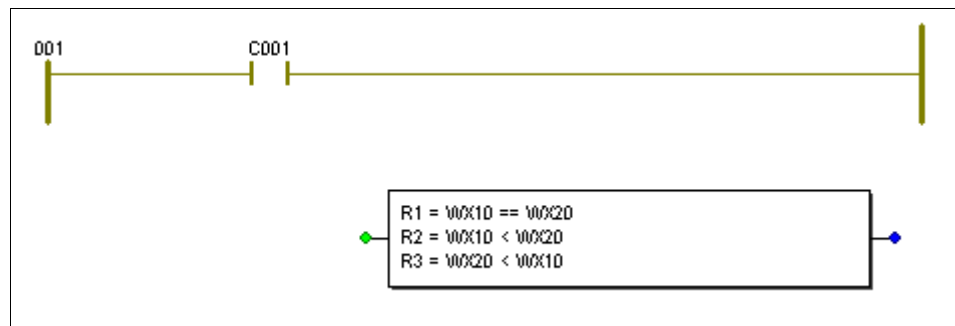


Figure A1-8: Simple LD network and arithmetic box inserted as a single object



When connecting an arithmetic box to a LD network using the connection mode, the left (green) connection point of the box must be connected to the output of a LD object, which represents the condition, whether the inline code is executed or not. This means the previous LD object must return the boolean value TRUE in order to execute the specific code statements.



When using the connection mode, objects can be linked only using a horizontal line. So your line should start at the connection point of the arithmetic box and move away from the box in a horizontal way.



### Connecting an arithmetic box to an existing LD network using the connection mode with the mouse

- Click on the icon 'Connect objects' in the toolbar. A symbol for a connection is added to the cursor.
- Click on the left (green) connection point of the arithmetic box.
- Move the mouse to the output of the last object in the LD network (in our example the output of contact 'C001').
- Click on the output (line) to insert the new connection line.



Your screen should look like the following figure now:

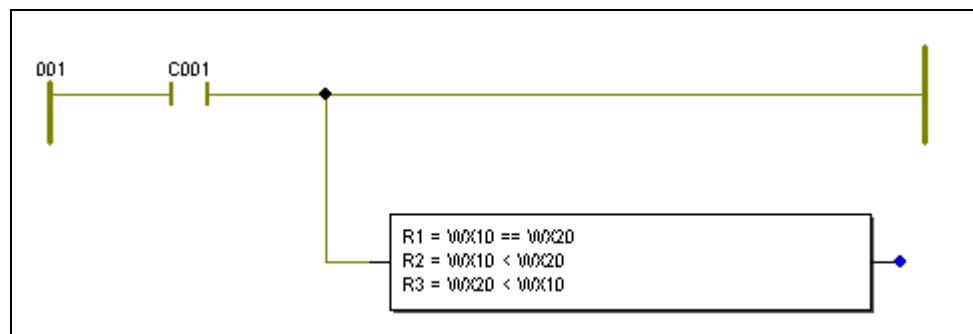


Figure A1-9: LD network with connected arithmetic box

As already mentioned above the right power rail is an optional object. Thus the right connection point of the arithmetic box must not be connected to the right power rail. Nevertheless you can connect the arithmetic box to the right power rail as described in the following procedure.



### Connecting an arithmetic box to the right power rail using the connection mode with the mouse

- Click on the icon 'Connect objects' in the toolbar. A symbol for a connection is added to the cursor.
- Click on the right (blue) connection point of the arithmetic box.



- Move the mouse to the right power rail and press the right mouse button when the mouse pointer is positioned on the power rail.  
The connection is established and the power rail is extended automatically as shown in the following figure.

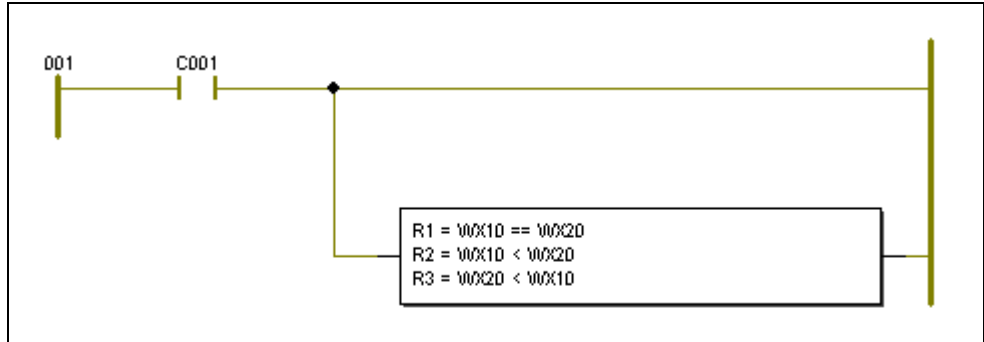


Figure A1-10: LD network and arithmetic box connected to the right power rail

---

# Online monitoring

## Compiling and downloading the project

When you have finished editing the worksheets containing specific code statements and you want to call these worksheets in online mode you first have to compile the contents of the worksheets to generate the specific PLC code. This is done by using the menu items from the submenu 'Build'. There is no difference in compiling worksheets containing specific code statements or not, i.e. in both cases the same functions and commands are provided by the programming system.



For a detailed description about compiling a project using the several menu items in the submenu 'Build' refer to the chapter 'Compiling, downloading and debugging' in this chapter and to the Help system.

If you are compiling a project or parts of it containing specific code statements for the first time, in many cases the compiler detects programming errors. In those cases a message appears in the sheet 'Build' of the message window, announcing the number of detected errors and warnings. This is regardless of whether the worksheets contain specific inline code or not. The corresponding error messages are then displayed in the sheet 'Errors' of the message window.

According to the general error handling, you can open directly the worksheet where the programming error is occurred by double clicking on the error message in the sheet 'Errors'. The corresponding line or the object is marked automatically in the worksheet.



If the specific code statements in an arithmetic box in LD worksheets contain an error, double clicking on the error message in the message window opens directly the arithmetic box and the corresponding line in the box where the error was detected is marked.

Having compiled your project using the menu items in the submenu 'Build' and the compiler detects no errors you have to download it to the simulation or PLC. This is performed using the dialog 'Download', which is called from the control dialog.



For a detailed description of downloading the project to the simulation or PLC and using the control dialog and the dialog 'Download' refer to section 'Downloading the project' in the chapter 'Compiling, downloading and debugging' in this manual.

## Calling worksheets in online mode and displaying online values in IL and LD worksheets

Having compiled and downloaded the project to the target it is possible to call the editors in online mode for displaying the online values. This is performed using the icon 'Debug on/off' in the toolbar or the menu item 'Open instance' in the context menu of a POU in the project tree or in the context menu of an already opened worksheet.



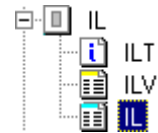
For a detailed description of calling worksheets in online mode and the notes you have to take into account refer to section 'Calling worksheets in online mode' in the chapter 'Compiling, downloading and debugging'.

For the next procedure let us assume that you want to display the online values in an IL worksheet which contains specific code statements.



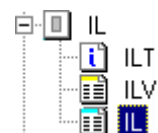
### Calling an IL worksheet containing specific code statements in online mode with the mouse

- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the IL worksheet. For this purpose double click on the POU name.
- Double click on the IL worksheet icon. The worksheet is opened.
- Click on the icon 'Debug on/off'. All opened worksheets are switched to online mode. If online mode is switched on, the icon appears 'pressed'.



### Calling an IL worksheet containing specific code statements in online mode with the keyboard

- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the IL worksheet as follows: Press < > or < > to highlight the POU name and press < > to open the subtree.
- Press < > or < > to mark the IL worksheet icon and then press <↓>. The worksheet is opened.
- Press <F10>. All opened worksheets are switched to online mode. If online mode is switched on, the icon 'Debug on/off' appears 'pressed'.



The following figure shows an IL worksheet in online mode, which contains both, IEC code and specific code.



1	FALSE	LD	X2	
2	FALSE	AND	( X0	
3	FALSE	OR	R0	IEC IL code
4			)	
5	FALSE	ANDN	R1	
6	FALSE	ST	R0	
7	TRUE	LD	TRUE	
8		BEGIN_SPECIFIC		(* introducing specific code *)
9	FALSE 100 200	R1 = WX10 == WX20		Specific code statements
10	TRUE 100 200	R2 = WX10 < WX20		
11	FALSE 200 100	R3 = WX20 < WX10		
12		END_SPECIFIC		(* finishing specific code *)
13				

Figure A1-11: IL worksheet with specific code statements in online mode

All textual worksheets in online mode contain a gray line, which separates the worksheet into two parts. On the left the online values are displayed. On the right the code body is shown.

For specific code statements the online values on the left side are displayed in the order of the code statement variables as shown in the above figure. This means the first online value on the left corresponds to the first variable on the right. In the above example the value 'FALSE' relates to the variable 'R1'.



In general up to 4 values can be displayed at once for a single code statement line.



You can change the width of the columns by positioning the mouse pointer on the gray separation line (when the correct position is reached, the cursor changes to a double line), pressing and holding the left mouse button while moving the cursor to the left or to the right.

In the following figure an example for a LD worksheet in online mode, which contains an arithmetic box with specific code statements is shown:

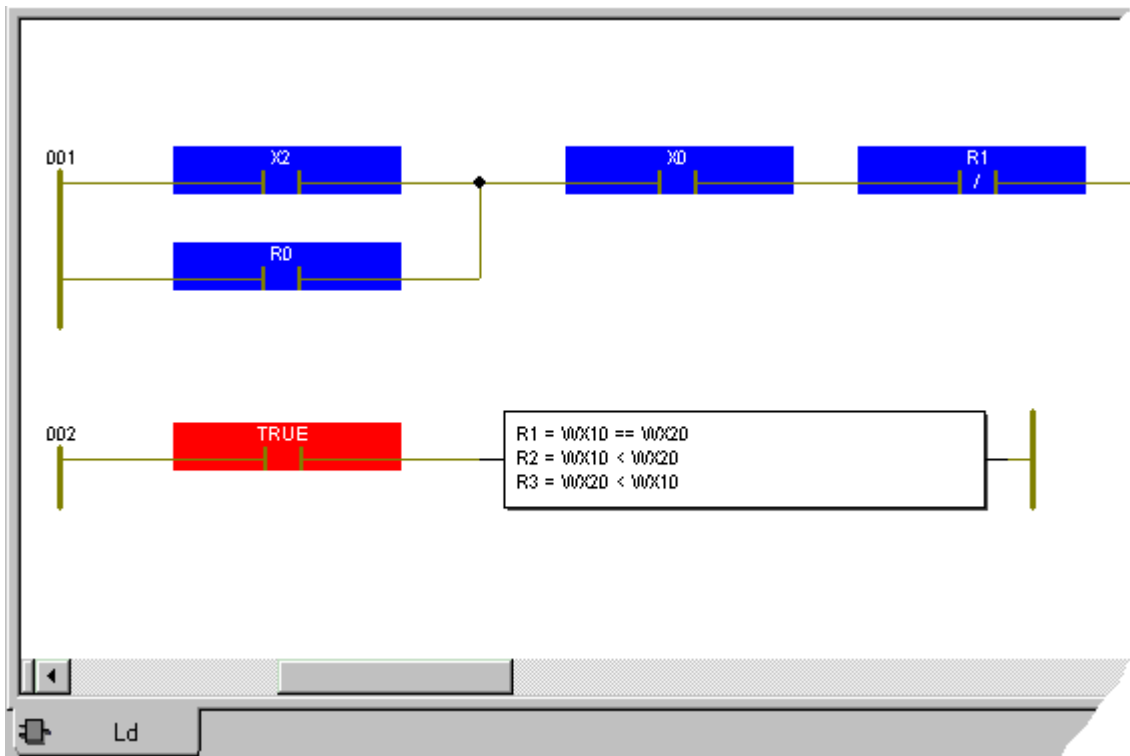


Figure A1-12: Part of a LD worksheet with included arithmetic box in online mode

If you want to display the online values of the specific code statements in an arithmetic box, you have to move and position the mouse cursor onto a particular line of the arithmetic box. Subsequently a tooltip is displayed containing several values according to the content of the line, i.e. the amount of data and the order they are displayed in.

In the following figure, the mouse has been placed on the second line of the arithmetic box. The tooltip values correspond to the PLC values in the order of displayed data, i.e. the first value 'TRUE' relates to the variable 'R2' in the second arithmetic box code line. The values in the tooltip are continuously updated, if the mouse cursor is not moved.

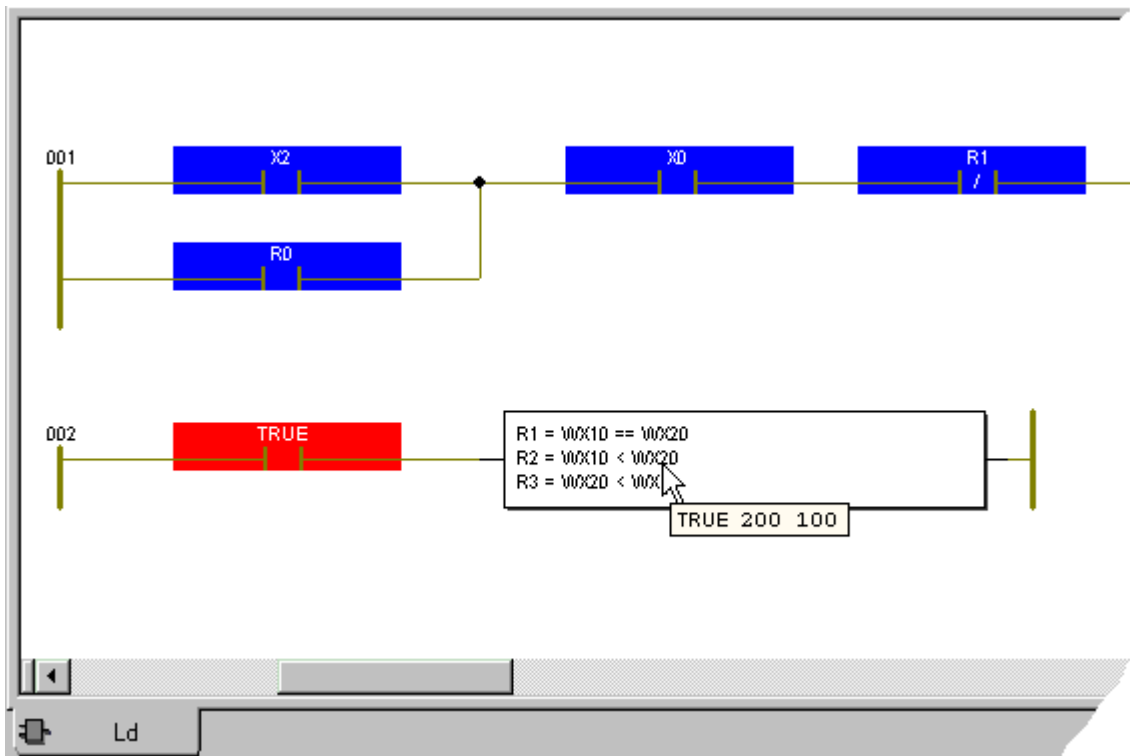


Figure A1-13: Part of a LD worksheet in online mode with online values displayed in the tooltip of the arithmetic box



In general up to 4 values can be displayed at once for a single code statement line.



For further information about displaying the online values and the used colors in textual and graphical worksheets refer to section 'Calling worksheets in online mode' in the chapter 'Compiling, downloading and debugging'.

# PRELIMINARY

## APPENDIX 2

### Compiling, downloading, debugging and uploading

**This chapter provides information about...**

- inserting configurations, resources and tasks in the project tree
- associating programs to tasks
- compiling the project
- patching POUs
- downloading the project
- calling worksheets in online mode
- debug possibilities
- uploading the project

# Compiling, downloading, debugging and uploading

---

## Inserting configurations, resources and tasks

The project tree normally has one configuration and one resource (and one DEFAULT task in case of an IPC\_30 simulator) if you create a new project. It is possible to insert either new configurations, resources or tasks using the project tree editor. The steps to be done are almost the same independently if you are inserting configurations, resources or tasks.



The PLC type of the configurations and the processor type of the resources depend on the connected PLC.



The configuration name and the resource name are fixed in case of a HIDIC PLC.

For the next steps let us assume that the project contains no configuration nor resource and you want to insert a new configuration 'Conf\_HIDIC' and a resource called 'RES\_EH150' for HIDIC EH-150 PLC as processor type.



### Inserting a new configuration with the mouse

- Click with the right mouse button on the icon 'Physical Hardware' in the project tree to open the context menu.
- Select the menu item 'Insert'. The dialog 'Insert' appears.





### Inserting a new configuration with the keyboard

- Press < > or < > to mark the icon 'Physical Hardware' in the project tree.
- Press <INS>. The dialog 'Insert' appears.

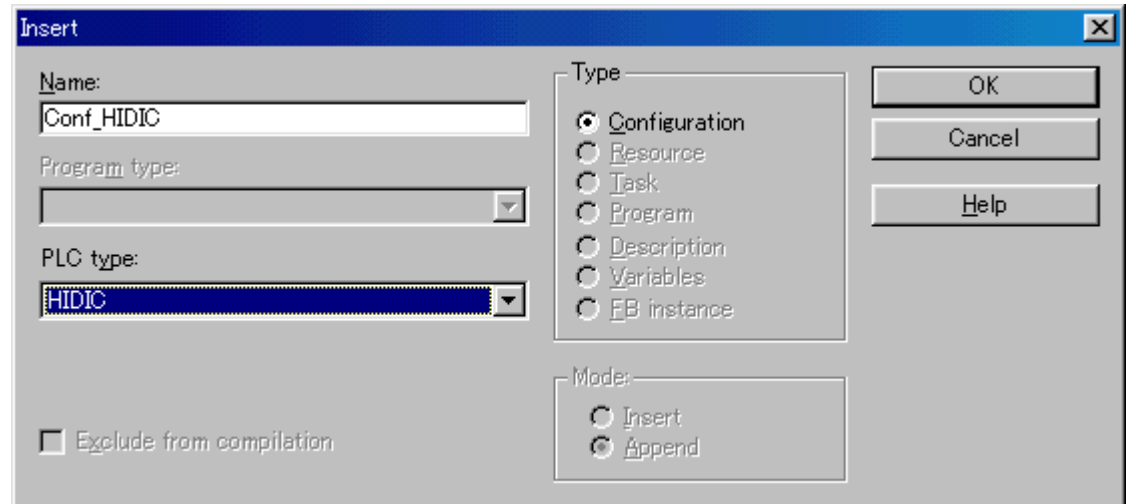


Figure A2-1: Dialog 'Insert'



### Using the dialog 'Insert'

- Enter 'Conf\_HIDIC' as the name of the configuration and select 'HIDIC' as the PLC type.
- Confirm the dialog. The configuration is inserted in the project tree.



### Inserting a new resource with the mouse

- Click with the right mouse button on the icon 'Configuration name' in the project tree to open the context menu.
- Select the menu item 'Insert'. The dialog 'Insert' appears.





### Inserting a new resource with the keyboard

- Press < > or < > to mark the icon '*Configuration name*' in the project tree.
- Press <INS>. The dialog 'Insert' appears.

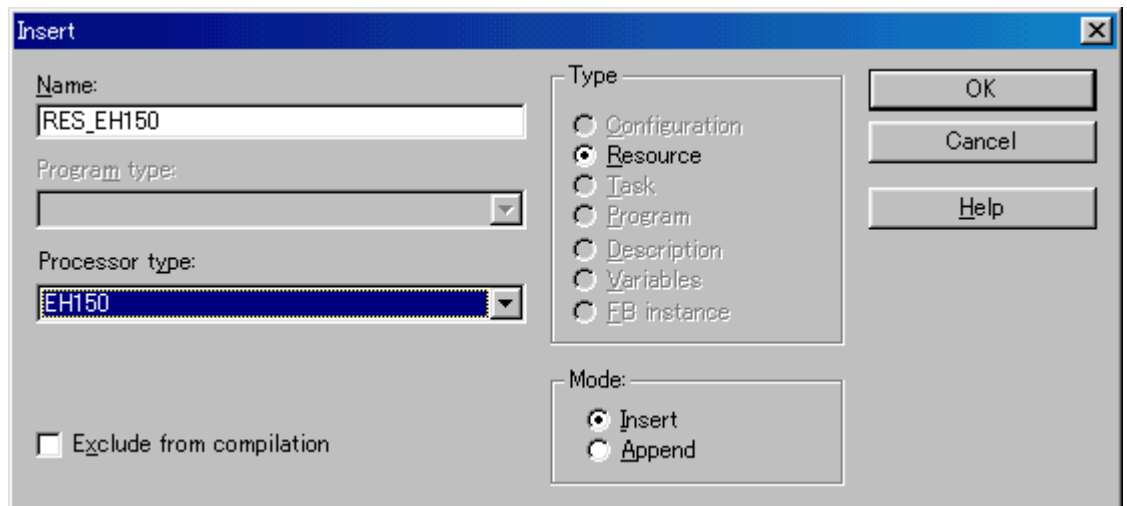


Figure A2-2: Dialog 'Insert'



### Using the dialog 'Insert'

- Enter 'RES\_EH150' as the name of the resource and select 'EH150' as the processor type.
- Confirm the dialog. The resource is inserted in the project tree.
- Select the menu item 'Settings...' in the context menu of the resource. The dialog 'Resource Settings for HIDIC' appears.

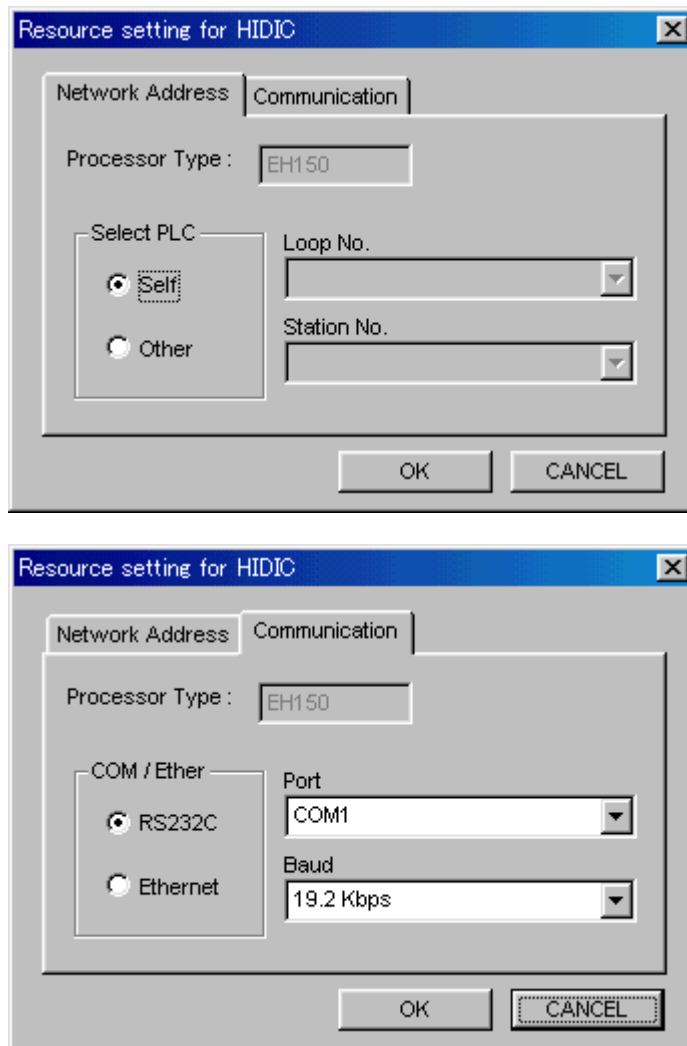


Figure A2-3: Dialog 'Resource Settings for HIDIC'



### Using the dialog 'Resource Settings for HIDIC'

- Activate the radio button 'Self' to send the PLC program to directly connected PLC while downloading. If you want to send the PLC program to other PLC, activate the radio button 'Other' and choose the Loop No. and Station No. from the pull down menus.
- Activate the radio button 'RS232C' you use to connect your PC to the PLC and choose the COM port and the baud rate from the pull down menus. If you want to use Ethernet, activate the radio button 'Ethernet' fill the hostname of the PLC and the port No.
- Confirm the dialog.





### Configuring other parameters for a resource with 'Resource Configuration' dialog

- Double click with left mouse button on the icon 'Resource Configuration' in the project tree.  
Then the 'Resource Configuration' dialog opens.

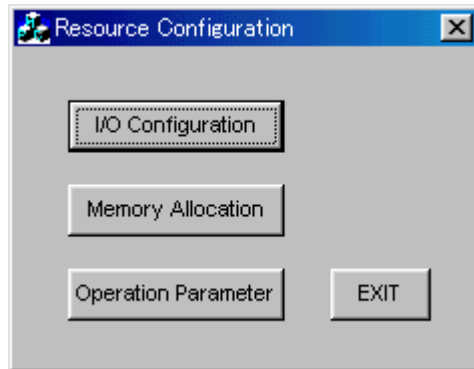


Figure A2-4: Dialog 'Resource Configuration'



### Using the dialog 'I/O Configuration'

- Click with left mouse button on the 'I/O Configuration' button in the 'Resource Configuration' dialog to open the 'I/O Configuration' dialog.
- Choose which PLC's I/Os of you want to configure from the 'Type' menu. You can choose from STANDARD (= the PLC directly connected to the PC which this software runs) and Remote Master PLC 1 to 4.
- Fill the necessary slots with I/O modules by double-clicking each slot and choosing the correct I/O module from the menu on the dialog 'SLOT assignment.'
- Confirm the dialog.

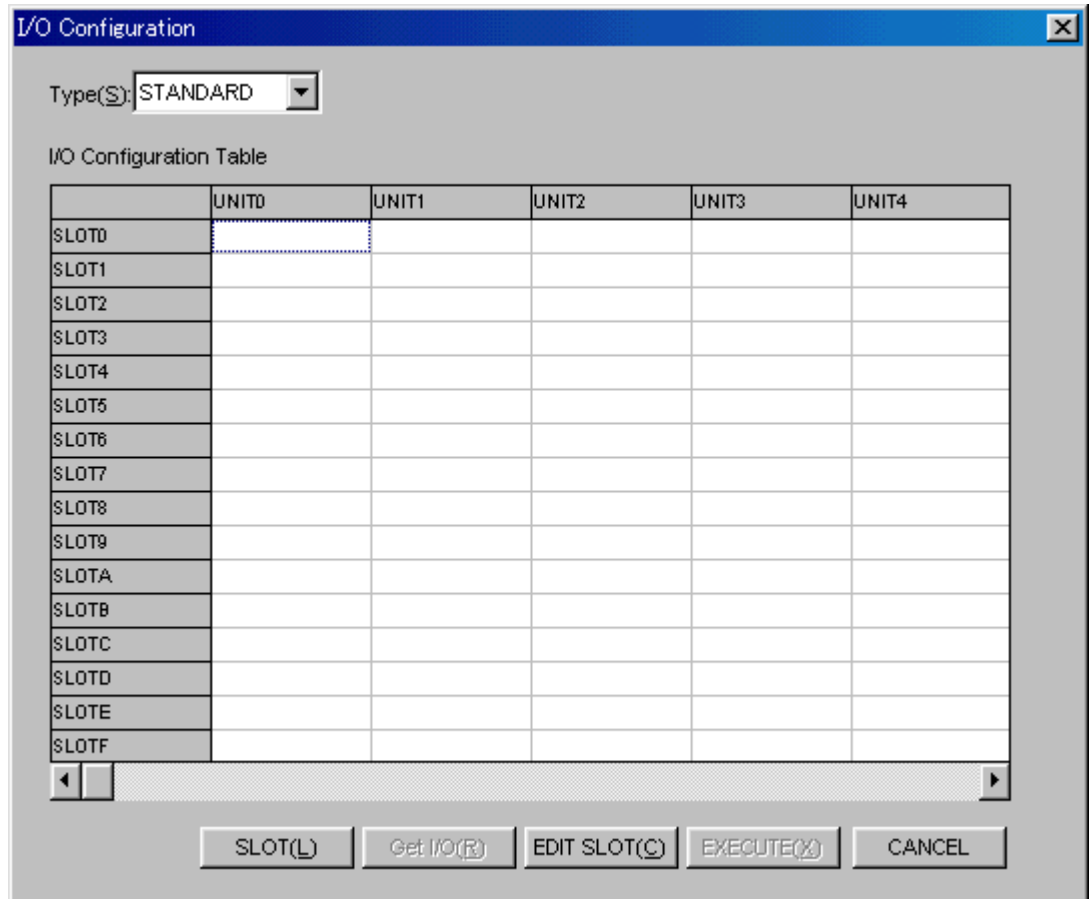


Figure A2-5: Dialog 'I/O Configuration'



### Using the dialog 'CPU settings'

- Click with left mouse button on the 'Memory Allocation' button in the 'Resource Configuration' dialog to open the 'CPU settings' dialog.
- Choose the correct memory size from the 'Memory Allocation' buttons.
- Set the retain areas on each memory areas if needed.
- Confirm the dialog.



For more detailed information on memory allocation, see your PLC's application manual.

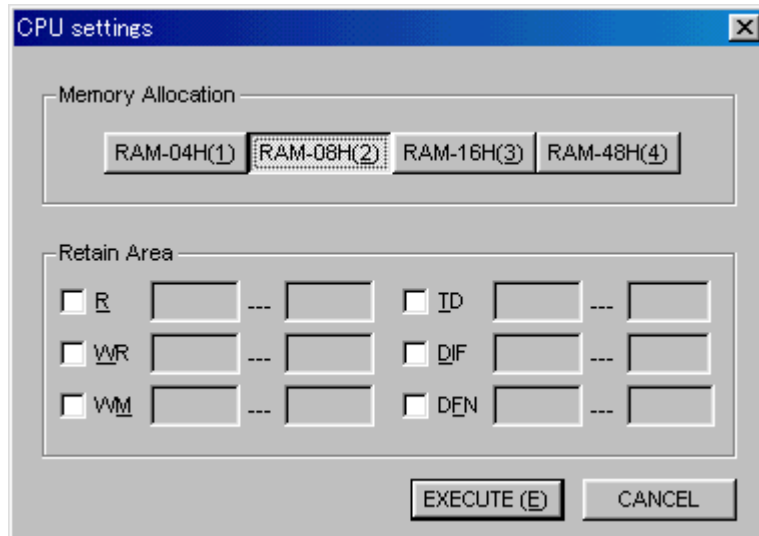


Figure A2-6: Dialog 'CPU settings'



### Using the dialog 'Operation parameters'

- Click with left mouse button on the 'Operation parameters' button in the 'Resource Configuration' dialog to open the 'CPU settings' dialog.
- Set each operation parameter.
- Confirm the dialog.

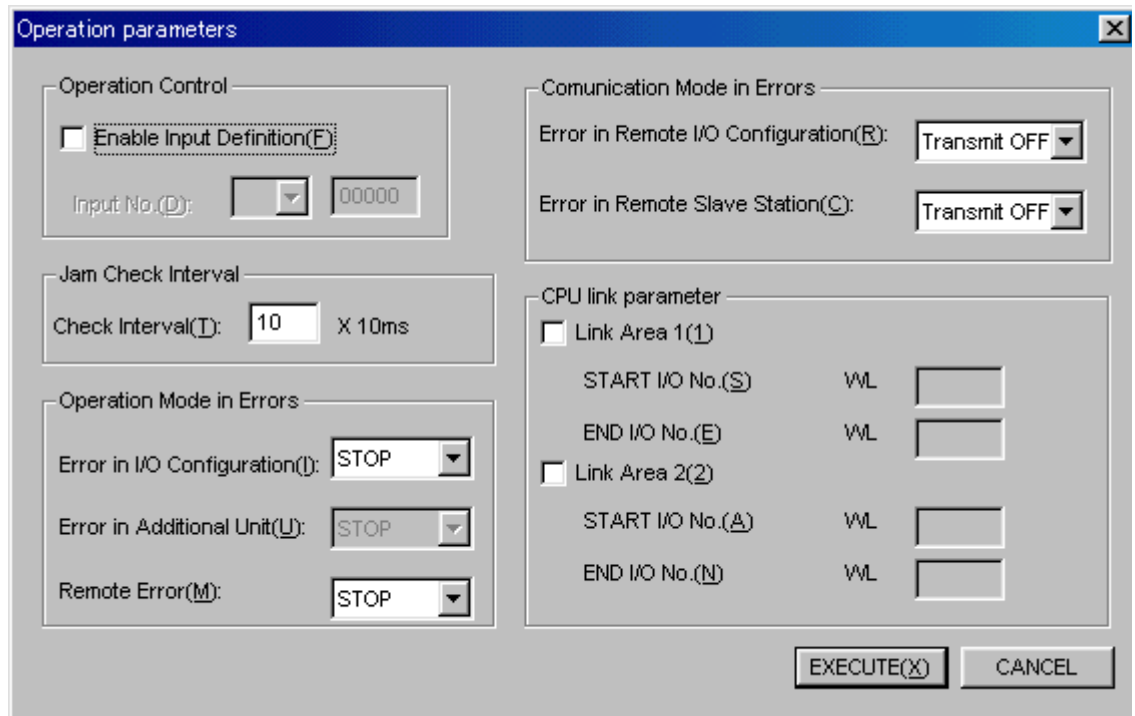


Figure A2-7: Dialog 'Operation parameters'



For more detailed information on these parameters, see your PLC's application manual.



### Inserting a new task with the mouse

- Click with the right mouse button on the icon 'Tasks' in the project tree to open the context menu.
- Select the menu item 'Insert'. The dialog 'Insert' appears.



### Inserting a new task with the keyboard

- Press < > or < > to mark the icon 'Tasks' in the project tree.
- Press <INS>. The dialog 'Insert' appears.

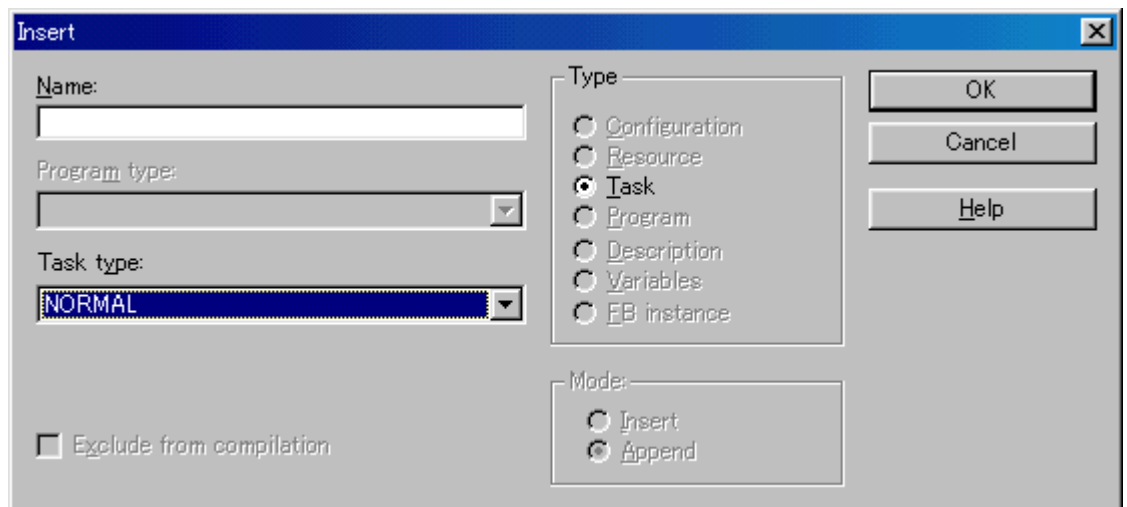


Figure A2-8: Dialog 'Insert'

## PRELIMINARY



### Using the dialog 'Insert'

- Enter the task name and select the task type.
- Confirm the dialog.  
If you choose 'CYCLIC' or 'EVENT' as the task type, you will need to set more parameters according to the task type setting dialog.



The easiest way to insert new configurations, resources or tasks is just marking the corresponding icon in the project tree and then repeating the same steps as described above.

---

## Associating programs to tasks

If you have inserted a new resource you have to insert one or several tasks. For the next steps let us assume that you have already inserted the task 'DEFAULT' in 'RES\_EH150' following the steps described in the previous section.

The next step to be done before compiling is to associate programs to tasks. This means deciding in which task a program is processed.



### Associating a program to a task with the mouse

- Click with the right mouse button on the task icon in the project tree to open the context menu.
- Select the menu item 'Insert'.  
The dialog 'Insert' appears



### Associating a program to a task with the keyboard

- Press < > or < > to mark the task icon in the project tree.
- Press <INS>.  
The dialog 'Insert' appears.



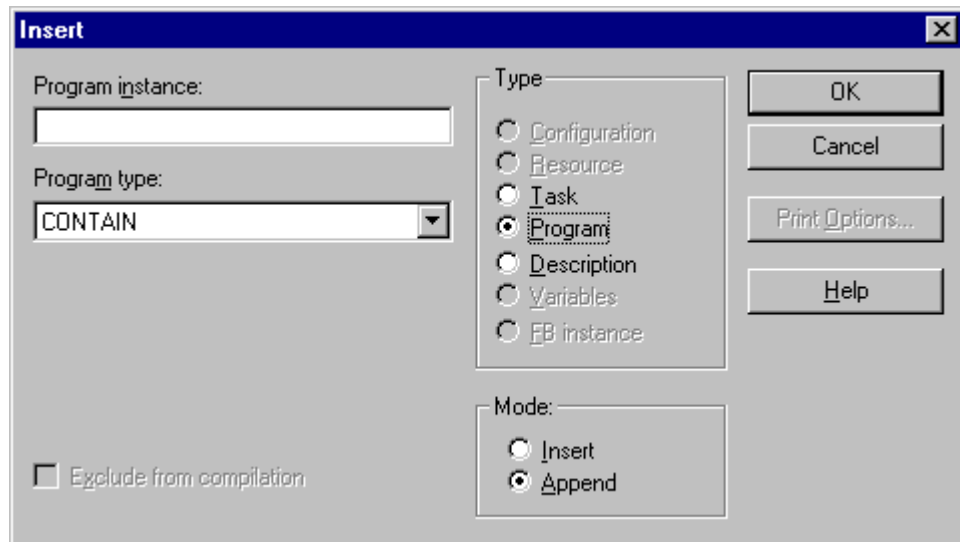


Figure A2-9: Dialog 'Insert'



### Using the dialog 'Insert'

- Activate the radio button 'Program' as shown in figure A2-5.
- Enter the instance name of the program in the field 'Program instance'.
- Choose the name of the program which is contained in the list box 'Program type'.
- Confirm the dialog.  
The icon of the program is inserted in the project tree.

---

## Compiling a project

Compiling means translating and transforming the contents of the worksheets in special code which can be executed by your PLC. The compilation process is performed in several steps. It starts with compiling (i.e. syntax checking) the different worksheets. During the second main step the compiled worksheets are linked together and an intermediate IEC code is generated. The last step generates the PLC code.

You have several possibilities for compiling either the whole project or only parts of it. In the following list the different possibilities are described, which are called using the menu items in the submenu 'Build' or the corresponding icons in the toolbar.

- \* 'Make' - This is the standard mode for compiling the project when you have finished editing. The menu item can be used to compile all worksheets which have been edited. These worksheets are marked with an asterisk in the project tree. After using 'Make' the PLC specific code is generated and the project is ready for downloading to the PLC.

## PRELIMINARY

- \* 'Patch POU' - This menu item is used to compile changes, which have been made e.g. after debugging a project. The changes are automatically downloaded to the PLC, so that you can view them immediately after switching into online mode.
- \* 'Compile worksheet' - This menu item is used to compile a single worksheet after editing it. Choosing this menu item means, that syntax errors within the current code body worksheet and the related variable worksheet are going to be detected by the compiler. All detected errors and warnings are displayed in the message window. By double clicking on an error or warning you can open the related worksheet, where the error was detected.  
When closing a worksheet it is compiled automatically.  
Using only the compiler no code is generated!
- \* 'Rebuild Project' - This menu item is used to compile the whole project for the first time after editing. It should only be used, if 'Make' generates compiler errors or you have unzipped your project without the front end code.  
Using 'Rebuild Project' all worksheets are going to be compiled and linked. Detected errors and warnings are displayed in the message window. After the syntax checking the IEC code as well as the PLC specific code is generated automatically. The project is then ready for downloading to the PLC.

The compilation modes 'Make' and 'Patch POU' are described in the following sections.

---

## Compiling a project using 'Make'

This section describes the steps how to compile the changes you have made in the worksheets. The other possibilities are described in the context-sensitive Help.

Using the menu item/icon 'Make' the changed worksheets are compiled, linked and the changed PLC code is going to be generated. After successful execution the changed project is ready for downloading to the PLC.



Before starting the compilation, ensure that the message window is visible. This window displays the compilation process, any detected errors and warnings and additional information to the process. If the window is not visible, press <CTRL> + <F2>.



### Compiling the changes with the mouse

- Click on the icon 'Make' in the toolbar. The compilation process is displayed in the sheet 'Build' of the message window. Errors and warnings detected during compilation are logged in the corresponding sheets of the message window.





### Compiling the changes with the keyboard

- Press <F9>. The compilation process is displayed in the page 'Build' of the message window. Errors and warnings detected during compilation are logged in the corresponding sheets of the message window.

In most cases while compiling for a first time the different compilers detect programming errors, such as a variable name which has been used twice or typing errors. In those cases a message appears in the sheet 'Build' of the message window, announcing the number of detected errors and warnings.

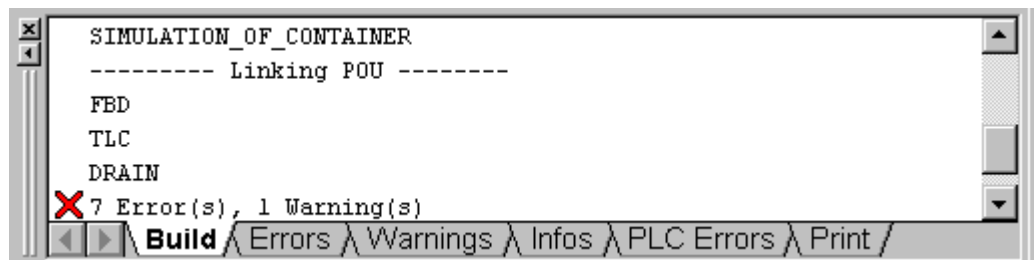


Figure A2-10: Announced errors after making a project

To display the detected errors, click on the tab 'Errors' in the message window. The error list is then shown in the message window.

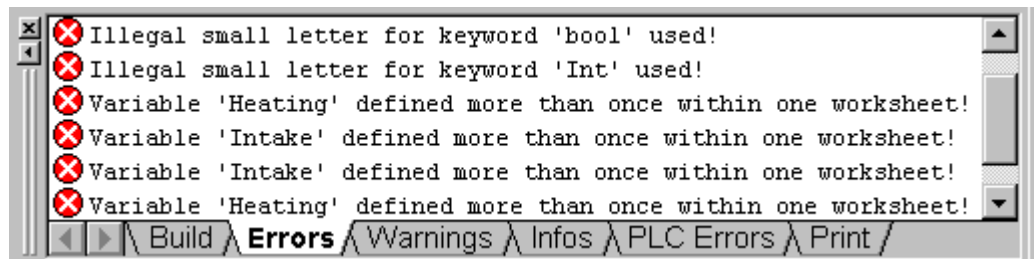


Figure A2-11: Error list, displayed in the message window

In order to display the list of warnings, just click on the tab 'Warnings'.

In most cases double clicking on a displayed error/warning will open directly the worksheet where the programming error/the reason for the warning is occurred. The corresponding line or the object is marked. You can also mark the error and press <F1> to get the corresponding help topic with information about the cause of the error and what steps have to be done now.



Having corrected the first error you can press <CTRL> + <F12> to go directly to the worksheet with the next error.



## Patching POU

To patch a POU means, that changes you have done while debugging a project are compiled, the corresponding code is generated and downloaded automatically to the PLC in one step.

If you have detected a programming error using the online mode and you have switched to the offline mode to remove the programming error you can use 'Patch POU' to compile these changes. The changes are downloaded automatically so that you can view them immediately having switched into online mode.

The following table lists the cases where Patch POU can be used.

Language	You can use Patch POU if ...
all	<ul style="list-style-type: none"> <li>* new local variables have been inserted.</li> <li>* new, empty lines has been inserted.</li> <li>* comments have been changed or added.</li> <li>* inserting global variables if they had already been declared as VAR_EXTERNAL in the POU before.</li> <li>* you have deleted variables.</li> </ul>
IL	<ul style="list-style-type: none"> <li>* you have changed or inserted IL operators.</li> <li>* you have changed the nesting level.</li> <li>* new local variables have been declared.</li> </ul>
ST	<ul style="list-style-type: none"> <li>* you have changed statements or expressions.</li> </ul>
FBD	<ul style="list-style-type: none"> <li>* existing networks have been changed.</li> </ul>
LD	<ul style="list-style-type: none"> <li>* existing networks have been changed.</li> </ul>
SFC	<ul style="list-style-type: none"> <li>* LD networks or variables connected to direct transitions have been changed.</li> <li>* the time interval of time action qualifier has been changed.</li> <li>* the variable name in action blocks has been changed.</li> </ul>

Figure A2-12: Premises for the use of the 'Patch POU' operation

## PRELIMINARY

Patch POU cannot be used in the following cases:

- \* after changing the physical hardware and the formal parameters of functions and function blocks (VAR\_INPUT, VAR\_OUTPUT and VAR\_IN\_OUT).
- \* after inserting functions, function block instances, global variables, POUs and libraries
- \* after deleting POUs or libraries.



The menu item 'Patch POU' is only available if you switch the worksheet in offline mode by clicking on the icon 'Debug on/off' in the toolbar.



### Patching POUs with the mouse

- Make sure that the worksheet is the active window.
- Make sure that the worksheet is in offline mode. In offline mode, the corresponding icon 'Debug on/off' appears not pressed (as shown beneath).
- Edit your worksheet and correct any detected programming errors.
- Click on the submenu 'Build' and select the menu item 'Patch POU'.  
The compilation process is started and its progress is displayed in the message window.  
Detected errors and warnings can be displayed in the corresponding sheets of the message window.



The steps to display error and warning messages and to open the corresponding worksheets are described in the previous section 'Compiling a project using Make'.

## Downloading the project

Having compiled your project using 'Make' or 'Rebuild Project' you have to download it to the simulation or PLC. If you have patched a POU the new generated code is downloaded automatically.



The target of the downloading process is set in the dialog 'Resource settings...'. Choose the menu item 'PLC Settings...' in the context menu of the resource to call the dialog. You can see to which simulation or port the PLC program is sent.



### Downloading the project with the mouse

- Click on the icon 'Show Control Dialog' in the toolbar. The control dialog appears directly if only one resource is available. If the project tree contains several resources, the dialog 'Select resource' appears. In this case choose the desired resource to which the project is to be downloaded and confirm the dialog to display the control dialog.



### Downloading the project with the keyboard

- Press <CTRL> + <F10>. The control dialog appears directly if only one resource is available. If the project tree contains several resources, the dialog 'Select resource' appears. In this case choose the desired resource to which the project is to be downloaded and press <↓> to display the control dialog.

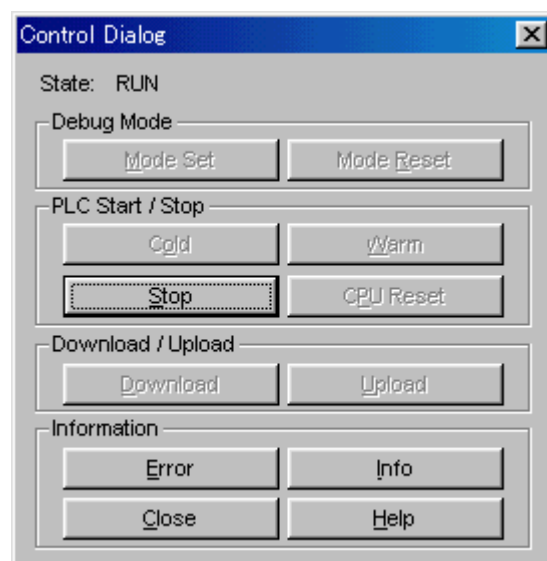


Figure A2-13: Control dialog

## PRELIMINARY



### Using the control dialog


- Press the button 'Download'.  
The full project including POU's and the configuration data is downloaded to the target.  
The PLC passes into the state 'STOP', which is displayed below the title bar in the control dialog.
- Press the button 'Cold' in the control dialog to start the program execution.



Please refer to the context-sensitive Help for detailed information about downloading.

## Calling worksheets in online mode

Having compiled and downloaded the project to the target it is possible to call the editors in online mode for debugging the worksheets. To call the worksheets in online mode you have the following possibilities:

- \* activate the icon 'Debug on/off' in the toolbar or press <F10>. By activating this icon or pressing <F10> all worksheets already opened are switched automatically to online mode. If you open a new worksheet from the project tree or instance tree, this is also called in online mode. The icon 'Debug on/off' appears pressed if online mode is switched on. Refer also to the following note. 
- \* choose the menu item 'Open instance' in the context menu of a POU in the project tree or in the context menu of an already opened worksheet. Refer also to the following note.



If already opened function block code bodies or program code bodies are instantiated several times and you want to debug these worksheets, i.e. calling in online mode by activating the icon 'Debug on/off', a warning message appears indicating that you have to use 'Open instance' to call these worksheets in online mode. In this case choose the menu item 'Open instance' in the context menu of the corresponding worksheets. Choosing this menu item the dialog 'Open Instance' appears where you have to choose the desired instance.

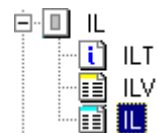
The dialog 'Open Instance' also appears if online mode is switched on and you want to open a function block code body or program code body from the project tree, which is instantiated several times.

For the next steps let us assume that you want debug a textual worksheet in online mode.



### Calling a textual worksheet in online mode with the mouse

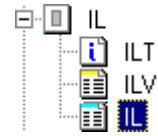
- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the worksheet. For this purpose double click on the POU name.
- Double click on the worksheet icon (e.g. 'IL'). The worksheet is opened.
- Click on the icon 'Debug on/off'. All opened worksheets are switched to online mode. If online mode is switched on, the icon appears 'pressed'.





### Calling a textual worksheet in online mode with the keyboard

- If the desired worksheet is not yet opened, open the corresponding subtree in the project tree, containing the worksheet as follows: Press < > or < > to highlight the POU name and press < > to open the subtree.
- Press < > or < > to mark the worksheet icon (e.g. 'IL') and then press <↵>. The worksheet is opened.
- Press <F10>. All opened worksheets are switched to online mode. If online mode is switched on, the icon 'Debug on/off' appears 'pressed'.



```

1      FALSE LD    %IX0.2    (*direct variables*)
2      FALSE AND   %IX0.3
3      FALSE OR    Action_INIT
4      FALSE ST    IL_VAR
5
6      FALSE LD    Input_IX0_0
7      JMPC   MANUAL
8
9      (* Timer FB TON *)
10     TRUE  LD    Timer_start
11     TRUE  ST    TON_IL.IN
12     1.500 LD    PT_TON_IL
13     1.500 ST    TON_IL.PT
    
```

Figure A2-14: IL worksheet in online mode

In textual worksheets in online mode a gray line separates the worksheet into two parts. On the left the online values are displayed. On the right the code body is shown.



You can change the width of the columns by positioning the mouse pointer on the gray separation line (when the correct position is reached, the cursor changes to a double line), pressing and holding the left mouse button while moving the cursor to the left or to the right.

In the following figure an example for a graphical worksheet in online mode is shown:

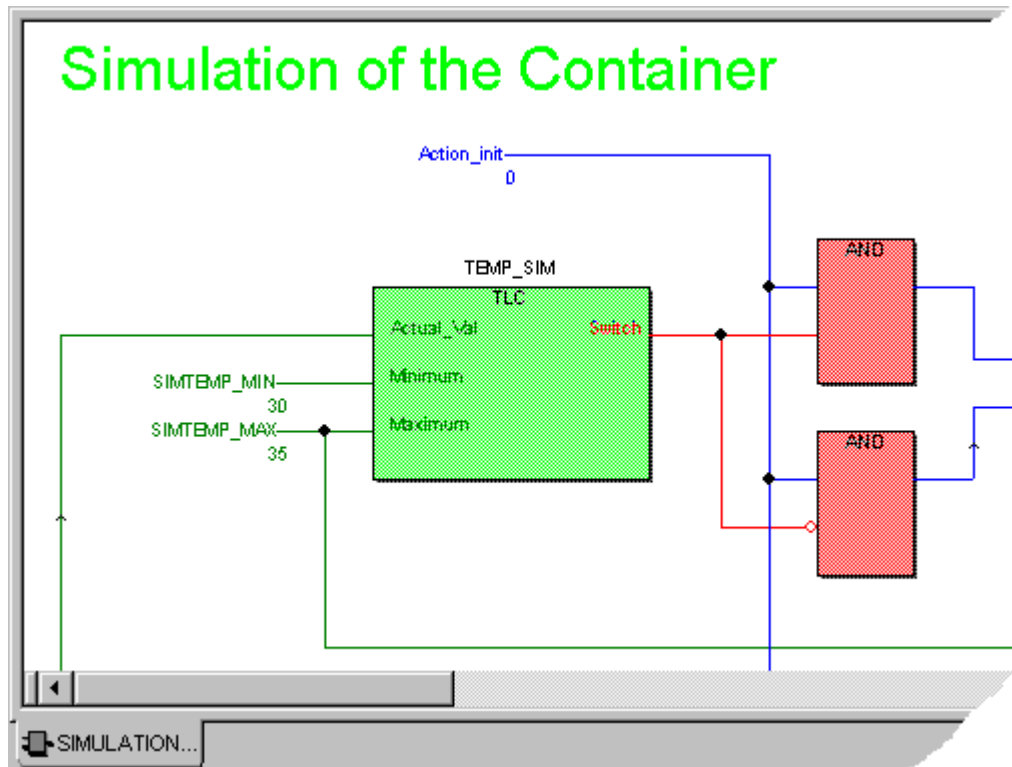


Figure A2-15: Graphical worksheet in online mode

The colors which are used in textual and in graphical worksheets have the following meanings:

Color	Meaning
Red	Boolean true
Blue	Boolean false
Green	integer values
Grey	a set breakpoint
Yellow	a reached breakpoint

Figure A2-16: Meaning of the colors in online mode



In graphical worksheets the way how the online values are displayed can be changed choosing the menu item 'Online Layout' in the submenu 'Online'.

---

## Switching between online and offline mode

If you have detected a programming error in any worksheet in online mode it is possible to switch to the offline mode to correct the programming error immediately. Having corrected the error the changes have to be compiled and sent to the target using 'Patch POU'.



### Switching between online and offline mode with the mouse

- Click on the icon 'Debug on/off' in the toolbar.  
All opened worksheets are switched automatically to offline mode. The icon appears not pressed if offline mode is switched on.
- Correct the programming errors.
- Choose the menu item 'Patch POU' in the submenu 'Build'. The changes you have done are compiled and sent to the target.
- Click on the icon 'Debug on/off' in the toolbar.  
All worksheets are switched automatically to online mode. The icon 'Debug on/off' appears pressed if online mode is switched on.



### Switching between online and offline mode with the keyboard

- Press <F10>.  
All opened worksheets are switched automatically to offline mode. The icon 'Debug on/off' appears not pressed if offline mode is switched on.
- Correct the programming errors.
- Press <ALT> + <F9>.  
The changes you have done are compiled and sent to the target.
- Press <F10>.  
All worksheets are switched automatically to online mode. The icon 'Debug on/off' appears pressed if online mode is switched on.



For information about calling function block code bodies and program code bodies in online mode, which are instantiated several times, refer to the previous section 'Calling worksheets in online mode'.



## Overwriting variables

In online mode variables can be overwritten. Overwriting means assigning to a variable a new value. The new value of the variable is only used for one working cycle. Having finished this cycle the variable is processed normally.



Be very careful overwriting variables while your PLC is running. Overwriting variables mean that the PLC program is executed with the values of the overwritten variable.



### Overwriting variables with the mouse

- Choose the menu item 'Online dialog...' in the context menu of the variable in your worksheet in online mode. The dialog 'Online Debug' appears.



### Overwriting variables with the keyboard

- Press the cursor keys to go to the variable in your worksheet in online mode.
- Press <SHIFT> and keep it pressed. Press < > or < > to mark the variable.
- Press <↵>. The dialog 'Online Debug' appears.

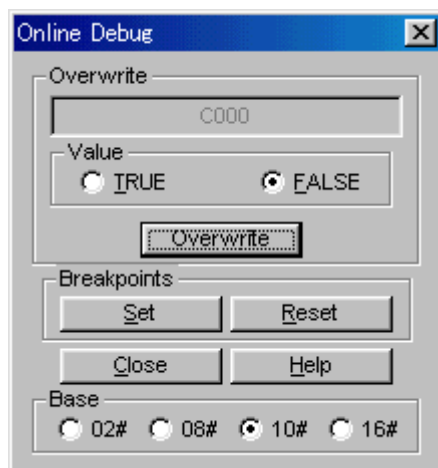


Figure A2-17: Dialog 'Online Debug'



### Using the dialog 'Online Debug'

- Activate the radio button 'TRUE' or 'FALSE'. If you force a non-Boolean variable the dialog includes an input field. In this case enter the desired value.

## PRELIMINARY

- Press the button 'Overwrite'.
- Confirm the dialog.  
The variable is overwritten.

---

## Setting and resetting breakpoints

“Setting and resetting breakpoints” is not supported.

---

## Debugging with set breakpoints

“Debugging with set breakpoints” is not supported.

---

## Using the watch window

The watch window can be used to collect variables from different worksheets to get an appreciation about how these variables work together. If a variable is added once to the watch window, the associated worksheet must not be opened to monitor the current value. Furthermore the watch window is used to debug elements of user defined data types such as arrays and structures.



The procedures how to debug user defined data types using the watch window is described in the following section 'Debugging user defined data types using the watch window'.

In the watch window you can insert and delete variables. There is no limit for the number of variables inserted in the watch window. In addition you can use the watch window to force and overwrite variables and to set and reset breakpoints. This is performed by choosing the menu item 'Online dialog...' in the context menu of a variable.

For the next description let us assume that you want to insert a variable to the watch window. The first step to do is to call the watch window and then to write the variable into it.



### Calling the watch window with the mouse

- Click on the submenu 'View'.  
The submenu is opened.
- Choose the menu item 'Watch Window'.  
The watch window appears.



### Calling the watch window with the keyboard

- Press <ALT> + <V> to open the submenu 'View'.
- Press <A> to open the watch window.

The watch window is displayed as follows:

Variable	Value	Type	Instance
Level	750	INT	C_IPC.R_IPC.T_100ms.PRG_FBD.Level
Temp	34	INT	C_IPC.R_IPC.T_100ms.PRG_FBD.Temp
Intake Q	FALSE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Intake_Q
Heater Q	FALSE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Heater_Q
Drain Q	TRUE	BOOL	C_IPC.R_IPC.T_100ms.PRG_FBD.Drain_Q
Container States		Container_Struct_Array	C_IPC.R_IPC.T_100ms.PRG_FBD.Container_States

At the bottom of the window, there are sheet tabs labeled 'Watch 1', 'Watch 2', 'Watch 3', and 'Watch 4'. 'Watch 1' is currently selected.

Figure A2-18: Watch window with several variables inserted

The watch window contains the following information:

- \* 'Variable': Displays the variable name. User defined data types such as arrays and structures are marked with a '+' (as for example 'Container States' in the above figure). To display the elements of these data types, click on the '+' sign. Normal variables are displayed such as 'Level' in the above example.
- \* 'Value': Displays the current value of the variable.
- \* 'Type': Displays the data type.
- \* 'Instance': Displays the instance path where the variable is used. The path always contains the configuration, resource, task, the associated program name and variable name.

The watch window allows to manage several pages where every page can be used independently. The individual pages are called by clicking on the sheet tabs at the bottom of the watch window.

## PRELIMINARY



### Inserting variables in the watch window with the mouse

- Mark the variable in the worksheet in online mode.
- Click the right mouse button to open the context menu.
- Choose the menu item 'Add to Watch Window'.  
The variable is inserted in the watch window.



If you want to delete variables in the watch window, mark the desired variable in the watch window and choose 'Delete' in the context menu of the variable.

## Uploading the project

You have the possibility to upload the project residing on the memory of the PLC.



The target of the uploading process is set in the dialog 'Resource settings...'. Choose the menu item 'PLC Settings...' in the context menu of the resource to call the dialog.



All POU's and tasks in your project you are now editing will be lost after uploading process. Make the backup copy of your current project before uploading. Uploading to the new blank project may be a good idea.



### Uploading the project with the mouse

- Click on the icon 'Show Control Dialog' in the toolbar. The control dialog appears.



### Uploading the project with the keyboard

- Press <CTRL> + <F10>. The control dialog appears.

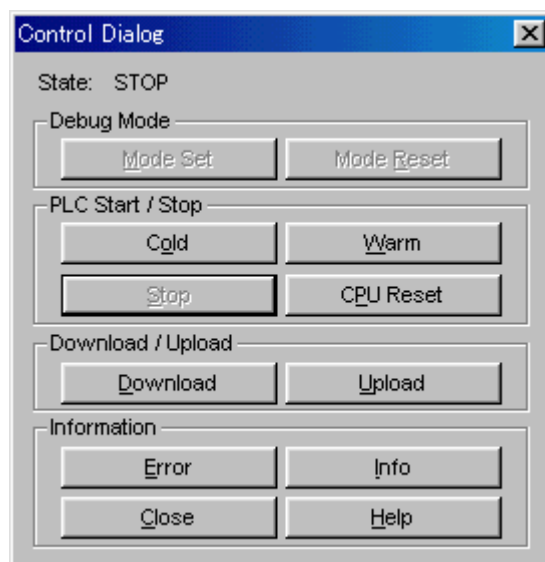


Figure A2-19: Control dialog



### Using the control dialog

- Press the button 'Upload'. The project on the PLC is uploaded and a POU including the program is generated in the project tree.

## PRELIMINARY



The uploaded program is a raw ladder program including some commands for controlling tasks for special purpose only . Be careful to reuse the uploaded program.



For more detailed information on these special commands, see your PLC's application manual.

**PRELIMINARY**

## **APPENDIX 3**

Directly Represented  
Variables Supported  
by HIDIC PLCs

**PRELIMINARY**

## Directly Represented Variables Supported by HIDIC PLCs

For further information, see the HIDIC PLC manuals.

Location		Directly Represented Variables		HIDIC Conventional Address Format
		IEC Standard	HIDIC Specific	
External Inputs	Bit	%I(X)0.[0-5].[0-10].[0-95]	%X0[0-5][0-A][00-95]	X0[0-5][0-A][00-95]
		%I(X)[1-4].[0-5].[0-10].[0-95]	%X[1-4][0-9][0-9][00-95]	X[1-4][0-9][0-9][00-95]
	Byte	%IB0.[0-5].[0-10].[0-7]	(Not Supported)	(Not Supported)
		%IB[1-4].[0-9].[0-9].[0-7]	(Not Supported)	(Not Supported)
Word	%IW0.[0-5].[0-10].[0-7]	%WX0[0-5][0-A][0-7]	WX0[0-5][0-A][0-7]	
	%IW[1-4].[0-9].[0-9].[0-7]	%WX[1-4][0-9][0-9][0-7]	WX[1-4][0-9][0-9][0-7]	
Double word	%ID0.[0-5].[0-10].[0-6]	%DX0[0-5][0-A][0-6]	DX0[0-5][0-A][0-6]	
	%ID[1-4].[0-9].[0-9].[0-6]	%DX[1-4][0-9][0-9][0-6]	DX[1-4][0-9][0-9][0-6]	
External Outputs	Bit	%Q(X)0.[0-5].[0-10].[0-95]	%Y0[0-5][0-A][00-95]	Y0[0-5][0-A][00-95]
		%Q(X)[1-4].[0-5].[0-10].[0-95]	%Y[1-4][0-9][0-9][00-95]	Y[1-4][0-9][0-9][00-95]
	Byte	%QB0.[0-5].[0-10].[0-7]	(Not Supported)	(Not Supported)
		%QB[1-4].[0-9].[0-9].[0-7]	(Not Supported)	(Not Supported)
Word	%QW0.[0-5].[0-10].[0-7]	%WY0[0-5][0-A][0-7]	WY0[0-5][0-A][0-7]	
	%QW[1-4].[0-9].[0-9].[0-7]	%WY[1-4][0-9][0-9][0-7]	WY[1-4][0-9][0-9][0-7]	
Double word	%QD0.[0-5].[0-10].[0-6]	%DY0[0-5][0-A][0-6]	DY0[0-5][0-A][0-6]	
	%QD[1-4].[0-9].[0-9].[0-6]	%DY[1-4][0-9][0-9][0-6]	DY[1-4][0-9][0-9][0-6]	
Internal Memory	Bit	%MX0.[0-1023].[0-15]	%XM[000-3FF][0-F]	M[000-3FF][0-F]
		%MX1.0.[0-2047]	%R[000-7FF]	R[000-7FF]
	Byte	%MB0.[0-1023]	(Not Supported)	(Not Supported)
		%MB1.[0-50175]	(Not Supported)	(Not Supported)
%MB2.[0-511]		(Not Supported)	(Not Supported)	
Word	%MW0.[0-1023]	%WM[000-3FF]	WM[000-3FF]	
	%MW1.[0-50175] <sup>*1</sup>	%WR0[0000-C3FF] <sup>*1</sup>	WR[0000-C3FF] <sup>*1</sup>	
	%MW2.[0-511]	%WRF[000-1FF]	WRF[000-1FF]	
Double word	%MD0.[0-1022]	%DM[000-3FE]	DM[000-3FE]	
	%MD1.[0-50174] <sup>*1</sup>	%DR0[0000-C3FE] <sup>*1</sup>	DR0[0000-C3FE] <sup>*1</sup>	
	%MD2.[0-510]	%DRF[000-1FE]	DRF[000-1FE]	
CPU Link area	Bit	%MX3.[0-1023].[0-15]	%L0[000-3FF][0-F]	L0[000-3FF][0-F]
		%MX4.[0-1023].[0-15]	%L1[000-3FF][0-F]	L1[000-3FF][0-F]
	Byte	%MB3.[0-1023]	(Not Supported)	(Not Supported)
		%MB4.[0-1023]	(Not Supported)	(Not Supported)
Word	%MW3.[0-1023]	%WL0[000-3FF]	WL0[000-3FF]	
	%MW4.[0-1023]	%WL1[000-3FF]	WL1[000-3FF]	
Double word	%MD3.[0-1022]	%DL0[000-3FE]	DL0[000-3FE]	
	%MD4.[0-1022]	%DL1[000-3FE]	DL1[000-3FE]	

<sup>\*1</sup> The area depends on PLC models



# PRELIMINARY

## APPENDIX 4

IEC 61131-3  
COMPLIANCE List  
for Pro-H 1.0 Kernel  
and PLC-Simulation  
based on ProConOS  
3.0 and HIDIC PLC-  
Adaptation

PRELIMINARY

**IEC 61131-3 COMPLIANCE LIST for Pro-H 1.0 Kernel and PLC-Simulation based on ProConOS 3.0 and HDIC PLC-Adaptation**

Legend

	Supported by ProConOS or HDIC PLC
--	Not supported by ProConOS or HDIC PLC

Pro-H Version	Feature	ProConOS (Simulator)	HIDIC PLC
1.0	2.1.1 Table 1 Character set 1 Windows character set 2 Lower case characters 3a Number sign (#) 4a Dollar sign (\$) 5a Vertical bar ( ) 6a left and right brackets "[ ]"		--
1.0	2.1.2 Table 2 Identifier features 1 Upper case and numbers 2 Upper and lower case, numbers, embedded underlines 3 Upper and lower case, numbers, leading or embedded underlines		
1.0	2.1.5 Table 3 Comments		
1.0	2.2.1 Table 4 Numeric literals 1 Integer literals 2 Real literals 3 Real literals with exponents 4 Base 2 literals 5 Base 8 literals 6 Base 16 literals 7 Boolean zero and one 8 Boolean FALSE and TRUE	--	--
1.0	2.2.2 Table 5 Character string literal features Feature 1		--
1.0	2.2.2 Table 6 Two-character combinations in character strings 2 Dollar sign 3 Single quote 4 Line feed 5 Newline 6 Formfeed (page) 7 Carriage return 8 Tab	--	--
1.0	2.2.3.1 Table 7 Duration literal features 1a Short prefix without underlines 1b Long prefix without underlines 2a Short prefix with underlines 2b Long prefix with underlines		--
1.0	2.2.3.2 Table 8 Date and time of day literals 1 DATE# 2 D# 3 TIME_OF_DAY# 4 TOD# 5 DATE_AND_TIME# 6 DT#	--	--
1.0	2.2.3.2 Table 9 Examples of date and time of day literals	--	--

**PRELIMINARY**

1.0	2.3.1 Table 10 Elementary data types 1      BOOL 2      SINT 3      INT 4      DINT 6      USINT 7      UINT 8      UDINT 10     REAL 12     TIME 13     DATE 14     TIME_OF_DAY / TOD 15     DATE_AND_TIME / DT 16     STRING 17     BYTE 18     WORD 19     DWORD		-- -- -- -- -- --
1.0	2.3.3.2 Table 12 Data type declaration feature 1      Direct derivation from elementary types 2      Enumerated data types 3      Subrange data types 4      Array data types 5      Structured data types	-- -- --	-- -- -- --
1.0	2.3.3.2 Table 13 Default initial values - target dependent		
1.0	2.4.1.1 Table 15 Directly represented variables 1      Input location 2      Output location 3      Memory location 4      Single bit size (Prefix X) 5      Single bit size (Prefix None) 6      Byte (8 bits) size 7      Word Size 8      Double word (32 bits) size	--	
1.0	2.4.3 Table 16 Variable declaration keywords VAR VAR_INPUT VAR_OUTPUT VAR_IN_OUT VAR_EXTERNAL VAR_GLOBAL AT RETAIN		

**PRELIMINARY**

1.0	2.4.3.1 Table 17	Variable type assignment features (-> Table 39)		
	1	Declaration of directly represented, non-retentive variables		
	2	Declaration of directly represented, retentive variables		
	3	Declaration of locations of symbolic variables		
	4	Array location assignment (by own data type declaration) <sup>1</sup>		--
	5	Automatic memory allocation of symbolic variables		
	7	Retentive array declaration (arrays as derived data Type)		--
	8	Declaration of structured variables		--
1.0	2.4.3.2 Table 18	Variable initial value assignment features (-> Table 39)		
	1	Initialization of directly represented, non retentive variables		
	2	Initialization of directly represented retentive variables		
	3	Location and initial value assignment to symbolic variables		
	4	Array location assignment and initialization (by own data type declaration) <sup>1</sup>	--	--
	5	initialization of symbolic variables		
	6	Array initialization (by own data type declaration)	--	--
	7	Retentive array declaration and initialization (by own data type declaration)		--
1.0	2.5.1.1 Table 19	Graphical negation of boolean signals		
	1	Negated input		
	2	Negated output		
1.0	2.5.1.2 Table 20	Use of EN input and ENO output		
	1	Required for LD (corresponding to changed IEC 1131-3)	--	
	2	Optional for FBD	--	--
	3	FBD without "EN" and "ENO"		
1.0	2.5.1.4 Table 21	Typed and overloaded functions		
	1	Overloaded functions		
	2	Typed functions		
1.0	2.5.1.5.1 Table 22	Type conversion function features		
	1	*_TO_*		--
	2	TRUNC		--
	3	BCD_TO_*		2
	4	*_TO_BCD		2

<sup>1</sup> Array location assignment not possible in variable worksheet. First the array need to be defined in data type worksheets. The assignment of direct addresses can be done later in the variable worksheet.

<sup>2</sup> INT and DINT are available for \*.

**PRELIMINARY**

1.0	2.5.1.5.2 Table 23 Standard functions of one numeric variable 1 ABS 2 SQRT 3 LN 4 LOG 5 EXP 6 SIN 7 COS 8 TAN 9 ASIN 10 ACOS 11 ATAN		-- -- -- -- -- -- -- -- -- -- --
1.0	2.5.1.5.2 Table 24 Standard arithmetic functions 12 ADD 13 MUL 14 SUB 15 DIV 16 MOD 17 EXPT 18 MOVE		--
1.0	2.5.1.5.3 Table 25 Standard bit-shift functions (FBD overloaded, IL typed) 1 SHL 2 SHR 3 ROR 4 ROL		-- --
1.0	2.5.1.5.4 Table 26 Standard bitwise boolean functions 5 AND 6 OR 7 XOR 8 NOT		
1.0	2.5.1.5.4 Table 27 Standard selection functions (typed, non extensible) 1 SEL 2a MAX 2b MIN 3 LIMIT 4 MUX		-- -- -- -- --
1.0	2.5.1.5.4 Table 28 Standard comparison functions (non extensible) 5 GT 6 GE 7 EQ 8 LE 9 LT 10 NE	--	--

**PRELIMINARY**

1.0	2.5.1.5.5 Table 29 Standard character string functions 1 LEN 2 LEFT 3 RIGHT 4 MID 5 CONCAT 6 INSERT 7 DELETE 8 REPLACE 9 FIND		-- -- -- -- -- -- -- -- --
1.0	2.5.1.5.7 Table 30 Functions of time data type 1 ADD (TIME, ...) 2 ADD (TIME_OF_DAY, ...) 3 ADD (DATE_AND_TIME, ...) 4 SUB (TIME, ...) 5 SUB (DATE, ...) 6 SUB (TIME_OF_DAY, ...) 8 SUB (TIME_OF_DAY, ...) 9 SUB (DATE_AND_TIME, ...) 10 MUL (TIME, ...) 11 DIV (TIME, ...) 12 CONCAT 13 DATE_AND_TIME_TO_TIME_OF_DAY 14 DATE_AND_TIME_TO_DATE	-- -- -- -- -- -- -- -- -- -- -- -- --	-- -- -- -- -- -- -- -- -- -- -- -- --
1.0	2.5.1.5.7 Table 31 Functions of enumerated data types (target dependant) 1 SEL 2 MUX 3 EQ 4 NE	-- -- -- --	-- -- -- --
1.0	2.5.2.2 Table 33 Function block declaration features (-> Table 39) 1 RETAIN qualifier on internal variables 2 RETAIN qualifier on output variables 4a Input/output declaration (textual)		
1.0	2.5.2.3.1 Table 34 Standard bistable function blocks 1 SR 2 RS		
1.0	2.5.2.3.2 Table 35 Standard edge detection function blocks 1 R_TRIG 2 F_TRIG		
1.0	2.5.2.3.3 Table 36 Standard counter function blocks 1 CTU 2 CTD 3 CTUD		-- <sup>3</sup> -- <sup>3</sup> -- <sup>3</sup>
1.0	2.5.2.3.4 Table 37 Standard timer function blocks 1 TP 2a TON 3a TOF		-- <sup>3</sup> -- <sup>3</sup> -- <sup>3</sup>

<sup>3</sup> Similar functions are available but not completely same as IEC standard functions.

**PRELIMINARY**

1.0	2.5.3 Table 39 Program declaration features 1 RETAIN qualifier on internal variables 2 RETAIN qualifier on output variables 11 Declaration of directly represented, non-Retentive variables 12 Declaration of directly represented retentive variables 13 Declaration of locations of symbolic variables 14 Array location assignment (by own data type declaration) <sup>5</sup> 15 Initialization of directly represented, non retentive variables 16 Initialization of directly represented retentive variables 17 Location and initial value assignment to symbolic variables 18 Array location assignment and initialization (by own data type declaration) <sup>2</sup> 19 Use of directly represented variables	--	--
1.0	2.6.2 Table 40 Step features 1 Step, initial step (graphical form) 3a, 3b Step flag		
1.0	2.6.3 Table 41 Transitions and transition conditions 2 Transition condition using LD language 3 Transition condition using FBD language 4,4a,4b Use of connector 7 Use of transition name 7a Transition condition using LD 7b Transition condition using FBD 7c Transition condition using IL 7d Transition condition using ST		
1.0	2.6.4.1 Table 42 Declaration of actions 1 Any boolean variable can be an action 2l Graphical declaration in LD language 2f Graphical declaration in FBD language 3s Textual declaration in ST language 3i Textual declaration in IL language		
1.0	2.6.4.2 Table 43 Step/action association 1 Action block 2 Concatenated action blocks		
1.0	2.6.4.3 Table 44 Action block 1 Qualifier 2 Action name		
1.0	2.6.4.4 Table 45 Action qualifiers 2 Non-stored 3 overriding Reset 4 Set (Stored) 5 time limited 6 time delayed 7 Pulse 8 Stored and time delayed 9 Delayed and stored 10 Stored and time limited		

<sup>5</sup> Array location assignment not possible in variable worksheet. First the array need to be defined in data type worksheets. The assignment of direct addresses can be done later in the variable worksheet.

**PRELIMINARY**

1.0	2.6.5 Table 46 Sequence evolution 1 Single sequence 2a Divergence of sequence selection 3 Convergence of sequence selection 4 Simultaneous sequences 5a Sequence skip 6a Sequence loop 7 Directional arrows		
1.0	2.7.1 Table 49 Configuration and resource declaration features 1 CONFIGURATION 3 RESOURCE 4 VAR_GLOBAL within RESOURCE 5a Periodic TASK 5b Non-periodic TASK 6a PROGRAM with PROGRAM-to-TASK association 6c PROGRAM with no TASK association 7 Declaration of directly represented variables		
1.0	2.7.2 Table 50 Task features 1a Textual declaration of periodic TASK (by project tree) 1b Textual declaration of non-periodic TASK (by project tree) 3a Textual association with PROGRAMs (by project tree) 4b Graphical association with FUNCTION BLOCKS <sup>6</sup> 5a Preemptive scheduling (Target dependent) 5b Non-preemptive scheduling (Target dependent)	--	--
1.0	3.2.2 Table 52 Instruction list (IL) operators 1 LD 2 ST 3 S, R 4 AND 6 OR 7 XOR 8 ADD 9 SUB 10 MUL 11 DIV 12 GT 13 GE 14 EQ 15 NE 16 LE 17 LT 18 JMP 19 CAL 20 RET 21 )		7 7 7 7 7 7
1.0	3.2.3 Table 53 Function block invocation features for IL language 2 CAL with load/store of inputs		

<sup>6</sup> We do not support of chapter 2.7.2 of the IEC 1131-3 part 7a and 7b, but the compiler shows a warning if this feature is used in the project.

<sup>7</sup> Data type BOOL is not supported for inputs.



**PRELIMINARY**

1.0	<p>3.3.1 Table 55 Operators on the ST language</p> <ol style="list-style-type: none"> <li>1 Parenthesization</li> <li>2 Function evaluation</li> <li>3 Exponentiation</li> <li>4 Negation</li> <li>5 Complement</li> <li>6 Multiply</li> <li>7 Divide</li> <li>8 Modulo</li> <li>9 Add</li> <li>10 Subtract</li> <li>11 Comparison</li> <li>12 Equality</li> <li>13 Inequality</li> <li>14 Boolean AND (&amp;)</li> <li>15 Boolean AND</li> <li>16 Boolean exclusive OR</li> <li>17 Boolean OR</li> </ol>		--
1.0	<p>3.3.2 Table 56 ST language statements</p> <ol style="list-style-type: none"> <li>1 Assignment</li> <li>2 FB invocation and FB output usage</li> <li>3 RETURN</li> <li>4 IF</li> <li>5 CASE</li> <li>6 FOR (only upwards)</li> <li>7 WHILE</li> <li>8 REPEAT</li> <li>9 EXIT</li> <li>10 Empty statement</li> </ol>		
1.0	<p>4.1.3 Table 57 Representation of line and block</p> <ol style="list-style-type: none"> <li>2 Horizontal lines</li> <li>4 Vertical lines</li> <li>6 Horizontal/vertical connection (with connection dot)</li> <li>8 Line crossings without connection (without gap)</li> <li>10 Connected and non connected corners</li> <li>12 Blocks with connecting lines</li> <li>14 Graphic connectors</li> </ol>		
1.0	<p>4.1.3 Figure 23 Feedback path</p> <ol style="list-style-type: none"> <li>a Explicit loop</li> <li>b Implicit loop</li> <li>c LD language equivalent</li> </ol>		--
1.0	<p>4.1.4 Table 58 Graphic execution control elements</p> <ol style="list-style-type: none"> <li>1 Unconditional jump FBD</li> <li>2 Unconditional jump LD</li> <li>3 Conditional jump FBD</li> <li>4 Conditional jump LD</li> <li>5 Conditional return LD</li> <li>6 Conditional return FBD</li> <li>8 Unconditional Return LD</li> </ol>		
1.0	<p>4.2.1 Table 59 Power rails</p> <ol style="list-style-type: none"> <li>1 Left power rail</li> <li>2 Right power rail</li> </ol>		
1.0	<p>4.2.2 Table 60 Link elements</p> <ol style="list-style-type: none"> <li>1 Horizontal link</li> <li>2 Vertical link</li> </ol>		

**PRELIMINARY**

1.0	4.2.3 Table 61 Contacts 1 Normally open contact 3 Normally closed contact 5, 7 Positive transition-sensing contact (target dependent)	--	--
1.0	4.2.4 Table 62 Coils 1 Coil 2 Negated coil 3 SET (latch) coil 4 RESET (unlatch) coil		

Pro-H Version	Feature	ProConOS (Simulator)	HIDIC PLC
1.0	<b>IEC 1131 Extensions</b> 2 TR2: constant type specifier e.g. INT#12, BOOL#1		

# List of figures

Figure 2-1:	An example of configuration elements.....	2-3
Figure 2-2:	Diagram of a default task with two programs.....	2-5
Figure 2-3:	Subtree 'Logical POUs' .....	2-6
Figure 2-4:	Project tree with the instances within the resource 'R_IPC' .....	2-7
Figure 2-5:	Worksheets of a function block in FBD.....	2-7
Figure 2-6:	Icons of a SFC POU .....	2-8
Figure 2-7:	Subtree 'Data Types' .....	2-9
Figure 2-8:	The project with its subtrees .....	2-10
Figure 2-9:	Subtree 'Library' .....	2-11
Figure 3-1:	Program user interface with sample project 'example' .....	3-4
Figure 3-2:	Sample tooltip (icon 'Save') with corresponding description displayed in the status bar.....	3-7
Figure 3-3:	Example of a detached toolbar window .....	3-8
Figure 3-4:	The 'Shortcut Manager' for adding/modifying keyboard shortcuts .....	3-9
Figure 3-7:	Default shortcuts in alphabetical order.....	3-12
Figure 3-8:	Worksheet tabs in the workspace.....	3-13
Figure 3-9:	Message window with its different page tabs.....	3-14
Figure 3-10:	Cross reference window.....	3-16
Figure 3-13:	Project tree window with sheet tabs.....	3-21
Figure 3-14:	Edit Wizard for the different editors.....	3-24
Figure 3-15:	Edit Wizard for variables and data type worksheets .....	3-25
Figure 3-16:	Navigating in a graphical worksheet using the overview window.....	3-27
Figure 3-17:	Dialog 'Pro-H' .....	3-29
Figure 4-1:	Dialog 'New Project' containing the available project templates .....	4-2
Figure 4-2:	Project 'Untitled' with program 'Untitled' and its worksheets .....	4-2
Figure 4-3:	Dialog 'Properties' while changing the properties of existing POUs.....	4-4
Figure 4-4:	Dialog 'Insert' while inserting a new POU .....	4-5
Figure 4-5:	Project with announced library 'Contain' .....	4-8
Figure 5-1:	Numeric literals .....	5-1
Figure 5-2:	Character string literals .....	5-2
Figure 5-3:	Duration literals .....	5-2
Figure 5-4:	Elementary data types.....	5-3

Figure 5-5:	Generic data types .....	5-4
Figure 5-6:	Type declaration of an array data type.....	5-5
Figure 5-7:	Programming example of an array data type.....	5-6
Figure 5-8:	Type declaration of an array of array .....	5-7
Figure 5-9:	Accessing elements of an array of array.....	5-7
Figure 5-10:	Initializing elements of an array.....	5-7
Figure 5-11:	Declaration of a structured data type .....	5-8
Figure 5-12:	Declaration of an array of structure.....	5-8
Figure 5-13:	Declaration of a structure of array.....	5-9
Figure 5-14:	Assigning values to components of a structure .....	5-9
Figure 5-15:	Declaration of a string data type .....	5-10
Figure 5-16:	Edit Wizard in a data type worksheet.....	5-11
Figure 5-17:	Data type worksheet 'Type' with pre-edited keywords, inserted using the Edit Wizard .....	5-11
Figure 5-18:	Declaration of a symbolic, a located variable and a directly represented variable.....	5-13
Figure 5-19:	Example for the declaration of a retentive variable.....	5-14
Figure 5-20:	Declaration and initialization of a symbolic, a located variable and a directly represented variable .....	5-15
Figure 5-21:	Table of keywords for variable declaration blocks .....	5-17
Figure 5-22:	Edit Wizard in a variable worksheet.....	5-18
Figure 5-23:	Variable worksheet with pre-edited keyword 'VAR', inserted using the Edit Wizard....	5-19
Figure 5-24:	Instantiation of a function block.....	5-20
Figure 5-25:	Example of an instance tree .....	5-21
Figure 6-1:	Structure of an assignment statement in ST.....	6-3
Figure 6-2:	Table of operators in ST .....	6-3
Figure 6-3:	Table of statements in ST .....	6-5
Figure 6-1:	Pre-edited CASE statement, inserted using the Edit Wizard.....	6-6
Figure 6-2:	Dialog 'Variable' .....	6-8
Figure 6-3:	Dialog 'Automatic Variables Declaration' .....	6-8
Figure 6-4:	Function call in ST.....	6-10
Figure 6-5:	Function block call in ST .....	6-10
Figure 6-6:	Pre-edited MAX function and CTU function block, inserted using the Edit Wizard.....	6-12
Figure 7-1:	Example of an instruction list .....	7-2
Figure 7-2:	Table of operators, modifiers and operands in IL .....	7-3
Figure 7-3:	Table of the modifiers in IL and their meaning.....	7-4
Figure 7-4:	Operator 'ADD' and FB 'RS', both inserted using the Edit Wizard .....	7-4
Figure 7-5:	Dialog 'Variable', called with a variable marked .....	7-7
Figure 7-6:	Dialog 'Automatic Variables Declaration' .....	7-7
Figure 7-7:	Dialog 'Variable' .....	7-9
Figure 7-8:	Example of a jump .....	7-9
Figure 7-9:	Example of calling a function .....	7-10

Figure 7-10:	Example of calling a function block.....	7-10
Figure 7-11:	Pre-edited MAX function and CTU function block, inserted using the Edit Wizard.....	7-12
Figure 8-1:	Dialog 'Function/Function Block' .....	8-5
Figure 8-2:	Dialog 'Variable' .....	8-7
Figure 8-3:	Dialog 'Automatic Variables Declaration' .....	8-8
Figure 8-4:	Function and function block before establishing the connection.....	8-10
Figure 8-5:	The connection is set .....	8-11
Figure 8-6:	The function is moved to a vacant position. The connection is routed automatically. .	8-11
Figure 8-7:	The output 'CV' of the FB 'CTU' is defined to be one end of the connection before the 'ADD' function is inserted .....	8-12
Figure 8-8:	The connection is established automatically when the new 'ADD' function has been inserted .....	8-12
Figure 8-9:	The function is moved to a vacant position. The connection is routed automatically. .	8-12
Figure 9-1:	Table of contacts and coils in LD .....	9-2
Figure 9-2:	Example of a LD network.....	9-3
Figure 9-3:	First LD network inserted .....	9-4
Figure 9-4:	First LD network with inserted new contact 'C002'.....	9-5
Figure 9-5:	LD network with a parallel branch .....	9-6
Figure 9-6:	LD network with a parallel branch, inserted in LD branch edit mode.....	9-7
Figure 9-7:	Dialog 'Contact/Coil' .....	9-9
Figure 9-8:	Dialog 'Automatic Variables Declaration' .....	9-9
Figure 9-9:	LD network with changed properties.....	9-10
Figure 9-10:	LD network with function block 'CTU'.....	9-13
Figure 10-1:	SFC network with one step and one transition.....	10-3
Figure 10-2:	SFC network with four steps .....	10-5
Figure 10-3:	Dialog 'Step' .....	10-6
Figure 10-4:	Dialog 'Divergence' .....	10-7
Figure 10-5:	SFC network with 2 alternative branches.....	10-8
Figure 10-6:	Dialog 'Divergence' .....	10-9
Figure 10-7:	Part of SFC network with an alternative and a simultaneous branch .....	10-10
Figure 10-8:	Dialog 'Action'.....	10-12
Figure 10-9:	Dialog 'Automatic Variables Declaration' .....	10-13
Figure 10-10:	Action block with variable name.....	10-14
Figure 10-11:	Dialog 'Transition'.....	10-15
Figure 10-12:	Transition, specified as direct connection with a green connection point .....	10-15
Figure 10-13:	Dialog 'Variable' .....	10-16
Figure 10-14:	Dialog 'Automatic Variables Declaration' .....	10-16
Figure 10-15:	Variable connected to a transition.....	10-17
Figure 10-16:	Dialog 'Insert'.....	10-19
Figure 11-1:	Dialog 'Insert'.....	11-2
Figure 11-2:	Dialog 'Resource Settings...' .....	11-3

Figure 11-3: Dialog 'Insert'.....	11-4
Figure 11-4: Announced errors after making a project.....	11-7
Figure 11-5: Error list, displayed in the message window .....	11-7
Figure 11-6: Premises for the use of the 'Patch POU' operation.....	11-8
Figure 11-7: Control dialog .....	11-10
Figure 11-8: Dialog 'Download'.....	11-11
Figure 11-9: IL worksheet in online mode .....	11-13
Figure 11-10: Graphical worksheet in online mode.....	11-14
Figure 11-11: Meaning of the colors in online mode .....	11-14
Figure 11-12: Meaning of the powerflow signs in text worksheets in online mode .....	11-16
Figure 11-13: Meaning of the powerflow signs in graphic worksheets in online mode .....	11-16
Figure 11-14: Dialog 'Online Debug'.....	11-18
Figure 11-15: Dialog 'Online Debug'.....	11-19
Figure 11-16: Control dialog if a breakpoint is set.....	11-21
Figure 11-17: ST worksheet in online mode with set breakpoint.....	11-22
Figure 11-18: Watch window with several variables inserted.....	11-23
Figure 11-19: Watch window with opened array of structure .....	11-25
Figure 12-1: Dialog 'Print Project' .....	12-2
Figure 12-2: Page 'Default Pagelayouts' in the dialog 'Options' .....	12-4
Figure 12-3: Pagelayout editor with a new pagelayout.....	12-6
Figure 12-4: Pagelayout with the new source area .....	12-7
Figure A1-1: Example of a specific code block in IL.....	A1-3
Figure A1-2: Example of a specific code block embedded in an IEC IL.....	A1-4
Figure A1-3: Instruction list including IEC IL code and specific code statements.....	A1-6
Figure A1-4: Simple LD network .....	A1-8
Figure A1-5: Dialog 'Inline Code' .....	A1-9
Figure A1-6: LD network with inseted arithmetic box .....	A1-10
Figure A1-7: Dialog 'Inline Code' .....	A1-10
Figure A1-8: Simple LD network and arithmetic box inserted as a single object.....	A1-11
Figure A1-9: LD network with connected arithmetic box .....	A1-12
Figure A1-10: LD network and arithmetic box connected to the right power rail .....	A1-13
Figure A1-11: IL worksheet with specific code statements in online mode .....	A1-16
Figure A1-12: Part of a LD worksheet with included arithmetic box in online mode.....	A1-17
Figure A1-13: Part of a LD worksheet in online mode with online values displayed in the tooltip of the arithmetic box .....	A1-18
Figure A2-1: Dialog 'Insert'.....	A2-2
Figure A2-2: Dialog 'Insert'.....	A2-3
Figure A2-3: Dialog 'Resource Settings for HIDIC' .....	A2-4
Figure A2-4: Dialog 'Resource Configuration'.....	A2-5
Figure A2-5: Dialog 'I/O Configuration' .....	A2-5

Figure A2-6: Dialog 'CPU settings'.....	A2-7
Figure A2-7: Dialog 'Operation parameters' .....	A2-7
Figure A2-8: Dialog 'Insert'.....	A2-8
Figure A2-9: Dialog 'Insert'.....	A2-10
Figure A2-10: Announced errors after making a project.....	A2-12
Figure A2-11: Error list, displayed in the message window .....	A2-12
Figure A2-12: Premises for the use of the 'Patch POU' operation.....	A2-13
Figure A2-13: Control dialog .....	A2-15
Figure A2-14: IL worksheet in online mode .....	A2-18
Figure A2-15: Graphical worksheet in online mode .....	A2-19
Figure A2-16: Meaning of the colors in online mode .....	A2-19
Figure A2-17: Dialog 'Online Debug'.....	A2-21
Figure A2-18: Watch window with several variables inserted.....	A2-23
Figure A2-19: Control dialog .....	A2-25

# Index

## A

- ABS • 2-4
- Action • 10-2, 10-12
  - detail • 10-18
  - qualifier • 10-2, 10-12
- Action block • 10-3
- ADD • 2-4, 7-3
- Addition • 7-3, 6-3
- Address status • 11-15
- Alternative branch in SFC • 10-2, 10-7
- AND • 2-4, 7-3
- Announcing a library • 4-8
- ANY\_BIT • 5-4
- ANY\_INT • 5-4
- ANY\_NUM • 5-4
- Archive • 4-9
- Arithmetic functions • 2-4
- Array data types
  - array of structure • 5-8
  - debug • 11-24
  - multidimensional array • 5-7
  - structures with arrays • 5-9
- Array data types • 5-5
- assignment statement • 6-3
- Associating programs to tasks • 11-4

## B

- B • see size prefix
- Bistable function blocks • 2-4
- Bitmap in pagelayout • 12-8
- Bit-string functions • 2-4
- BOOL • 5-3
- Boolean AND • 7-3, 6-3
- Boolean exclusive OR • 7-3, 6-3
- Boolean OR • 7-3, 6-3
- Branch
  - alternative in SFC • 10-2
  - parallel in LD • 9-3
  - simultaneous in SFC • 10-2
- Breakpoint • 11-18
- BY • 6-4
- BYTE • 5-3

## C

- CAL • 7-3, 7-10

## Calling

- function block in IL • 7-3
- functions and function blocks in IL • 7-10

- CASE • 6-4
- Character string functions • 2-4
- Character string literal • 5-2
- Code body part • 2-7
- Coil • 9-2
  - change properties • 9-8
  - inserting • 9-4
- Colors in online mode • 11-14
- Colors of the status bar • 3-19
- Comment
  - FBD • 8-2
  - IL • 7-2
  - LD • 9-3
  - SFC • 10-2
  - ST • 6-2
- Comparison • 7-3, 6-3
- Comparison functions • 2-4
- Compiling • 11-5
  - Compile worksheet • 11-5
  - errors • 11-6
  - Make • 11-6
  - Patch POU • 11-8, 11-15
- Complement • 6-3
- Configuration • 2-2, 2-3
  - insert • 11-1
- Configuration elements • 2-2
- Connecting graphical objects while inserting • 8-11
- Connecting objects in FBD • 8-9
- Connection line • 9-3
- Contact • 9-2
  - change properties • 9-8
  - inserting • 9-4
- Context-sensitive help • 3-20
- Correcting programming errors • 11-15
- Counter function blocks • 2-4
- Creating a new project • 4-1
- Cross reference window • 3-15
- Cyclic task • 2-2

## D

- D • see size prefix
- Data type
  - array • 5-5



- derived • 5-4
- elementary • 5-3
- generic • 5-4
- initial value • 5-15
- string • 5-10
- structure • 5-8
- user defined • 5-4
- Data type declaration
  - editing using the Edit Wizard • 5-11
- Data type worksheet • 3-23, 5-10
- Data types • 2-8
  - declaration • 2-8
- Debugging • 11-15, 11-24
- Declaration part • 2-7
- Default initial value • 5-15
- Default pagelayout • 12-4
- Derived data types • 2-8, 5-4
- Detail • 10-2, 10-18
- Dialog
  - Action • 10-12
  - Assign Shortcut • 3-10
  - Automatic FB Declaration • 7-12, 6-11, 9-12, 8-4
  - Automatic Variables Declaration • 10-13, 10-16, 7-7, 6-9, 9-10, 9-11, 8-8
  - Contact/Coil • 9-9
  - Control Dialog • 11-11, 11-21
  - Cross Reference Filter • 3-17
  - Customize • 12-4
  - Default Pagelayout • 12-4
  - Divergence • 10-7, 10-9
  - Download • 11-11
  - FB Instances • 7-12, 9-12, 8-4
  - Function/Function Block • 8-5
  - Insert • 4-6, 10-19, 11-2, 11-5
  - New Project • 4-2
  - Online Debug • 11-18, 11-19
  - Print Project • 12-2
  - Properties • 4-4
  - Resource Settings... • 11-3
  - Save changes? • 3-29
  - Save Project As • 4-10
  - Settings Environment Text • 12-9
  - Settings source area • 12-7
  - Shortcut keys • 3-9
  - Step • 10-6
  - Transition • 10-15
  - Variable • 10-16, 7-6, 7-7, 7-9, 6-8, 8-7
- DINT • 5-3
- Direct connection • 10-2
- Directly represented variable • 5-13
- Disk space, display in the status bar • 3-19
- DIV • 7-3
- Division • 7-3, 6-3
- DO • 6-4, 6-5
- documentation
  - context-sensitive help • 3-20
  - context-sensitive help • 1-2
  - Manual • 1-2
  - symbols • 1-3
  - textual conventions • 1-3
- Downloading • 11-10

- Drag & drop in graphic worksheets • 8-9
- Duplicating function inputs • 8-14
- Duration literal • 5-2
- DWORD • 5-3

## E

- Edge detection function blocks • 2-4
- Edit Wizard • 7-5, 8-2
  - Calling functions/FBs in IL • 7-10
  - calling functions/FBs in ST • 6-10
  - editing data type declarations • 5-11
  - general description • 3-24
  - in ST • 6-2
  - Inserting functions/FBs in FBD • 8-3
  - Inserting functions/FBs in LD • 9-11
  - Inserting instructions in IL • 7-4
  - Inserting statements in ST • 6-5
  - inserting variable declaration keywords • 5-18
  - listbox 'Group' • 3-25
  - replacing functions/FBs • 8-6
  - selection area • 3-25
- Editor
  - editing FBD worksheets • 8-1
  - editing IL worksheets • 7-1
  - editing LD worksheets • 9-1
  - editing SFC worksheet • 10-1
  - editing ST worksheets • 6-1
  - worksheets in online mode • 11-12
- Elementary data types • 2-8, 5-3
- ELSE • 6-4
- ELSIF • 6-4
- END\_CASE • 6-4
- END\_FOR • 6-4
- END\_IF • 6-4
- END\_REPEAT • 6-5
- END\_VAR • 5-17
- END\_WHILE • 6-5
- Environment item • 3-23, 12-9
- EQ • 7-3
- Equality • 6-3
- Error list (message window) • 11-7
- Error task • 2-2
- Errors
  - correct programming errors • 11-15
  - displayed in the message window • 3-14
  - while compiling • 11-6
- Event task • 2-2
- EXIT • 6-5
- Exiting
  - Pro-H • 3-29
  - worksheet • 3-28
- Exponentiation • 6-3
- Expression • 6-2, 6-3

## F

- FBD • 8-2
  - call editor • 8-1
  - comment • 8-2
  - connect objects • 8-9
  - insert variable • 8-6

- mix with SFC • 10-18
- negating inputs and outputs • 8-13
- network • 8-2, 8-9
- FBD worksheets • 3-22
- Filtering the cross reference window • 3-17
- Firmware library • 2-10, 4-8
- FOR • 6-4
- Forcing • 11-17
- Formal parameter • 7-10, 8-7, 8-13
- Function
  - call in IL • 7-10
  - call in LD • 9-11
  - call in ST • 6-10
  - changing the properties • 8-5
  - duplicating inputs • 8-14
  - firmware • 4-8
  - general description • 2-4
  - Inserting in FBD using the Edit Wizard • 8-3
  - negating inputs and outputs • 8-13
  - replacing using the Edit Wizard • 8-6
  - types of functions • 2-4
- Function block
  - call in IL • 7-3, 7-10
  - call in LD • 9-11
  - call in ST • 6-10
  - changing the properties • 8-5
  - declare instance • 5-15, 5-20
  - firmware • 4-8
  - general description • 2-4
  - instance • 2-6
  - negating inputs and outputs • 8-13
  - replacing using the Edit Wizard • 8-6
  - types of function blocks • 2-4
- Function block diagram • see FBD

## G

- GE • 7-3
- Generic data types • 2-8, 5-4
- Global variable • 5-14
- Graphic editor
  - edit FBD • 8-1
  - edit LD • 9-1
  - edit SFC • 10-1
  - general description • 3-22
- GT • 7-3

## H

- Hardware requirements • 3-1
- Help function • see context-sensitive help

## I

- I • see location prefix
- I/O configuration worksheet • 3-23
- I/Os, declaring • 5-14
- Icon
  - Add contact right • 9-5
  - Add contact/coil below • 9-6
  - Add function • 4-5
  - Add function block • 4-5
  - Add Program • 4-5
  - Add worksheet • 4-6, 4-7, 4-8
  - Change Node Properties • 4-3
  - code body worksheet name* • 4-6
  - Connect objects • 8-9
  - Contact network • 9-4
  - Data type worksheet* • 5-10
  - date type folder • 4-7
  - Duplicate FP • 8-14
  - Edit Wizard • 7-5, 7-11, 6-6, 6-10, 9-12, 8-3
  - Environment items • 12-9
  - FBD worksheet *name* • 8-1
  - Insert LD branch • 9-7
  - Insert SFC branch • 10-11
  - Insert Simultaneous/Alternative Branches • 10-7, 10-9
  - Insert step/transition • 10-3, 10-4
  - LD worksheet *name* • 9-1
  - libraries folder • 4-8
  - Line • 12-8
  - Make • 11-6
  - Normally closed contact • 9-8
  - Online on/off • 11-9, 11-15
  - POU name* • 4-9
  - Program name* • 4-3
  - Resource name'* • 11-1
  - Save • 3-28
  - SFC worksheet *name* • 10-1
  - Show Control Dialog • 11-10, 11-20
  - Source area • 12-6
  - ST Worksheet *name* • 7-1, 6-1
  - Task name* • 11-4
  - Toggle negation of FP • 8-13
  - Variable • 10-15, 7-6, 7-8, 6-7, 8-7
  - variable worksheet name* • 4-7
- Icons
  - description in the status bar • 3-7
  - tooltips, activating • 3-7
- IEC 61131 • 2-1
- IEC 61131-3
  - configuration • 2-2
  - configuration elements • 2-2
  - data types • 2-8
  - function • 2-4
  - function blocks • 2-4
  - instantiation • 2-6
  - POU • 2-4

- program • 2-5
- programming language • 2-11
- resource • 2-2
- task • 2-2
- variables • 2-8
- IEC code • 11-5
- IF • 6-4
- IL • 7-2
  - call editor • 7-1
  - call function • 7-10
  - call function block • 7-10
  - comment • 7-2
  - insert variable • 7-6
  - instruction • 7-2
  - Jump • 7-9
  - label • 7-9
  - modifier • 7-2, 7-4
  - operand • 7-2
  - operator • 7-2
- IL worksheets • 3-23
- Inequality • 6-3
- initial step • 10-2, 10-3
  - change • 10-6
- Initial value • 5-15
- Initializing variables • 5-15
- Installation • 3-2
- Instance • 2-6, 5-20
- Instance name • 2-6, 5-20, 11-5
- Instance tree • 2-6, 3-21, 5-20, 11-12
- Instantiation • 2-6, 5-20
  - of function blocks • 7-12, 9-12, 8-4
- Instruction
  - IL • 7-2
  - Inserting in IL using the Edit Wizard • 7-4
- instruction list • see IL
- INT • 5-3
- Internal memory • 2-5, 2-6, 5-20
- Interrupt task • 2-2
- Iteration statement • 6-4

## J

- JMP • 7-3, 7-9
- Jump • 7-3, 7-9

## K

- Keyboard Shortcuts • see shortcuts
- Keyboard, general usage • 3-3
- Keywords in ST • 6-2

## L

- Label • 7-9
- Ladder diagram • see LD
- Language element • 2-4
- LD • 9-2
  - Branch edit mode • 9-7
  - call editor • 9-1
  - call functions or function blocks • 9-11
  - change properties of contacts/coils • 9-8
  - coil • 9-2
  - comment • 9-3
  - connection line • 9-3
  - contact • 9-2
  - insert coil • 9-4
  - insert contact • 9-4
  - insert variable • 9-11
  - mix with SFC • 10-18
  - network • 9-3
  - operator • 7-3
  - Parallel branch • 9-3, 9-6, 9-7
  - power rail • 9-3
  - variable • 9-3
  - wired-OR • 9-3, 9-6
- LD worksheets • 3-22
- LE • 7-3
- Library • 2-10
  - announce • 2-11, 4-8
  - print • 12-1
- Line
  - pagelayout • 12-8
- Literal
  - character string • 5-2
  - duration • 5-2
  - general description • 5-1
  - numeric • 5-1
- Local variable • 5-14
- Located variable • 5-13
- Location • 5-13
  - prefix • 5-13
- LT • 7-3

## M

- M • see location prefix
- Main screen • 3-13
- Make • 11-5, 11-6
- Menu bar • 3-5
- Message window • 3-14
  - errors while compiling • 11-7
  - Warnings while compiling • 11-7
- mixing
  - FBD and LD • 9-11
- Modifier
  - IL • 7-2, 7-4
- Modulo • 6-3
- Mouse, general usage • 3-3
- MUL • 7-3
- Multiplication • 7-3, 6-3
- Pro-H
  - exit • 3-29
  - installation • 3-2

starting the program • 3-2  
user interface • 3-4

## N

NE • 7-3  
Negated coil • 9-2  
Negating inputs and outputs • 8-13  
Negation • 6-3  
Network  
  FBD • 8-2, 8-9  
  insert SFC network • 10-3  
  LD • 9-3  
  SFC • 10-2  
Normally closed contact • 9-2  
Normally open contact • 9-2, 9-3  
Numeric literal • 5-1  
Numerical functions • 2-4

## O

Online mode • 11-12  
  address status • 11-15  
  colors • 11-14  
  debug arrays and structures • 11-24  
  function worksheet • 11-16  
  Layout • 11-14  
  powerflow • 11-15  
  powerflow in graphic worksheets • 11-16  
  powerflow in text worksheets • 11-16  
  switching on/off • 11-15  
  variable status • 11-15  
  watch window • 11-22  
Operand  
  IL • 7-2  
  ST • 6-2, 6-3  
Operator  
  IL • 7-2  
  ST • 6-2, 6-3  
Optimized printing • 12-3, 12-4  
OR • 7-3  
Overview window • 3-27  
Overwriting variables • 11-17

## P

Pagelayout  
  creating a new • 12-5  
  defining as default • 12-3  
  preview • 12-10  
Pagelayout editor • 3-23  
  call • 12-5  
  define source area • 12-6  
  environment item • 3-23  
  insert a bitmap • 12-8  
  insert a line • 12-8  
  insert a rectangle • 12-8  
  insert a system item • 12-9  
  insert a text • 12-8  
  insert an environment item • 12-9  
Parallel branch in LD • 9-3, 9-6, 9-7  
parenthesization • 6-3

Patch POU • 11-5, 11-8, 11-15  
PLC code • 11-5  
POU  
  change properties • 4-3  
  Delete • 4-9  
  insert • 4-5  
  types of POUs • 2-4  
Power rail • 9-3  
Powerflow • 11-15  
Preview • 3-23, 12-10  
Printing • 12-1  
  print mode 'Optimized printing' • 12-3  
  print mode 'Standard' • 12-3  
  select items • 12-2  
  select range • 12-2  
Program  
  associate to task • 11-4  
  general description • 2-5  
  instance • 2-6, 5-21, 11-5  
  internal memory • 2-5  
Program modules • 2-4  
Program organization unit • see POU  
Programming errors • 11-15  
Programming language  
  FBD • 8-2  
  graphical • 2-11  
  IL • 7-2  
  LD • 9-2  
  ST • 6-2  
  textual • 2-11  
Project  
  compile • 11-5  
  create • 4-1  
  downloading • 11-10  
  Save • 4-9  
  template • 4-1  
  Untitled • 4-1  
  Zip into an archive file • 4-9, 4-10  
Project tree • 2-10  
  configuration elements • 2-2  
  insert configuration • 11-1  
  insert library • 4-8  
  insert POU • 4-5  
  insert resource • 11-1  
  insert task • 11-1  
  insert worksheet • 4-6  
  subtree 'Data types' • 2-9, 3-21  
  subtree 'Instances' • 2-6  
  subtree 'Library' • 2-11, 3-21  
  subtree 'Logical POUs' • 2-6, 3-21  
  subtree 'Physical Hardware' • 2-2, 3-21  
Project tree editor • 3-21  
Properties • 4-3

## Q

Q • see location prefix  
Qualifier in SFC • 10-2, 10-12

## R

R • 7-3  
Rail • 9-3  
REAL • 5-3  
Rebuild project • 11-6  
Rectangle in pagelayout • 12-8  
REPEAT • 6-5  
RESET coil • 9-2  
Resource • 2-2, 2-3  
    control dialog • 11-11, 11-21  
    insert • 11-1  
    Settings • 11-3  
RET • 7-3  
RETAIN • 5-14  
Retentive variable • 5-14  
    initial value • 5-15  
Return • 7-3, 6-5

## S

S • 7-3  
Save as... • 4-10  
Saving • 3-28  
Scope  
    variable • 5-14  
Selection functions • 2-4  
Selection statement • 6-4  
Sequential function chart • see SFC  
SET coil • 9-2  
Setup • 3-2  
SFC • 10-2  
    action • 10-2  
    action block • 10-2  
    action qualifier • 10-2, 10-12  
    alternative branch • 10-7  
    Branch edit mode • 10-11  
    call editor • 10-1  
    change network • 10-6  
    comment • 10-2  
    detail • 10-2, 10-18  
    insert network • 10-3, 10-4  
    insert variable • 10-12, 10-14  
    mix with FBD • 10-18  
    mix with LD • 10-18  
    network • 10-2  
    simultaneous branch • 10-9  
    step • 10-2  
    transition • 10-2  
SFC worksheets • 3-22  
Shortcut Manager • 3-9  
Shortcuts • 3-9  
    adding/modifying • 3-9  
    assigning new shortcuts • 3-10  
    list of default shortcuts • 3-11  
Simultaneous branch in SFC • 10-2, 10-9, 10-11

SINT • 5-3  
Size prefix • 5-13  
Software requirements • 3-1  
Sorting the cross reference list • 3-18  
Source area • 12-6  
ST • 7-3  
    assignment statement • 6-10  
    call editor • 6-1  
    call function • 6-10  
    call function block • 6-10  
    CASE statement • 6-4  
    comment • 6-2  
    edit assignment statements • 6-3  
    edit statements • 6-4  
    EXIT statement • 6-5  
    expression • 6-2, 6-3  
    FOR statement • 6-4  
    general description • 6-2  
    IF statement • 6-4  
    insert variable • 6-7  
    operand • 6-2, 6-3  
    operator • 6-2, 6-3  
    REPEAT statement • 6-5  
    RETURN statement • 6-5  
    statement • 6-2  
    statement keywords • 6-4  
    WHILE statement • 6-5  
ST worksheets • 3-23  
Standard function blocks • 2-4  
Standard print mode • 12-3  
Starting the program • 3-2  
Statement • 6-2  
    assignment statement • 6-3, 6-10  
    control statement • 6-4  
    Inserting in ST using the Edit Wizard • 6-5  
    iteration statement • 6-4  
    keywords • 6-4  
    selection statement • 6-4  
Status bar • 3-19  
Step • 10-2  
    change properties • 10-6  
    initial • 10-2  
    insert • 10-3, 10-4  
Step (debugging) • 11-21  
STRING • 5-4  
String data types • 5-10  
Structured data types • 5-8  
    debug • 11-24  
    initialized • 5-9  
    structures with arrays • 5-9  
Structured text • see ST  
SUB • 7-3  
Submenus, general description • 3-5  
Subtraction • 7-3, 6-3  
Subtree 'Data types' • 2-9, 3-21  
Subtree 'Instances' • 2-6  
Subtree 'Library' • 2-11, 3-21  
Subtree 'Logical POUs' • 2-6, 3-21  
Subtree 'Physical Hardware' • 2-2, 3-21  
Symbolic variable • 5-13  
System item • 12-9

## T

- Task • 2-2, 2-3
  - associate a program • 11-4
  - insert • 11-1
- Template • 4-1
- Text editor
  - declaring data types • 5-10
  - edit IL • 7-1
  - edit ST • 6-1
  - general description • 3-23
- TIME • 5-3, 5-4
- Time scheduling • 2-2
- Timer function blocks • 2-4
- TO • 6-4
- Toolbar • 3-7
- Tooltips • 3-7
- Trace • 11-21
- Transition • 10-2, 10-3
  - changing properties • 10-14
  - detail • 10-18
  - direct connection • 10-2
  - insert • 10-3, 10-4
  - variable • 10-14
- Type conversion functions • 2-4

## U

- UDINT • 5-3
- UINT • 5-3
- UNTIL • 6-5
- User defined data types • 2-8, 5-4
  - array • 5-5
  - strings • 5-10
  - structures • 5-8
- User interface
  - cross reference window • 3-15
  - Edit Wizard • 3-24
  - general description • 3-4
  - mainscreen • 3-13
  - menu bar • 3-5
  - message window • 3-14
  - overview window • 3-27
  - status bar • 3-19
  - submenus • 3-5
  - toolbar • 3-7
  - workspace • 3-13
- User library • 2-10, 4-8
- USINT • 5-3

## V

- VAR • 5-15
- VAR\_EXTERNAL • 5-14, 5-16
- VAR\_EXTERNAL\_FB • 5-16
- VAR\_EXTERNAL\_PG • 5-16
- VAR\_GLOBAL • 5-14, 5-17
- VAR\_GLOBAL\_FB • 5-17
- VAR\_GLOBAL\_PG • 5-17
- VAR\_IN\_OUT • 5-16
- VAR\_INPUT • 5-14, 5-16
- VAR\_OUTPUT • 5-14, 5-16
- Variable • 2-8
  - declare • 5-18
  - declare using the variable editor • 5-18
  - declare while editing a code body • 5-18
  - directly represented • 5-13
  - force • 11-17
  - global • 5-14
  - initialize • 5-15
  - insert in FBD • 8-6
  - insert in IL • 7-6
  - insert in LD • 9-11
  - insert in SFC • 10-12, 10-14
  - insert in ST • 6-7
  - inserting into watch window • 11-22
  - keywords • 5-15
  - local • 5-14
  - located • 5-13
  - overwrite • 11-17
  - retentive • 5-14
  - scope • 5-14
  - status in online mode • 11-15
  - symbolic • 5-13
  - watch window • 11-22
- Variable declaration • 2-7, 2-8
  - function block instance • 5-20
  - inserting keywords using the Edit Wizard • 5-18
  - instantiation • 5-15
  - keywords • 5-15
  - location • 5-13
  - location prefix • 5-13
  - size prefix • 5-13
  - types • 5-15
- Variable worksheets • 3-23

## **W**

W • see size prefix

Warnings, displayed in the message window • 3-14

Watch window • 11-22

  Debug arrays and structures • 11-24

  insert variable • 11-22

WHILE • 6-5

Wildcards • 3-17

Wired-OR • 9-3, 9-6

WORD • 5-3

Worksheet

  code body • 4-3

  delete • 4-9

  description • 4-3

  exit • 3-28

  insert • 4-6

  online mode • 11-12

  preview • 12-10

  print in online mode • 12-1

  save • 3-28

  variable • 4-3

Workspace • 3-13

## **X**

X • see size prefix

XOR • 7-3

## **Z**

  Zipping a project • 4-10

  zwt files • 4-10

**Imprint:**

Valid from Mar. 1999

English release 1.0

All rights reserved, including those of translation, reprint and reproduction of parts of this manual. No part of this work may be duplicated or reproduced in any shape by using electronic systems, nor for educational purposes, without the editor's authorization.

Modifications reserved, as we are continually working on improvements.

© **Hitachi Ltd. Industrial Components & Equipment Division**

46-1. Ooaza-Tomioka, Nakajo-machi,  
Kitakanbara-gun, Niigata-ken 959-2608, Japan

Tel. ++81 254 46 5513

Fax. ++81 254 46 3321

e-mail: [fa-plc@cm.nakajo.hitachi.co.jp](mailto:fa-plc@cm.nakajo.hitachi.co.jp)