

User Manual

OpenPCS Version 7.0

Content

1	A Quick Tour through OpenPCS	8
1.1	Installation	8
1.2	Hardware and Software Requirements	8
1.3	Starting OpenPCS	9
1.4	OpenPCS Samples	10
1.5	Guided Tour	10
1.5.1	Guided_Tour: Intro	10
1.5.2	Sample Program	11
1.5.3	Executing code	13
1.5.4	Monitoring Code	17
1.5.5	Control Data Analyzer	19
1.5.6	Online Edit	20
1.6	Additional	22
1.6.1	Adding Hardware Support	22
1.6.2	Templates	22
1.6.3	XML-Import/Export	23
1.6.4	About this manual	24
1.6.5	More Information	24
2	OpenPCS Tools	25
2.1	OpenPCS Framework	25
2.1.1	OpenPCS Framework: Introduction	25
2.1.2	Output Window	26
2.2	Browser	27
2.2.1	Browser: Introduction	27
2.2.2	Browser: Overview	27
2.2.3	Projects	31
2.2.4	Files	33
2.2.5	Resources and Tasks	33
2.2.6	OPC - I/O	35
2.2.7	Compiler	36
2.2.8	Online	36
2.2.9	Other Browser Features	40
2.3	Catalog	42
2.3.1	Catalog	42
2.3.2	Variable Catalog	43
2.4	Declaration Editor	44

2.4.1	Declaration Editor: introduction	44
2.4.2	Declaration Sections	45
2.4.3	Structure of a Declaration Line	47
2.4.4	Elementary Data Types	48
2.4.5	Directly represented variables	49
2.4.6	Derived data types	51
2.4.7	Declaration of array data types	52
2.4.8	Declaration of structured data types	52
2.4.9	Declaration of enumeration data types	53
2.5	Assignment Editor	54
2.5.1	Assignment Editor: Introduction	54
2.6	IL Editor	55
2.6.1	IL Editor: Introduction	55
2.6.2	Structure of Instruction List	56
2.6.3	Instructions in IL	56
2.6.4	IL Editor Online	57
2.7	ST Editor	57
2.7.1	ST Editor: introduction	57
2.7.2	Instructions in ST	58
2.7.3	Expressions in ST	58
2.7.4	Comments in ST	59
2.7.5	ST Editor Online	59
2.7.6	Tooltips for structs and elements of structs	60
2.7.7	AutoComplete / AutoDeclare	60
2.8	Ladder Diagram Editor	61
2.8.1	Ladder Editor: introduction	61
2.8.2	Ladder Logic: introduction	61
2.8.3	Network	61
2.8.4	Operators	62
2.8.5	Coils	62
2.8.6	Contact	63
2.8.7	Control Relay	64
2.8.8	Functionblocks and Functions	64
2.8.9	Ladder Editor Online	65
2.8.10	Check over Variable	65
2.8.11	AutoComplete / AutoDeclare	67
2.9	CFC Editor	67
2.9.1	Introduction CFC Editor	67

2.9.2	Working with Blocks	67
2.9.3	Connections	68
2.9.4	Margin Bars	68
2.9.5	CFC Editor Online	69
2.9.6	Advanced CFC topics	69
2.9.7	Compound Blocks	85
2.10	SFC Editor	87
2.10.1	SFC: introduction	87
2.10.2	Elements of a sequential function chart	88
2.10.3	Steps and initial steps	91
2.10.4	Transitions	91
2.10.5	Jumps	92
2.10.6	SFC Editor Online	92
2.10.7	Common errors	93
2.10.8	Selecting Elements	96
2.10.9	Advanced SFC topics	97
2.11	FBD Editor	98
2.11.1	Introduction FBD Editor	98
2.11.2	Working with Blocks	99
2.11.3	Connections	100
2.11.4	Margin Bars	101
2.11.5	Advanced	103
2.12	Test and Commissioning	105
2.12.1	Test and Commissioning: Introduction	105
2.12.2	Start and Stop	106
2.12.3	Watch variables	106
2.12.4	Set variables	106
2.12.5	Force Variables	106
2.12.6	Working with watchlists	107
2.13	Control Data Analyzer	108
2.13.1	Control Data Analyzer	108
2.13.2	Oscilloscope	110
2.13.3	Trigger	111
2.14	SmartSIM	112
2.14.1	Overview SmartSIM	112
2.14.2	Interrupt Tasks	112
2.15	OPC Server	113
2.15.1	About OPC Server	113

2.15.2	Remote OPC Server	113
2.16	Online Server	116
2.16.1	Online Server: Overview	116
2.16.2	Online Server Setup	116
2.17	Hardware drivers	119
2.17.1	Hardware drivers: Overview	119
2.18	Compiler	119
2.18.1	Compiler: Overview	119
2.18.2	Instruction List Compiler	120
2.18.3	Linker	121
2.18.4	Make	122
2.19	Licence Editor	123
2.19.1	Licence Editor: Overview	123
2.19.2	Usage without Licence Key	124
3	Advanced Topics	125
3.1	Runtime issues	125
3.1.1	Multitasking	125
3.1.2	Interrupts	126
3.1.3	Optimisation Settings	126
3.1.4	Multiple Resources	127
3.1.5	Variable Address	127
3.1.6	Performance	127
3.1.7	Adjusting order of cyclic tasks	128
3.2	Native Code Compiler	128
3.2.1	Native Code	128
3.2.2	Direct Calls	129
3.2.3	Exception Handling in native code	129
3.2.4	Unknown instructions	130
3.2.5	Span segments	130
3.2.6	NCC Intel Protected Mode	130
3.2.7	NCC Infineon C16x (huge model)	130
3.2.8	NCC Motorola 68K	131
3.2.9	NCC Hitachi H8/300H	131
3.2.10	NCC Motorola DSP563xx	131
3.2.11	NCC Intel Real Mode	131
3.2.12	NCC Motorola PowerPC	131
3.2.13	NCC ARM ARM Mode	131
3.2.14	NCC ARM THUMB Mode	131

3.3	Documentation	132
3.3.1	Crossreference	132
3.3.2	Cross-Reference (per variable)	132
3.3.3	Print IEC61131 Configuration	132
3.3.4	CFC Crossreference	132
3.3.5	Print-Options	136
3.3.6	Active Document Server	137
3.4	Libraries	137
3.4.1	Library: Overview	137
3.4.2	Create a Library	138
3.4.3	Install a Library	139
3.4.4	Adding a Library to a project	140
3.4.5	Uninstall Library	141
3.5	CANopen	141
3.5.1	CANopen: introduction	141
3.5.2	CANopen network variables	142
3.5.3	Configuration process	144
3.5.4	Insert a DCF-file into OpenPCS	146
3.5.5	Declaration of CANopen network variables	146
3.5.6	Synchronisation	148
3.5.7	CANopen constants	150
3.6	IEC61131-3	152
3.6.1	IEC61131-3 Details	152
3.6.2	IEC61131-3 Compliance Statement	156
3.7	Online Features	189
3.7.1	Breakpoints	189
3.7.2	Online Edit	190
3.7.3	Save System	191
3.7.4	Error Logs	191
4	Reference	192
4.1	Keywords (by category)	192
4.1.1	IEC61131 Standard Function Blocks	192
4.1.2	IEC61131-3 Standard Functions	192
4.1.3	IEC61131-3 operations	194
4.1.4	OpenPCS Functions and Function Blocks	194
4.1.5	Data Types	195
4.1.6	Declaration Keywords	196
4.1.7	Instruction List Instructions	196

4.1.8	Structured Text Keywords	198
4.1.9	CANopen	199
4.1.10	Others	200
4.2	Keywords (A..Z)	202
4.3	Errors and Warnings	300
4.3.1	How to Read Error Message	300
4.3.2	General Errors	301
4.3.3	Syntax Errors	301
4.3.4	Linker Messages	357
4.3.5	Compiler Messages	365
4.3.6	Make Messages	378
4.4	Shortcuts	379
4.4.1	Common Shortcuts	379
4.4.2	Editor depending Shortcuts	380
5	Index	381

1 A Quick Tour through OpenPCS

1.1 Installation

OpenPCS is delivered on CD-Rom. The CD auto-starts a screen where you can select the software you want to install. If auto-start is not activated or does not work, please start the SETUP.EXE from the subdirectory SOFTWARE\OPENPCS*<language>*.\.

At the end of you the installation, you will be asked if you want to install hardware drivers. If you got those with your PLC, enter the path to the hardware driver, else click "Quit". When you have got drivers for your PLC, you also got a licence key for OpenPCS. See [Licence Editor](#) for how to insert a licence key.

If you have not got a hardware driver nor a licence key, OpenPCS is still full functional, but restricted to "SIMULATION".

Note: Installations to substituted drives are not supported by Windows XP.

1.2 Hardware and Software Requirements

OpenPCS requires a PC with at least:

Pentium II, 1GHz

512 MB RAM

180 MB of free disk space

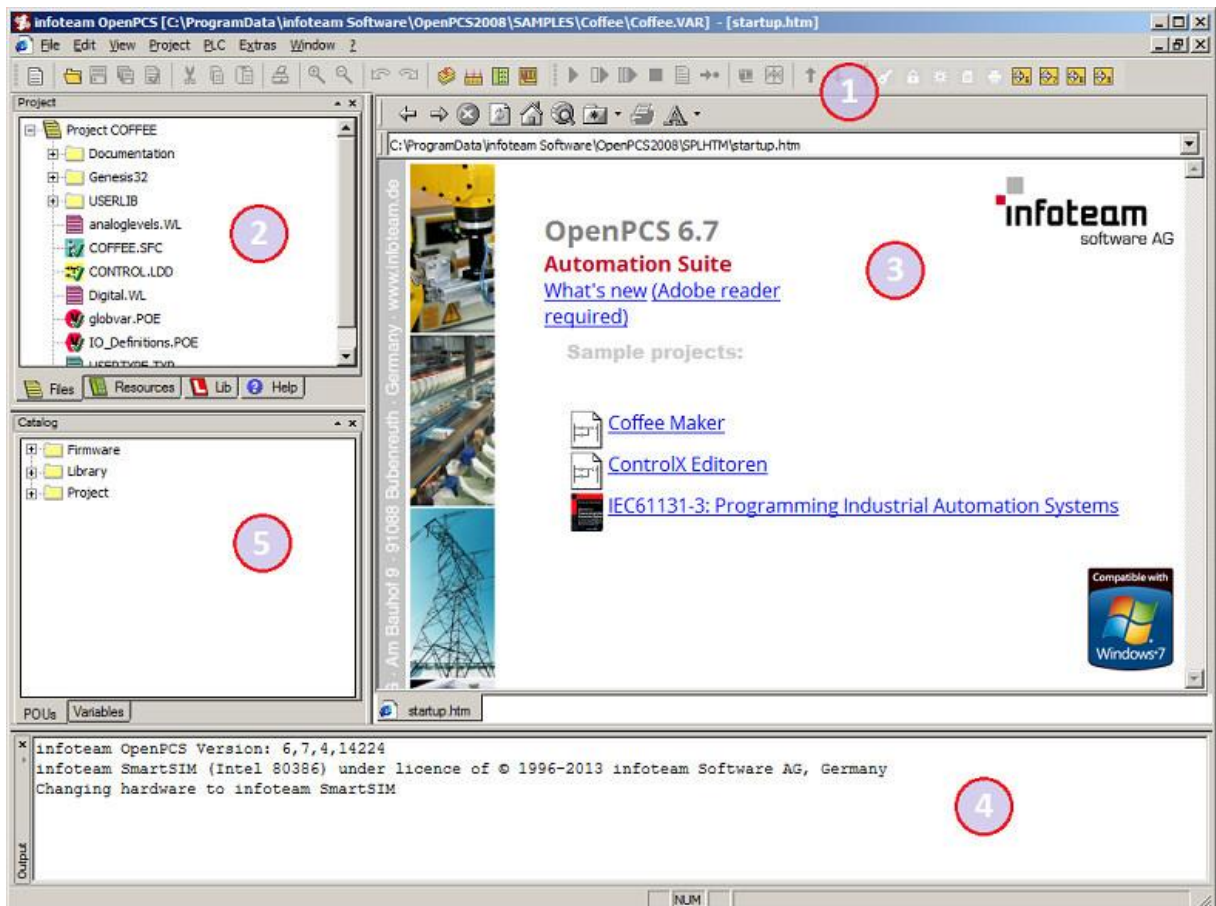
CD-ROM and 1024*768 resolution

Winows 2003, Windows XP SPII or Windows Vista 32bit

To support your specific PLC, more requirements may hold (e.g. more memory), and you may need additional hard- or software (e.g. interface cards, cables). If in doubt, consult the manual of your PLC.

1.3 Starting OpenPCS

Start Windows and choose Start->Programs->infoteam OpenPCS 2008->infoteam OpenPCS 2008 in the start-menu; this will open the [OpenPCS-Framework](#). If the project is created by OpenPCS versions prior to 7.0, the user will be asked to convert project files into UTF-8 format.



The screen is divided into 5 regions:

1. The top region with the menus and toolbars
2. The [Project-Browser](#)
3. The Editor-Window
4. The diagnostic [output window](#)
5. The [Catalog](#)-Window

The last opened project is displayed by default at start-up. An overview on delivered samples is given in the editor window, therefore it may differ from the shown screenshot.

1.4 OpenPCS Samples

OpenPCS comes with a variety of sample projects. When you start OpenPCS the first time a Startup-Screen with a list of sample projects will be displayed. Click on one of these and it will be opened.

Coffee	Emulates a coffee brewer. The program will be explained in detail on the following pages
ControlX	Demonstrates the common languages IL, ST, SFC and Ladder Diagram
BookExam	Example program of the Book IEC61131-3: Programming Industrial Automation Systems by Karl-Heinz John and Michael Tiegelkamp

If the startup-screen isn't displayed you can find the sample projects in the sample folder of your OpenPCS directory.

Notes:

1. Each sample project comes with a description that will automatically be shown when the project is opened.
2. The samples you see may vary due to OEM dependencies done by Brand Labelling OpenPCS.

1.5 Guided Tour

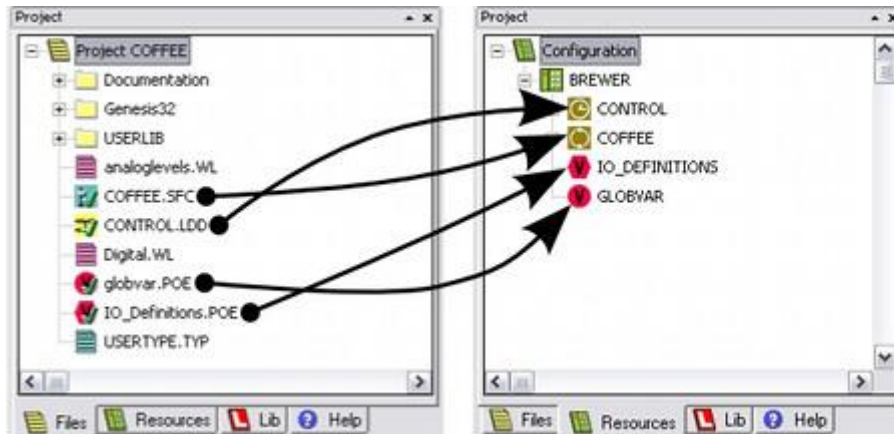
1.5.1 Guided_Tour: Intro

The guided tour hands the user a step-by-step introduction into the programming system OpenPCS.

The coffee sample is delivered with OpenPCS and is located in the OpenPCS sample directory.

The sample simulates a coffee machine and controls its flow. Both procedures are split into two tasks. The simulation of the machine is done via a sequential function chart (SFC) called coffee.sfc. Therefore timed routines are used which map the physics of the

brewer. A ladder diagram called control.ldd controls the tasks. Two declaration files globvar.poe and IO_Definitions.poe are necessary for the interaction of both tasks. The division into tasks is visible in the resource tab in the project browser, where the resource brewer consists of the two aforementioned tasks and definitions. Unfolding the tree shows the respective variables. The screenshot shows the relations between the files and the resources.

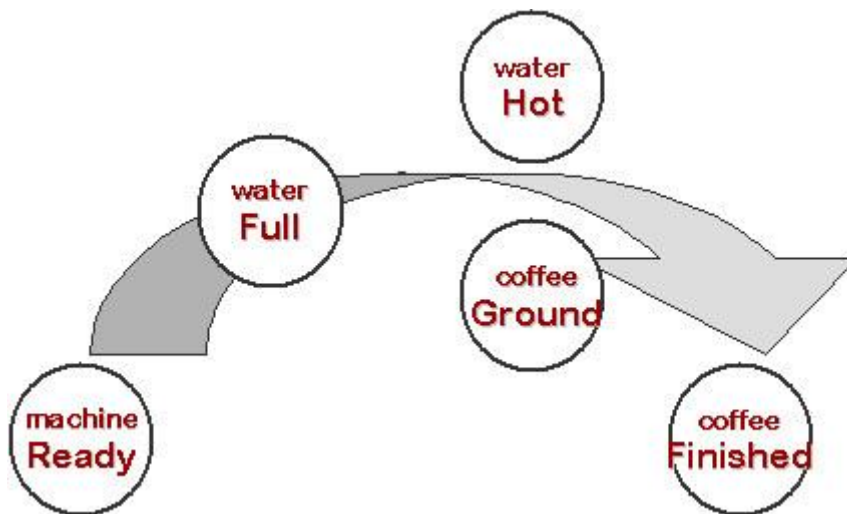


To add new files to a resource the user needs to right-click onto a file within the project browser and select "Link To Active Resource". A green check mark will be visible within the icon if the file is linked to the resource.

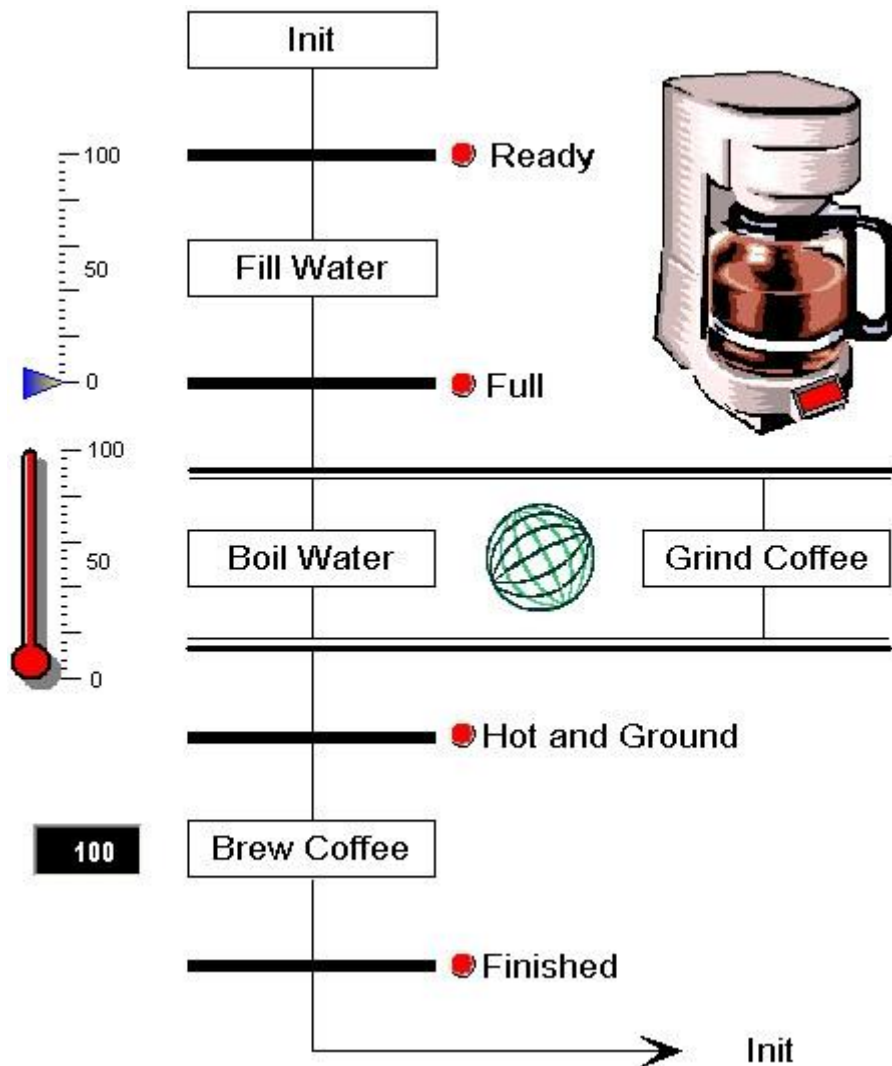
The Guided Tour is split into two sections. The first introduces the reader into the sample program and explains the used routines. The second part deals with compiling, executing and monitoring the program within OpenPCS.

1.5.2 Sample Program

A coffee brewer can be modeled with five states. A mandatory initial state called machine ready and the states water full, water hot, coffee ground and coffee finished as illustrated below. The programming coffee.sfc is based on these states. As the illustration shows grinding the coffee and heating the water can be done simultaneously.



Visualization via Iconics GraphWorkX illustrates the brewer. Illustrated is the chart coffee.sfc simulating the brewer as well as displays showing the temperature and water level. The states water hot and coffee ground are combined in a state called Hot and Ground.



1.5.3 Executing code

Please open the sample project coffee.var.

To execute the application, we need to compile it and transfer the code to the controller first. To build the code for the controller select **PLC->Build Active Resource** from the menu bar. In the output window, you will see the compilation proceed. The end of the output should look similar to the following:

```

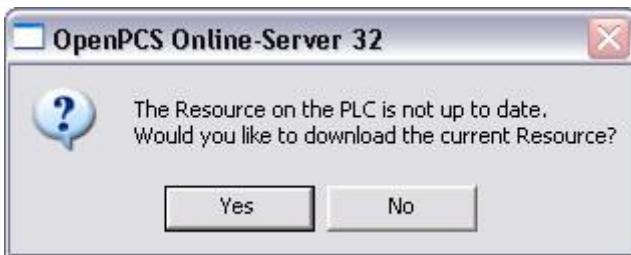
* Code size in bytes: 16097.
* Number of segments: 58.
0 error(s), 0 warning(s) - C:\PROJECTS\COFFEE\GEN\Resource\Resource.PCD.

VARTAB32: 6 variables added in 1 segments (229 bytes)

Persistency file creation disabled due to online linking.
Executing Post-Build-Steps:
Total:
0 error(s) 0 warning(s)

```

After compilation finished successfully, your code needs to be transferred to your controller. Now select **PLC->Online** to Connect to the resource. OpenPCS will detect, that your application needs to be downloaded and will prompt your permission to do so:

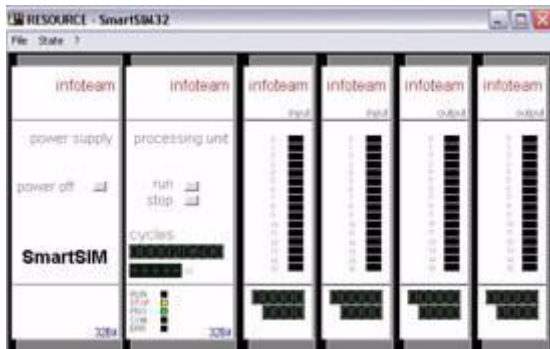


If a problem occurs, OpenPCS prints the error in the same window. The manual of OpenPCS hands a complete overview on all errors and hands possible solutions to the user.

Accept that with "yes". You will see a progress bar while the code is being transferred, but for this small example it should be finished very quickly. When download has finished, you will see that OpenPCS automatically opened another of its tools, the "Test and Commissioning". This is proof that OpenPCS is online:

Instancepath	Name	Value	Type	Address	Force	Comment

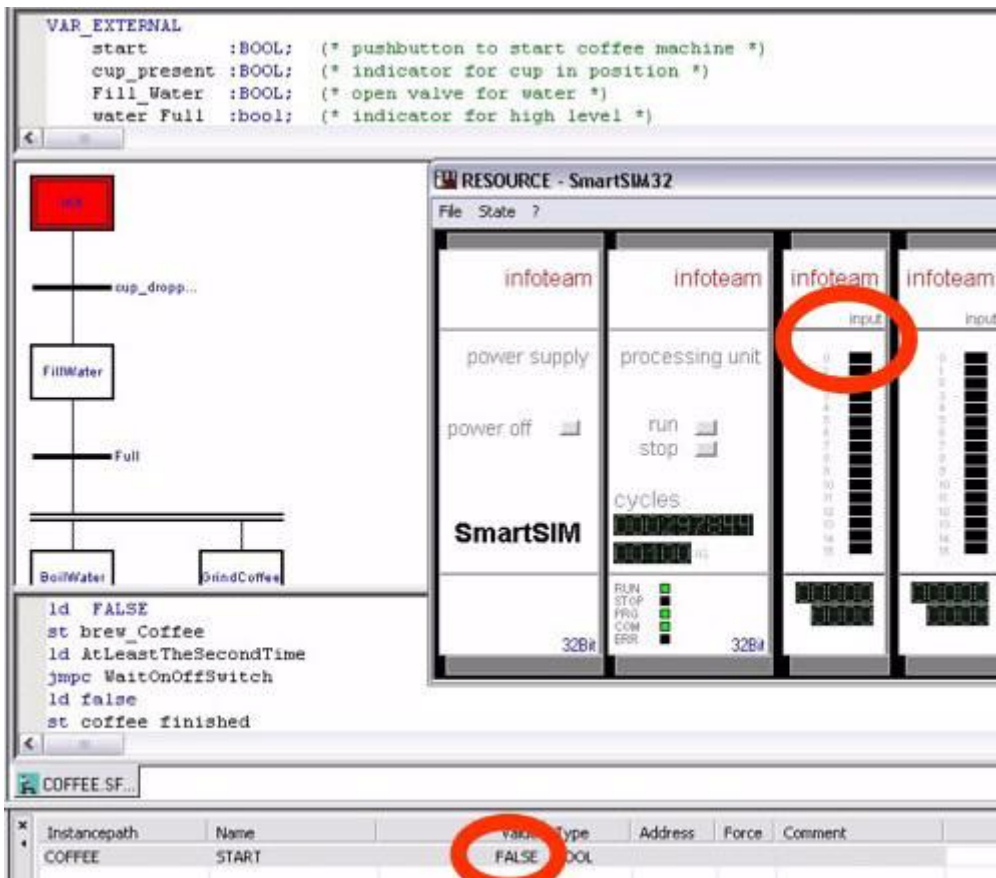
In this introduction, we are not using a real hardware controller. Instead, we are using the Windows Simulation tool that comes with OpenPCS, named SmartSIM, which OpenPCS starts automatically when downloading the program:



Use **PLC-> Coldstart** in the menu (or press the red arrow in the toolbar) to start execution of your code.



After starting the execution the following screen should be visible. The coffee brewer is initialized and the init-step is colored red.



The screenshot displays the infoteam SmartSIM32 software interface. At the top, a variable declaration block is visible:

```

VAR_EXTERNAL
  start      :BOOL; (* pushbutton to start coffee machine *)
  cup_present :BOOL; (* indicator for cup in position *)
  Fill_Water :BOOL; (* open valve for water *)
  water_Full :bool; (* indicator for high level *)

```

The main workspace is divided into several sections:

- Ladder Logic:** A diagram on the left shows a red step labeled "FillWater" connected to a transition labeled "cup_dropp...". Below this, there are two parallel paths leading to "BoilWater" and "DrindCoffee".
- Resource View:** A window titled "RESOURCE - SmartSIM32" shows four resource units. The third unit, labeled "input", has a red circle around its "input" indicator, which is currently active (filled).
- Code Editor:** A section at the bottom left contains the following code:

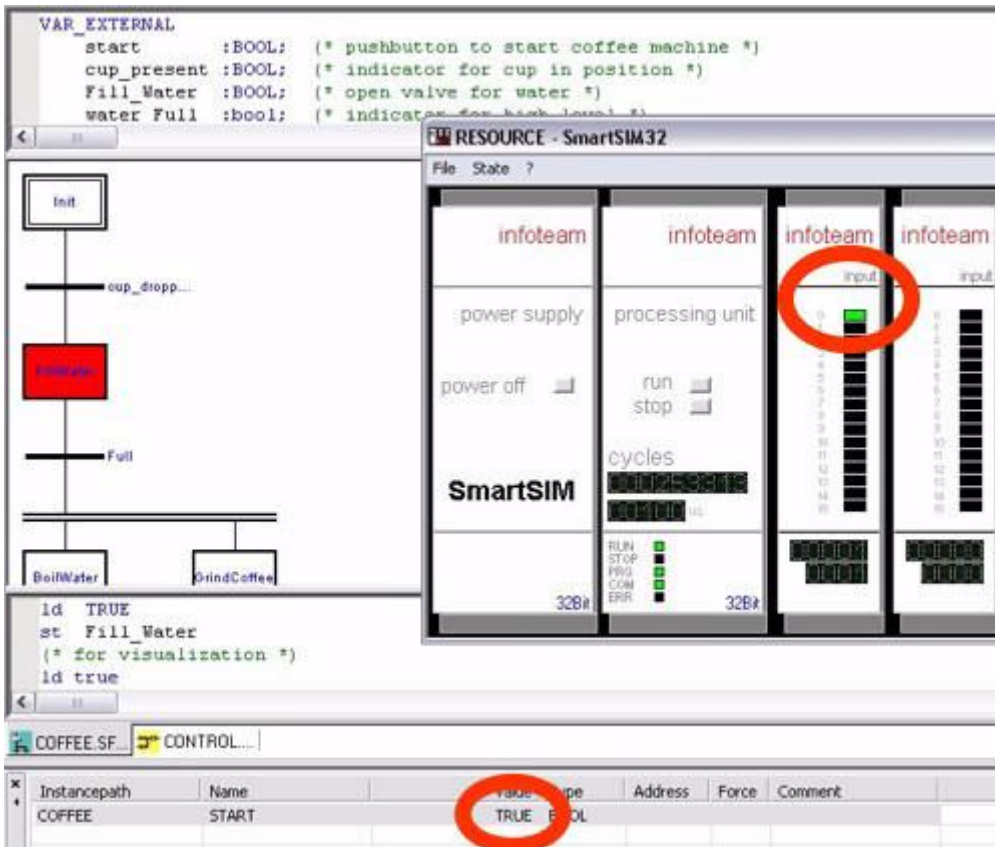

```

ld FALSE
st brew_Coffee
ld AtLeastTheSecondTime
jmpc WaitOnOffSwitch
ld false
st coffee finished

```
- Watchlist:** A table at the bottom shows the current state of variables:

Instancepath	Name	Value	Type	Address	Force	Comment
COFFEE	START	FALSE	BOOL			

Go to SmartSIM and activate the first input ("button"). The variable "start" turns to TRUE, as can be seen in the watchlist at the bottom of the screen, and the machine starts. Moreover, the SFC activates step "Fill_Water" which is now colored red.



```

VAR_EXTERNAL
start      :BOOL: (* pushbutton to start coffee machine *)
cup_present :BOOL: (* indicator for cup in position *)
Fill_Water :BOOL: (* open valve for water *)
water_Full :bool: (* indicator for high level *)

```

```

id TRUE
st Fill_Water
(* for visualization *)
ld true

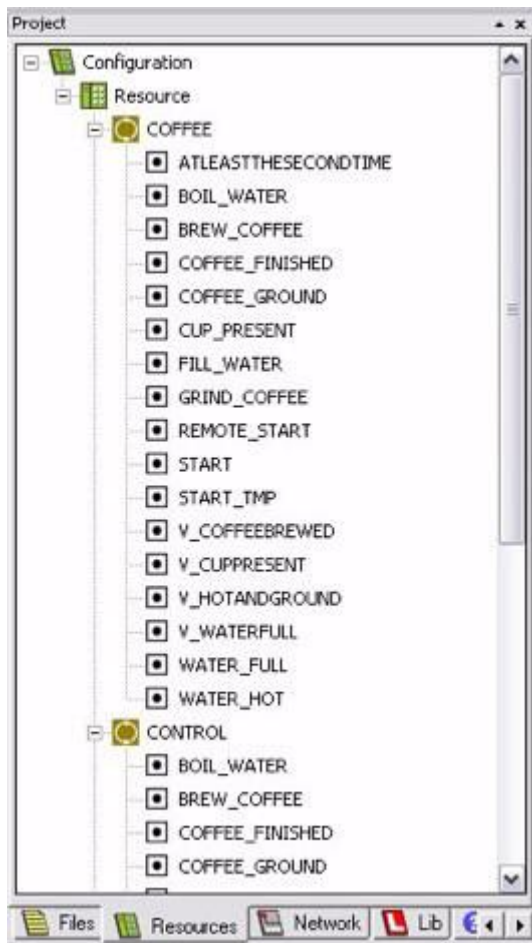
```

Instancepath	Name	Value	Type	Address	Force	Comment
COFFEE	START	TRUE	BOOL			

Another coffee cannot be brewed until the first cup is done.

1.5.4 Monitoring Code

Now that your application is running, go back to the [Project window](#) in the upper left and activate the [Resource tab](#). This tab offers an overview on the resources, its tasks and variables. Click all the small plus signs to open the entire tree under the resource entry. This will reveal the "instance tree", showing all instances of programs and function blocks and all variables that you used in your program:



Double-click some of the variable entries (white boxes with black dot), and see the corresponding variables added to the watch list in the Test & Commissioning:

Instancepath	Name	Value	Type	Address	Force	Comme
CONTROL	WATERTEMP	0	INT			
CONTROL	WATER_HOT	FALSE	BOOL			
CONTROL	WATER_FULL	FALSE	BOOL			
COFFEE	BOIL_WATER	FALSE	BOOL			
COFFEE	COFFEE_FINISHED	FALSE	BOOL			
COFFEE	START	FALSE	BOOL			

Debug OPC Variables Watchlist: Resource.WL

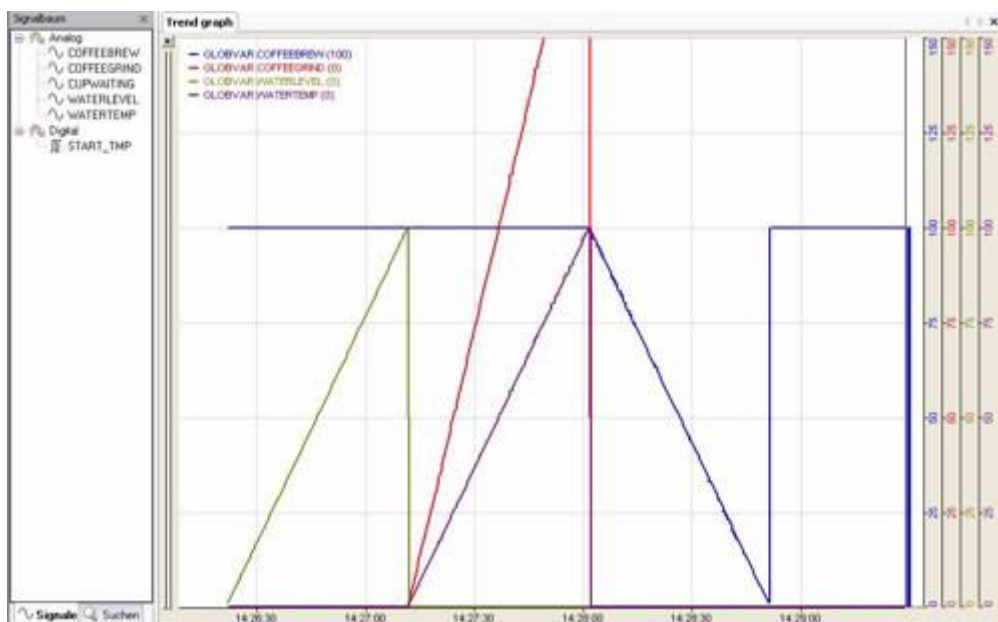
Go back to SmartSIM and modify the inputs to see the effect in the watch list.

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

Note: if SmartSIM does not stop when you set a breakpoint, you probably did not set [optimization settings](#) properly. Be sure your resource is configured for "size only".

1.5.5 Control Data Analyzer

The Control Data Analyzer hands the programmer the possibility to observe the development of variables over time. The Analyzer can be started via **View -> Control Data Analyzer**. This option is online available while being online. The screenshot illustrates the complete process of brewing a coffee. The water will be filled into the tank (yellow line), and then heated (red). Simultaneously the coffee gets grinded. When both steps are finished, the coffee gets brewed.



1.5.6 Online Edit

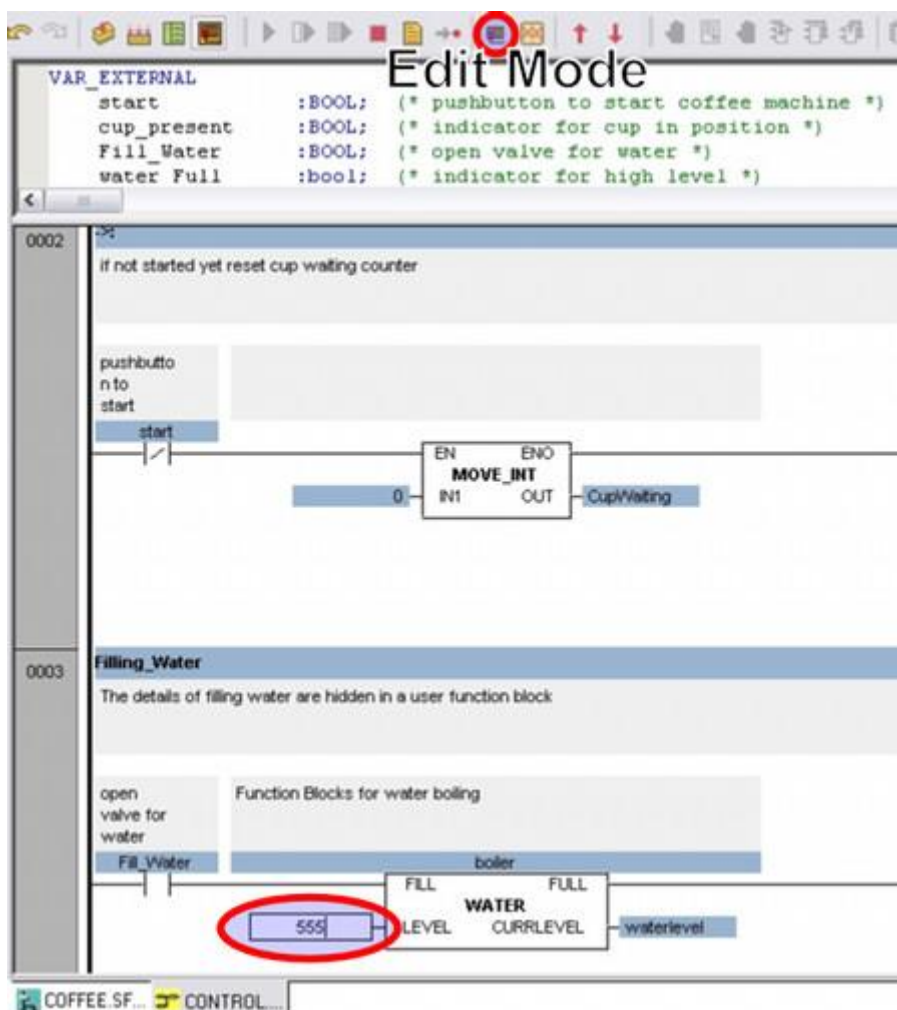
Online Edit (or Online Change) is a feature whereby program changes are applied to the PLC without the need to restart it. The following Steps need to be done in order to run Online Edit.

The program must be compiled and running on the PLC. The source is opened in an editor window.

The Editor can be switched from Monitor Mode (green colored symbol) to Edit Mode (red colored symbol) and back via **PLC->Online/Edit** or the corresponding button of the

toolbar 

Implement the desired changes and close the Edit Mode via **PLC->Online/ Edit** again. The screenshot below illustrates changes in the desired water level from 100 to 555.

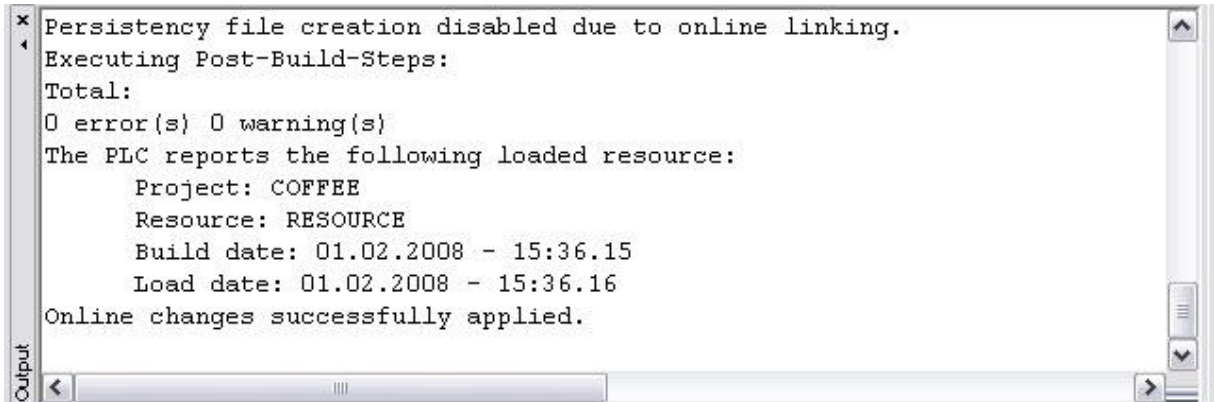


OpenPCS prints a dialogue to accept and download the changes



If the changes are accepted, OpenPCS recompiles the necessary unit and downloads them to the PLC without stopping the running cycle. The changes have bearing on the next cycle.

OpenPCS prompts a message in the output window, if the update is finished



Remark: The Changes will not be persistent on the PLC. Therefore you need SaveSystem, an optional target system feature.

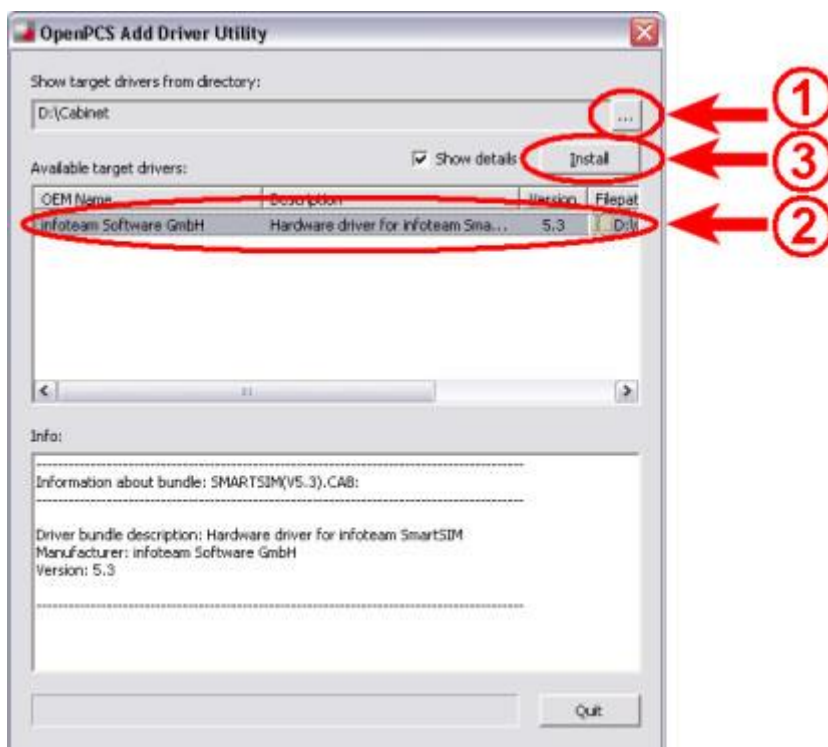
1.6 Additional

1.6.1 Adding Hardware Support

All Automation Network member companies are providing target drivers (*.cab files) for installing support for their hardware/controller. It is possible to install different target drivers for different controllers from different manufactures in one installation.

To install a target driver, select **Extras -> Tools -> Driver Install...**

First, specify the directory (e.g. X:\ for CD-ROM) for the target driver, then select the driver you want to install and thereafter click "Install".



Target drivers can contain hardware definitions, communication drivers, help files, libraries, templates for projects, resources, program files, etc. Please see the information provided with the target driver or the information provided by your PLC manufacturer.

1.6.2 Templates

OpenPCS 2006 supports file and project templates, to minimize the effort of creating solutions for specific tasks. Templates optimized to support particular PLCs can be provided

by the manufacturer of the PLC as part of a target driver. Templates can be used for resources, tasks, declarations, projects, program files, etc.

Note: Templates provided by one manufacturer may be incompatible with the hardware of a different manufacturer.

1.6.3 XML-Import/Export



OpenPCS supports the PLCOpen standard XML-Import/Export for IEC 61131-3 projects. The XML export/import provides a way to export an entire project to a single XML file, and also to import a project from a XML file.

To export a project, use "**Project->XML-Export...**". In the dialog, select a folder where to place the XML file. The filename of the XML file can not be selected, it will be the same as the name of the project. The export of the following files is supported:

1. ST
2. IL
3. SFC
4. Declarations for global variables
5. Declarations for direct global variables

The export of the following files is currently not supported:

1. Ladder
2. FBD
3. CFC
4. Files with user defined data types
5. Files for OPC variables
6. Subdirectories
7. Files in subdirectories
8. Non-OpenPCS files (e.g. PDF, DOC, ZIP, etc.)

For import, use "**Project->XML-Import...**". Select the XML file you want to import and thereafter choose a directory where to create a new project to contain the imported files.

To create a backup of your projects, it is recommended not to use the XML export function, because OpenPCS has its own backup function. Use **Project->Backup...** to create a backup. Select where to save the backup file (.BAK). You can restore a project by using **Project->Restore...** and selecting the .BAK file you want to restore.

The OpenPCS backup function saves all files within a project.

Remark: SFC files created with an OpenPCS version earlier than 5.2.0 must be resaved with a current OpenPCS version, because since 5.2.0 OpenPCS uses a XML file format for SFC files.

1.6.4 About this manual

This manual is organized in 4 main chapters:

Chapter 1 you have already read, gives a short introduction into the most common features of OpenPCS. If you have never used OpenPCS before, be sure to read this chapter.

Chapter 2 details on all OpenPCS tools, starting with all the different editors, plus the compiler and all not so visible tools. Read this to get an overview over features tool by tool.

Chapter 3 gives in-depth information on some Advanced Topics, most of these affecting more than one of the tools. Read this to get background information and an in-depth understanding of how to make best use of OpenPCS.

Chapter 4 is the Reference, find all keywords, functions of OpenPCS, all compiler messages and warnings plus many more items listed in alphabetical order here. Use this chapter to quickly locate pieces of information.

The index will help you find all the information. The user manual - printed or in electronic PDF format - has the same contents as the electronic online help. So if you can't find what you are looking for in the manual, try using the search function in the help file.

1.6.5 More Information

This is a user manual for the software OpenPCS only, not a training guide. If you need more information, we recommend to consult the installed user manual. For further reading:

1. Programming Industrial Automation Systems, by Karl-Heinz John and Michael Tiegelkamp, available in German ISBN 3-540-66445-9, English ISBN3-540-67752-6, and Chinese
2. infoteam Software, producer of OpenPCS, does offer on-site training courses on OpenPCS, IEC61131-3, Motion Control, Real-Time programming and related issues. Contact info@infoteam.de for pricing and availability.

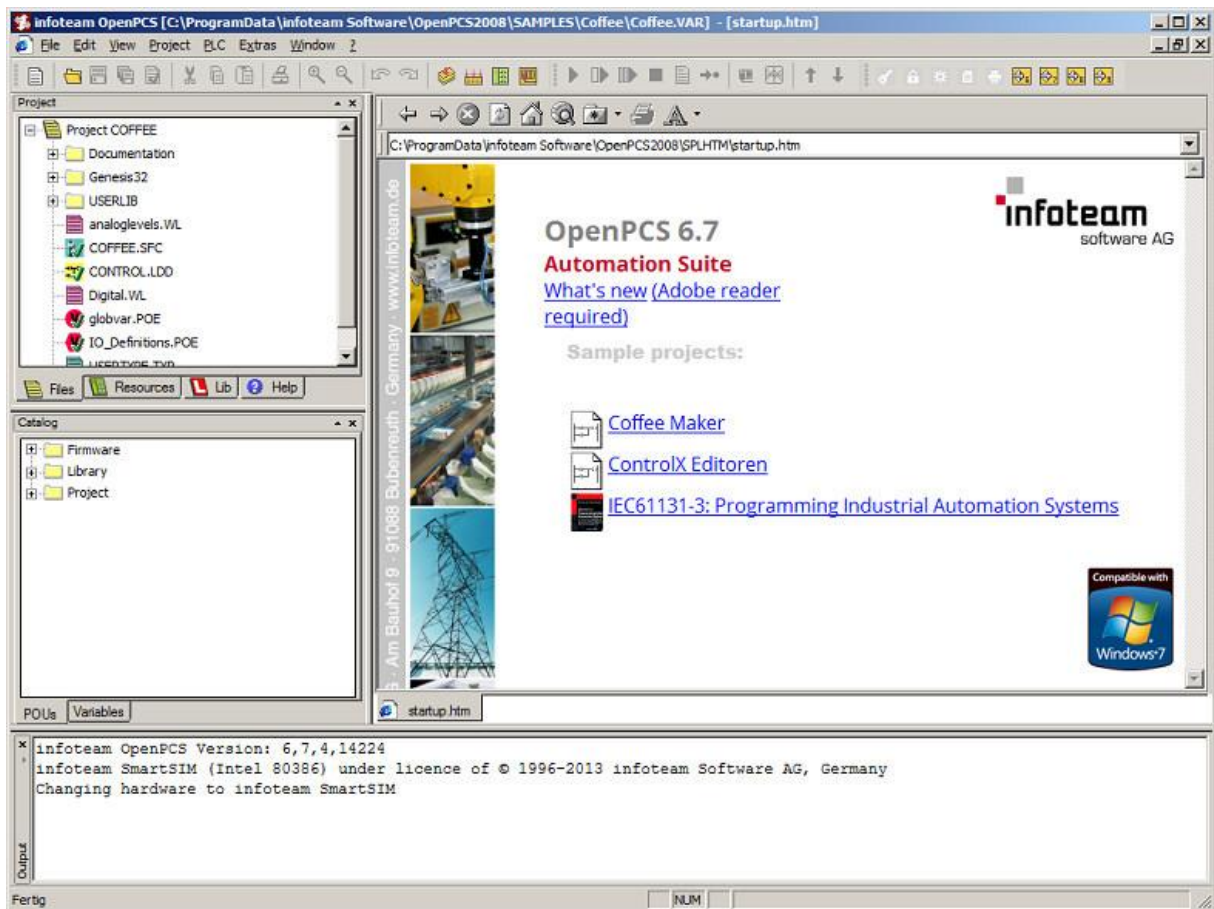
Please consider the homepage www.infoteam.de for further reading and additional downloads.

2 OpenPCS Tools

2.1 OpenPCS Framework

2.1.1 OpenPCS Framework: Introduction

The OpenPCS Framework hosts most of the tools of OpenPCS. Generally, the OpenPCS Framework will look similar to the following:



The project is shown in the [Project-Browser](#) on the left side. The editor-pane is located in the centre. Most editors will use split screen technology to edit declarations in the upper pane and instructions in the lower pane. While [declarations](#) look the same for all programming languages, instructions vary widely. The OpenPCS Framework can host many files at the same time.

Dagnostic messages will be shown in the [output window](#) at the bottom.

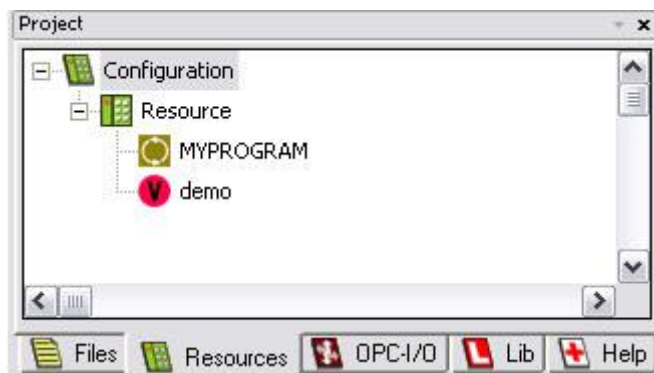
2.1.2 Output Window

The output window is located at the bottom of the [OpenPCS Framework](#) and used to display diagnostic messages.

2.2 Browser

2.2.1 Browser: Introduction

The Project-Browser is the File Manager of OpenPCS. Using the Browser, you will organize your work into files and projects. From the Browser, you will create and edit files, compile, download and monitor your application:

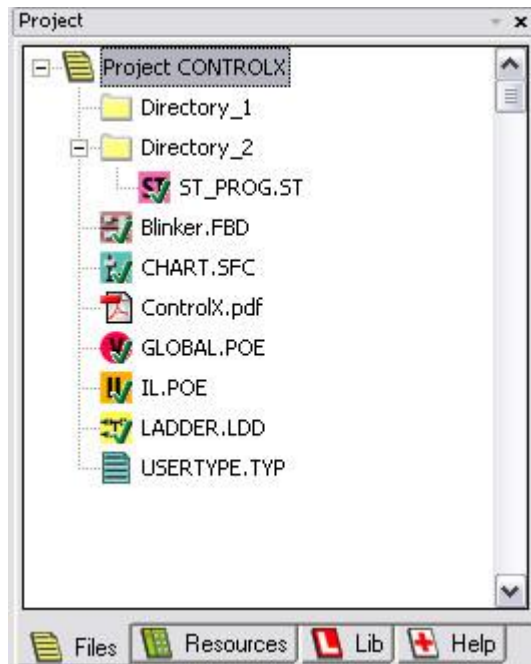


The Browser user interface consists of four different windows (panes):

1. The [File-Pane](#)
2. The [Resource-Pane](#)
3. The [OPC-I/O-Pane](#) (optional)
4. The [Library-Pane](#)
5. The [Help-Pane](#)

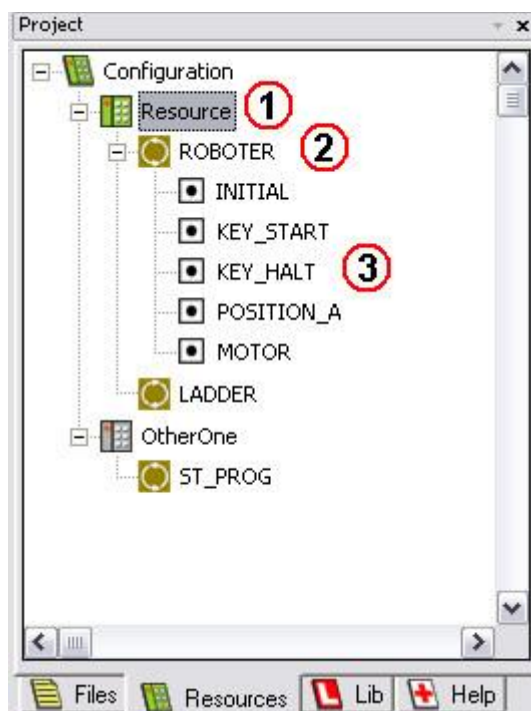
2.2.2 Browser: Overview

2.2.2.1 The File-Pane



The File-Pane contains a directory-tree with all your source files, collected under the current project (1). These are the files that you write yourself, with one of the editors of OpenPCS, or with different applications. All directories (2) and files (3) under the current project-path are shown.

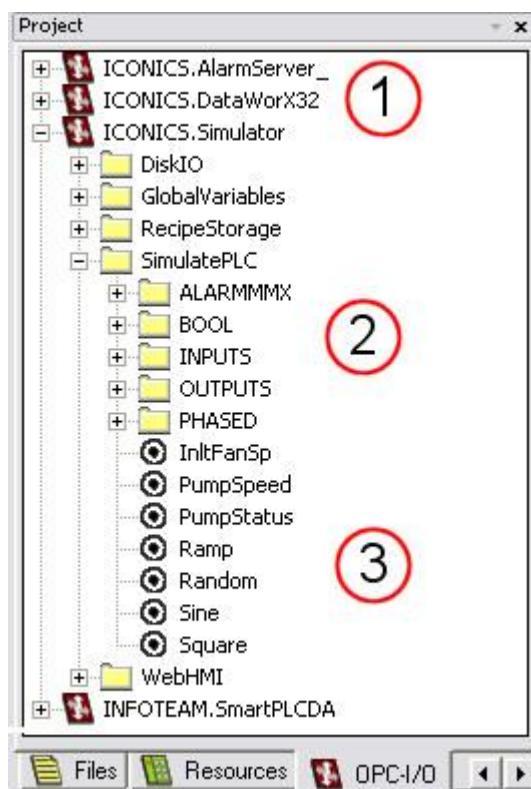
2.2.2.2 Resource-Pane



The Resource-Pane contains the instance tree, named "Configuration". It shows your controllers as resources (1), the tasks running in these controllers (2), the instances of functions and function blocks available within these, and all variables (3) defined in these. The [active resource](#) is shown with a green Button.

In the instance tree, there are only "links" to files and objects defined in the File-Pane: Tasks are referencing POU's of type PROGRAM, global variables are referencing global declaration files etc.

2.2.2.3 OPC - I/O-Pane



The OPC-I/O Pane contains the tree of locally available OPC-DA address spaces.

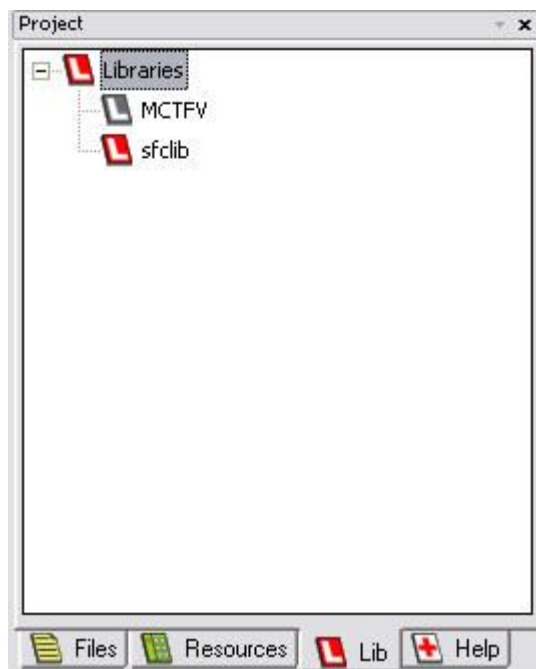
On the root level (1) it lists all the OPC-DA servers currently registered on the working PC. Underneath the root level you can find several layers of nested OPC folders (2) structuring the address space of the selected OPC-DA server. Finally, as leaves in the tree, there are the OPC-Tags (3) representing the I/O values of the OPC-DA server.

Since using the OPC-I/O Pane only makes sense for targets supporting OPC-I/O, you can enable or disable the display of this pane via the "Extras -> Browser Options..." dialog box.

Note: Non-local OPC-Servers are currently not supported. The infoteam OPC Server (infoteam.PadtOpcSvrDA) is not available in this list.

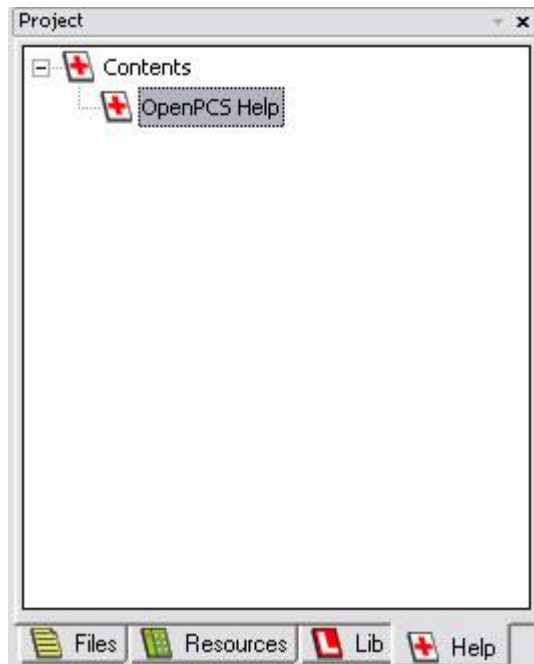
2.2.2.4 The Library-Pane

The Library-Pane (Lib) contains a tree with all installed libraries of the project. You can install new libraries with **Project -> Library->Install New...**



You can use a library in a project by selecting it and choosing **Project -> Library->Use in current project**. The libraries that are currently used in the project are shown with a red symbol.

2.2.2.5 The Help-Pane



The Help-Pane contains help-topics.

2.2.3 Projects

2.2.3.1 Create new project

If you have opened OpenPCS, you can start with the work. The first step is the creation of a new project. Select **Project-> New...**, or press the respective button in the toolbar.

Please note:

1. The name of an OpenPCS project should not contain blank (space) characters or special characters. Plus, for easy updates, it is recommended that you store your application separate from OpenPCS. To give an example, C:\PROJECTS is a good location to store your projects.
2. A subdirectory, which has the same name as your project, will be created automatically at the location you have entered. This directory contains all files which belong to your project.

2.2.3.2 Check project consistency

If you have any problems with your project, OpenPCS supplies you with the **Project->Check ...** dialog. It looks automatically for inconsistencies in the project files and tells you which are inconsistent enabling you to open them. The Project-Consistency-State-window has four buttons:

1. **Repair**: Fixes a POU name inconsistency.
2. **Open All**: Opens all files, which are currently shown in the window
3. **Open**: Opens a single file which was previously selected in the window

4. **Close:** Closes the Project-Consistency-State-window

The main reasons for inconsistency in project files are, on the one hand, obsolete POE files (e.g. the POE is older than the ST file), and on the other hand, missing POE files (to re-check for these inconsistencies, close the window and go to **Project->Check ...** again).

Opening, manually repairing the syntax and saving them again might repair them.

Note: This tool looks primary for inconsistencies and not for syntax errors.

2.2.3.3 Open Project

You have three possibilities to open a project:

In the "**File**"-menu: Here you find under the item "Recent Projects" a list of the last opened projects; the file, which you are looking for, could be contained in this list.

By the toolbar: Click on the button "**Open Project**".

By the menu: Click on "**Project ->Open...**" in the main menu.

Select the desired project in the dialogbox or look for it in the folders. The project files have the suffix ".var".

2.2.3.4 Import/Export

It is possible to export your project to a PLCopen standard file with **Project->Export** and to import one with **Project->Import**.

Note: The current version of OpenPCS supports only exporting/importing of POE, ST, SFC and MAK-files and the \$ENV\$ folder.

2.2.3.5 Search within project

All files, which are embedded in the current project, can be searched via **Project->Search in Files....** The result is given in the output window.

Double-clicking a result will open the document in the editor at the according position.

2.2.3.6 Refresh project information

Project -> Refresh project information refreshes the project information and writes the project internal newly. Thus e.g., prototypes are newly read in and libraries are re-freshed.

2.2.4 Files

2.2.4.1 Creating new files

You create files with OpenPCS from within the OpenPCS Framework. Select **File-> New** to see the many choices:

POU for programs, function blocks and functions, the basic code blocks defined by IEC61131-3. For each of these, you have the choice between the programming languages that come with OpenPCS, as far as appropriate.

Declarations for creating resource global, direct global, type and OPC variable declaration files.

Resources for creating new Resources

In **Projects** you find template projects with sample configurations.

In **Other** you find folders, GWX-files and watchlists.

Note: It is possible to find some other template files (or structure) there, depending on your OEM manufacturer.

2.2.4.2 File Operations

With the **File->File** submenu you are able to

1. Move a file to another directory
2. Copy a file
3. Rename a file
4. [Import](#) a file from another project/location
5. Export a file to another project/location

Note: The action belongs to the file selected in the browser.

2.2.5 Resources and Tasks

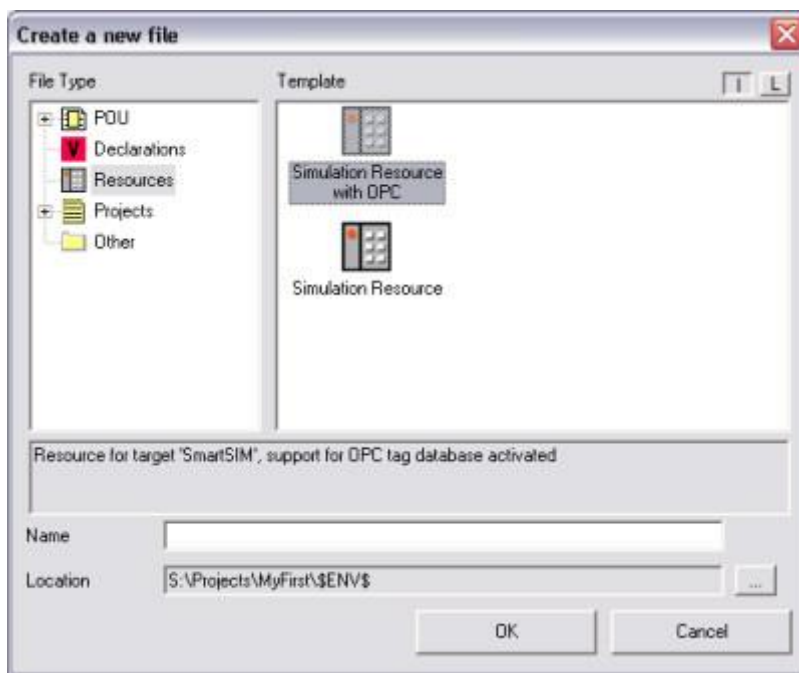
2.2.5.1 Resources: introduction

In general, a resource is equivalent to a PLC or a micro controller. A resource definition consists of a **name** for identification, the **hardware description**, i.e. Information about the properties of your PLC which will be used by OpenPCS, and a **connection name**, i.e. Information about the kind of communication between OpenPCS and the control system.

A resource maintains a list of tasks which are to be run on the control system.

2.2.5.2 Create resource

When creating a new project, OpenPCS will define one resource. If you want to create additional resources click on **File-> New...** In the following dialog-box go to "Others" and choose "Resource".



Press OK and the new resource will appear in the Resource-Pane.

2.2.5.3 Edit resource

To edit a resource, right-click on it and choose "Properties" in the context-menu.

A dialog-box will open, where you can change the following properties:

Under "Hardware Module", select the configuration file corresponding to the controller you are using. This configuration file should have been provided by the vendor of your controller. To use the windows simulation SmartSIM, use "SmartSIM".

Under "Network Connection", select the communication connection to connect to your target. Use **PLC-> Connections** to define new connections or to see or modify the properties of connections defined. The network connection is pre-selected as "Simulation" to work with the PLC-Simulation of OpenPCS.

Check "Enable Upload" to pack the sources of your application onto the target. This is helpful if at the end of debugging you want to save the project on the controller for later use by other service personnel.

"Generate Mapfile": after generating the code three text files will be created in which you find linker information. These files will be saved in the resource directory named "Pcedata.txt", "PceVars.txt" and "PceSegs.txt". Some other features of OpenPCS (GetVarAddr) need this feature to be enabled, so you better do not disable it without good reason.

For a description of optimization settings, see [Optimisation Settings](#) in Advanced Topics.

2.2.5.4 Add Task

In general, a task is equivalent to a program plus the information how the program can be executed. The definition of a task consists of the name, the Information about the execution of the task and a POU of type PROGRAM which should be executed in this task.

To add a task, mark the program you want to create the task of, and choose **PLC-> Link to resource**.

After adding of the task, you can double-click it in the Resource-Pane to change the task specifications.

Note that the task name depends on the program name, and can't be changed. To complete the task definition, you must specify the information, how the task can be executed: Cyclic , Timer controlled , [Interrupt](#) controlled. Task type, priority and time control the execution of this task and in co-operation with other tasks. To do this, right-click on the task and choose "Properties". For more information, see [Multitasking](#).

2.2.5.5 Active Resource

With every OpenPCS project, there may be many "resources" , see the Advanced Topics section for how to best work with [multiple resources](#). However, in order to make OpenPCS more user friendly and easier to use, there is at any time exactly one active resource. In the Browser, this will be shown with a green icon.

Many user commands - like compile, go online, download etc. - implicitly use the "active resource". So even when having many resources in one project, you will not have to specify which resource to use with these commands. If you want to use a different resource than the one currently active, right-click this other resource and select "set active" from the context menu.

2.2.6 OPC - I/O

2.2.6.1 OPC - I/O: Introduction

With the [OPC-I/O browser pane](#) you can browse the tree of locally available [OPC-DA](#) address spaces and use it to assign any leaf item of the tree to a global variable definition in the [assignment editor](#).

Thus you are able to use any values supplied by an OPC-DA server as a global input or output variable within your PLC program.

2.2.6.2 About OPC

OPC means **O**LE for **P**rocess **C**ontrol and is a series of standards specifications created by the OPC Foundation.

OpenPCS currently supports the Data Access Specification (OPC-DA) Version 2.0

For further information on OPC, please consult the web page of the OPC Foundation at

<http://www.opcfoundation.org>!Inet("http://www.opcfoundation.org")

2.2.7 Compiler

2.2.7.1 Build active resource

Build only those parts of your resource that have changed since last build due to modifications. Invoked by **PLC->Build active resource**.

OpenPCS will automatically build anything as necessary when going online, but it is good practice to recompile from time to time when programming to detect errors as early as possible.

2.2.7.2 Rebuild active resource

To rebuild all tasks of your active resource choose **PLC->Rebuild active resource** from the menu. This will completely recompile all parts of the active resource.

2.2.7.3 Rebuild all resources

Like "[Rebuild active resource](#)" but will rebuild all - active and inactive - resources.

2.2.8 Online

2.2.8.1 Going Online

To get into online mode, either double-click the resource you want to go online with, choose **PLC-> Online** or press the "go-online" button in the toolbar to go online with the active resource.

Repeat this to go offline again.

2.2.8.2 Download

OpenPCS will automatically prompt whenever a download seems necessary. If you like, you can at any time invoke a download yourself by using **PLC-> PC->PLC (Download)**.

it is possible to download the project during runtime (without stopping the PLC), If the current project already is on the resource. But so the download is not persistent. This mostly matches the [Online Edit](#) functionality.

If your PLC supports the Save System feature, we offer you the ability to persistently download your project without stopping the PLC.

Note: The usual download mode, stopping the PLC, is persistent anyway.

2.2.8.3 Watching variables

To add variables to the watch list of the Test&Commissioning, open the resource tree of your application and double-click any of the variables:



2.2.8.4 Starting Online Editor

Going online opens all your open POUs in online mode. To start the Editor in online mode for specific function block instances of your application, open the resource tree and locate the instance that you wish to monitor and double-click it.

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

Note:

Do not confuse an instance of your code (located under "Configuration" in the Browser) with the source code of that block, located under "Project Files" in the Browser.

2.2.8.5 Hardware information

This menu is only available in the online-mode. You get information about the used hardware.

Mark the active resource and select the menu item **PLC->PLC info**.

2.2.8.6 Resource information

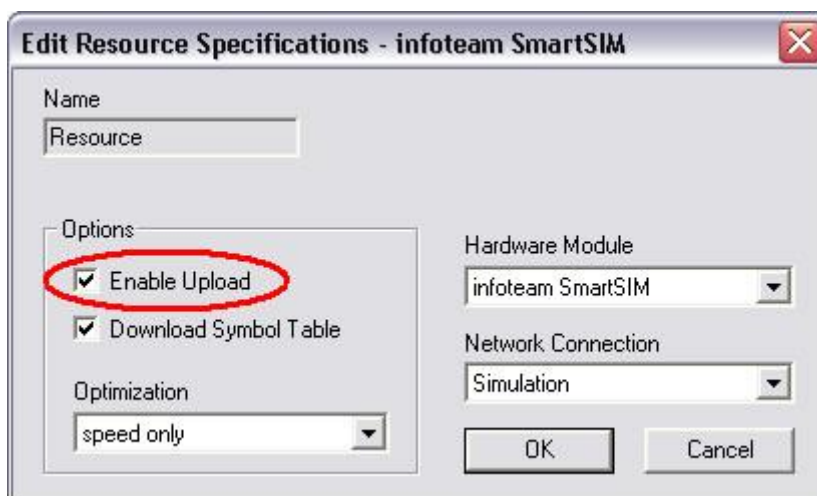
This menu is only available in the online-mode. The project name, the resource name, and the version number (which is internal created and assigned to a specific compilation) are displayed.

You can display the resource info by marking the resource and selecting the menu item **PLC->Resource info**.

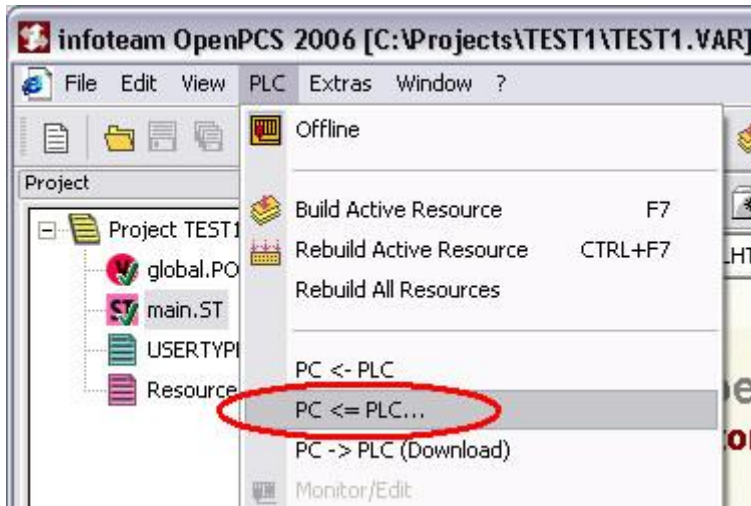
2.2.8.7 Upload

OpenPCS supports uploading of projects from your controller to your PC. Therefore, it is not necessary to have the source code of your project when updating your PLC, because you can upload the project.

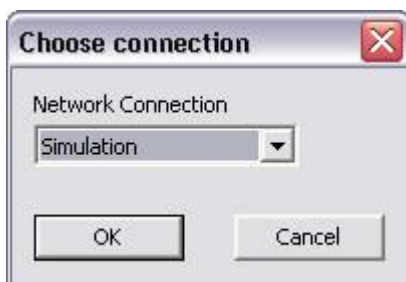
To enable this feature, the "enable upload" box has to be checked in the resource properties before compiling and downloading a resource to the PLC as shown in the figure below:



For uploading the project, make sure that the resource properties are set as described above, and not connect to any PLC. Then go to **PLC-> PC<= PLC**.



Now you have to choose the connection to the PLC for uploading the project.



After that, you will be asked, where to save the project (make sure that the project doesn't exist already):



The uploaded project will be opened automatically.

2.2.8.8 Erase

This is only available in online mode. To remove the entire program from the PLC select **PLC->Erase** from the menu or click the corresponding button in the toolbar.

Note: The exact reaction varies depending on the implementation of the OEM manufacturers of your PLC. If you want to know more about it, ask them.

2.2.9 Other Browser Features

2.2.9.1 Resource global variables

In OpenPCS, there are two kinds of global resource variables:

Global variables: these are variables without hardware-addresses, e.g. for intermediate results.

Direct global variables: these are variables with direct hardware-addresses together with the IO-declarations. These represent the interface to the hardware.

To create a new file with resource global variables, select **File->New->Declarations->Global** or **File->New->Declarations->Direct Global**. Edit these files, and link them to the resources you want to use them with.

2.2.9.2 Type definitions

By default, there is a file to hold user defined data types (`usertype.typ`) with each OpenPCS project. To have your own data types, edit this file or create respective files of your own. To use those data types with any resource, add the file to the respective resource.

2.2.9.3 Add files

OpenPCS allows to add any kind of file to OpenPCS projects. Use **File->File->Import...** and select the file of your choice. Beside files you have written with the editors of OpenPCS (IL, LD, FBD, ST, CFC, SFC) it is possible to import type definition and type declaration files as well as resources. Further more it is practicable to register files in one project, even if they were created by other programs, for example by: Microsoft Word, Microsoft Excel, Microsoft Project, AutoCAD.

Select the desired file type in the popup menu and open the corresponding directory. There you can select the file you want to copy. A multiple selection is possible when you keep pressing the left mouse button, the "Shift"- or the "Ctrl"-key. This files will be copied in the current directory of the browser and can be edited by a double click.

2.2.9.4 Browser Options

With **Extras->Options->Browser**, set Browser Options:

General:

"Show only OpenPCS file types": If the checkbox is filled, only OpenPCS file types are shown by the browser. All other files are hidden except of the types listed in "Additional file types".

"Don't show SFC variables" If the checkbox is filled, all variables with "_" as the first character are hidden in the resource tree.

"Don't show Tab captions" If the checkbox is filled, the tabs of the browser panes have only an icon and no caption.

Extended Settings:

"Enable OPC Browserpane" This checkbox is used to hide or show the [OPC Browserpane](#).

"Enable OPC Configuration Compiler" This checkbox is used to generate automatically the [OPC database](#).

"Enable GWX Embedded in OpenPCS" This checkbox is used to enable opening GWX files in the OpenPCS Framework.

"Save Watchlist" This checkbox is used to enable auto saving of the watchlist when going offline

Compiler:

"Compiler output level" This drop down menu is used to limit the amount of compiler error messages (for most details use 0).

"Warn if less than given percentage of PLC memory is left for resource" This checkbox is used to enable warnings about low memory of the PLC with the current resource. The minimal free memory can be set as percentage.

Note: this option is hardware specific. It depends on support of OEM.

"Warn if less than given percentage of max. segment size is left" This checkbox is used to enable warnings about large segments in the current resource. The minimal free memory in segments can be set as percentage.

2.2.9.5 CFC/FBD Options

Set the view options of the CFC and FBD editor via **Extras->Options->CFC Editor**. Changes show effect after new opening of a CFC or FBD file.

Block width in %: changes the width of a function block. 100 corresponds to the default width. The width may be set between 1 and 1000.

Margin width in %: changes the width of the margin. 100 corresponds to the default width. The width may be set between 1 and 1000.

2.2.9.6 Setting fonts and color

With the **Extras->Font/Color** dialog it is possible to set the appearance of the text editors used by the IL, ST, SFC, Ladder and FBD editors.

Note: The current version of OpenPCS provides only setting the font and size.

2.2.9.7 Custom Tools

The Browser provides one button in its toolbar to start an OEM specific tool.



OEMs of OpenPCS can configure OpenPCS to start any particular tool with this button. By default, the licence Editor will be launched.

2.2.9.8 Exclude from Project

It is possible to exclude subfolders from the project. Excluded folders are ignored in your project. You cannot navigate in excluded folders. This can be used e.g. to exclude sub-version folder or documentation folder etc.

To exclude a folder from your project, choose **Exclude from Project** in the context menu of the corresponding folder.

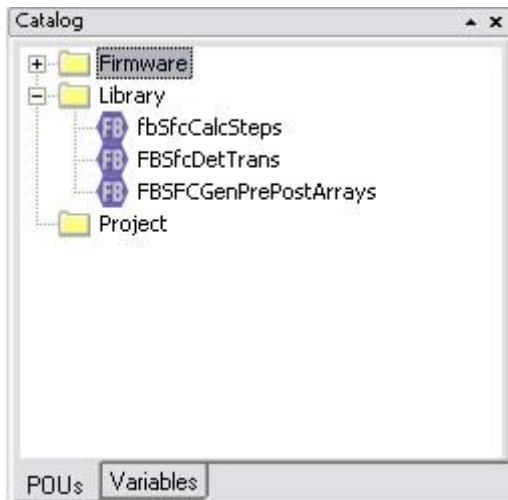
2.3 Catalog

2.3.1 Catalog

The Catalog is a tool to insert function blocks to your programs. The Catalog is visible below the project browser. If it is not there, go to **View->Catalog**.

With the catalog, you can insert function blocks to your programs by using drag'n drop.

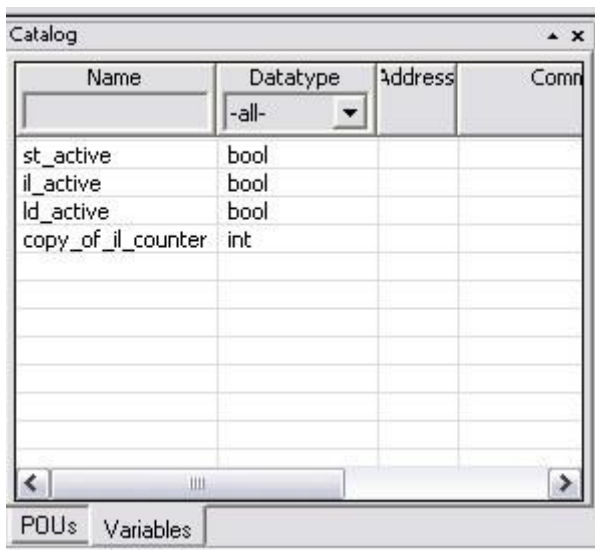
A double-click on an entry within the table opens the help on the function block.



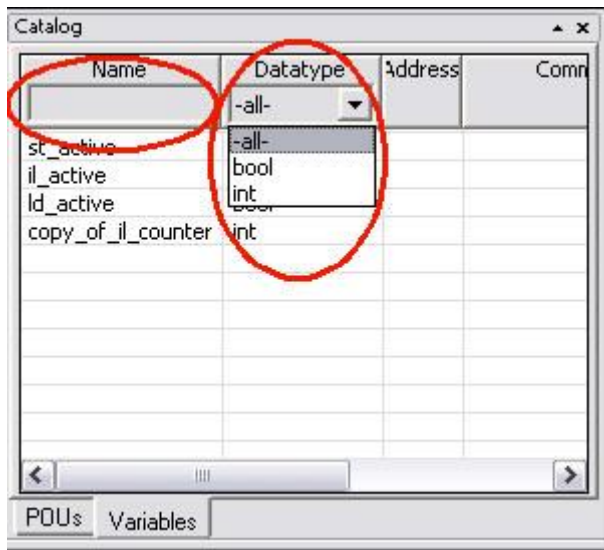
Using the Catalog, you don't have to write the names or go through the menus to insert a function block

2.3.2 Variable Catalog

The Variable Catalog is part of the Catalog. All global variables are shown in the Variable_Catalog. You can see their names, data types, addresses, comments (if available) and their scopes. At the moment the used flag is only supported by the CFC-Editor.



The Variable_Catalog enables you to insert global variables to your program by drag'n drop and also to filter global variables. When using drag'n drop, typos will not happen any longer. You can filter names, data types and also scopes, to see which variables are available and to get known the data type without looking for the definition.



Just insert the name and you will see all variables that fit to your input. You can also use asterisks (e.g. write "*A*" to the name field and you will get all variables which have an "A" in their names) and also use a combined filtering: First enter a name and then change the datatype.

When you create new global variables, they will not automatically be shown after saving the global variables file. Use a right-click into the variable grid and select refresh to update the Variable_Catalog.

2.4 Declaration Editor

2.4.1 Declaration Editor: introduction

The declaration editor is hosted by the [OpenPCS framework](#). Enter declarations as defined by IEC61131 here.

IEC61131-3 requires all data objects to be declared as variables. A set of different [declaration sections](#) is available to define variables on different scopes. IEC61131-3 comes with a set of predefined data-types, the so called [elementary data types](#). And, there are some means to define user-defined, so called [derived data types](#), using [structures](#), [arrays](#) and enumerations.

With most variables, storage is assigned by the compiler, without any programmer activity. For inputs, outputs, markers and potentially more types of variables, the programmer may specify a memory location, using [directly represented variables](#).

Declarations are entered in text-form just as defined by IEC61131-3.

2.4.2 Declaration Sections

Variables are declared in different sections of variables, so-called declaration blocks. A declaration block starts with a keyword and ends with END_VAR (e.g., VAR_GLOBAL ... END_VAR).

VAR_INPUT: If a variable block should only be read inside a POU, you must declare this variable as input-variable. It thereby isn't allowed to modify this variable in this POU. An input-variable can be used for the parameter transfer in a function or function block.

VAR_IN_OUT: An input-/ output-variable is accessed under the same name by a function block. The variable gets a reference (pointer) to the transferred variable and its memory location during the parameter transfer by the block-call. Because a write-operation has a direct effect to the content of an In_Out-variable, it isn't allowed to use a write-protected type for the transferred variable as INPUT-variables or variables with attribute CONSTANT.

VAR_OUTPUT: The Output-variables are declared in the function block that use them for the return of values. The calling POU can access them.

VAR_GLOBAL: A variable should be declared as global variable in the POU "program" if this variable should be valid in this POU and in the function blocks called by this POU. This variable must be declared as external variable (VAR_EXTERNAL) in all function blocks which intend to use this variable.

VAR_EXTERNAL: If a declared global variable will be used inside a function block, this variable must be declared as external variable inside this function block.

VAR: A local variable is only valid inside the POU in which it was declared. The declaration of local variables can be supplemented by the attributes "RETAIN" or "CONSTANT", or by an [address](#).

TYPE : The keyword "TYPE" is used for declaration of user defined (derived) [data types](#) with local scope in the POU-types "program" and "function block", or with global scope in the type definitions.

According to the POU-type only certain variable-sections can be used:

A POU of type Program may use Type, Local, Global and External

A POU of type Function block may contain Type, Input, Output, In_Out, Local and External

A POU of type Function max use Type, Input and Local.

CONSTANT may be used as a modifier to the keyword (e.g. VAR_GLOBAL CONSTANT) to declare all variables declared in this section as not to be modified by the application. The

compiler will issue a warning if such a variable is used in a context where it will or could be modified.

RETAIN may be used as a modifier to the keyword (e.g. VAR RETAIN) to declare all variables in this section as retentive, i.e. these variables will not be re-initialized on hot- or warm-start. If the target system supports retentive memory, this will result in the variable keeping their values over power failures.

OPC: The var qualifier OPC allows a user, to mark dedicated variables, to become part of the variable table, already within the declaration editor of OpenPCS. It can be used within the declaration of the following sections:

VAR (local variables)

VAR_GLOBAL (resource and task global variables)

VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT (input and output variables)

VAR_EXTERNAL (external variables)

Hereby these sections can be part of a **program, function block** or **function**.

Supported are the following data types of variables:

USINT, UINT, UDINT, SINT, INT, DIN, BOOL, BYTE, WORD, DWORD, REAL, LREAL
STRUCT, ARRAY

Not allowed are declarations of instances of function blocks (i.e. InstanceFB1 : FB1;).

The var qualifier "OPC" can be mixed with the other var qualifiers "CONSTANT" and "RETAIN". This allows declarations like:

VAR CONSTANT OPC

var1 : INT;

END_VAR

Supported is the following syntax:

CONSTANT OPC

OPC CONSTANT

RETAIN OPC

OPC RETAIN

Not allowed, since "RETAIN" and "CONSTANT" cannot be mixed, are sequences like "CONSTANT RETAIN OPC".

The variable table provides symbol information in the runtime system. This is used by the OPC server SmartLINK to provide OPC tags.

2.4.3 Structure of a Declaration Line

A declaration line has the following form, where optional parts are set in [square] brackets, and expressions are set between <sharp> brackets:

```
<variable name> [AT <Address>]: <Type> [:= <Initial value>]; [( * <Comment> * )]
```

First the variable name is given, followed by a colon. Behind the colon is the type, and eventually the hardware address introduced by the attribute "AT". Should the variable have a definite value on start, this value will be given after a ":= ". A line ends always with a semicolon (;). The line can be commented, and comments are set between (* and *).

Example:

```
Expvariable1 AT %I0.0: BOOL; (* variable of type BOOL at the address %I0.0 *)
```

```
Expvariable2 : BOOL := TRUE; (* variable of type BOOL with the start value TRUE *)
```

An exception is the direct address without variable names (these variables will be referenced by the address):

```
AT <Address> : <Type> [:= <Initial value>]; [( * <Comment> * )]
```

In this case the variable name is omitted, therefore the address statement is not optional.

Example:

```
AT %I0.0 : BOOL (* At the address %I0.0 is a data of type BOOL *)
```

The second way of addressing should be avoided for the sake of clarity, because the meaning of the variable relates to the variable name mostly. This is important if other people should read or edit this POU.

Some Examples:

```
Variable with no initial value: InterMedSum : INT;
```

```
Variable with initial value: Pieces : INT := 5;
```

Directly represented variable without name and with no initial value: AT %Q0.0 : BOOL;

Directly represented variable with name and with no initial value: Valve AT %Q0.2 :
BOOL;

Example function block: Counter1 : CTU;

Note:

1. Initial Values can only be given as literals. It is not possible to use other variables to initialize variables during declaration.
2. The significant length of a variable name is 64
3. It is not possible to initialize variables in process image

2.4.4 Elementary Data Types

keyword	name	range	size in bits
BOOL	"Boolean"	0 (FALSE), 1 (TRUE)	1 or 8
SINT	"Short Integer"	-128 to +127	8
USINT	"Unsigned Short Integer"	0 to 255	8
INT	"Integer"	-32 768 to +32 767	16
DINT	"Double Integer"	-2.147.483.648 to +2.147.483.647	32
UINT	"Unsigned Integer"	0 to 65 535	16
UDINT	"Unsigned Double Integer"	0 to 4.294.967.295	32
REAL	"Real number"	+/-3.4E+/-38	32
LREAL	"Long real number"	+/-1.8E+/-308	64
TIME	"Time duration"	t#-596h31m23s648ms to t#596h31m23s647ms	32
DATE	"Day, Month, Year (only)"	d#0001-01-01 to d#11759222-01-20	32

TIME_OF_DAY	"Time of day (only)" tod#00h00m00s000ms to tod#23h59m59s999ms	32
DATE_AND_TIME	"Date and Time"	64
STRING	"Character String"	length of string plus 2 bytes
WSTRING	"2-byte-character String"	length of wstring plus 2 bytes
BYTE	"Sequence of 8 bits" 0 to 255	8
WORD	"Sequence of 16 bits" 0 to 65535	16
DWORD	"Sequence of 32 bits" 0 to 4294967295	32

See also [Constants](#)

2.4.5 Directly represented variables

Directly represented variables are those variables that are mapped to a certain input, output or memory address specified by the programmer. The keyword AT is used to declare this, and the address is specified in a string starting with a percent sign (%).

Example: directly represented variables

Declaration of a directly represented variable with and without a symbolic name

```
PROGRAM dirvar1
```

```
VAR
```

```
AT %I0.0 : BOOL;
```

```
MyInput_1 AT %I0.1 : BOOL;
```

```
MyResult : BOOL;
```

```
END_VAR
```

```
LD MyInput_1
```

```
AND %I0.0
```

ST MyResult

END_PROGRAM

It is strongly recommended to use symbolic names for directly represented variables, as this eases rewiring to different addresses. Changing address I0.0 without usage of symbolic names means that you have to do the change in both sections, declaration and program. With usage of a symbolic name, here "MyInput_1" you just change the address e.g. to "I0.1" within the section of declaration.

Note: Directly represented variables may only be defined in POUs of type "program".

OpenPCS does not support the mapping to a physical PLC address (using AT%) for variables of types ARRAY, STRUCT and STRING.

If directly represented variables are declared global within the program POU, they may be used as external variables within an invoked function block. An alternative is to pass the variables to the function block as VAR_IN_OUT parameters. This is possible however for PCS outputs, markers and communication files RD, SD.

OpenPCS supports the following directly representable addresses:

I: Digital-Input

Q: Digital-Output

M: Marker

Which of these addresses are available depends on the hardware of the project.

As "size", these symbols may be used:

X, or nothing: (Bit), Size=1 Bit; Example: %IX0.0 or %I0.0

B (Byte), Size=8 Bit; Example: %IB0.0

W (Word): Size=16 Bit; Example: %QW0.0

D (Double Word) Size=32 Bit; Example: %ID0.4

L (Long Word) Size=64 Bit; Example: %IL0.0

2.4.6 Derived data types

Derived data types are defined by the manufacturer of your controller, or by yourself. These new data types are defined using keywords [TYPE](#) ... [END_TYPE](#) based on the elementary data types. After definition, they may be used just like predefined or elementary data types.

Example: Derived data types

In the following sample code, a new data type is defined to represent a "Pressure" value

```
TYPE
  Pressure : INT;
END_TYPE

VAR
  PreValvePressure: Pressure;
END_VAR
```

It is possible to combine different data types in a derived data type. [Arrays](#) and [structs](#) can be integrated as well. The following example defines a struct A. The struct itself consists of another struct called B and an integer array of size 5. Three new data types are derived within B: Stationname as string and Value1, Value2 as doubles.

```
TYPE
  A :
  STRUCT
    B :
    STRUCT
      Stationname : STRING
      Value1 : DOUBLE
      Value2 : DOUBLE
    END_STRUCT
    Arr_5_INT:ARRAY [1..5] OF INT;
  END_STRUCT
END_TYPE
```

```
VAR
```

```
Data1: A;
```

```
END_VAR
```

2.4.7 Declaration of array data types

Arrays contain multiple elements of the same data type. The keyword ARRAY is used to define an array. Each element of an array can be an elementary variable.

Example: Array data type

Type Arr1 will hold five elements of type INT

```
PROGRAM feld
TYPE
Arr_5_INT:ARRAY [1..5] OF INT;
END_TYPE
```

```
VAR
Arr1 : Arr_5_INT;
END_VAR
```

```
.
```

```
END_PROGRAM
```

2.4.8 Declaration of structured data types

A structure holds multiple elements of same or different data types, elementary. Keyword STRUCT is used to define a structure. The individual elements of a structure are called members of that structure, and are accessed by writing the structure, followed by a dot and the name of the member.

Example: Structured data type

```
PROGRAM struktur
```

```
TYPE
RobotArm :
STRUCT
Angle_1 : REAL;
```

```
Angle_2 : REAL;
```

```
Grip: BOOL;

Length: INT;

END_STRUCT;
END_TYPE

VAR
Robot1 : RobotArm;

Robot2: RobotArm;

END_VAR

LD Robot1.Grip

.
.
END_PROGRAM
```

2.4.9 Declaration of enumeration data types

A variable of an enumerated data type can take any one of a fixed list of values. The list of legal values is listed in the declaration of the enumeration data type, separated by commas. An initial value may be given after the closing ")"; if no initial value is given, the first value will be the default.

Example: Enumeration data type

Data type TrafficLight can be "red", "yellow" or "green". "Yellow" shall be the default.

```
TYPE TrafficLight:
(red,
yellow,
green):= yellow;
END_TYPE

VAR
MainRoad : TrafficLight;
CrossRoad : TrafficLight;

StopCar: BOOL;

END_VAR
```

In the instruction part of that POU, the defined enumerated values can be used:

Example: IL

LD MainRoad
EQ red
ST StopCar

2.5 Assignment Editor

2.5.1 Assignment Editor: Introduction

The Assignment Editor is hosted in the [OpenPCS framework](#). It is displayed as a document editor window.

	Name	IEC Type	Address	OPC Type	R/O	Comment
1	VarRampOppRO	INT	KONICS.Simulator/SimulatePLC.Ramp	INT	<input checked="" type="checkbox"/>	Ramp
2	VarBoolOpc	BOOL	KONICS.Simulator/SimulatePLC.OUTPUTS.BIT	BIT	<input checked="" type="checkbox"/>	bit-value
3	VarIntOpc	INT	KONICS.Simulator/SimulatePLC.OUTPUTS.INT	INT	<input checked="" type="checkbox"/>	integer
4	VarFloatOpc	REAL	KONICS.Simulator/SimulatePLC.OUTPUTS.FLOAT	FLOAT	<input checked="" type="checkbox"/>	floating point
5	IntFanSp	WORD	KONICS.Simulator/SimulatePLC.IntFanSp	not supp	<input checked="" type="checkbox"/>	comment
6	PumpSpeed	WORD	KONICS.Simulator/SimulatePLC.PumpSpeed	not supp	<input checked="" type="checkbox"/>	
7	PumpStatus	WORD	KONICS.Simulator/SimulatePLC.PumpStatus	BIT	<input checked="" type="checkbox"/>	
8	Random	BYTE	KONICS.Simulator/SimulatePLC.Random	not supp	<input checked="" type="checkbox"/>	
9	Sine	BYTE	KONICS.Simulator/SimulatePLC.Sine	not supp	<input checked="" type="checkbox"/>	
10	Square	INT	KONICS.Simulator/SimulatePLC.Square	not supp	<input checked="" type="checkbox"/>	
11	BeltAlarm	BOOL	KONICS.Simulator/SimulatePLC.ALARMMMX.BeltAlarm	BIT	<input checked="" type="checkbox"/>	
12	CoreTempAI	BOOL	KONICS.Simulator/SimulatePLC.ALARMMMX.CoreTempAI	BIT	<input checked="" type="checkbox"/>	
13	CoreTempAlarm	REAL	KONICS.Simulator/SimulatePLC.ALARMMMX.CoreTempAlarm	BIT	<input checked="" type="checkbox"/>	
14	FlowAlarm	INT	KONICS.Simulator/SimulatePLC.ALARMMMX.FlowAlarm	BIT	<input checked="" type="checkbox"/>	alarm
15	InternalPr	INT	KONICS.Simulator/SimulatePLC.ALARMMMX.InternalPr	BIT	<input checked="" type="checkbox"/>	
16	InternalPressure	WORD	KONICS.Simulator/SimulatePLC.ALARMMMX.InternalPressureAlar	BIT	<input checked="" type="checkbox"/>	
17	RadiationL	BYTE	KONICS.Simulator/SimulatePLC.ALARMMMX.RadiationL	BIT	<input checked="" type="checkbox"/>	
18	RadiationLevelAI	BYTE	KONICS.Simulator/SimulatePLC.ALARMMMX.RadiationLevelAlarm	BIT	<input checked="" type="checkbox"/>	
19	SecurityBr	INT	KONICS.Simulator/SimulatePLC.ALARMMMX.SecurityBr	BIT	<input checked="" type="checkbox"/>	
20	SecurityBreachAI	BYTE	KONICS.Simulator/SimulatePLC.ALARMMMX.SecurityBreachAlar	BIT	<input checked="" type="checkbox"/>	
21	Structural	WORD	KONICS.Simulator/SimulatePLC.ALARMMMX.Structural	BIT	<input checked="" type="checkbox"/>	
22	StructuralIntegrity	BYTE	KONICS.Simulator/SimulatePLC.ALARMMMX.StructuralIntegrityAI	BIT	<input checked="" type="checkbox"/>	

It is used to assign global variables to I/O ports, such as a value tag on an OPC-DA server.

The Editor is realised as a grid table, each row representing a variable assignment. The meaning of the columns is described as follows:

Name Valid name of the IEC variable to be assigned

IEC-Type IEC type of the variable to be assigned.

Address Address of the external data item to be assigned. The format depends on the type of the item to be assigned. In case of OPC, this column will contain the fully qualified OPC tag name.

OPC Type OPC specific type of the tag to be assigned.

R/O When checked, indicates that the variable shall be declared as read-only. When unchecked, the variable is readable and writable.

Comment Optional textual comment on the assignment. It is just stored in the assignment document but not used by the compiler and other tools.

You can either manually enter the names of the OPC-Tags in the "Address"-Field or use the [OPC-I/O Browser pane](#) to browse the address space and assign the tags wanted using the menu items Edit -> Add tag or Edit -> Assign Tag.

Edit / Add tag (or double-click): Inserts a new declaration line into the assignment editor for the OPC tag currently selected in the browser pane. If this OPC tag is already defined within the assignment editor, the definition line is made the current line in the assignment editor.

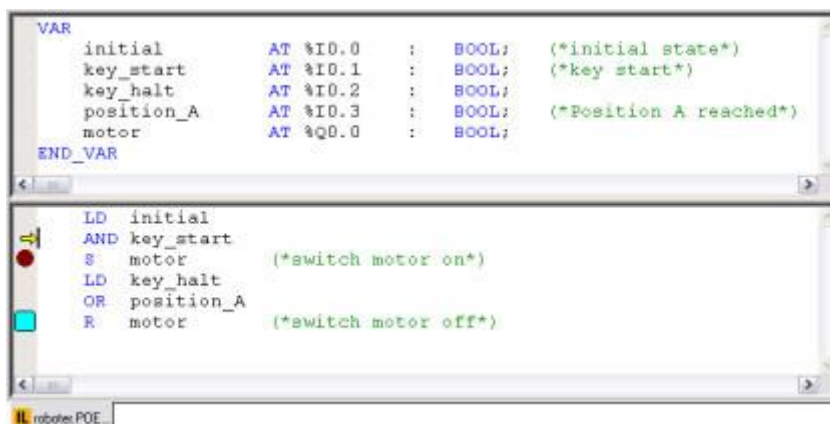
Edit / Assign tag: Assigns the tag from the browser pane to the currently selected line of the assignment editor. This works only if exactly one line is currently selected.

Using one of these functions, the OPC type of the assigned tag is automatically determined and set to the "OPC Type" column. As the default IEC variable name, the final component of the OPC tag name is used and can be changed by the user as wanted. The IEC type of the variable has to be specified manually by the user.

2.6 IL Editor

2.6.1 IL Editor: Introduction

The IL-Editor is hosted in the [OpenPCS framework](#). In the upper part of the IL-Editor, enter the [declarations](#) of the POU. In the lower pane, enter IL instructions:



```

VAR
  initial          AT %I0.0   :  BOOL;  (*initial state*)
  key_start        AT %I0.1   :  BOOL;  (*key start*)
  key_halt         AT %I0.2   :  BOOL;
  position_A      AT %I0.3   :  BOOL;  (*Position A reached*)
  motor           AT %Q0.0   :  BOOL;
END_VAR

LD  initial
AND key_start
S  motor      (*switch motor on*)
LD  key_halt
OR  position_A
R  motor      (*switch motor off*)
  
```

The IL-Editor supports bookmarks (to mark locations of interest for easy navigation while editing a file) and [Breakpoints](#).

2.6.2 Structure of Instruction List

An IL-line has the following form, when optional parts are set in [square] brackets, and expressions are set between <sharp> brackets:

```
[<Label>:] <Operator> <Operand1> [,<Operand2>,<Operand3>,...] [( * <Comment> *)]
```

At the beginning is a label if the line represents a jump target. After that an operator is placed followed by the operands and separated by commas. Comments are enclosed by (* and *).

Example:

```
Start: LD a (* Load a in the register *)
```

```
ADD b (* Add b to the register *)
```

```
ST c (* store result to variable c *)
```

A call to a function block instance is done using operator CAL and CALC respectively; the operand is the instance name, followed by arguments supplied in parentheses:

```
[<Label>:] CAL/CALC <Instance name>(
[<Input1>:=<Value1>,<Input2>:=<Value2>,...]
|
[<Variable1>:=<Output1>,<Variable2>:=<Output2>,...]
)
```

The parameter transfer consists of two parts. In the first part the parameters are transferred to the function block by setting values to the INPUT- and IN_OUT-variables respectively. The variables, which get no value, retain the value of their last call and their initial value respectively. Separated by a "|" from the first part, output parameters are specified.

2.6.3 Instructions in IL

For a list of all instructions supported in IL, please see the reference section, [Instruction List Keywords](#).

2.6.4 IL Editor Online

To debug and monitor code written in IL, use the IL Editor in monitor mode.

There are mainly three ways to debug and monitor IL code:

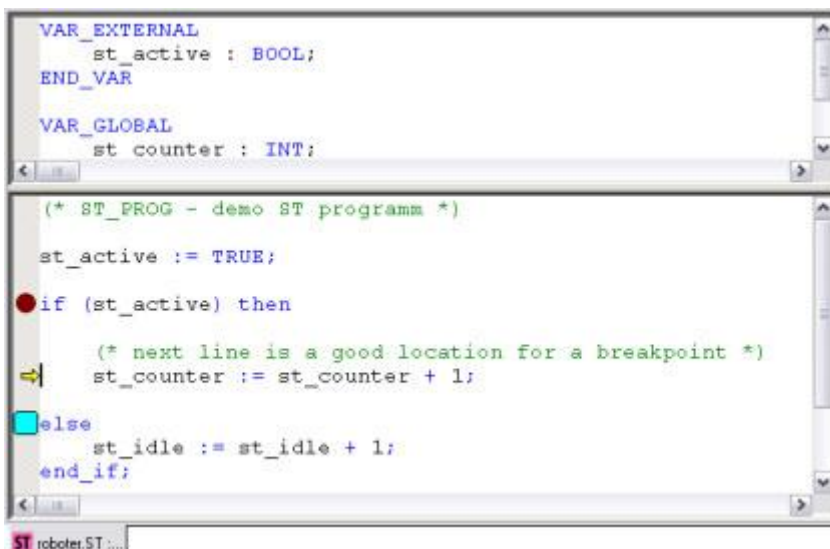
1. Use [Breakpoints](#) to stop execution, single-step through your code. Use this to understand, follow and find problems in the logic flow of the application.
2. Move the mouse cursor over a variable and see a tiny "toolbox" appear, displaying the variable's name, type and value. The value is permanently updated. Use this to quickly examine the current value of different variables within a region of your code, with or without stopping execution, at a breakpoint or while single-stepping.
3. Use the watch list in the [Test+Commissioning](#) to monitor a set of variables, which may be from any part of your applications. Use this to keep an eye on a set of variables while examining different parts of your application's code.

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.7 ST Editor

2.7.1 ST Editor: introduction

The ST-Editor is hosted in the [OpenPCS framework](#). In the upper part of the ST-Editor, enter the [declarations](#) of the POU. In the lower pane, enter ST instructions:



```
VAR_EXTERNAL
    st_active : BOOL;
END_VAR

VAR_GLOBAL
    st_counter : INT;

(* ST_PROG - demo ST programm *)
st_active := TRUE;
if (st_active) then
    (* next line is a good location for a breakpoint *)
    st_counter := st_counter + 1;
else
    st_idle := st_idle + 1;
end_if;
```

The ST Editor supports bookmarks (for marking lines of interest while editing a file) and [Breakpoints](#).

2.7.2 Instructions in ST

Code written in ST is a sequence of ST-instructions. ST-instructions are terminated with a semicolon.

Linefeeds are not significant, i.e. more than one instruction can be on one line, and one instruction can use one or more line.

For a list of all instructions supported in ST, please see the reference section, [Structured Text Keywords](#).

2.7.3 Expressions in ST

Operands known in ST are:

Literal variables, e.g. 14, "abc", t#3d_5h

Variables, e.g.: Var1, Var[2,3]

Function Call, e.g.: Max(a,b)

While operators are parts of ST-language, expressions are constructions which must be constructed by aid of ST-elements. Operators need operands to build expressions.

Parentheses	()
function call	
Exponentiation	**
Negation	-
Complement	NOT
Multiplication	*
Division	/
Modulo	MOD

Addition	+
Subtraction	-
Comparison	<, >, <=, >=
Equality	=
Inequality	<>
Boolean AND	&, AND
Boolean exclusive OR	XOR
Boolean OR	OR

2.7.4 Comments in ST

Like all modern programming languages, ST supports comments. A comment is any text included between "(" and ")", e.g.

```
(* Comments are helpful *)
```

The compiler will ignore comments when generating executable code, so your program will not accelerate in any way if you omit comments. Comments may span multiple lines, e.g.

```
(* This comment
is long and
needs more than one
line
```

```
*)
```

2.7.5 ST Editor Online

To debug and monitor code written in ST, use the ST Editor in monitor mode.

There are mainly three ways to debug and monitor ST code:

1. Use [Breakpoints](#) to stop execution, single-step through your code. Use this to understand, follow and find problems in the logic flow of the application.

2. Move the mouse cursor over a variable and see a tiny "toolbox" appear, displaying the variable's name, type and value. The value is permanently updated. Use this to quickly examine the current value of different variables within a region of your code, with or without stopping execution, at a breakpoint or while single-stepping.
3. Use the watch list in the [Test+Commissioning](#) to monitor a set of variables, which may be from any part of your applications. Use this to keep an eye on a set of variables while examining different parts of your application's code.

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.7.6 Tooltips for structs and elements of structs

It is now possible to watch the whole structure information in any depth in the ST Editor tooltips.

```
control.speed := 15;
```

```
control : my_struct
STRUCT
  is_running : BOOL;
  speed : DINT;
END_STRUCT
```

If the Editor is in the "Edit" mode, the struct and it's first level members will be shown with data type information. In the "Online" mode, the values will be shown behind the resolvable members.

```
control.speed := 15;
```

```
my_struct control
STRUCT
  BOOL is_running = FALSE
  DINT speed = 15
END_STRUCT
```

2.7.7 AutoComplete / AutoDeclare

If a variable is typed, which is not declared, and CTRL-SPACE (RETURN in Ladder editor) is used, the declaration dialog will appear.

If there is already a variable with the given name, nothing happens.

If a variable is typed, which is the first part of a declared one, the declared variable will be inserted at the given position.

If there is a couple of variables available, a list will appear and the user can navigate with UP-arrow and DOWN-arrow to the entry he wants. Pressing RETURN in that case will in-

sert the variable at the current position. Pressing ESCAPE will hide the list and return back into normal edit mode.

2.8 Ladder Diagram Editor

2.8.1 Ladder Editor: introduction

The Ladder-Editor is hosted in the [OpenPCS framework](#). In the upper part of the Ladder-Editor, enter the [declarations](#) of the POU. In the lower pane, enter Ladder instructions.

2.8.2 Ladder Logic: introduction

The basic principle of Ladder Logic is currency flow through networks. Generally, Ladder Logic is restricted to processing Boolean signals (1=True, 0=False).

A Network is restricted by so called margin connectors to the left and to the right within the Ladder Editor. The left margin connector has the logical value 1 (current). There are connections that conduct currency to elements (variables) that conduct currency to the right hand side or isolate depending on their logical state. The result of the procedure depends on the arrangement of elements and the way they are connected (AND = serial; OR = parallel).

Networks consist of the following graphical objects:

Connections (horizontal or vertical lines, and soldered points).

[Contacts](#), [Coils](#), [Control Relays](#)

[Function blocks and Functions](#)

Jumps (Graphical elements for control flow).

2.8.3 Network

The instruction section of the Ladder Diagram Editor is subdivided into so called networks, which help structuring the graphic.

A network consists of: Network label, Network comment and Network graphic.

Network label: Each network that may be a jump target from within another network will automatically be assigned a preceding alphanumerical identifier or an unsigned decimal integer. By default, networks will be numbered. This numbering of all networks will be automatically updated whenever a new network is inserted. The numbering simplifies

finding a certain network an corresponds to line numbers of textual programming languages.

Network comment: The Network Comment is represented as a square area in the ladder diagram. To enter a commentary text, double click on this square. The comment is always placed below the network label. Note that the first network additionally contains a ladder diagram comment above the network label and the network comment.

Network graphic: The network graphic consist of graphical objects, which may be graphical symbols or connections. Connections transport data between graphical symbols, which process the data at their inputs and transfer the processed data to their outputs. Note that the connections may also cross.

2.8.4 Operators

Within a ladder diagram, the term operator designates the graphical objects contact, coil and jump.

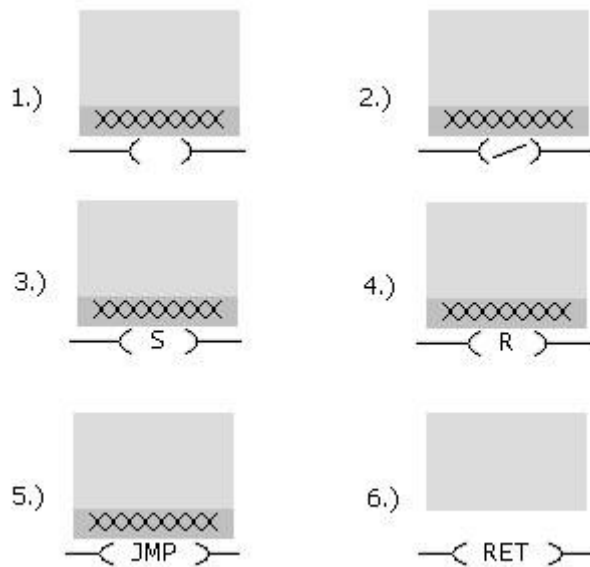
Contacts: A contact associates the value of an incoming connection with the value of an assigned variable. The kind of association depends on the type of contact. The result value will be transferred to the connection on the right hand side. There are triggers and interruptors (The Boolean value of the variable will not be changed).

Coils: Coils serve to assign values to output variables of networks. A coil copies the state of the connector on its left hand side to its connector on its right hand side without any changes. Furthermore, the coil saves a function of the state or the transition of the left connector into a Boolean variable.

Jump: Jumps manipulate the control flow of programs. They make it possible to directly invoke certain networks in a defined order. When encountering a jump operator, control flow continues at a different network. Thus, jumps are an exception from the basic principle that networks are always processed in a top down fashion.

2.8.5 Coils

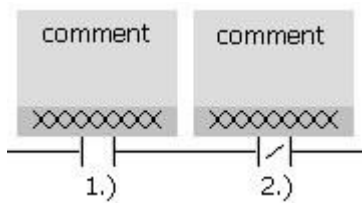
The output variable is always situated to the right hand side of the network and is connected to the right currency rail.



1. The result of the logical connection will directly be assigned to the output variable.
2. The output variable will be assigned the negation of the result of the logical connection.
3. The result of the logical connection will "permanently set" the output variable: If the result of the logical connection is "1", the output variable will be set to "1". If, however, the result of the logical connection is "0", this will have no implications.
4. The result of the logical connection will "permanently reset" the output variable: If the result of the logical connection is "1", the output variable will be set to "0". If, however, the result of the logical connection is "0", this will have no implications.
5. Jump operations manipulate control flow. With jumps, networks may be executed only if certain conditions hold. Jumps may be conditioned by a binary combination result, or unconditioned, i.e. obligatory. The jump target must always be the beginning of a network, designated by its network label.
6. Return jumps stop program execution within the current POU, and continue at the point where the POU was invoked from. Return jumps may be conditioned by a binary connection result, or unconditioned.

2.8.6 Contact

There are two contact symbols for Boolean input variables:



1. Left is the contact symbol for a variable that must have the value "1" to make the corresponding Boolean connection true. If the variable is associated with a physical address, the state "1" corresponds to a released interruptor or a pressed trigger.
2. Right is the contact symbol for a variable that must have the value "0" to make the corresponding Boolean connection true. If the variable is associated with a physical address, the state "0" corresponds to a pressed interruptor or a released trigger.

2.8.7 Control Relay

Control relays are contacts that are inserted in front of coils. Control relays may be used as breakpoints in manual execution, for example. There can always be one control relay before each coil only.

Insert-> Control Relay: Use this command to insert a control relay additional to the logical symbol.

2.8.8 Functionblocks and Functions

To insert Function Blocks or Functions to a network, click on a connection and use **Insert -> Functionblock...** or **Insert -> Function...** to insert it at this position. You can then choose the desired block or function from a list of available blocks/functions. Only predefined functions can be chosen.

Attention:

A function block: can only be added to a network if it satisfies the following criteria:

- The first input-parameter of the block has to be of type **BOOL** and has to have the name "EN". If this parameter is set to FALSE in a network, the corresponding block won't be started or even get parameters passed.
- The first output-parameter of the block has to be of type **BOOL** and has to have the name "ENO". This parameter has to be set to TRUE if the block has worked correctly and without errors.

2.8.9 Ladder Editor Online

When you have the Ladder Editor in monitor mode, it will automatically start displaying live values of contacts, coils, function and function block inputs and outputs as far as possible.

If the online editor can't get a value of a variable from the runtime system, it will display "-!-".

Displaying values in the online editor of variable types, that use more than 4 bytes (strings, arrays, structs), is not supported by the current version of the Ladder Editor. To view them use the [Test and Commissioning](#).

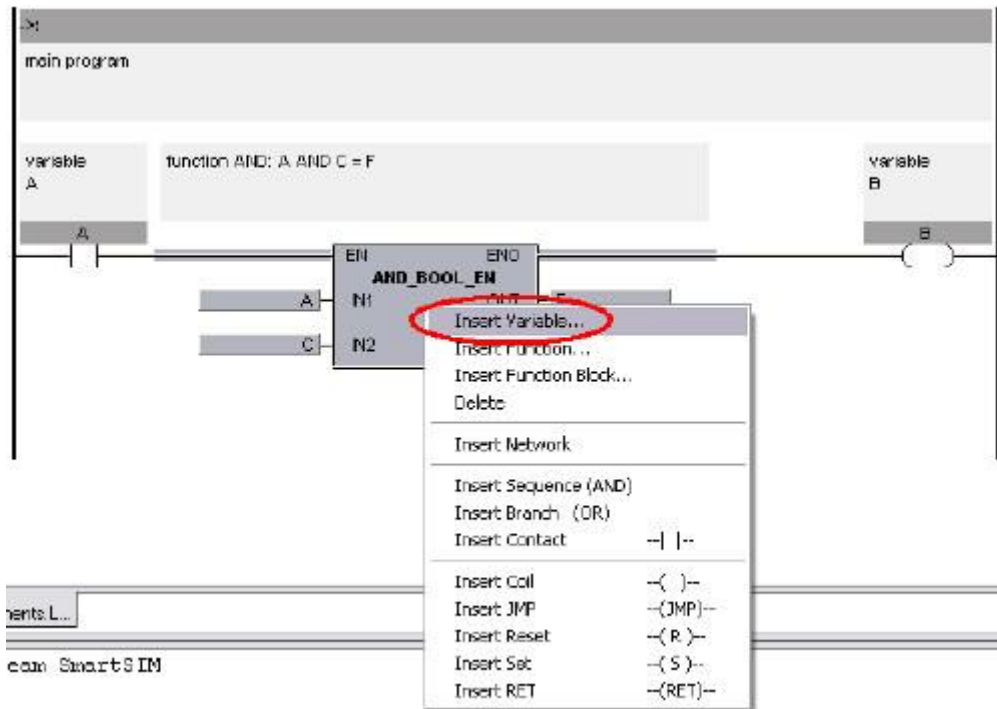
OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.8.10 Check over Variable

The Ladder Editor contains a comment check method, that marks comments if the semantic of a program has changed. To mark comments that might be wrong, OpenPCS pre-writes "[CHECK!]" to such comments. Then it's up to you to check if these comments are still correct.

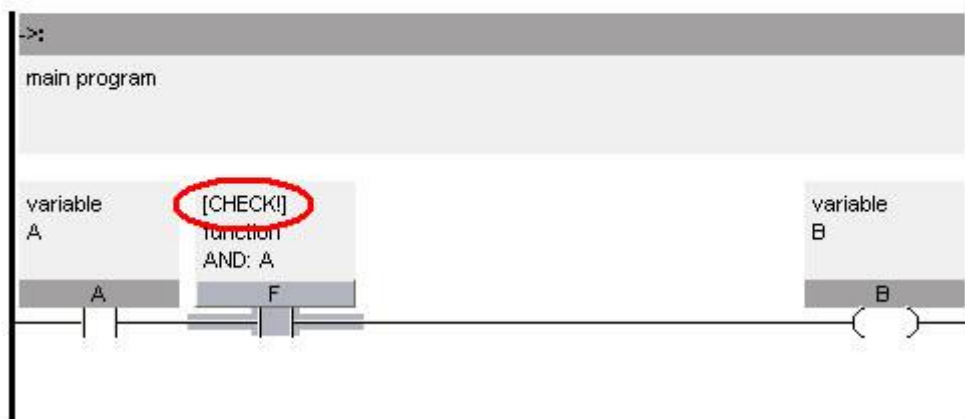
The reason therefore is that when using the ladder editor, it is possible to replace a function (block) by a contact with a variable or vice versa. This changes the semantic of the program and so the comments above the function (block) or variable might be wrong.

To illustrate this, look at the following figures. Choose a function that you want to be replaced by a contact with a variable. Select it with the right mouse button and choose "Insert Variable" from the context menu.



After replacing this function by a contact, the comment above the function is changed. Now, there's pre-written "[CHECK!]".

A (very) simple demonstration program!



The main reason here is, that the semantic of the program has changed, but the comment is still the same. This is a hint, to verify if this comment's still correct.

2.8.11 AutoComplete / AutoDeclare

If a variable is typed, which is not declared, and CTRL-SPACE (RETURN in Ladder editor) is used, the declaration dialog will appear.

If there is already a variable with the given name, nothing happens.

If a variable is typed, which is the first part of a declared one, the declared variable will be inserted at the given position.

If there is a couple of variables available, a list will appear and the user can navigate with UP-arrow and DOWN-arrow to the entry he wants. Pressing RETURN in that case will insert the variable at the current position. Pressing ESCAPE will hide the list and return back into normal edit mode.

2.9 CFC Editor

2.9.1 Introduction CFC Editor

The OpenCFC^(R)-Editor (Continuous Function Chart Editor) is an engineering tool used to create automation programs graphically.

The main elements of a CFC chart are [Blocks](#) (firmware blocks, user defined blocks, compound blocks), that can be freely arranged on the chart, [Margin Bars](#) (left and right), which provide links to IEC61131 variables and virtual links within the chart, and [connections](#), to connect one output (block or margin bar) to one or more inputs (block or margin bar).

2.9.2 Working with Blocks

To add blocks to your CFC chart, use Insert->Block for firmware or user-defined blocks, Insert->Textblock for [text blocks](#), or Insert->CompoundBlock for [compound blocks](#).

The mouse cursor will change, click the chart where you want to insert the new block.

To re-arrange blocks, select the blocks and drag-and-drop them to their new location.

When adding new blocks or moving existing blocks, the CFC Editor will make room by moving aside existing blocks as appropriate.

To remove blocks from your chart, select them and press DEL.

Click twice on a block give it an [alias name](#).

2.9.3 Connections

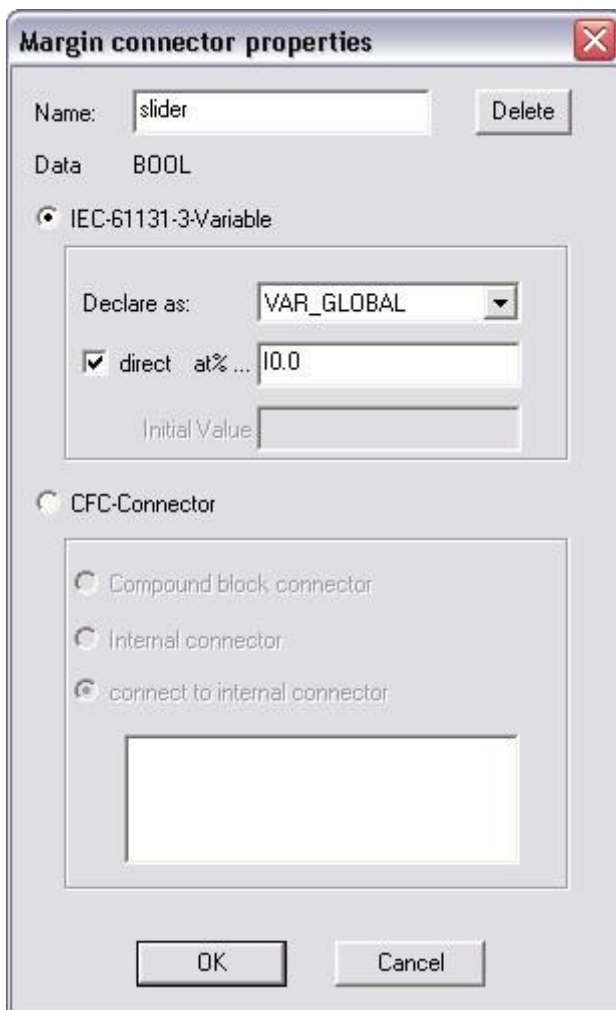
To connect two objects, first select the output object (output of a block, or item on the left margin bar), then select the input (input of a function block, or item on the right margin bar), then press **Insert->Connection**.

OpenPCS also supports [Multiple Connections](#)

2.9.4 Margin Bars

Margin Bars connect the logic contained in the CFC chart to other parts of the same CFC chart, or to other parts of the application or the process to be controlled.

To configure any element of the margin bar, right-click it and select "Properties" from the context menu:



In Name, enter the name of the object. This should be a valid IEC61131-3 variable name.

If you want the CFC-Editor to declare a variable for this margin bar object, select IEC61131-Variable. Otherwise, if you select "CFC-Connector", the object is used only virtually, and all information is immediately propagated to the connected outputs. This may be more economic in runtime and memory consumption, but it prevents online monitoring.

For IEC61131-3 variables, select the declaration section from the combo-box. The selection offered here depends on the type of block and the type of margin bar. For some kinds of variables, you may choose to select a physical address or an initial value.

For CFC-connectors, you can choose "compound block connector", i.e. a connection from within a compound block to the outside, "(connect to) internal connector", i.e. virtually connecting one entry on the right margin bar back to one on the left margin bar. "Internal connector" and "connect to internal connector" are similar, but the first is only available on a right margin bar (where internal connectors are defined), whereas the latter is available only at a left margin bar, where internal connectors may be used.

2.9.5 CFC Editor Online

When you have the CFC Editor in monitor mode, it will automatically start displaying live values of blocks, connections and margin bar entries as far as possible.

If the online editor can't get a value of a variable from the runtime system, it will display "-!-".

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.9.6 Advanced CFC topics

2.9.6.1 Text Block

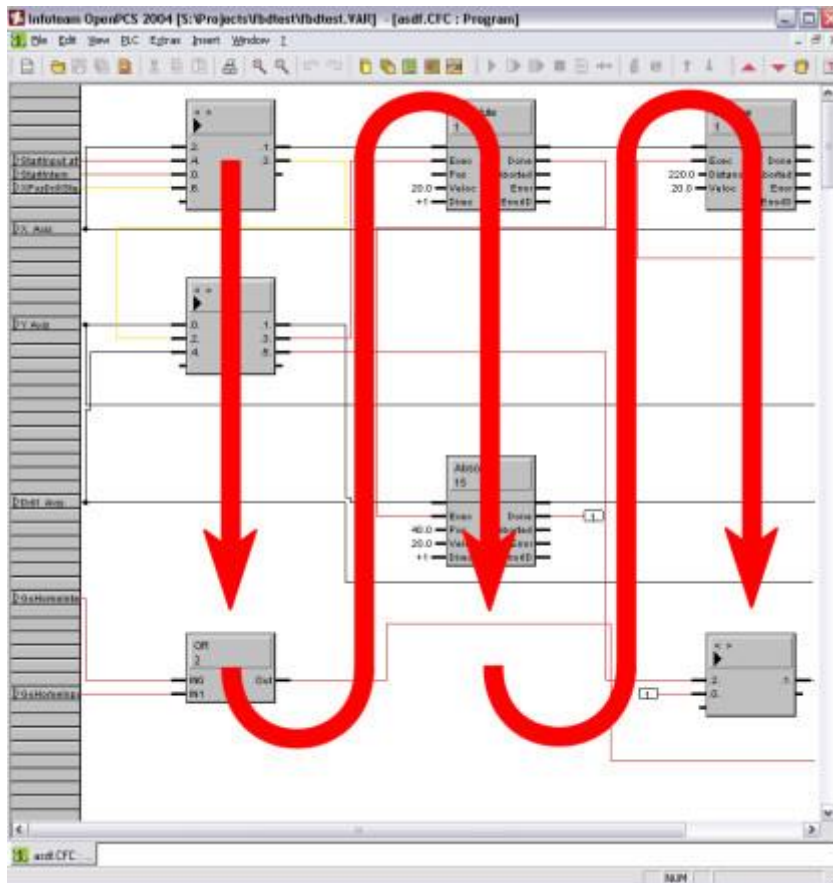
Use Insert->Textblock to insert a text block into your chart. A text block is only for documentation purposes and does not add anything to the code being executed.

2.9.6.2 Using constants as inputs

To use a constant value as the input to a block, select the input (or margin bar entry), right click it with the mouse, select "properties" and enter the constant value in the edit field "value" on sheet "default value".

2.9.6.3 Execution Order

The arrangement of the blocks on a chart is directly related to the sequence of execution: Blocks are executed first column first from top to bottom, then second column top to bottom, and so on. To modify execution sequence, rearrange the blocks as required.



Compound blocks will be executed as a whole at that moment in the execution order where the compound block is located. The contents of the compound block will be executed in itself following the same rules. This is very similar to subroutines in modern programming languages.

The CFC-Editor offers you several possibilities for printing. Use File->Print to print the current level of a chart, and File->Print All to print all levels of the loaded CFC chart.

2.9.6.4 Multiple Connections

The CFC editor supports connections between one output and multiple inputs. To create a multiple connection first create a connection between the desired output and one input. Now, mark the next input and click in the output. The connection, created in the first step and the output are now marked. Choose **Insert->Connection** to create the multi-

ple connection between the output and the two inputs. You can now add more inputs the same way.

To remove an input from a multiple connection, mark the input and hit the delete-key. Only the connection between this input and the output will be removed.

2.9.6.5 Replacement of Blocks

The CFC editor supports the replacement of a firmware or user-defined block by a block of another type by selecting the block(s) and choosing Edit->Replace Block from the menu.

A dialog box analogue to the Insert->Block dialog will appear, allowing the user to select the desired new block type from a list of known firmware and user-defined blocks.

Additionally the user may check the option "automatically replace all instances of the block type in current plan", which causes the replacement of all instances (even the non-marked ones) of the currently marked block's block type inside the entire CFC-plan.

After selection of a new block type, another dialog box is shown, allowing the user to map the connectors of the old and new block type for reconnection after replacement. The left column of the displayed table lists the connectors of the old block type together with the type and kind (VAR_INPUT/VAR_OUTPUT) of the connector (*1). The right-hand column displays a list of adequate connectors of the new block type.

The user can assign a corresponding connector for each connector of the old block type. Note, that each connector of the new block may only assigned once.

If a connector shall or can not be reconnected, "do not reconnect automatically" can be chosen.

After clicking OK the CFC editor replaces the block(s) by (a) block(s) of the new block type and rewires the connectors as specified in the assignment dialog.

(*1): VAR_IN_OUT connectors will show twice in the list of connectors: Once as VAR_INPUT& and once as VAR_OUTPUT&. The "&" marker signals, that the connector actually represents an VAR_IN_OUT parameter.

2.9.6.6 Finding Errors in CFC

The CFC Editor will locate you close to the location of an error if you double-click the respective error message in the [output window](#) of the framework.

2.9.6.7 Block specific help

It is possible to get a block specific help. Right-click on the block, you want help for, and select the menu-item "Show documentation". If OpenPCS finds no reference, you will be prompted. If one reference is found, it will be displayed and if more than one reference you will be prompted to choose which one to display.

2.9.6.8 Extensible inputs

The following CFC (and FBD) functions are extensible. This means we can add one or more inputs as a copy of the first input:

AND, ANDN, OR, ORN, XOR, XORN, MUL, ADD, MUX, MIN, MAX, CONCAT

Appending an input is done via selecting one of those functions and calling (context) menu entry "Append Input". If you want to delete again an added input, select input and call (context) menu entry "Delete Input".

2.9.6.9 Functions with negatable inputs

For all of the following logical CFC (FBD) functions you can negate each Boolean input:

AND, ANDN, OR, ORN, XOR, XORN, NOT

Negating an input is done via selecting the input and calling (context) menu entry "Negate Input". A negation circle is drawn at the connector.

The next call of (context) menu entry "Negate Input" removes the negation.

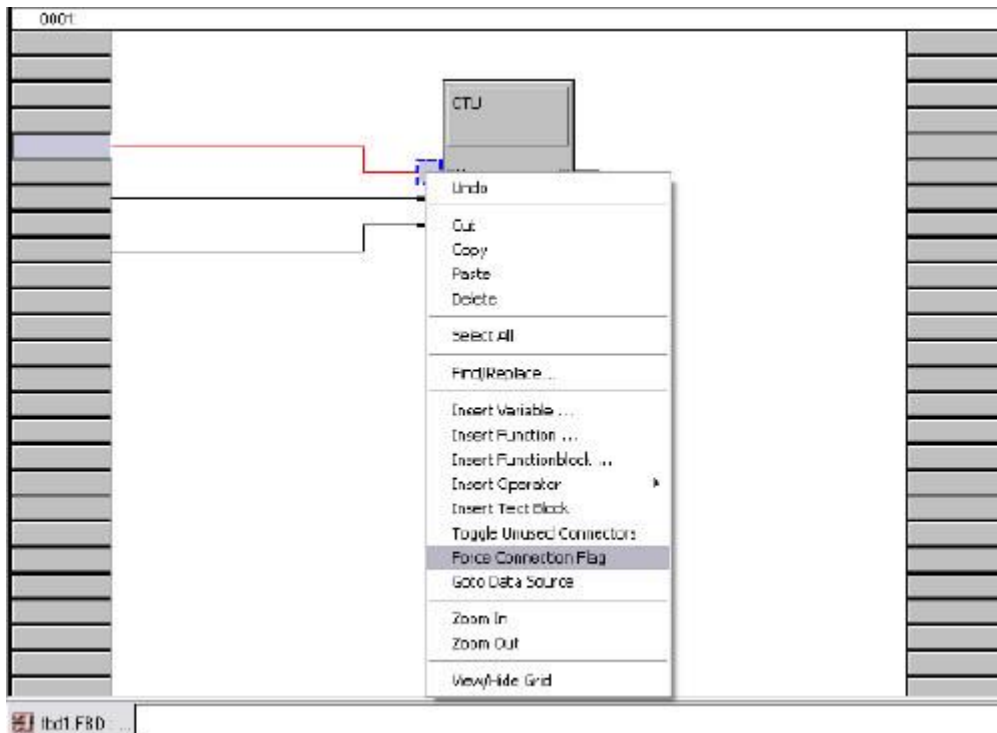
2.9.6.10 Syntax check at CFC connections

After inline editing values or IEC identifiers on all CFC connectors the user input is checked for correct syntax: If a constant value is entered that does not fit the data type of the connector a message like

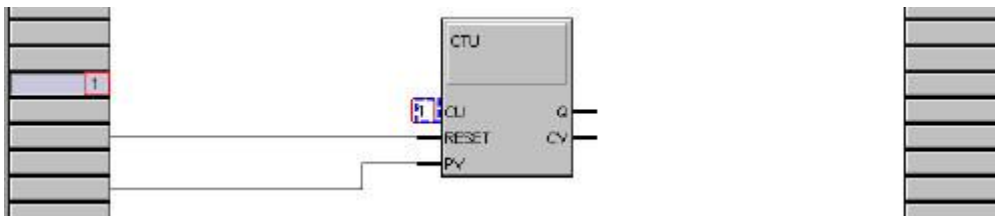
"Syntax error: Invalid constant for data type xxx."
is shown. Anyway the value is accepted.

2.9.6.11 Connection flag

To reduce the number of connection lines we can suppress single connections and force so called connection flags via (context) menu entry "Toggle force connection flag":



Use connection flags for this single connection.



The suppression of connection lines is saved with plan and restored after reloading.

Connection flags are also used if a connection exists between connectors with different page numbers. These flags are not visible in the program but if *Print comments and flags* (see [Print Form](#)) is used for printing the chart these flags can be printed.

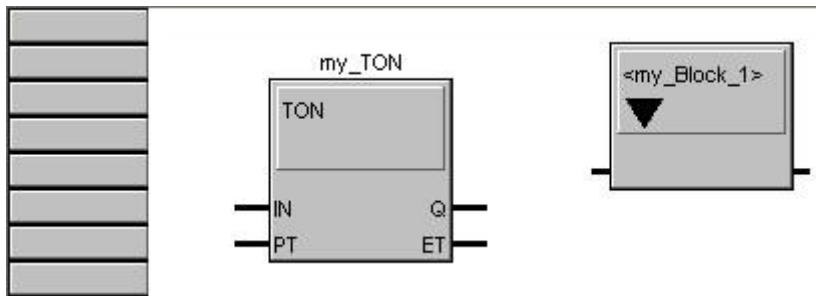
Flags are numbered pagewise in Hex-Format.

2.9.6.12 Copying blocks with inputs

If at least one block is selected, there is a new (context) menu entry active: "Duplicate blocks". Calling it copies the selected block(s) into the internal plan clipboard and set editor into duplicate mode - mouse cursor and caret style behave and look like they do in paste mode: Everywhere you click or press space bar the duplicate(s) of the block(s) is/are inserted and all input connections are duplicated. Until you right-click the mouse, press ESCAPE or click into a "no-paste-allowed" area, the editor stays in duplicate mode so you can insert more duplicates.

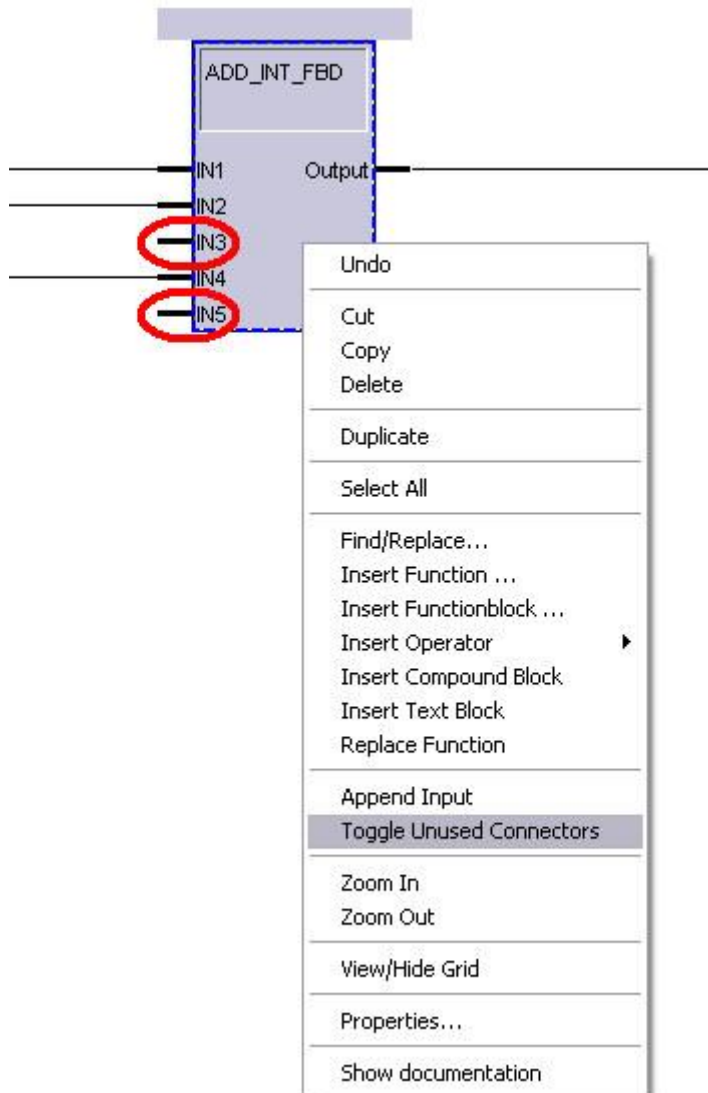
2.9.6.13 Alias names

The user can enter alias names for blocks to mark and quick find special blocks. Alias names for functions and function blocks are drawn and inline editable above the block body. Alias names for compound blocks are drawn and inline editable within the block body.

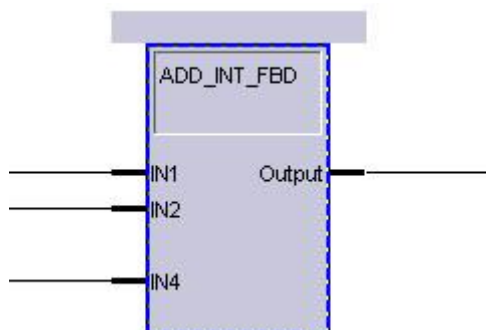


2.9.6.14 Masking of unused connectors

For more clarity there is a new (contex) menu entry "Toggle Unused Connectors". Calling it hide/shows **all** unused block connectors. Unused connectors are connectors without any connections and values.



Unused connectors are not shown.



If unused connectors are hidden

1. we cannot find them by searching.

2. we cannot navigate onto neither by mouse nor by keyboard.
3. we still can find them by double clicking on a compiler/syntax error/warning.

2.9.6.15 Global ID

For each object (block, connector) a global ID is assigned to be uniquely addressed. For blocks this ID is displayed below the name of the block. The global ID can also be displayed via tooltip.

2.9.6.16 Keyboard handling for CFC and FBD editor

2.9.6.16.1 Fundamentals for keyboard usage

For keyboard navigation, a small caret is displayed which shows the current input focus for the user.

The CFC/FBD editor can be used with mouse and keyboard simultaneously. **The cursor will not follow the caret.** The form of the cursor will not automatically change due to the state of the caret. The state of the cursor will of course follow the position of the cursor and not the position of the caret

2.9.6.16.2 Caret and selection

The current selection follows the caret. Exceptions or special cases are:

If the caret is navigated to an empty grid cell, the selection is canceled (nothing is selected).

To detach the caret position from the current selection for generating a connection, the caret must be navigated while <shift>-key is pressed. As the <shift>-key is released the selection is enlarged by the element at the current caret position (equivalent to a left-click on the element in the caret). The current implementation takes care that only permitted states of selections can be made.

Multiple selections with other elements can be made using <ctrl> while navigating. (Multiple selections consisting of isolated blocks is not allowed.)

2.9.6.16.3 Representation of the caret

The caret is always visible. Even if the element, on which the caret is located, is selected.

In special cases the caret is represented in a different way.

The caret is always visible even if the selection is done by mouse.

The caret can not be switched off.

The caret will not be printed.

2.9.6.16.4 Positioning of the caret

The caret is positioned at the marked point by left or right mouse click.

follows in general the selection by mouse.

2.9.6.16.5 Caret position by selected moves

It must be granted that (even in co-use of mouse and keyboard) there is always a valid caret position. The caret position is defined for the following actions which remove the element at a valid caret position:

Selection by mouse: The caret follows in general the selection by mouse and automatic functions

Removing/cutting a block: Thereafter the caret will expand to the whole grid cell which was occupied by the removed/cutted block.

Removing/cutting a set of blocks: Thereafter the caret will select the left upper grid cell which was occupied by the set of blocks.

Removing/Cutting the input of a block: The caret will jump to the input that is above the removed/cutted input. If there isn't any, the caret will expand to the whole block.

Removing/cutting a network: The caret will jump to the network above the removed/cutted network. If there isn't any, the caret will jump to next possible network below.

Removing/cutting a set of networks: The caret will jump to the network that is above the uppermost network. If there isn't any, it will jump to the first network below.

Decreasing the number of rows in a network: The caret will jump to the grid row above, the grid column will be the same. The caret refers at first to the grid cell even if there is a block contained in it.

Caret position after "select all": After the call of "select all", the caret jumps to left uppermost grid cell in the map. The map is scrolled upwards for uncovering the caret. Internally the same method is called as by using the shortcut <ctrl>+<pos1>.

2.9.6.16.6 Automatic positioning of the caret

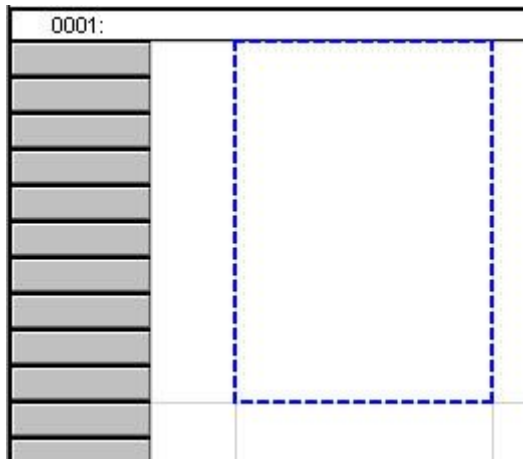
After a file is loaded, the caret is placed at the upper left grid cell. The position of the caret is not saved with the map.

After the entering of a compound block, the caret will be placed at the upper left grid cell.

By using undo/redo, the caret follows the position which is provided by the operation. For this purpose, the caret position is saved before undo/redo and will be restored according to network number and position (row, column). If the network or the concerning cell doesn't exist anymore, the caret will jump to the next network/cell above.

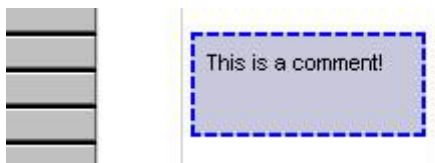
Below, the defaults for the positioning of the caret are listed, depending on the driven CFC/FBD element. How the navigate between these positions is described in a future chapter (Caret navigation).

Caret in empty grid cells



In empty grid cells, the caret takes the size and position of the whole cell.

Caret and comments



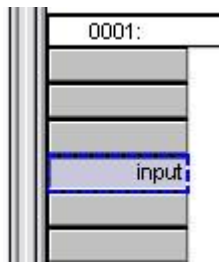
At grid cells with comments, the caret takes the position and size according to the selected comment.

Caret at the (FBD) network label



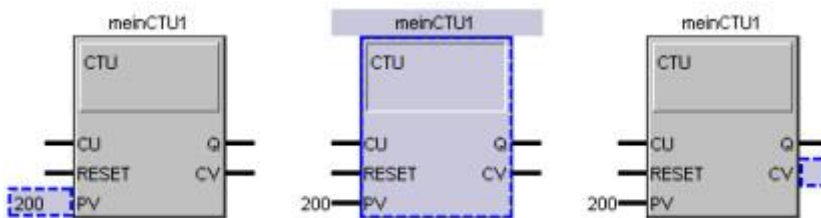
At the network label, the caret takes the position and the size according to the network title line (according to the measures of the selected network label).

Caret at a margin connector



At a margin connector, the caret takes the position and size according to the measures of the selected margin connector.

Caret in grid cells with blocks



The caret surrounds either the block field or a connector. The size of the caret at a connector/block corresponds to the selection of a connector/block. The name of an entity will not be surrounded by the caret.

2.9.6.16.7 Caret navigation

In the following is described how to navigate with the caret inside a CFC/FBD map.

2.9.6.16.8 Navigating at margin

At margin, you can jump to the underlying margin element or the element above by using <UP> or <DOWN> arrow keys.

2.9.6.16.9 Navigating between (FBD) networks and network labels

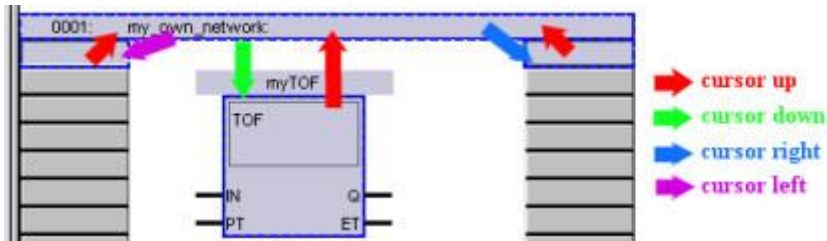
If the caret is on the upper or lower margin connector, you can jump to the network label of the underlying network or network above by using <UP> or <DOWN> arrow keys (see picture below).

If the caret is on a grid cell or element in the upper row of a network you can jump to the network label of the network above by using <UP>

If the caret is on a grid cell or element in the lower row of a network, you can jump to the network label of the underlying network by using <DOWN>

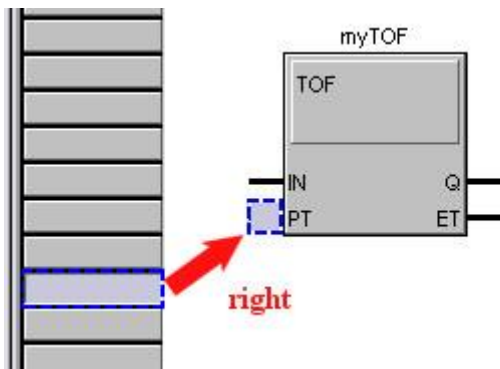
If the caret is on a network label, you can jump to the left lower grid cell (resp. grid element or connector) of the network above by using <UP>

If the caret is on a network label, you can jump to the left upper grid cell (resp. grid element or connector) of the network belonging to the network label by using <DOWN>. With <RIGHT> or <LEFT> the caret jumps to the upper connector of the left or right margin.

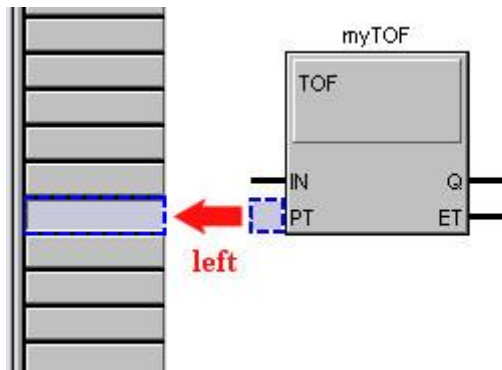


2.9.6.16.10 Changeover margin to block

By using <RIGHT> or <LEFT> when the caret is located at left or right margin, the caret jumps to the grid cell resp. element of the grid cell which is opposite to the margin connector. A margin connector at the level of a connection channel is always assigned to the grid cell above the connection channel. If the grid cell contains a block, the caret jumps to the closest connector in consideration of the starting position (margin connector).



If the caret is positioned on a grid cell or on a block connector besides the margin, it jumps to the closest margin connector.



2.9.6.16.11 Up and down at inputs/outputs

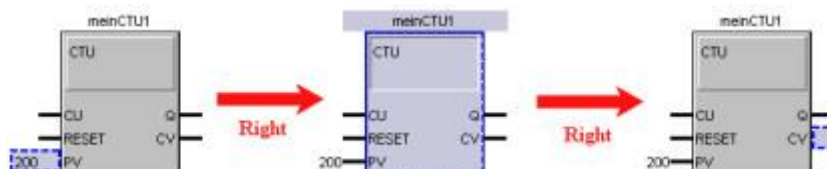
<UP> or <DOWN> navigates the caret to the input or output of a block.
If the caret is located on the lowest input/output, you jump to the underlying grid cell or the label of the next network by using <DOWN>.

2.9.6.16.12 Left and right at inputs and outputs

<LEFT> or <RIGHT> navigates the caret between input/output and the block field itself.

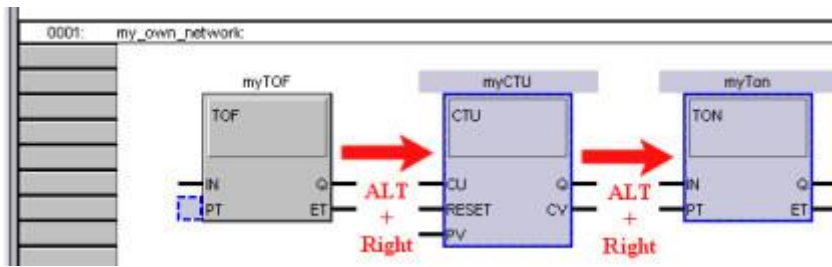
Observe the behavior of the caret by navigating from the inputs/outputs of a block to the outputs/inputs of the same block.

For this purpose, the last caret connector row/column is buffered. Thus, a behavior as in the following picture is possible.



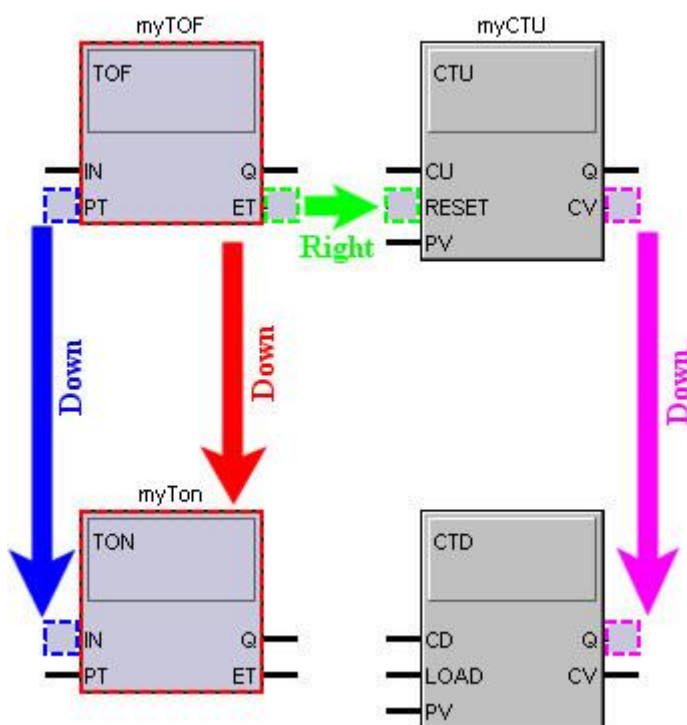
By navigating onto the block field, the caret connector row is not changed and will be evaluated by the next usage of <RIGHT>. The same behavior happens for the caret connector column how we will see in one of the following chapters.

For navigating faster between grid cells with blocks, you can jump directly to the block field by using <ALT> + <UP/DOWN/LEFT/RIGHT>.



2.9.6.16.13 Navigating between grid cells

Observe the behavior by navigating between grid cells with blocks. By navigating on an empty cell or a cell with a comment, the caret is placed on the comment or the whole grid element with no respect to the starting position. For navigating between grid cells with blocks, the principle of buffering the caret connector row/column as described above is essential.



If there is no connector which fits to the current connector row or column (e.g. JMPC), the caret will jump to the block field.

2.9.6.16.14 Navigating along connections

The caret can jump to all connected inputs starting at an output connector. With the methods defined in the chapter "Methods for navigating the caret", you can jump from every input connector to all connected output connectors and vice versa.

Attention: The next output connector is always that one which was connected to the input connector with respect to time.

For these actions, there are entries in the (context) menu:

Goto Data Source : jump to data source

Goto Next Data Destination : jump to next data sink

Goto Previous Data Destination : jump to previous data sink

2.9.6.16.15 Fast navigation with the caret

2.9.6.16.16 Pos1 and End

Pos1 and End **refer only to the grid itself** (the margin is excluded) and locate the caret on the grid in the current row far left or far right.

2.9.6.16.17 Ctrl+Pos1 and Ctrl+End

Ctrl+Pos1 and Ctrl+End **refer only to the grid itself** (the margin is excluded) and locate the caret at the upper left or lower right corner of the grid. I.e. Ctrl+Pos1 in FBD jumps to the upper left corner of the first network and Ctrl+End to the lower right corner of the last network.

2.9.6.16.18 Page Up/Down

By using Page Up/Down, the visible clip is always aligned to the top edge of a grid cell. It is scrolled only by the number of visible grid cells.

2.9.6.16.19 Automatic post scrolling

While navigating, the visible clip shall always be scrolled in that way, that the caret (plus a certain amount of tolerance) is visible.

2.9.6.16.20 Revoking the selection

The usage of the <ESC> key revokes the current selection but doesn't change the position of the caret.

2.9.6.16.21 Selecting multiple elements

By using <CTRL>+<LEFT/RIGHT/UP/DOWN>, multiple elements can be selected. Still, only consistent and valid selections are permitted. (e.g.: blocks and border line connectors cannot be selected at the same time)

Attention: While working with the caret, there is no rectangle selection (rubber band selection) possible!

2.9.6.16.22 Inline edit at the caret position

If the caret is located on an element, which is inline editable, the element will be selected and opened in the inline edit modus as soon as the user starts to write an alphanumeric sign.

There's an ambiguity with regard to the decision for caret and selection: If there's already another inline editable element selected, that element, which is currently covered by the caret, is set to the inline edit modus.

2.9.6.16.23 Insertion of blocks by keyboard usage

The insertion of blocks by keyboard works according to the following procedure:

Call the choosing block dialog by shortcut.

Chose the block type to be inserted.

Close the choosing block dialog and the insert modus is automatically activated.

For finally inserting the block, the caret must be moved to the insert position. **Navigation is only allowed between grid cells.** The caret will be shown as described as in "Caret in empty grid cells". (Even if there is a block in it)

If the caret is moved to a position at which inserting a block is not allowed, the caret will change its figure according to properties for exception situations.

(see caret properties)

If a valid location for inserting a block was chosen, the block is inserted by using <SPACE> and the caret is placed on the block field.

If an invalid position was chosen and <SPACE> pressed, an event is sent to the automation suite that the insert operation was not successful. The insert operation is aborted and the standard caret is shown.

2.9.6.16.24 Moving/copying blocks and margin connectors by keyboard

Blocks can be moved by using <CTRL>+<SHIFT>+<UP/DOWN/LEFT/RIGHT>. As soon as the <CTRL>+<SHIFT> keys are released, the insert operation at the current caret position is made (equivalent to releasing the left mouse button while moving a block/margin connector by mouse). The figure of the caret on invalid positions is according to inserting blocks.

Margin connectors can be moved by using <CTRL>+<SHIFT>+<UP/DOWN>. As soon as the <CTRL>+<SHIFT> keys are released, the insert operation at the current position of the caret is made. (equivalent to releasing the left mouse button while moving a

block/margin connector by mouse). The figure of the caret on invalid position is according to inserting blocks.

Copying blocks and margin connectors is made by using copy and paste. **Thereby you can only move between grid cells.**

2.9.6.16.25 Insert connections by keyboard

For inserting a connection by keyboard, two "compoundable" elements (block connectors and/or margin connectors) have to be marked by the caret. Afterwards a new connection can be inserted by using the shortcut for the menu "Insert -> Connection".

More comfortable and faster: If the shift key is released while two or more connectors are selected, which allow a connection, this connection is inserted automatically.

2.9.6.16.26 Keyboard combinations for navigating the caret

Alt + arrow keys : fast navigation for blocks

Ctrl + arrow keys : multiple selection (e.g. connectors or blocks)

Alt + Ctrl + arrow keys : fast multiple selection only for blocks

Shift + arrow keys : release the caret from selection

Shift + Alt + arrow keys : release the caret from selection using fast navigation

Ctrl + shift + arrow keys : moving of blocks or margin connectors

2.9.7 Compound Blocks

2.9.7.1 Compound Blocks: Introduction

Compound Blocks are a way to structure your application

The work area of the CFC-Editor is limited to one page width. By selecting the paper size, you determine the number of blocks that can be placed horizontally. Vertically, a function chart can grow unlimited.

Although in fact you are not limited in the length of your CFC chart, it is easy to lose overview on a too lengthy chart. Compound Blocks are a means to finer structure your application, hiding groups of logically related blocks inside one "Compound Block".

Signals between the blocks inside a Compound Block are not visible to the outside. Outside a Compound Block, only those signals are visible that enter or leave the Compound Block.

On screen, double-click the Compound Block to see it's contents. Use **"View->Level up"** or in the toolbar to get back to the location where the Compound Block is being invoked.

Compound Blocks can be nested, i.e. inside a Compound Block you can define, or use, other compound blocks. The contents of a Compound Block can be edited, you can add or delete blocks, rewire connections, add, modify or delete connections leaving or entering the Compound Block.

On screen, the last input and output connector of a Compound Block is shorter than any other connector, so you can easily distinguish a Compound Block from other Blocks.

2.9.7.2 Create compound block

To create a new, empty Compound Block,

1. Select "Insert->Compound block..."
2. The mouse cursor changes
3. Click the mouse where you want to insert the new Compound Block

You can now fill the Compound Block first, by double-clicking and editing it just like any other function chart. Or, add inputs and outputs to the Compound Block first, editing its contents later using the already provided inputs and outputs then.

Whenever you run out of space on a chart, or think readability would be increased by more hierarchically grouping, you can collapse some of your already wired blocks into a Compound Block:

1. Have the Block(s) selected
2. Select "Insert->Compound block..."
3. CFC-Editor will prompt you to verify you want to convert the blocks to a Compound Block
4. The selected Blocks will be removed from the chart and replaced by a Compound Block. All signals between these blocks will be moved with the Blocks, all signals to other blocks will be kept and changed to interface signals of the Compound Block.

Notes:

Currently there is no support for reverting the process of converting a group of blocks to a compound block.

2.9.7.3 Adding input or output to compound block

You can edit the contents of a Compound Block just like any other function chart. When you need to provide additional inputs, or need to provide additional outputs, you need to change the interface of the Compound Block accordingly. You can do this from the surrounding (top-down) or from within the Compound Block (bottom-up).

Top-Down:

1. Any Compound Block has one very last connector which is shorter than the others. This is always the last connector, one on the left side as an input, one on the right side as an output.
2. Wire this last input or output
3. As soon as you use this last connector, it will be shown in full length, and another shorter connector will be added to the end.

Bottom-Up:

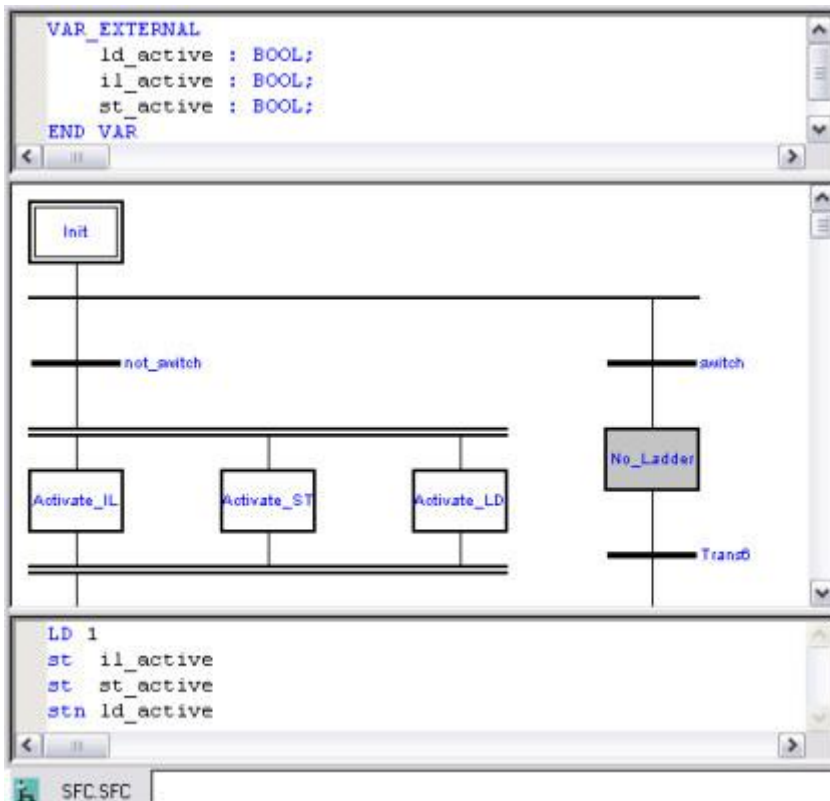
1. Double-click a compound block you want to add a connector.
2. Wire a connection of a block inside the compound block to the left or right margin bar (depending whether you want create an in- or output)
3. Click right on the connector and open the "Properties..." dialog box via the context menu.
4. Mark the items "CFC-Connector" and "Compound block connector" name it and close the dialog box by clicking "OK".

If you go one level up by clicking the appropriate symbol you see that another shorter unused connector has been added to the compound block.

2.10 SFC Editor

2.10.1 SFC: introduction

The SFC-Editor is hosted in the [OpenPCS framework](#). It is separated into three parts. In the upper part of the SFC-Editor, enter the [declarations](#) of the POU. In the middle part edit the chart and in the lower part edit the code of the elements.



Note: The code language is selected once at the creation of the program. The current version supports [IL](#) and [ST](#).

2.10.2 Elements of a sequential function chart

SFC-plans are a tool for formulation of control flow of technical process, which are characterised by change of states. Every state transition is coupled on certain conditions.

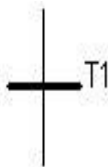
The sequential function chart offers the following language elements:



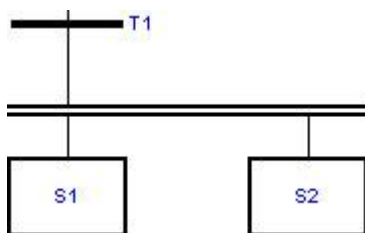
Step: A step contains many actions. Actions contain code fragments. A step, which is executing, is called "active". If a step is active, the contained actions will execute. A step can be activated by: switching of a previous transition, a jump element, setting the initial flag (c.f. initial step).



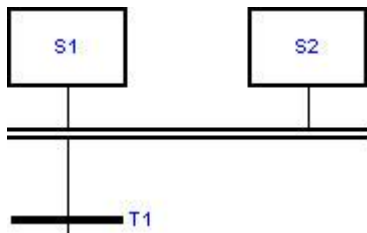
Initial step: Initial steps are active at the beginning of the program. Positions in the plan could be marked by initial steps, at which the execution starts on program start.



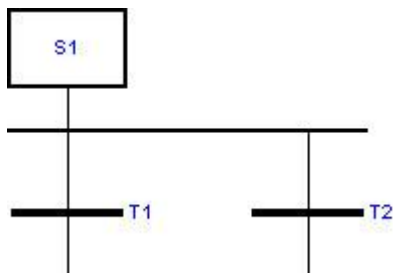
Transition: The program flow is controlled temporally and structurally by switching of transitions. A transition will switch if the transition condition is true and all previous steps are active. Once the transition switches, all previous steps become inactive and all following steps become active.



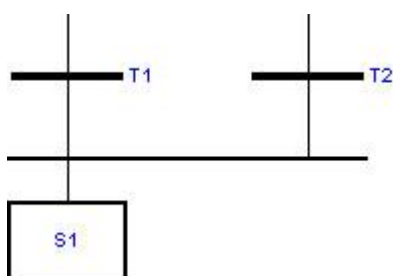
Simultaneous sequences: One transition may set active multiple steps at the same time, starting a parallel chain. If all previous steps of transition T1 are active and the transition condition is TRUE, all following steps (e.g., S1, S2) of the simultaneous sequence will activate.



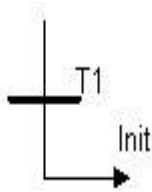
Convergence of simultaneous sequences: The chains of a simultaneous sequence are converged into a single transition. If all previous steps (e.g., S1, S2) are active and the condition of the following transition (T1) is TRUE, all previous steps will be deactivated while the steps following transition T1 will be activated.



Divergence of sequence selection: Selection of a sequence step chain. If the step before the divergence is active, all transitions (e.g., T1, T2) are checked from left to right. The first transition evaluating to TRUE will switch, deactivating the step before and activating the step after.



Convergence of sequence selection: The chains of a divergence of sequence selection are converged into a step. If one of the transitions switches, the steps before it will be deactivated, and the Step following it will be activated.



Jump: The program flow is continued at another location. The name of the jump is the name of the activate step if the previous transition (T1) switches.

2.10.3 Steps and initial steps

The code of a step is executed cyclic if and only if this is an active state. In principle, one can say that the code is surrounded by a loop which is entered if a previous transition switches, and is left if a following transition switches. If a step was activated, its code is executed at least one time. Initial steps are always active at program start that means that no preceding transition is necessary. In standard, the entry-point into the program is the first element in the chart (initial step). Every step can be converted into an initial step by activating the control box "initial step" in the properties window. De-activating this switch will turn the initial step back into a normal step.

The name of a step must meet following syntax: The first character of the step name is a letter ("a"- "z", "A"- "Z"); every further character is a letter or a number ("0" - "9") or a underline ("_"). Valid step names are "Step1" and "S_1", invalid step names are "_Step1" "1Step" and "Heater off"

Step names have a maximum length of 31 characters. You get more information in the topic [Jumps](#) .

2.10.4 Transitions

Transitions are responsible for the change of the active state of previous step(s) to the following step(s). Transitions show the possible change in form of a true, Boolean statement (transition condition).

The code of the transition has to be written so that the current result at the end of the code is of type BOOL. The transition switches if and only if the accumulator (in IL), or the corresponding variable (in ST) is TRUE. The communication with other SFC-elements occurs by local variables.

Example (IL): (* Transition switches off if temperature is more than 70° C *)

```
LD variable_of_temperature
```

GT 70

Example (ST): (* Transition switches off if temperature is more than 70° C *)

```
IF variable_of_temperature > 70 THEN
```

```
    result := true;
```

```
ELSE
```

```
    result := false;
```

```
END_IF
```

```
trans1 := result;
```

Note:

1. The last line of a transition code should always load a Boolean value (in IL), or assign the variable, named like corresponding transition (i.e. trans1) with the intended result (in ST).
2. The declaration of the variable with the transition name in ST are automatically generated and not shown. Do not use their names for other variables (only in ST).

2.10.5 Jumps

Jumps are elements of a SFC-plan for controlling the flow of execution. With the up to now introduced elements, the activation of the steps happens always from top to bottom. For programming of cycles and similar things, a further possibility is necessary to activate previous steps. Jumps exist to provide this functionality.

The predecessor of a jump element is always a transition. The target of a jump is always a step. The target of the jump is fixed by giving the jump the same name as the selected target-step. If a step is given as a target of a jump, its name must be unique. If a jump-target is not or more than once available, corresponding error messages are created during the syntax control.

To guarantee the consistency of an SFC-plan, the insertion of a jump is possible only as the last element of a divergence of sequence selection.

2.10.6 SFC Editor Online

When you have the SFC Editor in monitor mode, it will automatically display status information. Small red rectangles will be displayed in all active steps. This information is

updated as frequently as possible. However, the target controller may be too quick for all intermediate states to be displayed.

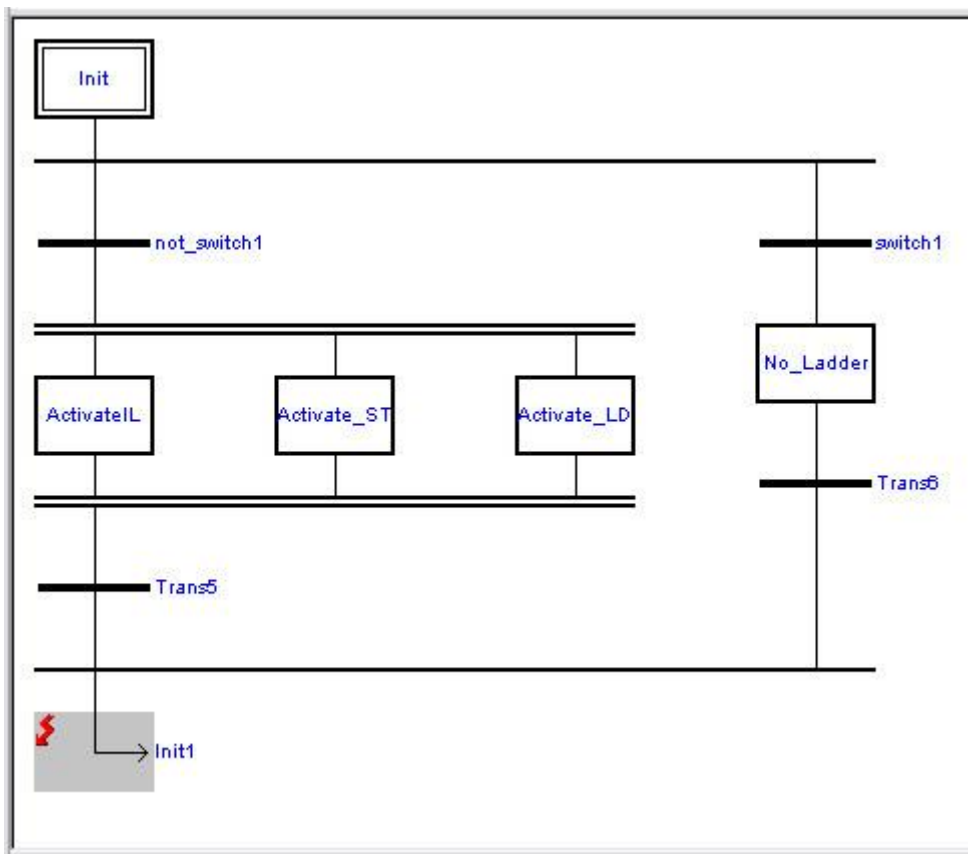
You can view the contents of steps and transitions while online, but not see status information for these. If the contents of steps or transitions grows complex enough to require debugging, it is strongly recommended to move it to individual function blocks.

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.10.7 Common errors

Errors in the chart of the SFC- Editor are indicated by a red arrow, as soon as the project is saved. The most common errors are exemplarily shown via the ControlX-sample enhanced in OpenPCS.

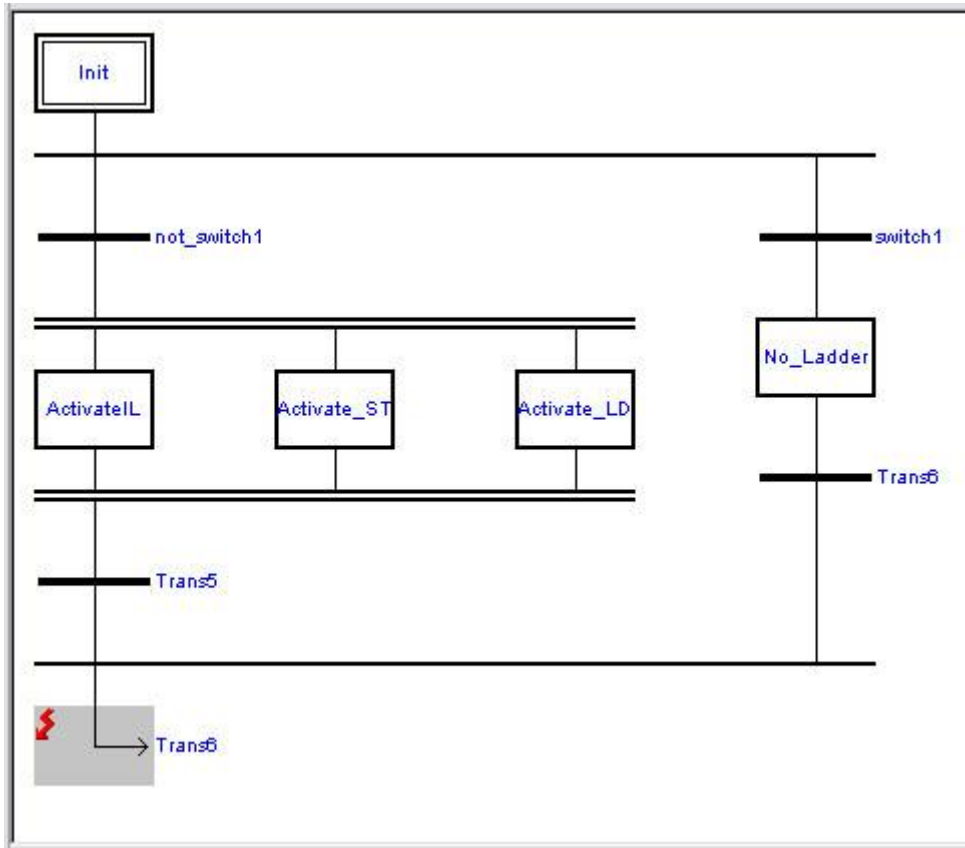
Target of jump is not valid



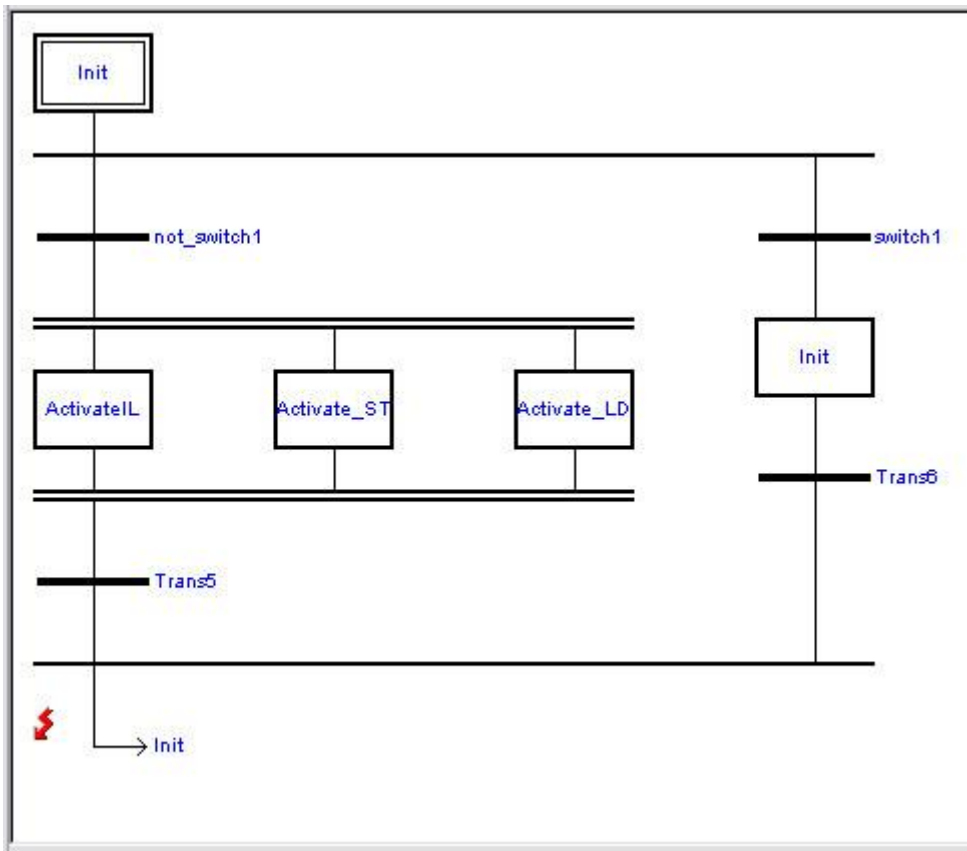
Init1 was defined as target which does not exist within the project. Within the Output-window, OpenPCS prompts:

Error : "Init1" is not a valid jump target (no Stepp with the name "Init1" exists).

Only Steps are valid targets. Thus the same message is prompted, if a transition is defined as a target.



No unique target



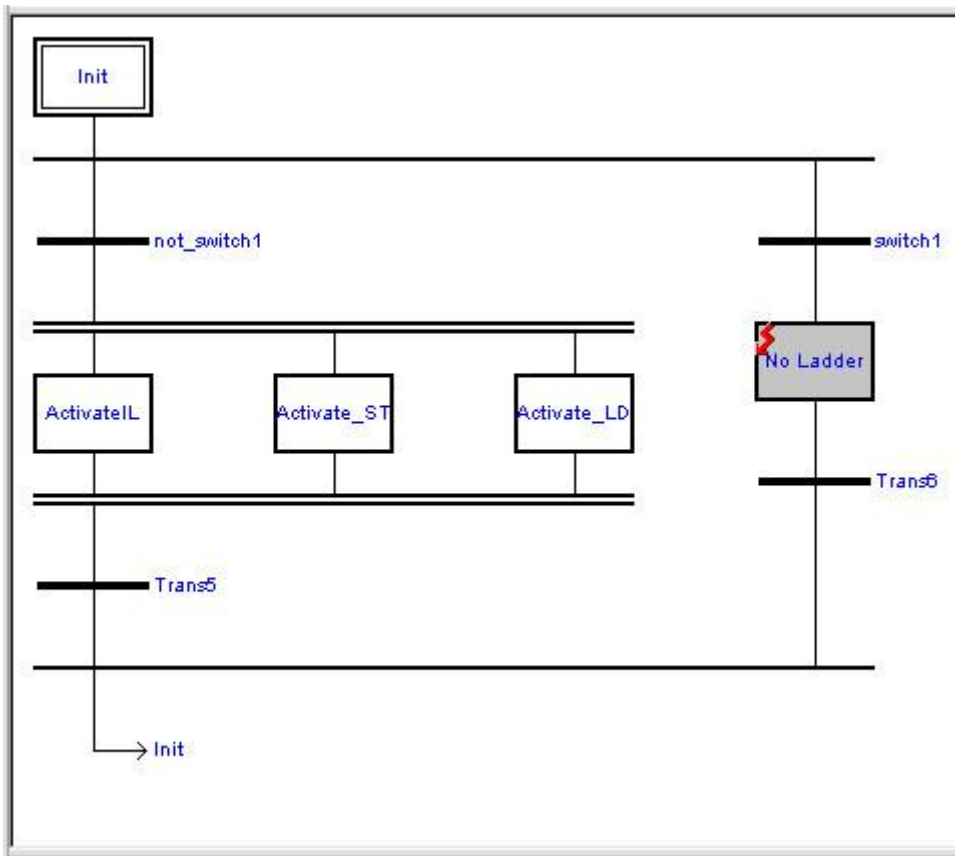
Two Steps are named *Init*. Since the step is chosen as target, OpenPCS cannot decide which step should be used. It prompts the following message in the output.

Error : The jump target "Init" is ambiguous - the following 2 steps with the name "Init" exist:

...\CONTROLX\CHART.SFC(10,100,4) : Object : Step "Init"

...\CONTROLX\CHART.SFC(10,100,5) : Object : Step "Init"

No valid names



Spaces are not allowed within the name of steps .

Error : The name of the object "No Ladder" contains invalid charcters.

2.10.8 Selecting Elements

2.10.8.1 Marking a single element

Click the left mouse button to mark an element with the mouse showing its code in the below text editor.

The mark of a single element can be transformed into a neighbour element by aid of the cursor keys (←, →, ↑, ↓) of the keyboard.

2.10.8.2 Region marks

Mark one element first, using mouse or keyboard.

Hold the Shift-key pressed, and click another element with the mouse to select an entire region of the chart.

or hold the Ctrl-key pressed and click other elements to add them to the selection.

In this version are no region marks possible by keyboard.

2.10.8.3 Marking several elements

In order to execute a function on several elements of the SFC-plan, all corresponding elements must be marked.

Mark one element first, using mouse or keyboard.

Hold the Shift-key pressed, and click another element with the mouse to select an entire region of the chart.

or hold the Ctrl-key pressed and click other elements to add them to the selection

In this version are no region marks possible by keyboard.

2.10.9 Advanced SFC topics

2.10.9.1 Exception handling

During an execution of a SFC-program, situations could happen which require a specific change to execution logic in the program. The modelling of this "exception handling" is possible with additional standard plan elements (transitions, jumps, steps), but reduce the clarity of the program.

The SFC-editor offers macros for the solution of this problem on the IL-level to activate or inactivate steps purposefully.

The following commands are available:

```
@ACTIVATE_STEP(StepName) /* Activating of a step */
```

```
@DEACTIVATE_STEP(StepName) /* Inactivating of a step */
```

```
@DEACTIVATE_ALL_STEPS() /* Inactivating of all steps */
```

These macros manipulate the internal execution control so that the given steps will be (in-)activated in the next cycle additionally.

Attention: If the above commands are used in the IL-code, unsure or not executable networks could arise!

Note: The current version does not support those commands for the ST editor.

2.10.9.2 Finding error position

Edit -> Goto IL Line: Use this command to find the necessary code fragment in the SFC-plan by a number of an incorrect line in the generated POE file. The number of an incorrect line is taken down in the OpenPCS-system during the compiling.

2.10.9.3 Using languages other than IL / ST

SFC expects code written in steps and transitions to be in IL or ST. To use other languages (Ladder Diagram, CFC, FBD), write your code to a function block (or function, if applicable) and invoke an instance of that function block from within the step or transition.

Remember to declare an instance of that function block in the declarations of your SFC program.

You can reuse one instance of such a function block in different steps and transitions, or use different one, as required by your application. With more complex code, this will not only yield a cleaner structure of your application, but also reduce memory consumption and increase the ability to debug.

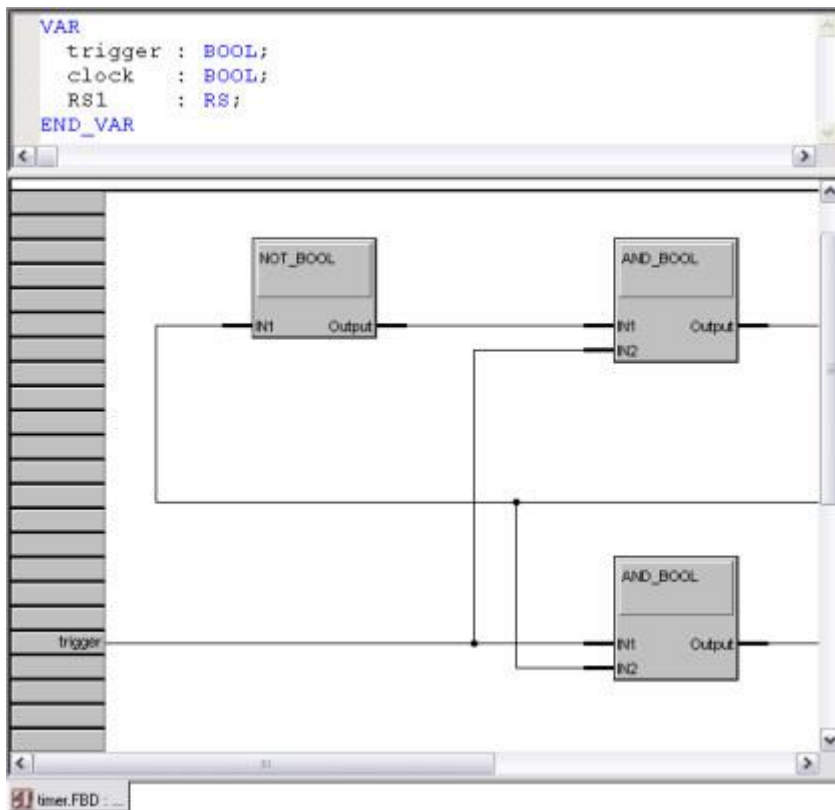
2.11 FBD Editor

2.11.1 Introduction FBD Editor

The FBD-Editor (Function Block Diagram Editor) is an engineering tool used to create automation programs graphically.

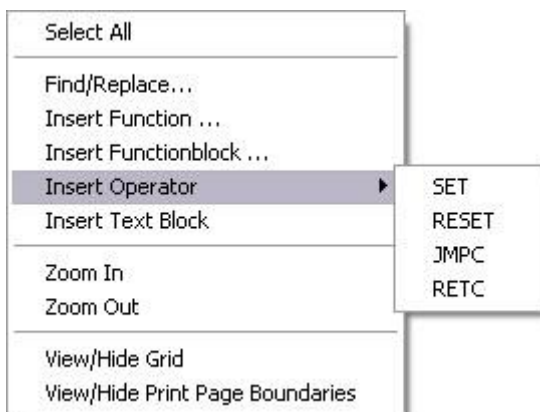
The FBD-Editor is hosted in the [OpenPCS framework](#). In the upper part of the FBD-Editor, enter the [Declarations](#) of the POU.

The main elements of a FBD chart are [Blocks](#) (functions, functionblocks, operators), that can be freely arranged on the chart, [Margin Bars](#) (left and right), which provide links to IEC61131-variables and [Connections](#) to connect one output (block or margin bar) to one or more inputs (block or margin bar).



2.11.2 Working with Blocks

To add blocks to your FBD chart, use Insert->Function, Insert->Functionblock for firm-ware or user-defined blocks, or Insert->Textblock for documentation. You can also press the right mouse button within the instruction pane and select "Insert Function", "Insert Functionblock", "Insert Textblock" or "Insert Operator" from the popup menu.



The mouse cursor will change, click the chart where you want to insert the new block.

To re-arrange blocks, select the blocks and drag-and-drop them to their new location.

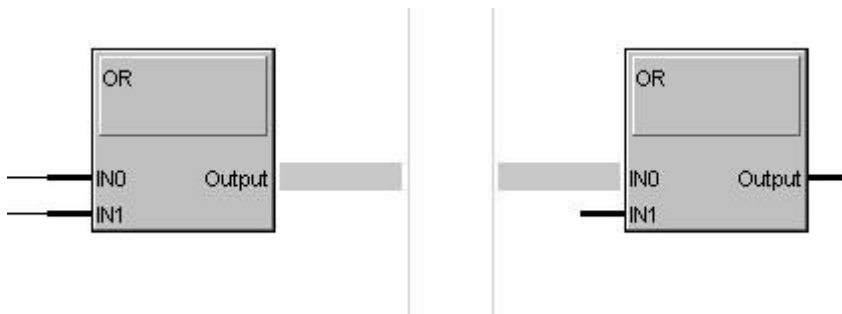
When adding new blocks or moving existing blocks, the FBD Editor will make room by moving aside existing blocks as appropriate.

To remove blocks from your chart, select them and press DEL.

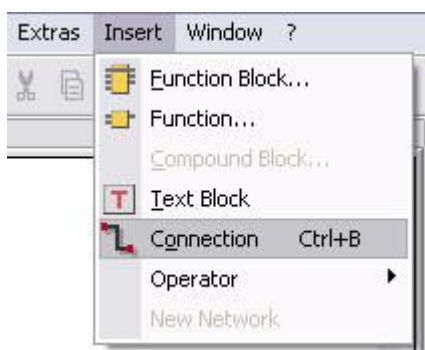
2.11.3 Connections

To connect variables, functions, function blocks etc. just left click the margin bar or the node at function etc. symbols. The margin bars or nodes highlighted you connect with menu "Insert" and there "Connection" or just use the short cut [CTRL] [B]. The new connection now is selected and therefore red. Without selection it will be black.

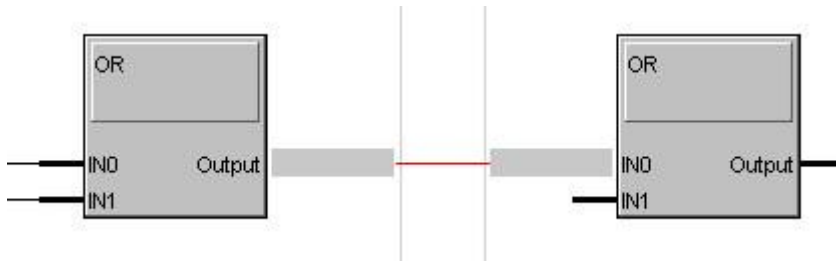
1. Left click



2. Do the connect



3. See the connection



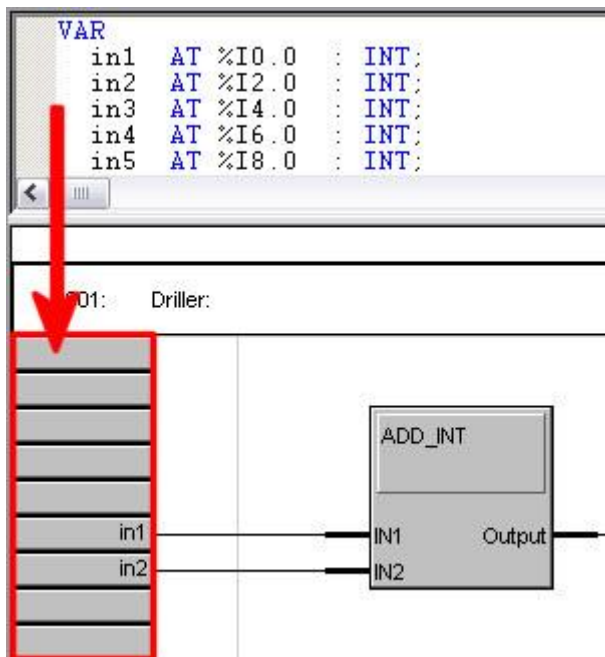
To select a connection just left click on one of its ends (margin bar or node). A selected connection can be deleted just with [DEL] or right click on margin bar or node and selecting "Delete".

Note: There are multiple connections possible starting from one source (it is not possible to merge multiple connections).

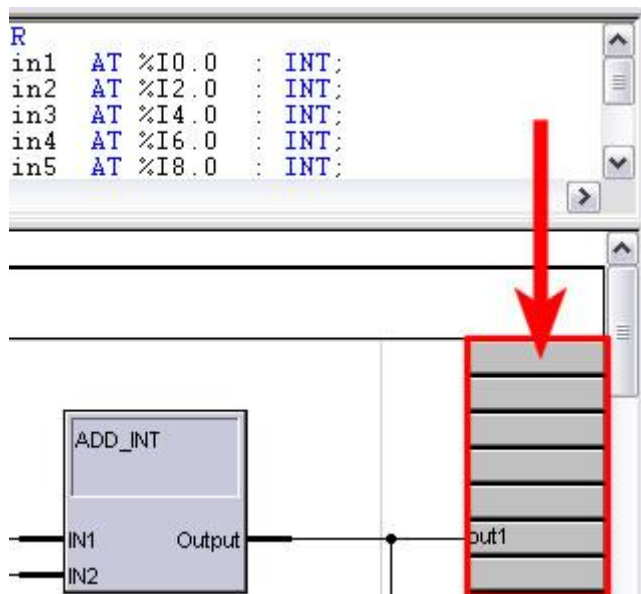
2.11.4 Margin Bars

The margin bars on the left and the right side allow you to connect the logic contained in the FBD with input (left) and output (right) variables. Just double click into a margin bar and type in the variable name.

Left margin bar:



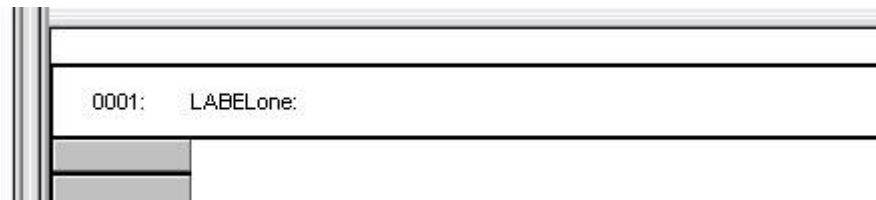
Right margin bar:



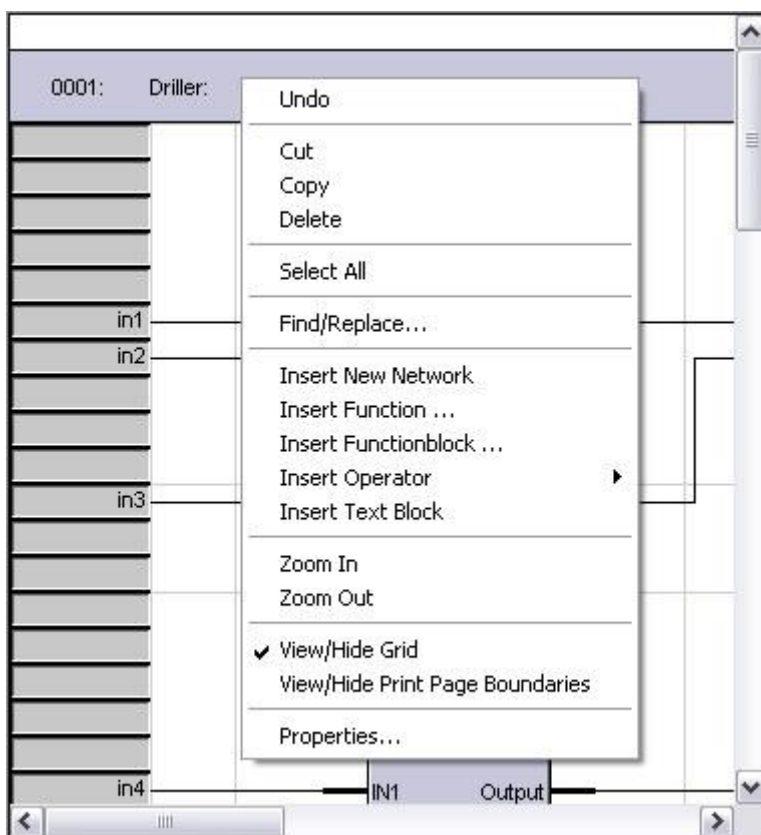
2.11.5 Advanced

2.11.5.1 Working with Networks

You can structure your project with networks. Each network automatically gets a four digit number for identification shown on the top of each network. A click or double-click within that area allows you to type in a name to label the network.



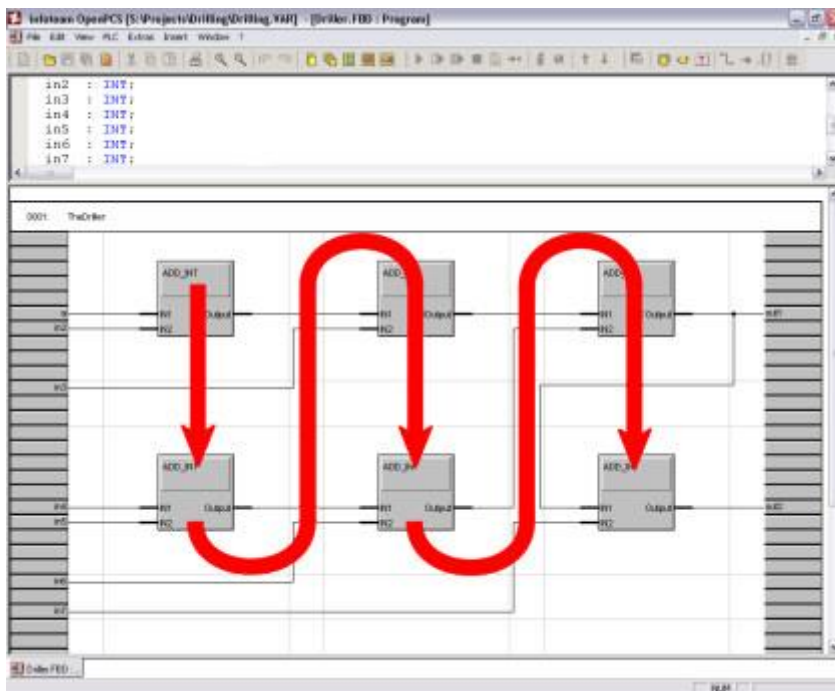
Using the right mouse button a pop-up menu occurs that allows you to add a new network, jump to the next or jump to the previous.



2.11.5.2 Execution Order

The networks are executed one after another beginning from top to bottom.

In each network the arrangement of the blocks is directly related to the sequence of execution: Blocks are executed first column first from top to bottom, then second column top to bottom, and so on. To modify execution sequence, rearrange the blocks as required.



2.11.5.3 Replacement of Blocks

The FBD editor supports the replacement of a firmware or user-defined block by a block of another type by selecting the block(s) and choosing Edit->Replace Function / Edit->Replace Function Block from the menu.

A dialog box analogue to the Insert->Function / Insert->Function Block dialog will appear, allowing the user to select the desired new block type from a list of known firmware and user-defined blocks.

Additionally the user may check the option "automatically replace all instances of the block type in current plan", which causes the replacement of all instances (even the non-marked ones) of the currently marked block's block type inside the entire FBD-plan.

After selection of a new block type, another dialog box is shown, allowing the user to map the connectors of the old and new block type for reconnection after replacement. The left column of the displayed table lists the connectors of the old block type together with the type and kind (VAR_INPUT/VAR_OUTPUT) of the connector (*1). The right-hand column displays a list of adequate connectors of the new block type.

The user can assign a corresponding connector for each connector of the old block type. Note, that each connector of the new block may only assigned once.

If a connector shall or can not be reconnected, "do not reconnect automatically" can be chosen.

After clicking OK the FBD editor replaces the block(s) by (a) block(s) of the new block type and rewires the connectors as specified in the assignment dialog.

(*1): VAR_IN_OUT connectors will show twice in the list of connectors: Once as VAR_INPUT& and once as VAR_OUTPUT&. The "&" marker signals, that the connector actually represents an VAR_IN_OUT parameter.

2.11.5.4 Finding Errors in FBD

The FBD Editor will locate you close to the location of an error if you double-click the respective error message in the [output window](#) of the framework.

2.11.5.5 FBD Editor Online

When you have the FBD Editor in monitor mode, it will automatically start displaying live values of blocks, connections and margin bar entries as far as possible.

If the online editor can't get a value of a variable from the runtime system, it will display "-!-".

OpenPCS supports "online edit", for further information see [Online Edit](#) in the user manual.

2.11.5.6 Keyboard handling for CFC and FBD editor

See OpenPCS Tools->CFC Editor->Advanced CFC Topics->Keyboard handling for CFC and FBD editor for information.

Or start reading here: [Fundamentals for keyboard](#)

2.12 Test and Commissioning

2.12.1 Test and Commissioning: Introduction

Test and Commissioning is the tool to maintain all online operation of OpenPCS. Use the T+C to monitor the value of variables, to start and stop your controller, and to change online blocks while running the application.

2.12.2 Start and Stop

Test and Commissioning supports three different ways of starting the application: "Cold Start" will reset all variables to their initial value, "Hot Start" will not reset any variable, while a "Warm Start" will re-initialize only those variables which are not declared [RETAIN](#).

2.12.3 Watch variables

During a program test, it is important to know which values the variables have, or which value produce an error. Therefore, we have the possibility to watch variables.

Change to the Resource-Pane.

Open the branch of the task the variables you want to watch belong to.

Double click on the variable which you want to watch.

The variable appears in the "Test and commissioning"-window where instance path, type, value, and status are displayed. These variables are permanently updated during the program execution on the PLC. If OpenPCS can't get a value for a variable from the runtime system (e.g. the variable is not available in the currently running program), a "-!" is shown

To remove variables from the list you have three possibilities as well. Mark the variable by clicking it with the left mouse button then: click on the corresponding symbol in the toolbar or use the "del"-key or select the item **Remove Variable** in the menu "Edit".

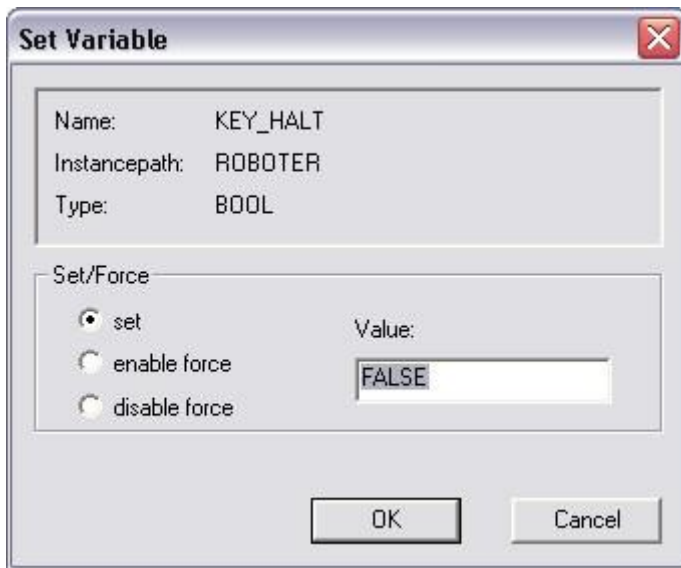
Double click on an array variable opens a dialog where you should enter the index you want to watch. Indexes for multi-dimensional arrays have to be comma separated.

2.12.4 Set variables

To influence the behavior of your control program for test cases, you can set variables to specific values. Mark the variable in the T+C, and select the menu item **"PLC->Set variable"**, or click directly on the variable in the T+C. Enter the new value and accept by "Set"-button. See also [Force Variables](#)

2.12.5 Force Variables

Besides watching and setting values of variables, OpenPCS support "forcing" of variables. If a variable is forced, the value will be reset to the value specified at the end of each cycle (before writing to the outputs). Forcing is controlled by three buttons labeled "set", "enable force" and "disable force" in the variable set dialog:



In the column "Force" of the Test & Commissioning-Window, OpenPCS will display if a variable is currently forced or not.

The action performed when pressing OK depends on which of the three buttons "set", "enable force" and "disable force" is selected:

if the variable is currently **not forced**, "set" will once set the variable to the value specified. If the variable is modified by the application, this might have a very short effect only. "enable force" will force the variable to the value specified, i.e. set the variable to the specified value at the end of each cycle, "disable force" will have no effect

if the variable is currently **forced**, "set" will disable forcing for this variable and set the variable once to the value specified, "enable force" will continue to force the variable, but with the value specified now, "disable force" will not set the variable, but only disable forcing for the variable

Please note

Forcing only resets the variable at the end of each cycle. Modifications during one cycle are possible and not prevented.

Forcing is not restricted to directly represented variables (AT %...)

Removing a variable from the watchlist will automatically disable forcing this variable

2.12.6 Working with watchlists

The Test & Commissioning's list of variables can be saved to a so-called Watch List file. This allows for switching between different Watch Lists while being online.

There is always a default Watch List file with the name *<name of your resource>.WL* in the project root directory.

While online, a Watch List is saved through the main menu command: **SPS -> Save Watch List As...**

The saved Watch List will then show up in the Browser's File pane. After saving, all subsequent modifications of the variable list will be stored in this Watch List.

To restore a different saved Watch List simply open it by double-clicking it in the Browser. Or by choosing File->Open while the Watch List is selected in the Browser.

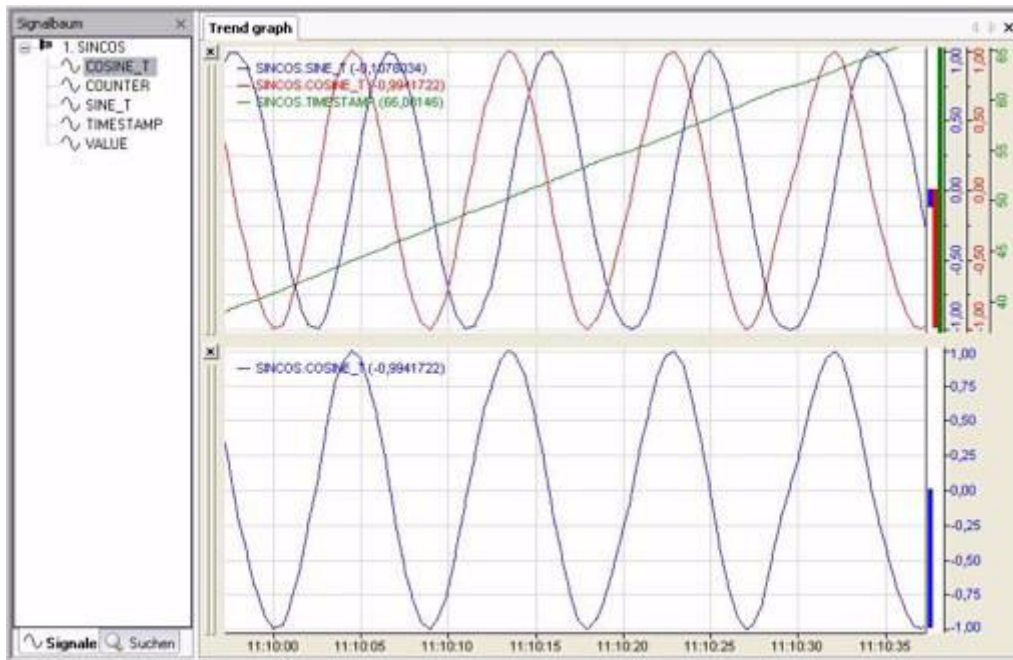
An empty Watch List can be created by selecting **File->New / Others / Watch List**.

2.13 Control Data Analyzer

2.13.1 Control Data Analyzer

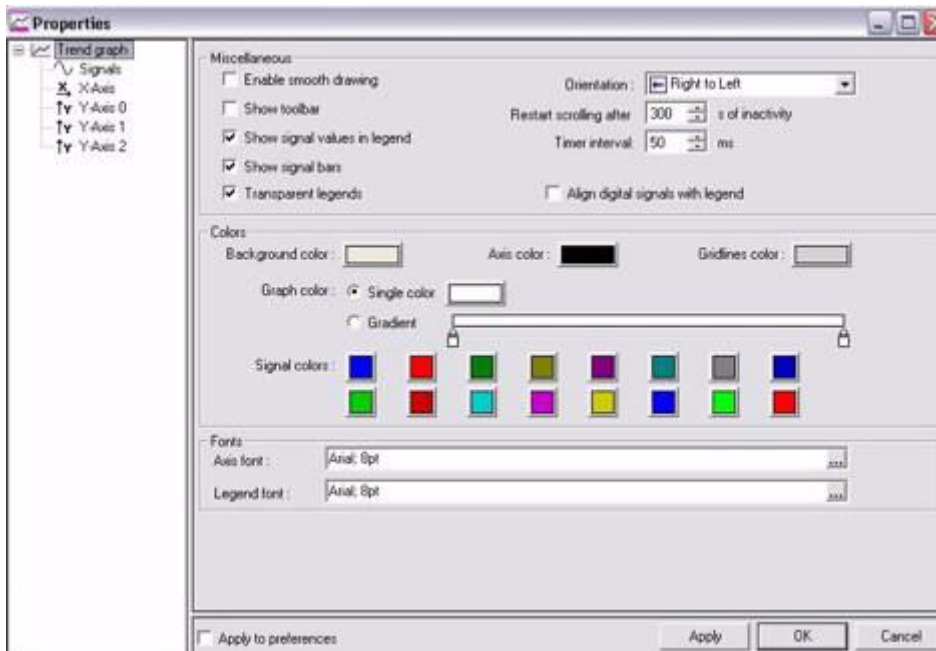
Control Data Analyzer hands the user the opportunity to plot the development of variables stored in the variable table of the controller.

Control Data Analyzer is only available while OpenPCS is online and can be started via **View -> Control Data Analyzer**. Control Data Analyzer will be shown in the area of the editor window.



The OPC variables are visible in the left column, called signal tree. The signal can be grouped by their type (analogue or digital) or the files, they are defined in. A new plot can be started via double clicking a variable. The user can append variables to the plot by dragging the variables onto the graph. Each graph gets a unique color and y-axis.

The user can open the properties via right-clicking within the plot.



The user can edit the colors and fonts of a plot and the general appearance of the analyzer. Under signals, the user can give graphs an offset and differ the design. The axis can be set up in the respective sections.

Notes:

To use the Control Data Analyzer the used PLC has to support TFR (Transient Fault Recorder). This is a real-time database for all runtime variables configured to be captured by the TFR. The runtime system provides the data through its standard communication channels of the online server. The infoteam SmartSim and the infoteam SmartPLC/OPC do so.

2.13.2 Oscilloscope

The oscilloscope is a Cartesian coordinate system displaying signals over time. Markers support measuring tasks. Three modes and triggers support display control.

Signals are added by the signal tree. The trigger event is displayed at the middle of the x-axis. Each time data is relative to that event. The y-axis is centered to 0.

The toolbar enables switching between three modes: Run, Stop and OneShot:

Run: the arriving data is constantly checked to the defined trigger. If the trigger event happens, the signal chart is centered on it.

Stop: The triggering is stopped immediately; thereby a happening trigger event is irrelevant. The view switches to the navigator.

OneShot: Once a trigger event is identified, the mode switches to Stop mode.

Note: The toolbar can be activated via the context menu.

2.13.3 Trigger

The current trigger can be accessed via the context menu of the caption, the trigger button or the oscilloscope properties dialog.

There are two different triggers: a fixed one and a delayed one. The fixed one is just a simple trigger, the delayed one consists of three simple triggers: A, B and Reset. The delayed one is only satisfied, if B happens after A before Reset happens.

A simple trigger is satisfied depending on the set conditions and its mode. Conditions are value and kind of edge of the connected signal.

The value may only be in whole numbers - the data of the signal is rounded down.

The edge of the signal can be rising, falling, alternating (vertex), or "rising or falling".

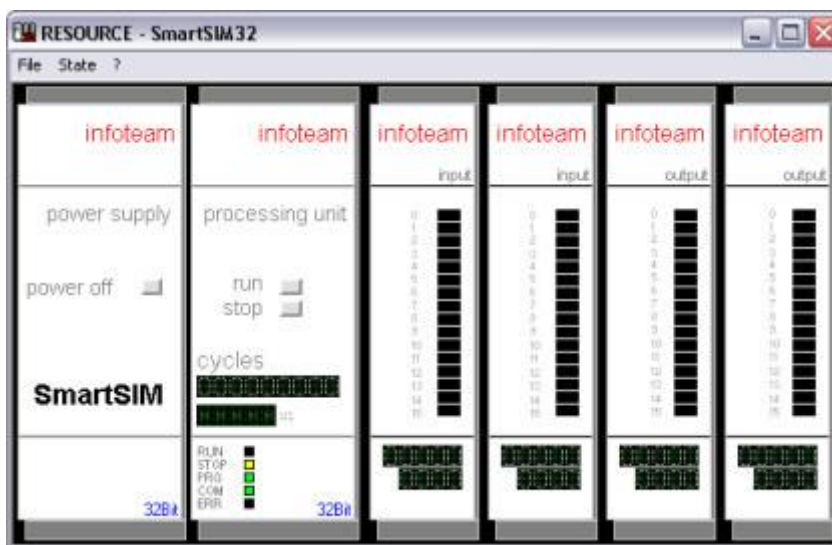
Depending on the mode the trigger is treated:

1. Normal: the trigger is satisfied if all conditions match.
2. Automatic: the trigger is satisfied if all conditions match or a defined count of data samples is processed without matching conditions.
3. Automatic (level): the trigger is satisfied if all conditions match. If the conditions do not match for a certain count of data samples, the value is automatically set to the half of the amplitude of the signal and the flow continues as in automatic mode.

2.14 SmartSIM

2.14.1 Overview SmartSIM

In order to test a program, you need a PLC. You can use the real control system which you bought together with OpenPCS. To be independent of the different control systems in this manual, with which OpenPCS is distributed, we use only the SmartSIM32 here.



SmartSIM32 can be started as a stand-alone executable, but typically it will be launched automatically by OpenPCS as soon as you try to go online with a resource which is using [connection](#) "simulation".

Note:

To prevent SmartSIM from loading the program stored on disk, keep CTRL and SHIFT pressed at start of SmartSIM.

2.14.2 Interrupt Tasks

The SmartSim also supports running interrupt tasks. A interrupt can be provoked by pressing a key on the keyboard. So the only thing you have to do is to bind a key to the interrupt task. For that select the task properties in the [resource pane](#) of the task, be sure that its [tasktype](#) is set to interrupt and name the field interrupt to a key.

Note:

1. Possible keys are "+", "-", and all ASCII signs in the interval from 0x1F to 0x7F.
2. To provoke the interrupt pressing a key, the SmartSim needs to have the focus on.

2.15 OPC Server

2.15.1 About OPC Server

The infoteam SmartLink OPC server is a gateway between OPC clients and one or multiple controllers based on IEC 61131-3 runtime systems. It is available at www.infoteam.de for free. A demo connection to the infoteam SmartSim is supplied.

2.15.2 Remote OPC Server

For configuring the remote access from OPC-Clients to OPC-Server SmartPLCDA via DCOM, please follow the following steps.

Check Requirements:

OS Requirement: WindowsXP with SP2

The password must not be blank or "admin".

For performing the next steps, you must have administrative privileges to the local PC to change the DCOM settings.

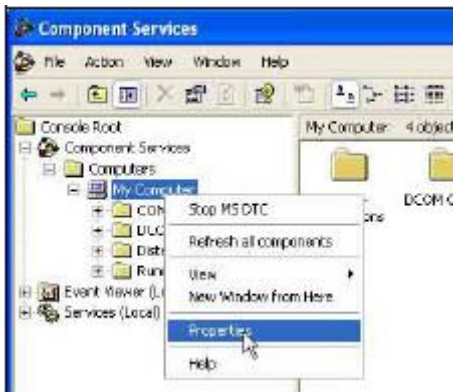
DCOM must be enabled in the hardware.ini-file

```
[ENABLE_OPC_PANE]
DCOM=1
```

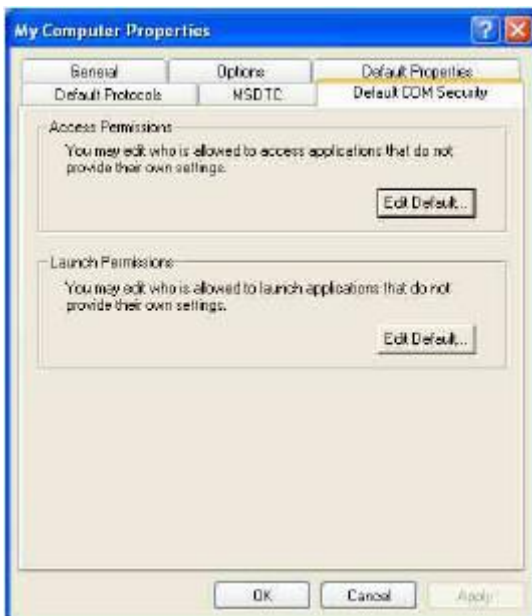
Setting up DCOM communication

Select **Start** -> **Run** and type "dcomcnfg", this opens the component service manager of WindowsXP.

Go to Console Root -> Component Services -> Computers. Right click to My Computer and select Properties.



Select the **Default Properties** tab and select the options as in figure below:

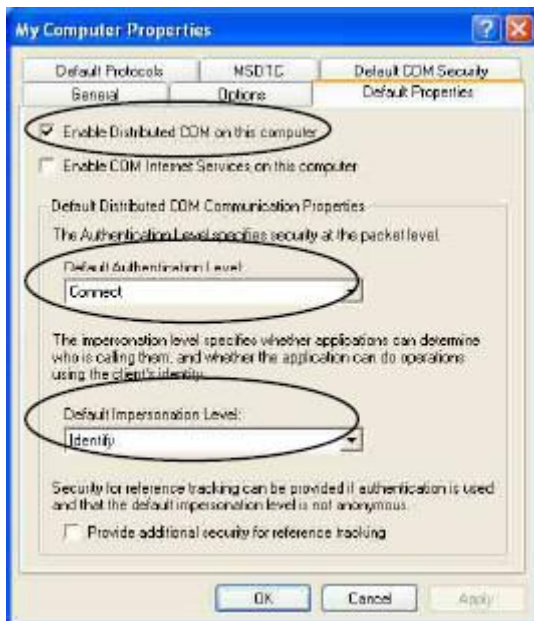


Setting permissions for DCOM

Default permissions have to be changed to establish communication.

Select the **COM Security** tab

Select Edit Default for Access Permissions



Make sure the access permissions window contains at least the following entries:

1. Everyone
2. Interactive
3. Network
4. System
5. Domain Administrators
6. Domain Users

If these entries are not shown, click the **Add** button, then click **Advanced** and **Find Now**. This lists all groups and users (if not, make sure that for object types **Groups** is selected). Select the groups and press OK, also in the next window. This adds these groups.


For domain users proceed as described in 3., but for location specify the domain name.

Repeat steps 2. to 4. for **Launch and Activation Permission**. Afterwards close the **My Computer Properties** window. Now the setting changes are finished.

The computer has to be restarted to accept changes.

2.16 Online Server

2.16.1 Online Server: Overview

The Online Server is the gateway between all OpenPCS tools and your controller(s). It is started automatically in the background and routing all traffic and commands between all applications and the controller. A small icon  will be displayed in the system tray when the Online Server is active. A click on that icon will offer to terminate the server, which you should generally avoid to do.

2.16.2 Online Server Setup

2.16.2.1 Online connections: introduction

"Connections" are symbolic names for potential links to your controller. Each resource is configured to use exactly one connection, and each connection is bound to one "driver" and given a parameter set for that driver. OpenPCS comes with several drivers but by default only one connection, which is bound to the IPC driver.

By default, the following drivers are available:

IPC - Interprocess Communication, for a connection to the SmartSIM

TCP - For a TCP/IP connection to target systems older than version 4.3.1

TCP_432 - For a TCP/IP connection to target systems with versions 4.3.1 or newer

RS232 - For a serial connection to target systems with version 4.0 or newer

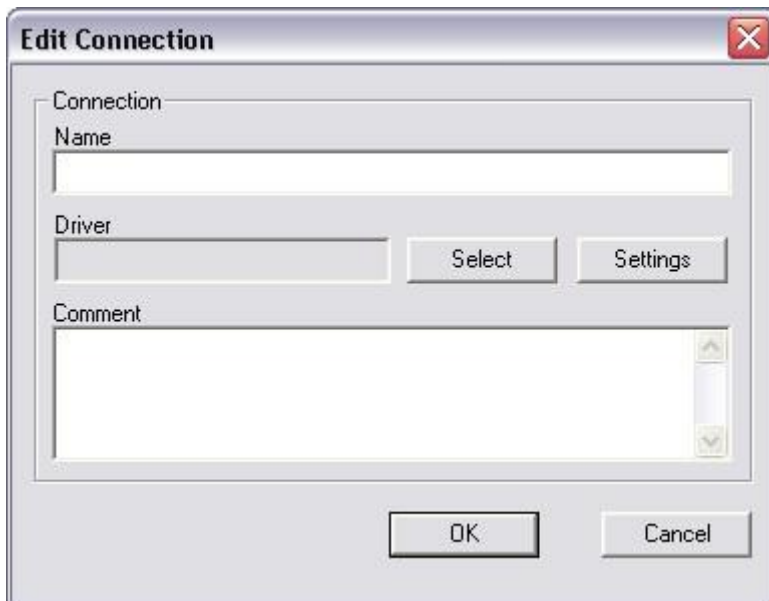
RS232_35 - For a serial connection to target systems older than version 4.0

2.16.2.2 Create new connection

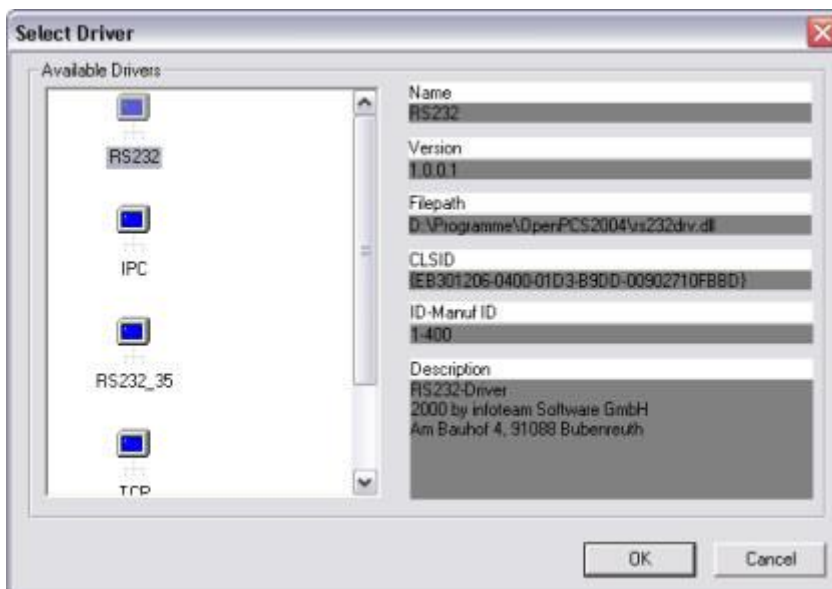
Choose PLC->Connections...

When the Connection Setup Window appears, click on the new button.

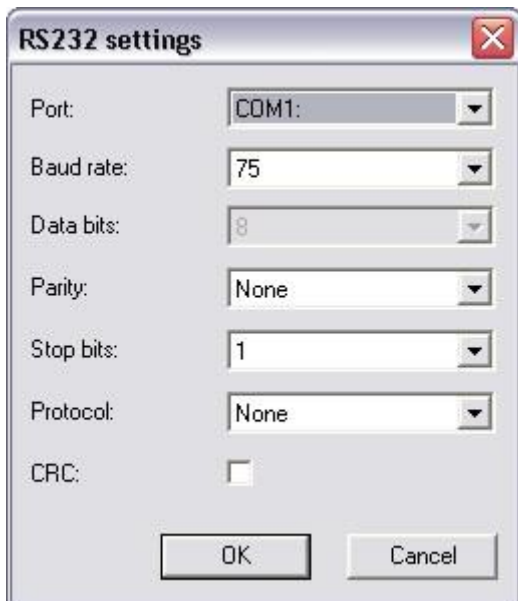
The Connection Properties dialog box opens:



Now enter the name of the connection to be created. Take care that the connection name has not got any spaces in it. Use the underscore (" _ ") instead of space. Select a driver by clicking on the select button, the Driver Select window showing driver settings opens:



Click the desired driver and then the OK button. The Connection Properties window now becomes active again. Click the settings button. The Driver's Settings dialog box opens:



You can modify the settings individually. When you agree with the adjusted settings, just click OK, otherwise click cancel. If you want to place a remark, you can do this at the Connection Properties window. Finally, click OK from the Connection Properties window.

2.16.2.3 Delete Connection

Select a connection by clicking on the connection's name from the Available Connections list. Then click the **"Remove"** button, the selected connection is deleted and disappears from the list.

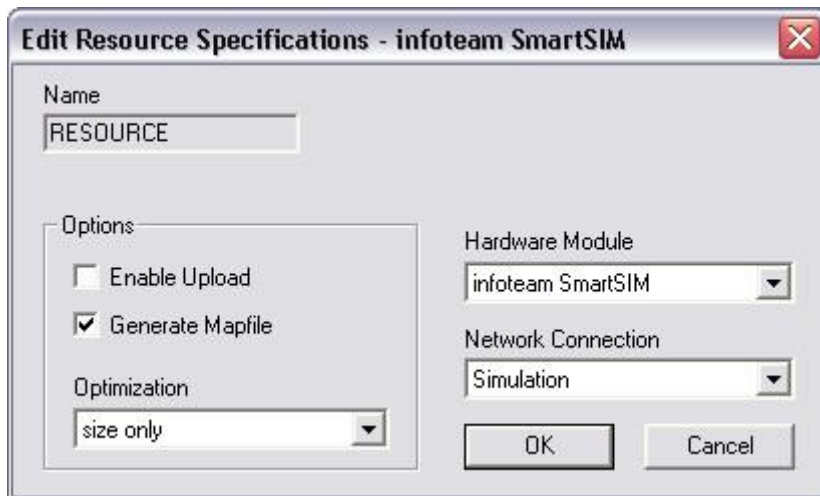
2.16.2.4 Edit connection properties

By clicking on the connection's name from the Available Connections list, it will become selected. Click on the **Properties** button and the Connection Properties dialog box appears.

Go on like as described in the topic [Create a new connection](#).

2.16.2.5 Select Connection

Right-click the active resource (marked green) in the browser and choose "Properties..." from the context menu, the "Edit Resource Specifications" dialog box opens:



To select the connection use **Network Connection** selection bar.

2.17 Hardware drivers

2.17.1 Hardware drivers: Overview

With the tool AddDriver (AddDrv.exe) you can automatically install driver packages into OpenPCS that are supplied by your OEM-manufacturer. These driver packages are MS-Cabinet files (extension .CAB) with defined content, which are shipped together with OpenPCS by your hardware manufacturer or, e.g. are offered in the internet.

If AddDrv.exe is executed without command line parameters (e.g. from the menu **Extra -> Tools -> Driver Install...**) the desired driver package can be chosen in a dialog box.

2.18 Compiler

2.18.1 Compiler: Overview

The Compiler is the tool converting the sources of your application that you write in different editors into executable code, which is [UCODE](#) or [NATIVE CODE](#).

The Compiler is automatically launched by the Browser whenever there is need to recompile your application, and the Compiler can manually be invoked from the Framework. In general, there should be no need to invoke the compiler from the command line.

2.18.2 Instruction List Compiler

2.18.2.1 Compiler Command Line

```
ILC [ -s | -c | -p | -i | -l ] <Poe file names> -v <Varfilename> -d <Devicename> -r  
<ResourceName> { -o <OutputDir> } { -g | -m } { -x } { -b } { -y } { -w<OutputLevel> }
```

ILC command <Poe file names> -v <Varfilename> -d <Devicename> -r
<ResourceName> options

The commands and options are preceded by either a slash (/) or a dash (-) and are not case sensitive. Multiple commands per call are allowed.

Commands:

-s: performs a IL syntax check on the files following the command

-c: compiles all files following the command

-p: creates prototypes for all program organization units specified in the list of files following the command

-i: creates include files for all program organization units specified in the list of files following this command

-l: creates a file with all dependencies for the program organization units specified in the list of following this command.

If more than one command is used in a call of ILC.EXE than every single command applies to all files following the commands.

Options:

-o: with this option the user can specify an output directory or the target path name if a single resource is built. If -o is followed by a directory name the targets are stored in the specified directory.

-g: the input files (*.poe files) are no located resource global variables

-m: the input files (the files specified after the command) are located resource global variables

-x: dump object files

-f: suppress final error information

-b: Use short file names. ILC presupposes that the filenames of POUs whose names are longer than eight characters are cut down to eight characters.

-w: by using this flag the user can specify the output level for the output information. It has to be followed by an integer that indicates the output level. The following values are defined for the "OutputLevel":

0: print all available informations. I. e. errors, warnings and info-messages.

2: suppress warnings.

4: suppress info-messages (e.g. Compiling c:\test\test.poe).

6: suppress warnings and info-messages.

Please note that error messages are printed always and can not be suppressed.

-y: write the initial data segment of the POUs to be compiled in a text file. This command should be used, to obtain the initial data segment of firmware POUs for the <hardware>.ini file.

2.18.3 Linker

2.18.3.1 Linker Command Line

ITLink [-r | -t] <source file names> -v <Varfilename> -m <makefile name> {-g <resource global object filename>} {-s <packed sources file>} {-o <OutputDir>} {-x}

The commands and options are preceded by either a slash (/) or a dash (-) and are not case sensitive. Multiple commands per call are not allowed.

Commands:

-r: link the files specified in the file list following the command to a resource object (*.pcd file).

-t: link files specified in the file list following the command to a task object (*.crd file)

In both cases the linker requires the project file path and the make file path. The project file path has to be prefaced by "-v" and the make file path by "-m".

Options:

-o: specify an output directory or the target path name if a single resource is built. If -o is followed by a directory name the targets are stored in the specified directory.

-g: the input files (the files specified after the command) contain object information about the resource global variables.

-s: the file specified after the option contains the packed sources of the resource. The content of this file is inserted in the resource global segment table. This option is valid only in combination with the command "-r".

-x: dump object files

-w: by using this flag the user can specify the output level for the output information. It has to be followed by an integer that indicates the output level. The following values are defined for the "OutputLevel":

0: print all available information. I. e. errors, warnings and info-messages.

2: suppress warnings.

4: suppress info-messages (e.g. Compiling c:\test\test.poe).

6: suppress warnings and info-messages.

Please note that error messages are printed always and can not be suppressed.

2.18.4 Make

2.18.4.1 Make Command Line

```
ITMake [ -p | (-m <makefilename> | -u <ArchiveFileName> -v) | -y ] <Varfilename> { -o <OutputDir> | <Outputfilename> } { -i } { -s } { -w<OutputLevel> } { -b }
```

The commands and options are preceded by either a slash (/) or a dash (-) and are not case sensitive. Multiple commands per call are not allowed.

Commands:

-p: build all resources in the specified project(s). This command must be followed by at least one project-file-path (VAR-file-path).

-m: build the specified resources. This command must be followed by a list of resource file names and the specifier -v followed by the project-file-path. The resources to be build, must all be part of the project specified by the VAR-file-name.

-u: uncompress archive. This command must be followed by a file name which contains the archive to uncompress.

-y: create the initial data segment of all POUs in the project and write it in a text file. This command should be used, to obtain the initial data segment of firmware POUs for the <hardware>.ini file.

Options:

-o: with this option the user can specify an output directory or the target path name, if a single resource is built. If -o is followed by a directory name the target(s) is stored in the specified directory.

-e: this option is reserved for future implementations. It enables the user to specify a target path for output messages. If an error file is specified, the user output is printed in the specified file instead on stdout. If -e is followed by a directory name an error file is created for every compiled or linked entity and the error files are stored in the specified directory.

-i: Incremental build, i. e. only changed files and files affected by this changes are recompiled and linked.

-s: Use short file names. ITMake presupposes that the filenames of POUs whose names are longer than eight characters are cutted down to eight characters.

-w: by using this flag the user can specify the output level for the output information. It has to be followed by an integer that indicates the output level. The following values are defined for the "OutputLevel":

0: print all available information. I. e. errors, warnings and info-messages.

2: suppress warnings.

4: suppress info-messages (e.g. Compiling c:\test\test.poe).

6: suppress warnings and info-messages.

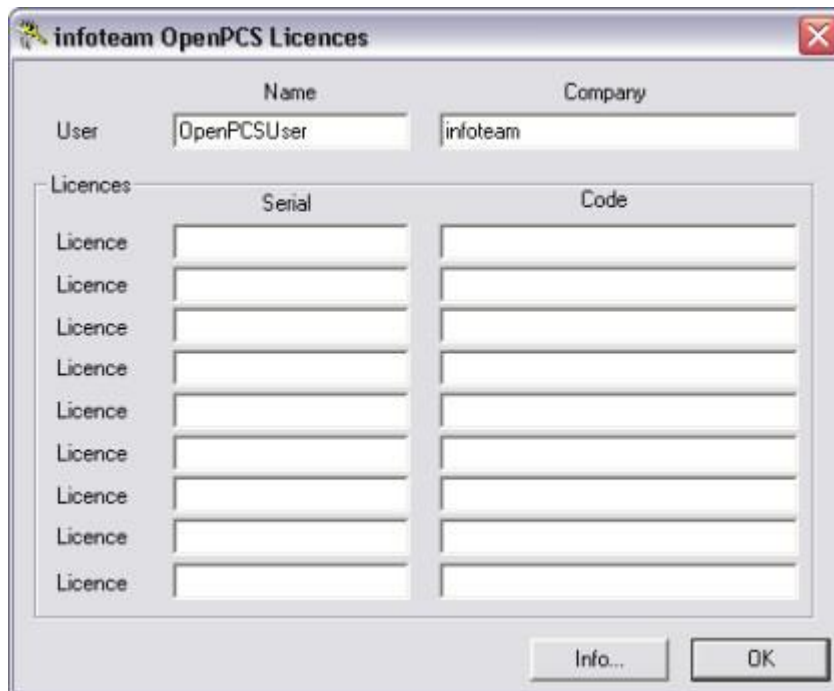
Please note that error messages are printed always and can not be suppressed.

-b: use short file names. If this option is specified and the name of a POU exceeds eight characters, the compiler uses the first eight characters of the POU name as it's file name.

2.19 Licence Editor

2.19.1 Licence Editor: Overview

The Licence Editor is invoked automatically during Setup of OpenPCS. It can later be launched by selecting "Licence" from the Help menu of the OpenPCS Browser:



	Name	Company
User	OpenPCSUser	infoteam

Licences	Serial	Code
Licence		
Licence		
Licence		
Licence		
Licence		
Licence		
Licence		
Licence		
Licence		

Info... OK

Enter your name and company information on top, and up to nine pairs of serial numbers and licence codes in the boxes below.

Notes:

1. If you have got a valid licence, run this licence.
2. For details on licences installed, press "Info".

2.19.2 Usage without Licence Key

Without a licence OpenPCS is still full functional, but restricted to the [Simulation](#) .

3 Advanced Topics

3.1 Runtime issues

3.1.1 Multitasking

Multitasking is highly target dependent. The behavior described here is the standard as implemented by OpenPCS, but can be different for any particular target device. If in doubt, check with the supplier of your controller.

OpenPCS supports all three tasks types defined by IEC61131-3:

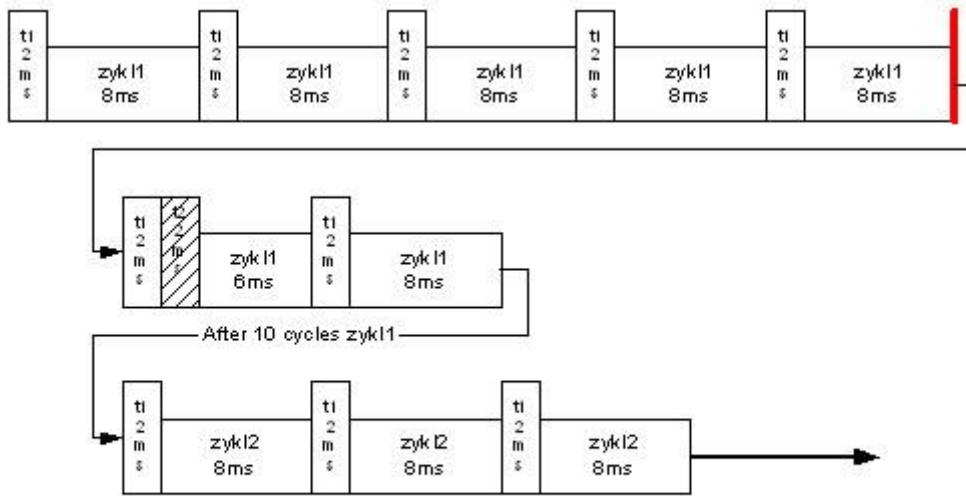
Cyclic tasks will be executed when no timer or interrupt tasks are ready to run. The priority that can be specified in the task properties will be interpreted as a cycle interleave, e.g. priority = 3 will have this task executed only every third cycle. No particular execution order is defined by OpenPCS amongst multiple cyclic tasks

timer tasks will be executed every N milliseconds, with N specified in the task properties.

[Interrupt tasks](#) will be executed as soon as the interrupt occurs they are linked to. The SmartSIM simulation does support interrupt tasks (see [how to provoke them](#)).

As an example a resource has four tasks like in the following table:

Task	time	Priority
zykl1	needs 40 ms	1
zykl2	needs 20 ms	10
timer1	every 10ms; needs 2ms	1
timer2	every 50ms; needs 2ms	2



3.1.2 Interrupts

Interrupt tasks are executed immediately after an interrupt rises. There are three different interrupt types pre-defined in OpenPCS supported by infoteam SmartSim and infoteam SmartPLC/OPC. Please see documentation of your OEM for supported interrupt types.

Interrupt types:

1. STARTUP: rises each time the PLC is started (Coldstart/Warmstart/Hotstart).
2. STOP: rises each time the PLC is stopped.
3. ERROR: rises each time an error is risen.

These tasks are executed once. To gain specific information and manipulate it use firmware function blocks like [Resume](#) and [ETRC](#).

3.1.3 Optimisation Settings

OpenPCS supports optimization settings "speed", "size" and "normal". "Size" will generate only UCODE and no native code. A native code compiler will not be invoked. "Normal" will compile UCODE and native code with mixing of both enabled for optimum debugging support. If the native code compiler supports "direct calls", these will not be used. "Speed" will yield an error message if any portion of the application cannot be compiled with native code. If the native code compiler supports direct calls, these will be used.

3.1.4 Multiple Resources

OpenPCS supports multiple resources. To work with multiple resources, just create new resources in your project. Remember that source code (from the project in the file-pane) can easily be used on different resources. To go online, download or compile, set the [active resource](#) before.

If you have not only a few, but plenty of resources (20, 50, or 200) that you need to update, download and start simultaneously, there are "batch online" features available as options to OpenPCS. Contact your OEM or infoteam for availability.

3.1.5 Variable Address

In some applications, it is necessary to determine at run-time the address of a variable, given its name. Some Human Machine Interfaces, and some Network interfaces rely on that being possible.

OpenPCS can download symbolic information to support this. Compile Option "[Mapfile](#)" needs to be enabled. Use function block [GetVarFlatAddress](#) to get address information then.

As applications can contain many symbolic variables, only a filtered sub-set is downloaded to the controller. By default, only global variables are downloaded, but this filtering is open to modification by OEMs, e.g. to filter by variable name.

3.1.6 Performance

There are some obvious and some not so obvious factors influencing the performance of your application:

Obvious factors include the performance of your platform, including I/O and networks, the size of your application, measured in lines of code or in bytes required at runtime. A native code compiler, if available, will typically increase performance.

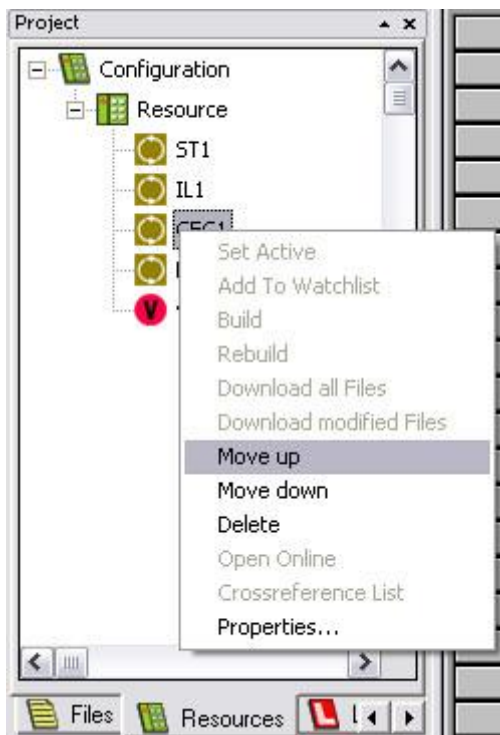
Not so obvious factors include

1. **Task structure** of your application: more tasks will typically reduce performance due to additional overhead in task switching. Removing code from fast, or cyclic tasks, and moving it to less often executed timer tasks, or executing them only when needed in interrupt tasks, can on the other hand tremendously increase throughput.
2. While **native code** typically executes faster, task switching is less responsive. So even when native code is available, there is reason to leave the cyclic task with UCODE ([optimization](#) size) and have only timer and interrupt tasks in native code for maximum performance and minimum task switch and jitter.

3. While all code compiled to the target uses Instruction List as the common intermediate language, code produced by the **different editors** varies. While for some applications some languages might be the best choice, things can be completely different for other applications. Carefully evaluate the different editors and languages and pick your favourite ones for the different applications.

3.1.7 Adjusting order of cyclic tasks

The compiler uses the task order of the resource makefile, as shown on the resource pane, to link the tasks into the resource. The runtime system uses this order to execute cyclic tasks. On the resource pane of the browser two new context menu entries have been added. With "Move up" and "Move down" the order of the tasks can be modified. The priority, set in the task properties still has the same meaning for cyclic tasks.



3.2 Native Code Compiler

3.2.1 Native Code

OpenPCS supports UCODE and NATIVE CODE.

Native Code is optional, but available for most platforms. While UCODE is optimized for portability between platforms, fast task switches and small memory footprint, NATIVE

CODE is optimized for execution speed on one particular platform. On application level, the programmer can select which code to use via the [Optimisation Settings](#) on resource or task level.

Some debugging features of OpenPCS are available with UCODE only, so if you intent to use one of these, be sure to set [optimisation](#) to "size": this includes [Breakpoints](#) and online display in Ladder Diagram as well as the POU Editor, Online Change.

The OpenPCS RunTime System typically allows for task switches to be triggered after each and every UCODE instructions, i.e. quite frequently even within small loops. This is not possible with native code. If no direct calls are being used, the run-time checks for task switches at each call/return. With direct calls, it checks for task switches only after one task has completely finished.

Execution of native code typically is faster than execution of UCODE, with the speed-up factor depending on the processor, the implementation of the run-time system and the native code compiler, the C-compiler used to compile the run-time system, the application, the memory architecture, e.g.: if not using "direct calls", calling a function block with native code may be slower than calling the function block with UCODE, due to calling convention conversion. Hence, if your application uses plenty of tiny blocks, speed-up may be below expectation. With complex instructions (e.g. sine or real-divide), UCODE overhead may be comparably small, hence the speed-up gained with native code may be small as well.

The virtual UCODE machine within the run-time system will do some checks while executing the UCODE. In native code, most of this checking is omitted because it would add significant run-time overhead and the ultimate reason to use native code is speed improvement. It is strongly recommend that you carefully test and debug your application with UCODE first before using native code. If you need a native code compiler containing checking, contact infoteam. Checks not available with native code include array subscript overflow/underflow and string length checking

3.2.2 Direct Calls

Some Native Code Compilers implement a feature named "direct calls". This will avoid overhead during switching of function blocks, i.e. with CAL and RET, executing these directly as a call to a subroutine, and return respectively, in assembly language. This yields performance gain, but reduces responsiveness to task switching. If you are using one task only, this should greatly increase performance, if you are using multiple tasks, this could result in much larger jitter.

Direct calls will only be used with [optimization](#) speed.

3.2.3 Exception Handling in native code

Unless noted otherwise, code created by the native code compilers will, for performance reasons, not check for exceptions, like division by zero or invalid array index. Therefore it

is recommended that you carefully test your application using [optimization](#) setting size first, and only then switch to native code.

3.2.4 Unknown instructions

While infoteam Software attempts to build all native code compilers in a way so they can always compile the full UCODE instruction set, there is always the possibility that some native code compiler can not compile some UCODE instruction, for a variety of reasons.

If that happens, the Native Code Compiler will report this and not create native code for the POU containing that code. A double click on that message in the Browser's Output Window should locate you to that line.

If optimization is set to "speed", this message will be an error. Either modify your code to not contain that instruction, or set optimization to "default".

In optimization "default", this POU will be executed in UCODE, while all others would be executed in native code, if possible.

3.2.5 Span segments

With OpenPCS, all segments are limited to 64k Bytes in size. As native code typically is larger than UCODE, this can easily lead to the fact that some application can only be run in UCODE and not in native code due to this limitation. Some native code compilers implement a feature "span segments", which will allow segments to exceed 64k for that reason.

3.2.6 NCC Intel Protected Mode

The NCC86 will check for invalid array index and division by zero [exceptions](#).

The NCC86 does implement the [span segments](#) feature.

The NCC86 does implement the [direct calls](#) feature.

3.2.7 NCC Infineon C16x (huge model)

The NCC167H does implement the [direct calls](#) feature.

3.2.8 NCC Motorola 68K

The NCC68K does implement the [span segments](#) feature.

The NCC68K does implement the [direct calls](#) feature.

3.2.9 NCC Hitachi H8/300H

The NCCH8300H will check for [exceptions](#) invalid array index and division by zero.

3.2.10 NCC Motorola DSP563xx

This NCC is only available in custom versions of OpenPCS.

3.2.11 NCC Intel Real Mode

This NCC is only available in custom versions of OpenPCS.

3.2.12 NCC Motorola PowerPC

The NCC will check for invalid array index and division by zero [exceptions](#).

The NCC does implement the [direct calls](#) feature.

3.2.13 NCC ARM ARM Mode

The NCC ARM will check for invalid array index and division by zero [exceptions](#).

The NCC ARM does implement the [span segments](#) feature.

The NCC ARM does implement the [direct calls](#) feature.

3.2.14 NCC ARM THUMB Mode

This NCC is only available in custom versions of OpenPCS.

3.3 Documentation

3.3.1 Crossreference

See also [Cross-Reference \(per variable\)](#) and [CFC Crossreference](#).

To create a cross reference list for your project, right-click the active resource and select "crossreference list..." from the context menu.

A preview of the cross reference will be displayed, which can either be viewed and navigated online, or printed.

3.3.2 Cross-Reference (per variable)

To view a cross reference list for a certain variable right click the variable in the resource pane and select "Crossreference List" from the context menu.

See also [Cross-Reference](#) list for visualizing a Cross-Reference information.

3.3.3 Print IEC61131 Configuration

In order to get a printed documentation of the configuration of your resource and tasks, select the configuration in the Browser's resource view and choose "Print Configuration" in the context-menu.

3.3.4 CFC Crossreference

The CFC cross-reference is a valuable aid in debugging and understanding execution of CFC charts.

The OpenPCS standard cross-reference is of limited use to CFC programmers, as most symbols listed in that cross-reference will be symbols which names have been created automatically by the CFC Editor and have no meaning to the programmer.

To create the CFC cross-reference, select File -> Crossreference, or print the chart to see the cross-reference on paper. The cross-reference stored in file is less legible, but better suited to automatic post-processing with third party tools (like grep, awk).

The CFC cross reference is listed in the form

source: name [chart] page line

destination1: name [chart] page line

destination2: name [chart] page line

where

1. source is a name on the right margin bar, i.e. designs a signal leaving one compound block
2. destination is a name on the left margin bar, i.e. designs a signal entering a compound block
3. name is the variable name automatically generated by the CFC editor for that signal. Use that name to find this signal in the Test and Commissioning Tool to monitor the value of that signal.
4. chart is a path of names of compound blocks. Use that to find the location either in CFC-Editor by opening one sub-compound block after the other in the specified order, or by locating the printed chart via the table of contents.
5. page is the page of the printout, where the corresponding source/destination is found.
6. line is the position of the connection at the block corresponding to the margin bar.

The entries are sorted by source/destination, refer to the file stored if you need other sort sequences.

Note: If IEC61131-variables are used as connectors, there may be more than one source line. They have the following form

varname{scope}: ...

where

varname is just the name of the variable.

scope is represents the [declaration section](#) of the variable.

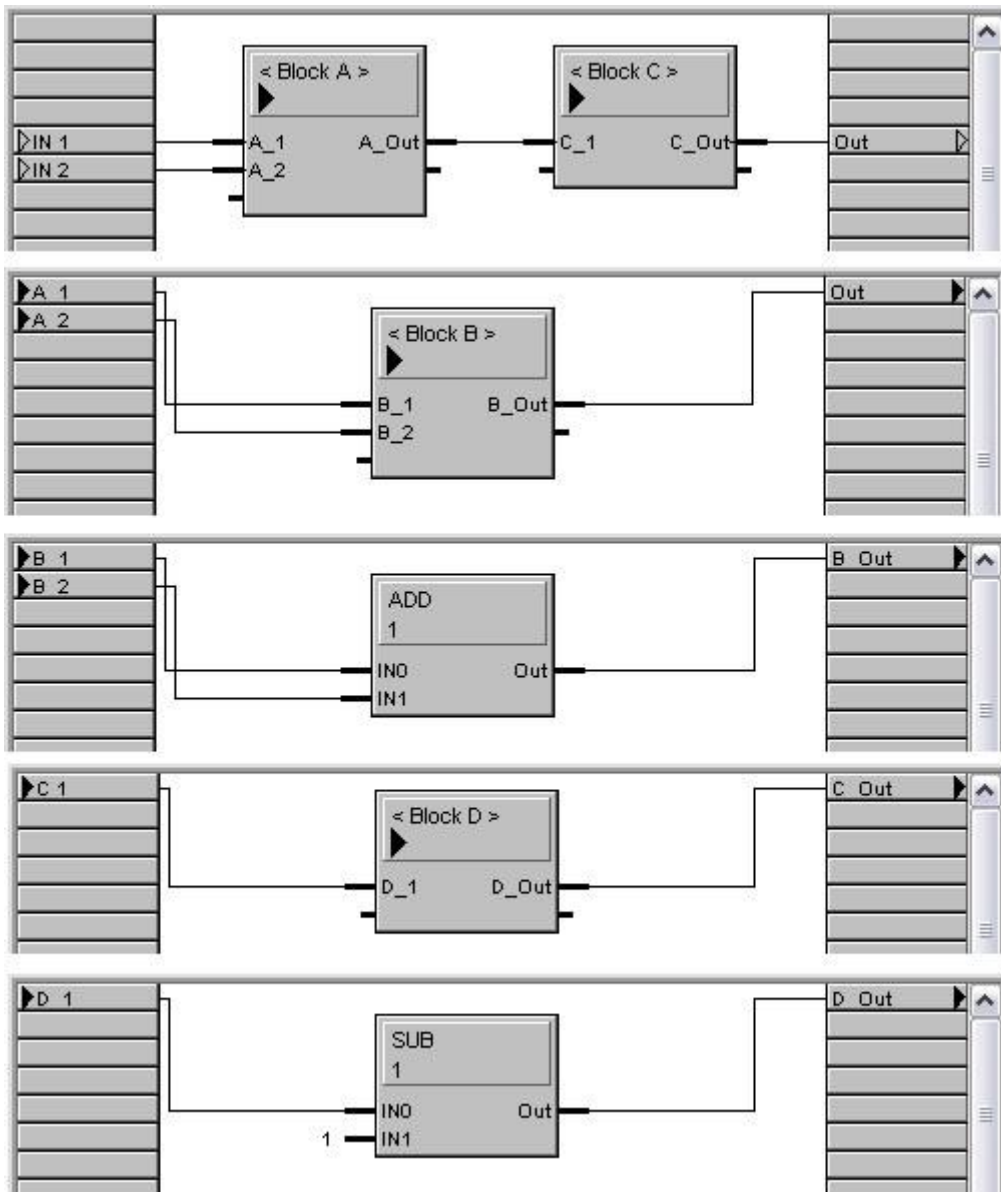
CFC Cross Reference sample

We use a small sample to demonstrate the CFC cross reference.

Set up a small CFC program, using two blocks (ADD and SUB), to add 23 to one input variable, then subtracting one from the result:



Now move block ADD into a compound block A and block SUB into a compound block C. Open block A and move ADD further down into a new compound block B. Open block C and move the SUB block further down into a new compound block D. Enter reasonable names for all margin bar entries. If you open all blocks, the result will look like that:



With this small sample, output of the CFC cross-reference will look like:

B_Out: FCT_10_10_10_1_ADD_OUT [SAMPLE.chart 1.Block A.Block B] page 4 line 5

D_1: FCT_10_30_10_1_SUB.IN0 [SAMPLE.chart 1.Block C.Block D] page 6 line 5

B_Out: FCT_10_10_10_1_ADD_OUT [SAMPLE.chart 1.Block A.Block B] page 4 line 5

D_1: FCT_10_30_10_1_SUB.IN0 [SAMPLE.chart 1.Block C.Block D] page 6 line 5

in1{VAR}:

B_1: FCT_10_10_10_1_ADD.IN0 [SAMPLE.chart 1.Block A.Block B] page 4 line 5

in2{VAR}:

B_2: FCT_10_10_10_1_ADD.IN1 [SAMPLE.chart 1.Block A.Block B] page 4 line 6

With this, the following questions are easily answered:

Looking at the ADD block: where does this output signal go to? Find the name of the output signal, B_Out. See cross-reference to find it goes to named D_1 in block chart1.BlockC.BlockD.

looking at the SUB-block: where does the input signal come from? Find the name of the input signal D_1, locate D_1 in the cross-reference and find it comes from B_Out. (as the list is sorted by source names, this is easier to find by opening the file with some editor than by looking at the printed cross-reference)

How can I monitor that signal entering the SUB-block online? Find the name of the SUB-blocks input in the margin bar (D_1), locate that in the cross-reference and read the name of the IEC61131-variable associated to it (FCT_10_30_10_1_SUB.IN0). Find that variable in the Browser's instance tree and double click it to have it added to the watch list.

3.3.5 Print-Options

All OpenPCS tools support forms for printing, and will automatically use the print form assigned to the project. If no print form is assigned to a project a default header and footer is printed.

To configure the project print form choose **Extras->Tools->Print-Options**: There the print form can be selected, print form depending aliases (*) can be set, date and time format can be localized.

Additional options:

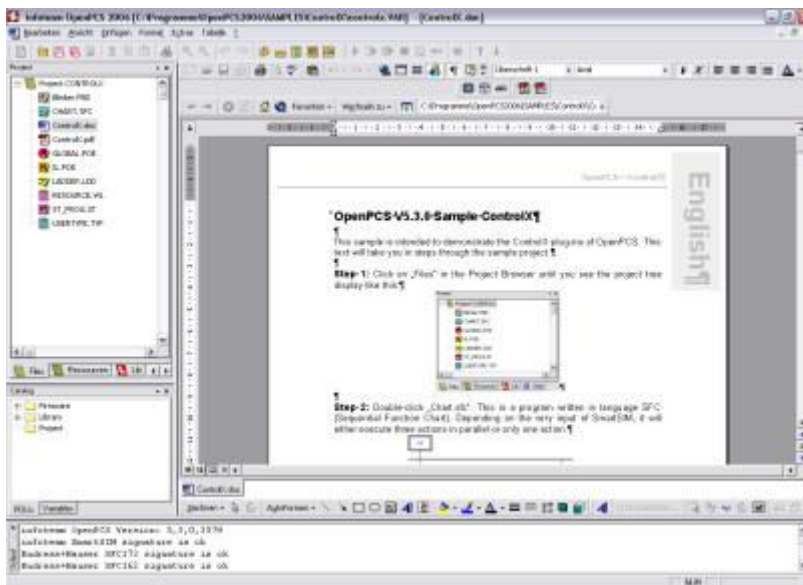
1. **Print front page**: a front page is printed consisting of the current print form and a rich text file "cover.rtf" located in the project folder. If "cover.rtf" is missing, only the print form is printed.
2. **Print comments**: cfc commentaries are printed on additional pages (after the corresponding chart) (only cfc).
3. **Print comments and flags**: cfc commentaries and an itemization of existing [connection flags](#) including name, global ID, page number and grid position are printed on additional pages (after the corresponding chart). (only cfc)
4. **Print comments on previous page**: The additional pages for comments are printed preceding to the the corresponding chart. (only cfc)

Note: The aspect ratio of the print form should correspond to the printer configuration (especially when the print form contains images). The size is independent.

3.3.6 Active Document Server

OpenPCS contains an Active Document Server Interface, this means that all registered active documents are supported by OpenPCS, can be opened by OpenPCS and can be edited by OpenPCS.

When opening such a file, the document is opened in the editor window part of OpenPCS as in the figure below.



Attention: Depending on the system configuration and installed applications with active document server, the files that can be edited by OpenPCS may vary from PC to PC.

Warning: If the active document server, which is not part of OpenPCS, is not stable, this will also lead to an unstable performance of OpenPCS.

3.4 Libraries

3.4.1 Library: Overview

Libraries are collections of functions and function blocks that can be re-used over different OpenPCS projects.

Working with libraries involves several steps: a library is first [created](#), pretty much like any other OpenPCS project. If creator and user are different, it is then distributed via

Floppy Disk, CD-ROM, or Internet, and made available to the user. The user will [install](#) the library, i.e. transfer the library to his own PC. To use a library with an OpenPCS project, the library has to be [added](#) to this project, this making the contents of the library available for use.

To get rid of a library within a project, the library can be removed from this project. This can be necessary if a different implementation of the same library should be used instead.

To remove a library completely from a PC, the library can be uninstalled. This can be necessary if the library should be used on a different PC and licensing conditions require it to be removed prior.

The following chapters will give a sample on how to do a library of your own.

3.4.2 Create a Library

To create a library, proceed just like creating any normal OpenPCS project. Be sure to perform a syntax check when finished creating POU's (functions or function blocks) in your library project.

If you receive a library from your supplier, you will not have to "create" that library. Proceed with "installing" this library instead!

Example:

*Start the Browser and create a new project named "MyLib" using **Project->New...***

Create a function block named "det_edge" (for edge detection): **New-> Functionblock->IL**. Implement this function block as shown below:

```
VAR_INPUT
    input : BOOL ;
END_VAR
VAR_OUTPUT
    output : BOOL ;
```

END_VAR

VAR

tempvar : BOOL ;

END_VAR

LD input

ANDN tempvar

ST output

LD input

ST tempvar

Invoke a syntax check with **File->Syntaxcheck** .

3.4.3 Install a Library

Before you can use a library, you have to install it on your PC. Use **Project->Library->Install New...**

Use the "browse"-button to locate the .VAR file representing your project. If you created the library yourself, this will be in the directory you specified when creating the library project with **Project->New....** If you received the library on a disk, this can be something beginning with "**A:**". During installation, the library project will be copied into a sub-directory of <windows>\openpcs.500\Lib.

Example:

Create a new project in the Browser using **Project->New....** Name that new project "TEST".

Select **Project->Library-> Install New....**

Now use the browse-button to locate the MyLib-project you created just before and press "Ok".

3.4.4 Adding a Library to a project

After installation, all files needed for the library will be present on your computer. But the functions and function blocks in that library will not be automatically available in your projects. You have to "add" the library to the project first using **Project->Library->Use in current project**.

Example:

Mark the Library "MyLib" in the Library-Pane and select **Project->Library->Use in current project** .

Create a new POU of type PROGRAM, named "main". Select **Insert->Functionblocks....** to see your library functions. To use your function block DET_EDGE, implement program "main" as shown below:

```
VAR
```

```
sig1 AT %I0.0 : BOOL ;
```

```
anEdge : DET_EDGE;
```

```
count : UINT ;
```

```
END_VAR
```

```
CAL anEdge (
```

```
input := sig1
```

```
|
```

```
:=output
```

```
)
```

```
LDN anEdge.output
```

```
JMPC ende
```

```
LD count
```

ADD 1

ST count

ende:

Compile that program, add it to a resource of your choice and execute it. Change input %i0.0 and see variable count incremented.

3.4.5 Uninstall Library

If you want to get rid of a library installed on your PC, make sure the library is not used any more, mark it and select **Project->Library->Uninstall**. In the dialog shown, select the library to get rid of and press OK.

Example:

Mark the Library "MyLib" in the Library-pane.

Select **Project->Library->Uninstall**. In the dialog, select <Windows>\openpcs.500\MyLib".

Press OK, and "MyLib" is no longer available as a library.

3.5 CANopen

3.5.1 CANopen: introduction

This manual describes the integration of CANopen services in PLC programs according to the IEC61131-3 standard. Such CANopen integration allows use of networked variables, as well direct access to CANopen parameters and functions via predefined function blocks. This requires a PLC with a CANopen interface.

CANopen services for PLC programs according to the IEC61131-3 standard are defined in the CiA (CAN in Automation e.V.) Draft Standard 405. These standards are the basis for providing these CANopen functions.

Using Networked Variables

Networked variables are the easiest way of data exchange in a CANopen network system. Within the PLC, program access to the network variables occurs in the same way as access to internal, local variables on the PLC. From the point of view of a PLC programmer it is unimportant whether a input variable is assigned to a local input on the PLC device or to an input on a networked expansion device. The use of networked variables only requires basic knowledge of CANopen. In general, a CANopen configuration tool as well as the availability of EDS files for the individual CANopen devices are required for integrating network variables into a PLC.

Using CANopen Function blocks

CANopen function blocks enable direct access to specific CANopen services, thus offering a high degree of flexibility in the target application. Furthermore, using these function blocks does not require an additional CANopen configuration tool or EDS files. However, using the CANopen function blocks assumes that the user has detailed knowledge about CANopen and its services.

3.5.2 CANopen network variables

A variety of control devices are capable of exchanging data via CANopen network features, or can be expanded by connecting additional CANopen I/O modules to the CAN bus. Data exchange always occurs via network variables on the PLC program level. According to the IEC61131-3 standard, these variables are declared as "VAR_EXTERNAL" and therefore marked as "*external of the control unit*". The PLC itself keeps a local copy of these variables, whereas the network layer is responsible for assigning and synchronizing the actual values with the value of the networked CANopen device.

From the point of view of CANopen, the PLC represents a "regular" I/O module, with its inputs and outputs not available to the user on standard connectors, but in form of network variables mapped into the process image. The appearance of the PLC, in regards to its networked inputs and outputs, can change depending on the number and size of the network variables used in the PLC program. This means that the same PLC can appear differently to the CANopen network if different PLC programs are executed. In order to support such flexible behavior, the PLC utilizes a dynamic Object Dictionary, a structure for managing variables as well as communication and mapping parameters similar to the ones used in databases. Regular CANopen I/O modules usually have a static Object Dictionary.

PLC network variables are stored in the index range A000h - AFFFh within the Object Dictionary in accordance with the CiA Draft Standard 405.

The following terms are important for further explanations of the integration of control units with decentralized I/O expansion modules:

CANopen I/O module:

The CANopen I/O module represents a device that provides certain resources, such as input and output channels to the network. Such a module is considered as a Slave device from the Network Management's (NMT) point of view.

Mapping:

Mapping describes the connection between variables, as well as inputs and outputs to the corresponding bytes and bit positions within a CAN message.

CANopen Configurator:

The CANopen Configurator or Configuration Tool is a special software tool that enables design and management of CANopen networks, interconnection of inputs and outputs on various devices, as well as configuration of network parameters. Furthermore, the CANopen Configurator is used to connect network variables of a PLC program to the corresponding inputs and outputs on the CANopen I/O module. **A CANopen Configurator is always a separate software tool and NOT included in the delivery contents of the OpenPCS IEC61131 programming system.** We recommend the program "ProCANopen" available from the company Vector Informatik.

EDS File:

The EDS file (**E**lectronic **D**ata **S**heet) is provided by the manufacturer of a CANopen device. The file describes the basic device characteristics, such as available I/Os, factory default mapping and network communication configuration, as well as parameters that can be modified by the user.

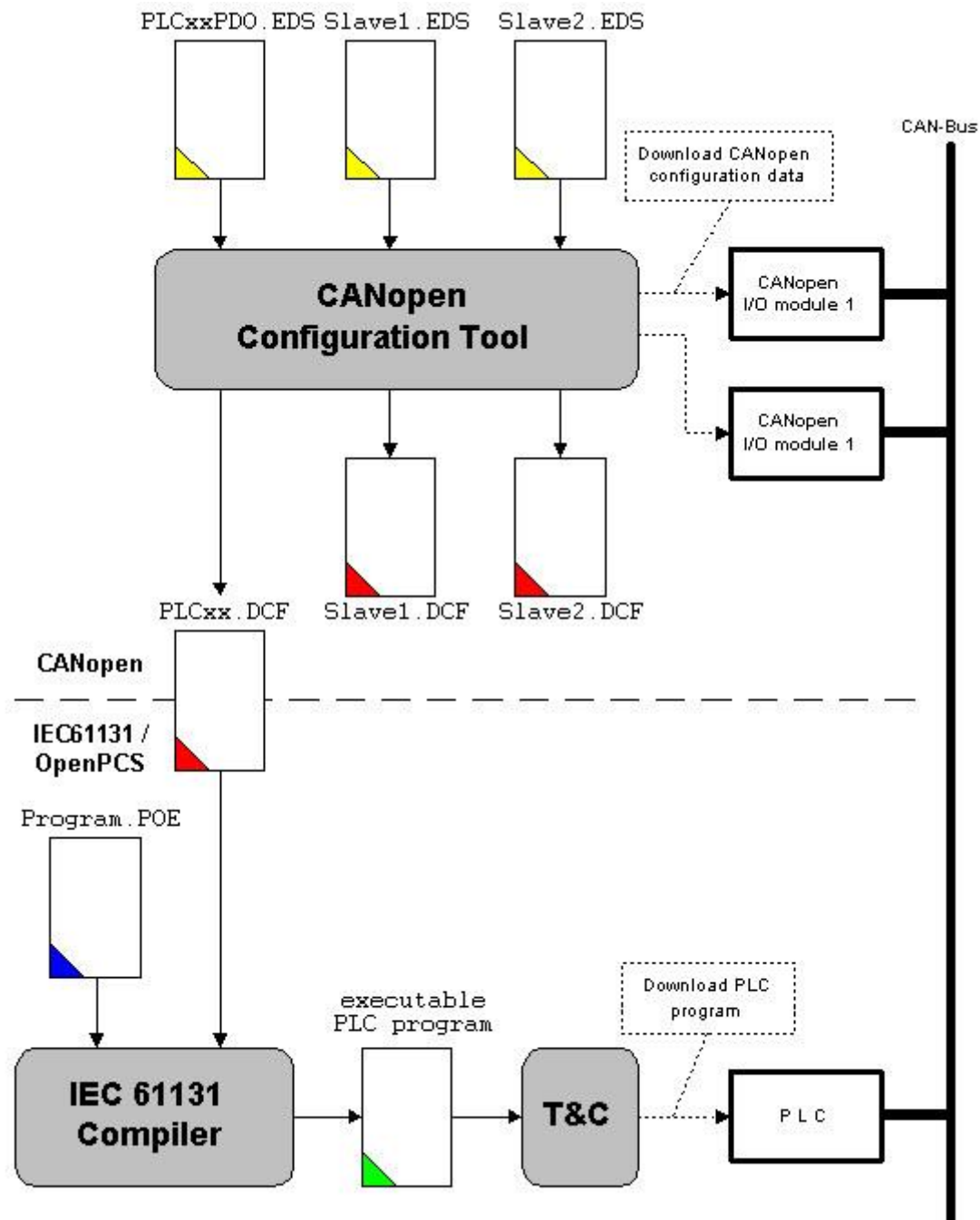
DCF File:

The DCF file (**D**evice **C**onfiguration **F**ile) is generated by the CANopen Configurator as a result of the configuration process. The CANopen Configurator uses the EDS file as a template and adds available entries via parameters configured by the user including identifier and mapping.

In order to assign decentralized I/Os to a PLC, the active network variables are linked to applicable inputs and outputs on the CANopen I/O module within the PLC program. In general this requires the use of a CANopen Configurator. In contrast to standard CANopen I/O devices, there is no EDS file available for a PLC that indicates the number and type of available inputs and outputs. On a PLC only the user defines, with its specific application program, the number and type of inputs and outputs are accessible via the CANopen network. This is done by defining the corresponding network variables. For this reason, only a generic EDS file is available for PLCs which define that the control unit supports dynamic objects.

3.5.3 Configuration process

The EDS file is the most important item for the configuration process of I/O units. The CANopen Configurator reads the EDS file and allows access to resources provided by the CANopen I/O module from the users application. In this configuration process, the user defines characteristics, such as which inputs and outputs are used, which bit or byte position of a CAN message carries the corresponding I/O value (mapping) and which identifier is used for the CAN message. As a result of the configuration process, the CANopen Configurator creates a DCF file for each individual node. Such manual configuration, however, is only required if the user needs to change the default parameters (identifier, mapping) given by the device manufacturer. The standard parameters are calculated and assigned using a defined algorithm with an adjustable node number (node ID). Further information about this process can be found in the applicable Systems or Hardware Manual for the CANopen I/O device.



In contrast to I/O devices with static inputs and outputs, a PLC features dynamic objects, in other words, the current network variables defined in the corresponding PLC program. Because the manufacturer has no knowledge about objects created during runtime, corresponding object entries do not appear in the EDS file. Therefore, a Configurator is always required when linking dynamic objects. The results of the configuration process are then stored in the DCF file. The IEC61131 programming system utilizes the DCF file generated by the Configurator in order to assign the network variables declared as

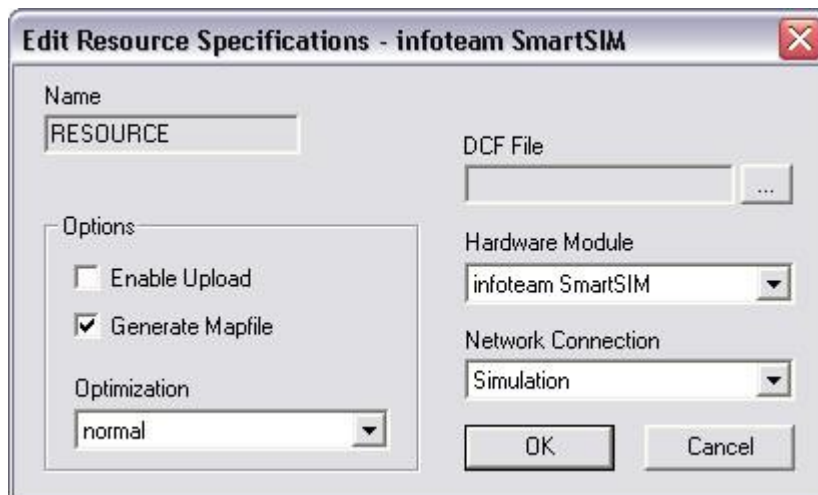
VAR_EXTERNAL and, furthermore, creates the necessary control information for the network layer.

The network parameters used to exchange process data are defined according to user requirements with the help of a CANopen Configurator. These parameters are, for example, the transmission mode (synchronous, asynchronous), the assigned identifiers or the mapping. Furthermore, the Configurator enables the user to create a linking system that is essential for network variables between the PLC and the CANopen I/O modules. Symbolic names are assigned to process data (usually inputs and outputs) on the individual I/O module. This allows the PLC program to easily access these network variables at a later point.

The DCF file for the PLC functions acts as the interface between the CANopen Configurator and the **OpenPCS** programming environment. This configuration file needs to be connected to the corresponding hardware. This provides the control unit with all necessary network information for process data exchange with the CANopen I/O modules.

3.5.4 Insert a DCF-file into OpenPCS

If your hardware supports CANOpen, you can insert a DCF-file into your OpenPCS project with the dialog [Edit resource](#) :



If this input field does not exist in your dialog box, please contact the manufacturer of your hardware.

3.5.5 Declaration of CANopen network variables

Network variables used in a PLC program are declared with the keywords **VAR_EXTERNAL ... END_VAR**. Thus, they are marked as *"outside of the program"* and

furthermore as *"outside of the PLC"*. However, declaration of network variables is the same as for local variables.

```
VAR_EXTERNAL
```

```
NetVar1 : BYTE ;
```

```
NetVar2 : UINT ;
```

```
END_VAR
```

Similar to the section VAR in the free variable editor the section VAR_EXTERNAL must be entered by hand. When using the syntax controlled variable editor the network variables must be defined in the section External. Switching between free and syntax controlled variable editor is done with the menu <Extras -> Variable Editor> located in the *"POE-Editor"* program.

The same symbolic names that are defined for the process data in the corresponding DCF file must be used as **names for the network variables**. The variable name is the common relation between the IEC61131 PLC and CANopen.

A data type that is compatible for both IEC61131 and CANopen must be chosen as **type of the network variables**.

According to the IEC61131 standard the chosen data type must match the usage of the network variable (logic, arithmetic) when declaring the variable in the PLC program. No clear arrangement between IEC61131 and CANopen is however available.

Note:

No clear arrangement in regards of data types between IEC61131 and CANopen is available. For this reason the IEC61131 data type used in the PLC program must be select according to the usage of the corresponding variable.

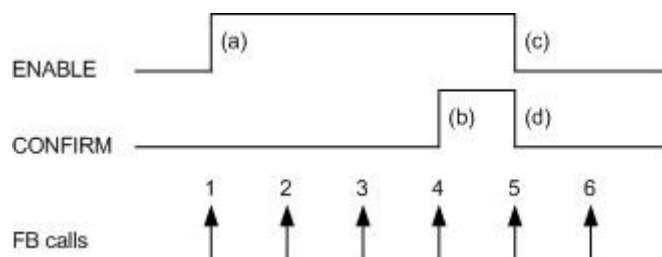
Data Type Assignment between IEC61131 and CANopen:

IEC61131	CANopen	Usage	Data Size (Bit)
----------	---------	-------	--------------------

BOOL	Boolean	Logic	1
BYTE	Unsigned8	Logic	8
USINT	Unsigned8	Arithmetic (unsigned)	8
SINT	Integer8	Arithmetic (with sign)	8
WORD	Unsigned16	Logic	16
UINT	Unsigned16	Arithmetic (unsigned)	16
INT	Integer16	Arithmetic (with sign)	16
DWORD	Unsigned32	Logic	32
UDINT	Unsigned32	Arithmetic (unsigned)	32
DINT	Integer32	Arithmetic (with sign)	32
REAL	Float	Arithmetic	32

3.5.6 Synchronisation

The majority of the CANopen function blocks for the IEC61131-3 are implemented asynchronously with the PLC program. The process synchronization between CANopen and the PLC program occurs with the help of the component signals ENABLE and CONFIRM.



Process Synchronization Between CANopen and the PLC Program

A CANopen service implemented asynchronously with an PLC program is processed completely in the following steps:

After the PLC program has initialized all the input variables, it sets the input ENABLE to TRUE and gives the command that the CANopen component be called (call #1). The component recognizes a positive transition on the input ENABLE, and subsequently takes on all input values and starts the corresponding CANopen service (step (a)). Finally the component returns to the PLC program, whereby the initiated CANopen service continues to be processed in the background.

By the time the CANopen service has been completely processed the function block will be called multiple times by the PLC program. The input ENABLE must remain set to TRUE during this time, in order to allow for the execution of the CANopen service in the background (calls 2 and 3).

After successful completion of the CANopen service, the function block sets its output CONFIRM to TRUE. This signals to the PLC program the end of the service by CANopen and also shows that any additional output variables are now set with valid values (for example with the data read from a node, step (b), call 4).

The PLC program provides the function block with proof that the CANopen service has been completed by setting the input ENABLE to FALSE. At the same time the PLC program signals that it has adopted the output variable delivered by the function block (step (c), call 5). In the last step the function block sets its output CONFIRM back to FALSE, so that the system is back in its output state (step (d)).

Note:

As a rule, the network layer allows the execution of only one CANopen service that is not simultaneous with the execution of the PLC program. With the start of this service by setting the input ENABLE to TRUE (step 1), access to the network layer is locked, preventing use by other function blocks. This lock state is maintained until the function block in question is called once again (step 4) by setting the ENABLE input to FALSE again after completion of the service (FB sets its output CONFIRM to TRUE, step 3). The intermediate call of the other CANopen function block will lead to the error report TRANSFER_BUSY on the ERROR output.

The output CONFIRM changes from FALSE to TRUE only after successful completion of the current CANopen service. Possible errors are shown on outputs ERROR and ERRORINFO. Thus it is required that an PLC program monitors the value of ERROR in addition to the output CONFIRM, in order to be able to evaluate errors that have occurred as well.

Calling the function block with the ENABLE input set to FALSE will cause the CANopen service that is active in the background to abort, and result in an internal reset of the function block. At the same time the output CONFIRM will be set to FALSE and the outputs ERROR and ERRORINFO will be set to the value NO_ERROR.

3.5.7 CANopen constants

To characterize the internal error state of the network layer, the CiA Draft Standard 405 defines the specific data type "CiA405_CANOPEN_KERNEL_ERROR". The error states which could occur within the local network layer of an PLC are summarized below. These error codes are used by various function blocks as output parameter ERROR.

Constants for Data Type "CIA405_CANOPEN_KERNEL_ERROR"

Constant Error Code

16#0000 (= 00 dec) NO_ERROR

16#0001 (= 01 dec) OTHER_ERROR

16#0002 (= 02 dec) DATA_OVERFLOW

16#0003 (= 03 dec) TIME_OUT

16#0010 (= 16 dec) CAN_BUS_OFF

16#0011 (= 17 dec) CAN_ERROR_PASSIVE

16#0021 (= 33 dec) GENERIC_ERROR

16#0022 (= 34 dec) FUNCTION_NOT_AVAILABLE

16#0023 (= 35 dec) NO_MASTER_MODE

16#0024 (= 36 dec) INVALID_DEVICE

16#0025 (= 37 dec) TRANSFER_BUSY

16#0030 (= 48 dec) NO_SDO_CHANNEL_AVAILABLE

16#0031 (= 49 dec) SDO_BUSY

16#0032 (= 50 dec) SDO_INITIALIZE_ERROR

16#0033 (= 51 dec) SDO_LENGTH_ERROR

16#0040 (= 64 dec) NO_VALID_DATA_AVAILABLE
16#0041 (= 65 dec) COBID_ALREADY_REGISTERED
16#0042 (= 66 dec) NO_FREE_COBID_TABLE_ENTRY
16#0043 (= 67 dec) NO_SUCH_COBID_REGISTERED
16#0044 (= 68 dec) NO_FREE_RECEIVE_CHANNEL
16#0045 (= 69 dec) DATA_LENGTH_ZERO_NOT_ALLOWED

To characterize the state of a CANopen device, the CiA Draft Standard 405 defines the specific data type "CIA405_STATE".
The state value UNKNOWN and NOT_AVAIL are extensions to the existing standard. All other constant values agree with the corresponding definitions of the CiA Draft Standard 301.

Constants for the Data Type "CIA405_STATE"

Constants State Value

16#0000 INIT
16#0001 RESET_COMM
16#0002 RESET_APP
16#0003 PRE_OPERATIONAL
16#0004 STOPPED
16#0005 OPERATIONAL
16#0006 UNKNOWN
16#0007 NOT_AVAIL

To characterize the state that a CANopen device is suppose to switch to, the CiA Draft Standard 405 defines the specific data type "CIA405_TRANSITION_STATE". The constant values match the corresponding definition of the CiA Draft Standard 301.

Constants for the Data Type "CIA405_TRANSITION_STATE"

Constants State Value

16#0000 START_REMOTE_NODE

16#0001 STOP_REMOTE_NODE

16#0002 ENTER_PRE_OPERATIONAL

16#0003 RESET_NODE

16#0004 RESET_COMMUNICATION

In addition the CiA Draft Standard 405 defines the specific data types "CIA405_SDO_ERROR" and "EMCY_ERR_CODE" and "EMCY_ERR_REGISTER". These data types represent the error messages generated by another node.

The data type "CIA405_SDO_ERROR" is used for the parameter ERRORINFO of the SDO function block and delivers the communication parameter's SDO abort code. The general abort codes are defined in the CiA Standard 301, but can be expanded by the manufacturer of the CANopen subassembly currently in use.

The data types "EMCY_ERR_CODE" and "EMCY_ERR_REGISTER" are used for the corresponding parameters of the function blocks CAN_RECV_EMCY and CAN_RECV_EMCY_DEV. They contain the emergency error information of the node that generated the corresponding emergency message. The general emergency errors are defined in the CiA Draft Standard 301, but can be expanded by the manufacturer of the CANopen subassembly currently in use.

3.6 IEC61131-3

3.6.1 IEC61131-3 Details

3.6.1.1 Character String Literals

A string constant is sequence of characters enclosed in "". Special characters can be embedded within a character string literal by using escape sequences starting with the \$ sign, as listed in the following table:

Predefined character constants	Meaning
"\$"""	The Apostrophe ""
"\$\$"	The \$ sign itself

"\$L" or "\$l"	Line Feed
"\$N" or "\$n"	New Line
"\$P" or "\$p"	Form Feed
"\$R" or "\$r"	Carriage Return
"\$T" or "\$t"	Tabulator

Example

Character Constant	Meaning and Length
"A"	Single character A, length=1
" "	Blank character, length=1
""	No character, length=1
"\$R\$L"	Carriage Return, Line Feed, length=2
"\$0D\$0A"	Carriage Return, Line Feed, length=2

3.6.1.2 Maximum String Length

Each string is delimited by a maximum length. The default maximum length of a string is 32 characters. It can be changed setting an individual maximum string length in round brackets immediately after the keyword [STRING](#).

The maximum string length can be set to all values from 0 to 251. However this may differ at other hardwares.

Examples:

TYPE

name: STRING(15) := "John Q. Public"; (*maximum string length 15*)

address: STRING(50) := "Main Street 1, 12345 Springfield, ???"; (*maximum string length 50*)

END_TYPE

VAR

user: name; (*maximum string length 15*)

id: string(8) := "12345678"; (*maximum string length 8*)

phone : STRING; (*maximum string length 32*)

END_VAR

3.6.1.3 Constants

Within a literal constant, underscores are allowed to increase readability. Such underscores have no meaning regarding the value of a constant. Literal constants for some data types require a special prefix

Constant Data Type	Example	Meaning
INT	-13 45165 or 45_165 +125	Integer -13 Integer 45165 (both) Integer 125
REAL	-13.12 123.45 0.123 -1.23E-3	Real -13,12 Real 123,45 Real 0,123 Real -0,00123
Dual number	2#0111_1110 or 126	126
Octal number	8#123 or 83	83
Hexadecimal number	16#123 or 291	291
BOOL	0 and 1 TRUE and FALSE	Boolean TRUE and FALSE values
STRING	"ABC"	Character string ABC
WSTRING	"ABC"	2-byte-character string ABC
TIME	T#12.3ms or TIME#12.3ms	Time duration of 12,3 milliseconds
	T#12h34m or T#12h_34m	Time duration of 12 hours and 34 minutes

	T#-4m	Negative time duration of 4 minutes
DATE	DATE#1995-12-24 or D#1995-12-24	Date 24.12.1995
TIME OF DAY	TOD#12:05:14.56 or TIME_OF_DAY#12:05:14.56	12 hours05 minutes and 14,56 seconds PM
DATE AND TIME	DT#1995-12-24-12:05:14.56 or DATE_AND_TIME#1995-12-24- 12:05:14.56	Date and time: 12 hours05 minutes and 14,56 seconds PM on 24.12.1995

Literal constants of data types TIME, DATE and DATE_AND_TIME uses keywords plus a hash sign "#". The keywords can be written in long (e.g. DATE_AND_TIME) or short form (e.g. DT).

Note: DATE, TIME_OF_DAY and DATE_AND_TIME are currently not supported by OpenPCS.

See also [Elementary Data Types](#)

3.6.1.4 Single Bit Access

With OpenPCS, each individual bit of BYTE or WORD variable can be accessed by writing the bitnumber, separated by a dot, after the variable name

Example

```
PROGRAM Only_1_Bit

VAR
Bitpattern1 : BYTE := 2#10101010;
Bitpattern2 AT %IW0.0 : WORD;
END_VAR

LD Bitpattern2.15 (* Copy bit 15 *)
ST Bitpattern1.0 (* into bit 0 *)
.
.

END_PROGRAM
```

Please note that this feature might not be available on all hardware platforms for all data types due to implementation restrictions.

3.6.1.5 Passing Output Parameters

IEC61131 defines two ways of passing parameters. OpenPCS provides, as a legal extension to IEC61131, a means to directly pass output parameters. You can pass output parameters within the line of the CAL instruction by using a vertical slash "|" instead of a comma, and giving the actual parameter on the left side of the assignment:

Example

```
CAL SR_Instance_1(SET1 := On,  
RESET := Off  
|  
Result := Q1)
```

3.6.1.6 Nested Comments

Nested comments are **not** allowed.

3.6.1.7 Block Type: Program, Function, Function Block

A **program** in OpenPCS has the following characteristic properties, as defined by IEC61131: Only the program is allowed to declare variables to be mapped to physical addresses; A program is allowed to call functions and instances of function blocks.

A **function block**, as defined by IEC61131, has the following characteristic properties: It may have one, more than one, or no inputs; It may have one, more than one, or no outputs; Multiple instances can be created of a function block, and each instance will keep a private copy of all data associated with that function block (input, output, intermediate data); a function block cannot be called, only instances can be called. The function block has a "memory", i.e. all data (input, output, local) will keep it's value from one call to the next. On a call, it is not necessary to supply all input data; those not provided will simply keep the value from the previous call (or the default value if there was no call before). A function block can call functions and instances of other function blocks.

A **function**, as defined by IEC61131, has the following characteristic properties: It has one or more inputs (but no input is not allowed); It has exactly one output value (which may be a structure); A function has no "memory" from one call to the next, and it will return always the same output when given the same inputs. On every call to a function, all inputs have to be supplied. A function may use local variables for intermediate storage, but the value of these local variables will not be kept from one call to the next. A function may call other functions, but it is not allowed to call instances of function blocks.

3.6.2 IEC61131-3 Compliance Statement

3.6.2.1 Compliance Statement

The following tables have the same numbering as those in the IEC 1131-3/EN 61131-3 standard. Tables showing features not yet supported by this version of OpenPCS are not listed. Some tables in IEC61131-3 do not contain features, so missing table numbers do not necessarily imply missing features. To understand this document, you will want to consult IEC61131-3.

This version of OpenPCS complies with the requirements of IEC61131-3, for the following language features:

Table 1: Character Set Features

No.	Description	Yes	No
1	Required character set	x	
2	Lower case	x	
3a	Number sign (#)	x	
3b	or Pound sign (£)		x
4a	Dollar sign (\$)	x	
4b	or Currency sign		x
5a	Vertical bar ()		x
5b	or Exclamation mark (!)	x	
6a	Subscript delimiters: brackets []	x	
6b	or parentheses ()		x

3.6.2.2 Table 2: Identifier features

No.	Description	Yes	No
1	Upper case and numbers	x	
2	Upper and lower case, numbers, embedded underlines	x	
3	Upper and lower case, numbers, leading or embedded underlines	x	

3.6.2.3 Table 3: Comment features

No.	Description	Yes	No
-----	-------------	-----	----

1	Comments	x	
---	----------	---	--

3.6.2.4 Table 4: Numeric Literals

No.	Description	Yes	No
1	Integer literals	x	
2	Real literals	x	
3	Real literals with exponents	x	
4	Base 2 literals	x	
5	Base 8 literals	x	
6	Base 16 literals	x	
7	Boolean zero and one	x	
8	Boolean FALSE and TRUE	x	

3.6.2.5 Table 5: Character string literal features

No.	Description	Yes	No
1	Empty string (length zero)	x	
	String of length one containing the single character A	x	
	String of length one containing the "space" character	x	
	String of length one containing the "single quote" character	x	
	String of length two containing CR und LF	x	

	String of length five which would print as "\$1.00"		
--	---	--	--

3.6.2.6 Table 6: Two character combinations in character strings

No.	Description	Yes	No
2	Dollar sign (\$\$)	x	
3	Single quote (\$")	x	
4	Line feed (\$L or \$l)	x	
5	New line (\$N or \$n)	x	
6	New page (\$P or \$p)	x	
7	Carriage return (\$R or \$r)	x	
8	Tab (\$T or \$t)	x	

3.6.2.7 Table 7: Duration literal features

No.	Description	Yes	No
	Duration literals without underlines:		
1a	Short prefix	x	
1b	Long prefix	x	
	Duration literal with underlines		
2a	Short prefix	x	
2b	Long prefix	x	

3.6.2.8 Table 8: Date and time of day literals

No.	Description	Yes	No

1	Date literals (long prefix: DATE#)	x	
2	Date literals (short prefix: D#)	x	
3	Time of day literals (long prefix: TIME_OF_DAY#)	x	
4	Time of day literals (short prefix: TOD#)	x	
5	Date and time literals (long prefix: DATE_AND_TIME#)	x	
6	Date and time literals (short prefix: DT#)	x	

Table 10: elementary data types

No.	Keyword	Data type	Yes	No
1	BOOL	Boolean	x	
2	SINT	Short integer	x	
3	INT	Integer	x	
4	DINT	Double integer	x	
5	LINT	Long integer		x
6	USINT	Unsigned short integer	x	
7	UINT	Unsigned integer	x	
8	UDINT	Unsigned double integer	x	
9	ULINT	Unsigned long integer		x
10	REAL	Real numbers	x	
11	LREAL	Long real numbers	x	
12	TIME	Duration	x	
13	DATE	Date (only)	x	

14	TIME_OF_DAY or TOD	Time of day (only)	x	
15	DATE_AND_ TIME or TD	Date and time	x	
16	STRING	Variable-length character string	x	
17	BYTE	Bit string of length 8	x	
18	WORD	Bit string of length 16	x	
19	DWORD	Bit string of length 32	x	
20	LWORD	Bit string of length 64		x

3.6.2.9 Table 12: Data type declaration feature

No.	Description	Yes	No
1	Direct derivation from elementary types	x	
2	Enumerated data types	x	
3	Sub range data types		x
4	Array data types	x	
5	Structured data types	x	

3.6.2.10 Table 13: Default initial values

Description	Initial value	Yes	No
BOOL, SINT, INT, DINT, LINT,	0	x	
USINT, UINT, UDINT, ULINT	0	x	
BYTE, WORD, DWORD, LWORD	0	x	
REAL, LREAL	0.0	x	

TIME	T#0s	x	
DATE	D#0001-01-01	x	
TIME_OF_DAY	TOD#00:00:00	x	
DATE_AND_TIME	DT#0001-01-01-00:00:00	x	
STRING	"(the empty string)"	x	

3.6.2.11 Table 14: Data type initial value declaration features

No.	Description	Yes	No
1	Initialization of directly derived types	x	
2	Initialization of enumerated data types	x	
3	Initialization of sub range data types		x
4	Initialization of array data types	x	
5	Initialization of structured data types	x	
6	Initialization of derived structured data types	x	

3.6.2.12 Table 15: Location and size prefix features for directly represented variables

No.	Description	Yes	No
1	I: Input location	x	
2	Q: Output location	x	
3	M: Marker location	x	
4	X: (Single) bit size	x	

5	None: (Single) bit size	x	
6	B: Byte (8 bits) size	x	
7	W: Word (16 bits) size	x	
8	D: Double word (32 bits) size	x	
9	L: Long word (64 bits) size		x

3.6.2.13 Table 16: Variable keywords for variable declaration

Keyword	Yes	No
VAR	x	
VAR_INPUT	x	
VAR_OUTPUT	x	
VAR_IN_OUT	x	
VAR_EXTERNAL	x	
VAR_GLOBAL	x	
VAR_ACCESS		x
RETAIN	x	
CONSTANT	x	
AT	x	

3.6.2.14 Table 17: Variable type assignment features

No.	Description	Yes	No
1	Declaration of directly represented, non-	x	

	retentive variables		
2	Declaration of directly represented, retentive variables	x	
3	Declaration of locations of symbolic variables	x	
4	Array location assignment		x
5	Automatic memory allocation of symbolic variables	x	
6	Array declaration	x	
7	Retentive array declaration	x	
8	Declaration of structured variables	x	

3.6.2.15 Table 18: Variable initial value assignment features

No.	Description	Yes	No
1	Initialization of directly represented, non-retentive variables		x
2	Initialization of directly represented, retentive variables		x
3	Location and initial value assignment to symbolic variables		x
4	Array location assignment and initialization		x
5	Initialization of symbolic variables	x	
6	Array initialization	x	
7	Retentive array declaration and initialization	x	
8	Initialization of structured variables	x	
9	Initialization of constants	x	

3.6.2.16 Table 19: Graphical negation of Boolean signals

No.	Description	Yes	No
1	Negated input	x	
2	Negated output		x

3.6.2.17 Table 20: Use EN input an ENO output

No.	Description	Yes	No
1	Use of EN and ENO		x
2	Use of EN and ENO		x
3	FBD without EN and ENO	x	

3.6.2.18 Table 21: Typed and overloaded functions

No.	Description	Yes	No
1	Overloaded functions (non type-dependent)	x	
2	Typed functions	x	

3.6.2.19 Table 22: Type conversion function features

No.	Description	Yes	No
1	*_TO_**	x	
2	TRUNC	x	
3	BCD_TO_**		x
4	*_TO_BCD		x

Comment:

If you are using TIME-values, only TIME_TO_DINT, TIME_TO_REAL and DINT TO_TIME are implemented. Other TIME-cast-functions are only available within the Ladder-Diagram-Editor.

For no. 1, (*) is the input variable data type and (**) is the output variable data type. The following data types are supported:

BOOL

BYTE

DINT

DWORD

INT

REAL

SINT

STRING

TIME

UDINT

UINT

USINT

WORD

3.6.2.20 Table 23: Standard functions of one numeric variable

No.	Description	Yes	No
1	ABS	x	
2	SQRT	x	
3	LN	x	

4	LOG	x	
5	EXP	x	
6	SIN	x	
7	COS	x	
8	TAN	x	
9	ASIN	x	
10	ACOS	x	
11	ATAN	x	

3.6.2.21 Table 24: Arithmetic standard functions

No.	Name	Symbol	Yes	No
12	ADD	+	x	
13	MUL	*	x	
14	SUB	-	x	
15	DIV	/	x	
16	MOD		x	
17	EXPT	**	x	
18n 18s	MOVE	:=	x	x

3.6.2.22 Table 25: Standard bit shift functions

No.	Name	Yes	No
1	SHL	x	

2	SHR	x	
3	ROR	x	
4	ROL	x	

3.6.2.23 Table 26: Standard bitwise Boolean functions

No.	Name	Yes	No
5	AND	x	
6	OR	x	
7	XOR	x	
8	NOT	x	

3.6.2.24 Table 27: Standard selection functions

No.	Name	Yes	No
1	SEL		x
2a	MAX	x	
2b	MIN	x	
3	LIMIT	x	
4	MUX		x

3.6.2.25 Table 28: Standard comparison functions

No.	Name	Yes	No
5	GT	x	
6	GE	x	

7	EQ	x	
8	LE	x	
9	LT	x	
10	NE	x	

3.6.2.26 Table 29: Standard character string functions

No.	Name	Yes	No
1	LEN	x	
2	LEFT	x	
3	RIGHT	x	
4	MID	x	
5	CONCAT	x	
6	INSERT	x	
7	DELETE	xX	
8	REPLACE	xx	
9	FIND	x	

3.6.2.27 Table 30: Functions of time data types

No.	Name	Operation	Yes	No
1	ADD	TIME + TIME = TIME	x	
2		TOD + TIME = TOD	x	
3		DAT + TIME = DAT	x	
4	SUB	TIME - TIME = TIME	x	

5		DATE - DATE = TIME	x	
6		TOD - TIME = TOD	x	
7		TOD - TOD = TIME	x	
8		DAT - TIME = DAT	x	
9		DAT - DAT = TIME	x	
10	MUL	TIME * ANY_NUM = TIME		x
11	DIV	TIME / ANY_NUM = TIME		x
12	CONCAT	DATE TOD = DAT		x
		Type conversion functions		
13		DATE_AND_TIME_TO_TIME_OF_DAY		x
14		DATE_AND_TIME_TO_DATE		x
15	RTC		x	

3.6.2.28 Table 31: Functions of enumerated data types

No.	Name	Yes	No
1	SEL		x
2	MUX		x
3	EQ		x
4	NE		x

3.6.2.29 Table 33: Function block declaration features

No.	Description	Yes	No
1	RETAIN qualifier on internal variables	x	

2	RETAIN qualifier on output variables	x	
3	RETAIN qualifier on internal function blocks		x
4a	Input/output declaration (textual)	x	
4b	Input/output declaration (graphical)		x
5a	Function block instance name as input (textual)		x
5b	Function block instance name as input (graphical)		x
6a	Function block instance name as input/output (textual)		x
6b	Function block instance name as input/output (graphical)		x
7a	Function block instance name as external variable (textual)		x
7b	Function block instance name as external variable (graphical)		x
	Textual declaration of		
8a	- rising edge inputs	x	
8b	- falling edge inputs	x	
	Graphical declaration of		
9a	- rising edge inputs		x
9b	- falling edge inputs		x

3.6.2.30 Table 34: Standard bistable function blocks

No.	Name	Yes	No
1	SR	x	
2	RS	x	

3	SEMA		x
---	------	--	---

3.6.2.31 Table 35: Standard edge detection function blocks

No.	Name	Yes	No
1	R_TRIG	x	
2	F_TRIG	x	

3.6.2.32 Table 36: Standard counter function blocks

No.	Name	Yes	No
1	R_TRIG	x	
2	F_TRIG	x	

3.6.2.33 Table 37: Standard timer function blocks

No.	Name	Yes	No
1	TP (Pulse)	x	
2a	TON (on-delay)	x	
2b	T---0 (on-delay)		x
3a	TOF (off-delay)	x	
3b	0---T (off-delay)		x
4	RTC (real-time clock)	x	

3.6.2.34 Table 39: Program declaration features

No.	Description	Yes	No
-----	-------------	-----	----

1	RETAIN qualifier on internal variable	x	
2	RETAIN qualifier on output variable		x
3	RETAIN qualifier on internal function blocks		x
4a	Input/output declaration (textual)		x
4b	Input/output declaration (graphical)		x
5a	Function block instance name as input (textual)		x
5b	Function block instance name as input (graphical)		x
6a	Function block instance name as input/output (textual)		x
6b	Function block instance name as input/output (graphical)		x
7a	Function block instance name as external variable (textual)		x
7b	Function block instance name as external variable (graphical)		x
8a	Textual declaration of: - rising edge inputs		x
8b	- falling edge inputs		x
9a	Graphical declaration of: - rising edge inputs		x
9b	- falling edge inputs		x
10	Formal input and output parameters		x
11	Declaration of directly represented, non-retentive variables	x	
12	Declaration of directly represented, retentive variables	x	

13	Declaration of locations of symbolic variables	x	
14	Array location assignment		x
15	Initialization of directly represented, non-retentive variables		x
16	Initialization of directly represented, retentive variables		x
17	Location and initial value assignment to symbolic variables		x
18	Array location assignment and initialization		x
19	Use of directly represented variables	x	
20	VAR_GLOBAL .. END_VAR Declaration within a PROGRAM	x	
21	VAR_ACCESS .. END_VAR Declaration within a PROGRAM		x

3.6.2.35 Table 40: Step features

No.	Description	Yes	No
1	Step graphical	x	
	Initial step graphical	x	
2	Step textual		x
	Initial Step textual		x
3a	Step flag general form		x
3b	Step flag - direct connection of Boolean variable		x
4	Step elapsed time		x

3.6.2.36 Table 41: Transitions and Transition conditions

No.	Description	Yes	No
1	Transition condition using ST language		x
2	Transition condition using LD language		x
3	Transition condition using FBD language		x
4	Use of connector		x
4a	Transition condition using LD language		x
4b	Transition condition using FBD language		x
5	Textual transition in ST	x	
6	Textual transition in IL	x	
7	Transition name	x	
7a	Transition condition using LD language		x
7b	Transition condition using FBD language		x
7c	Transition condition using IL language	x	
7d	Transition condition using ST language	x	

3.6.2.37 Table 42: Declaration of actions

No.	Description	Yes	No
1	Boolean variable as action		x
2l	graphical declaration in LD language		x
2s	inclusion of SFC elements in action		x
2f	graphical declaration in FBD language		x
3s	textual declaration in ST language		x
3i	graphical declaration in IL language	x	

3.6.2.38 Table 43: Step/action association

No.	Description	Yes	No
1	action block		x
2	concatenated action blocks		x
3	textual step body	x	
4	action block "d" field		x

3.6.2.39 Table 44: Action block features

No.	Description	Yes	No
1	qualifier as per 2.6.4.4		x
2	action name	x	
3	Boolean indicator variables		x
4	IL language		x
5	ST language		x
6	LD language		x
7	FBD language		x
8	action blocks in ladder diagrams		x
9	action block in function block diagrams		x

3.6.2.40 Table 45: Action qualifiers

No.	Description	Yes	No
1	None	x	

2	N (non-stored)	x	
3	R (overriding reset)		x
4	S (set stored)		x
5	L (time limited)		x
6	D (time delayed)		x
7	P (pulse)		x
8	SD (stored and time delayed)		x
9	DS (delayed and stored)		x
10	SL (stored and time limited)		x

3.6.2.41 Table 46: Sequence evolution

No.	Description	Yes	No
1	single sequence	x	
2a	divergence of sequence selection (left-to-right)	x	
2b	divergence of sequence selection (with priorities)		x
2c	divergence of sequence selection (with mutual exclusion)		x
3	Convergence of sequence evolution	x	
4	simultaneous sequence divergence	x	
5	simultaneous sequence convergence	x	
5a	sequence skip (left-to-right)	x	
5b	sequence skip (with priorities)		x
5c	sequence skip (with mutual exclusion)		x

6a	sequence loop (left-to-right)		x
6b	sequence loop (with priorities)		x
6c	sequence loop (with mutual exclusion)		x
7	directional arrows		x

3.6.2.42 Table 52: Instruction list (IL) operators

No.	Operator	Modifiers	Yes	No
1	LD	N	x	
2	ST	N	x	
3	S R		x x	
4	AND	N,(x	
5	&	N,(x	
6	OR	N,(x	
7	XOR	N,(x	
8	ADD	(x	
9	SUB	(x	
10	MUL	(x	
11	DIV	(x	
12	GT	(x	
13	GE	(x	
14	EQ	(x	
15	NE	(x	
16	LE	(x	

17	LT	(x	
18	JMP	C, N	x	
19	CAL	C, N	x	
20	RET	C, N	x	
21)		x	

3.6.2.43 Table 53: Function block invocation features for IL language

No.	Description	Yes	No
1	CAL with input list	x	
2	CAL with load/store of inputs	x	
3	Use of input operators		x

3.6.2.44 Table 55: Operators of the ST language

No.	Description	Yes	No
1	Parenthesation	x	
2	Function evaluation	x	
3	Exponentiation	x	
4	Negation	x	
5	Complement	x	
6	Multiply	x	
7	Divide	x	
8	Modulo	x	

9	Add	x	
10	Subtract	x	
11	Comparison	x	
12	Equality	x	
13	Inequality	x	
14	Boolean AND	x	
15	Boolean AND	x	
16	Boolean Exclusive XOR	x	
17	Boolean OR	x	

3.6.2.45 Table 56: ST language statements

No.	Description	Yes	No
1	Assignment	x	
2	Function block invocation and FB output usage	x	
3	RETURN	x	
4	IF	x	
5	CASE	x	
6	FOR	x	
7	WHILE	x	
8	REPEAT	x	
9	EXIT	x	
10	Empty Statement	x	

3.6.2.46 Table 57: Representation of lines and block

No.	Description	Yes	No
Horizontal lines:			
1	ISO/IEC 646 "minus" character		x
2	graphic or semigraphic	x	
Vertical lines:			
3	ISO/IEC 646 "vertical line" character		x
4	graphic or semigraphic	x	
Horizontal/vertical connection:			
5	ISO/IEC 646 "plus" character		x
6	graphic or semigraphic	x	
Line crossing without connection:			
7	ISO/IEC 646 characters		x
8	graphic or semigraphic	x	
Connected and non-connected corners:			
9	ISO/IEC 646 characters		x
10	graphic or semigraphic	x	
Blocks with connecting lines			
11	ISO/IEC 646 characters		x
12	graphic or semigraphic	x	
Connectors using ISO/IEC 646 characters:			
13	Connector, Continuation of a connected line		x
14	graphic or semigraphic	x	

3.6.2.47 Table 58: Graphic execution control elements

No.	Description	Yes	No
	Unconditional Jump		
1	FBD language	x	
2	LD language	x	
3	Conditional Jump (FBD language)	x	
4	Conditional Jump (LD language)	x	
	Conditional Return		
5	LD language	x	
6	FBD language	x	
	Unconditional Return		
7	from Function	x	
	from Function Block	x	
8	Alternative Representation in LD language	x	

3.6.2.48 Table 59: Power rails

No.	Description	Yes	No
1	Left power rail	x	
2	Right power rail	x	

3.6.2.49 Table 60: Link Elements

No.	Description	Yes	No
-----	-------------	-----	----

1	Horizontal link	x	
2	vertical link with attached horizontal links	x	

3.6.2.50 Table 61: Contacts

No.	Description	Yes	No
	Normally open contact		
1		x	
2		x	
	Normally closed contact		
3		x	
4		x	
	Positive transition-sensing contact		
5			x
6			x
	Negative transition-sensing contact		
7			x
8			x

3.6.2.51 Table 62: Coils

No.	Description	Yes	No
1	Coil	x	
2	Negated Coil	x	
3	SET (latch) coil	x	
4	RESET (unlatch) coil	x	

5	Retentive (Memory) coil		x
6	SET retentive (Memory) coil		x
7	RESET retentive (Memory) coil		x
8	Positive transition-sensing coil		x
9	Negative transition-sensing coil		x

3.6.2.52 Table 63: Reserved Names

Names of data types cannot be used for file or variable names. The following names are also not allowed for variables and/or files:

- _____
 - D
- _____
 - L
- _____
 - N
- _____
 - P
- _____
 - Q

3.6.2.53 FBD language Elements

[Note: this chapter is empty because the corresponding chapter in IEC61131-3 does not list any features to be referenced here].

3.6.2.54 Table D.1: Implementation-dependent parameters

Clause	Parameter	Values
1.5.1	Error handling procedures	see next chapter
2.1.1	National characters used	see table 1 above
2.1.2	Maximum length identifiers	256
	Significant length identifiers	64

2.1.5	Maximum comment length	>512
2.2.3.1	Range of values of duration	+/- 24,85 days
2.3.1	Range of values for variables of type TIME Precision of representation of seconds in type TIME_OF_DAY and DATE_AND_TIME	+/- 24,85 days -
2.3.3	Maximum - number of array subscripts - array size - number of structure elements - structure size - number of variables per declaration	6 < 4KB per POU < 8KB per POU
2.3.3.1	Maximum number of enumerated values	< 64 KB per POU
2.3.3.2	Default maximum length of STRING variables Maximum permissible length of STRING variables	32 253 [see note 1]
2.4.1.1	Maximum number of hierarchical levels Logical or physical mapping	5
2.4.1.2	Maximum number of subscripts Maximum number of subscript values Maximum number of levels of structures	- - >512
2.4.2	Initialization of system inputs	The value of the system inputs corresponds to their physical values
2.4.3	Maximum number of variables per	

	declaration	< 64 KB per POU
2.5	Information to determine execution times of program organization units	No
2.5.1.1	Method of function representation	Textual
2.5.1.3	Maximum number of function specifications	limited only by available memory
2.5.1.5	Maximum number of inputs of extensible functions	IL: 2, LD/FBD: unlimited
2.5.1.5.1	Effects of type conversions on accuracy	Truncated
2.5.1.5.2	Accuracy of functions of one variable Implementation of arithmetic functions	Currently not supported
2.5.2	Maximum number of function blocks and instantiations	ca. 8000 [see note 2]
2.5.2.3.3	PVmin, PVmax of counters	minimum/maximum value of respective data type
2.5.3	Program size limitations	limited only by available memory
2.6	Timing and portability effects of execution control elements	-
2.6.2	Precision of step elapsed time Maximum number of steps per SFC	-
2.6.3	Maximum number of transitions per SFC and per step	-
2.6.4	Action control mechanism	-
2.6.4.2	Maximum number of action blocks per step	-

2.6.5	Graphic indication of step state Transition clearing time Maximum width of diverge/converge constructs	-
2.7.1	Content of RESOURCE libraries	-
2.7.2	Maximum number of tasks Task interval resolution Pre-emptive or non-pre-emptive scheduling	-
3.3.1	Maximum length of expressions Partial evaluation of Boolean expressions	unlimited no
3.3.2	Maximum length of statements	Unlimited
3.3.2.3	Maximum number of CASE selections	Unlimited
4.1.1	Graphic/semigraphic representation Restrictions on network topology	Graphic
4.1.3	Evaluation order of feedback loops	-

note 1: OpenPCS is highly configurable, so this parameter may vary depending on your hardware. If in doubt, consult the documentation of your hardware.

note 2: The maximum number of function blocks is less, if variables are declared in the same segment.

3.6.2.55 Table E.1: Error conditions

2.3.3.1	Value of a variable exceeds the specified sub range	Syntax error reported for initialization in declaration; ignored at runtime
2.4.2	Length of initialization list doesn't match the number of	Syntax error

	array entries	
2.5.1.5.1	Type conversion errors	Ignored
2.5.1.5.2	Numerical result exceeds range for data type. Division by zero	firmware blocks report that at ENO, ignored elsewhere
2.5.1.5.4	Mixed input data types to a selection function Selector (K) out of range for MUX function	not supported
2.5.1.5.5	Invalid character position specified Result exceeds maximum string length	-
2.5.1.5.6	Result exceeds range for data type	Restriction to maximum value (see 2.2.3.1)
2.6.2	Zero or more than one initial step in the SFC network User program attempts to modify step state or time	-
2.6.2.5	Simultaneously true, non-prioritized transitions in a selection divergence	-
2.6.3	Side effects in evaluation of transition condition	-
2.6.4.5	Action control contention error	-
2.6.5	"Unsafe" or "Unreachable" SFC	-
2.7.1	Data type conflict in VAR_ACCESS	-
2.7.2	Tasks require too many processor resources Execution deadline not met	-

	Other task scheduling conflicts	
3.2.2	Numerical result exceeds range for data type	Scan via functions
3.3.1	Division by zero Invalid data type for operation	Syntax error can be monitored
3.3.2.1	Return from function without value assigned	-
3.3.2.4	Iteration fails to terminate	-
4.1.1	Same identifier as connector label and element name	-
4.1.4	Uninitialized feedback variable	-
4.1.5	Numerical result exceeds range for data type Division by 0	-

3.7 Online Features

3.7.1 Breakpoints

OpenPCS supports Breakpoints in textual languages ST and IL. Breakpoints are currently not supported in [Native Code](#), so set [optimization](#) to "size". Breakpoints are not supported with all targets due to hardware restrictions. Breakpoints are not saved, so set new breakpoints before starting a newly downloaded application.

If a breakpoint is reached in any one task of the OpenPCS application, execution of all tasks will immediately be stopped. When single-stepping, continuing to the next breakpoint, etc. it is undefined and left to the target if other tasks should be executed in the meantime. Therefore it is recommended to have one task only when single-stepping intuitively.

Stopping a controller with breakpoints and single-stepping can disable many of the safety precautions in your controller and your application, so be sure to take appropriate measures so guarantee damage to be avoided.

3.7.2 Online Edit

Online Edit (or Online Change) is a feature whereby program changes are online applied to the PLC without the need to restart it.

To perform an Online Edit, proceed as follows:

1. In Online Mode, switch an editor to edit mode by PLC->Monitor/Edit (or use toolbar button Monitor/Edit)
2. Modify declarations and code in the editor as required
3. Switch back to Monitor Mode by using Monitor/Edit
4. Now, you will be prompted to update the target. Select "Yes" to save any modifications, recompile the application and download your modifications to the target without stopping the program.
5. Select "No" to abort Online Edit and to discard all changes (also: no modifications will be saved to file).

Changes such as adding a variable, renaming a variable, changing data type of a variable lead to resetting to its initial value. As a restart is not necessary, variable values of program parts that are not affected by the changes will keep their current values (i.e. they will not be reset to their initial values).

Any changes of initial values do not lead to any noticeable results without restart.

Not supported features: add/remove task, change task property, change resource property, change task order.

Attention: If the target system does not support [Save System](#) the changes are not persistent. The System should be saved afterward via **PLC -> Save System...** if the changes should be persistent on the controller. For further Information see the [respective section](#).

Note 1: Strictly, functions are also POU's. Since they are stateless, they need not be treated by Online Edit, however.

Note 2: Online Edit is supported from OpenPCS version 5.3.0 on. It requires Target System version 5.2.2 or above.

Note 3: Additionally there are the following configuration rules:

RuntimeSystem supports "Download without Stop" & OnlineLinking = 1	Data remain the same, Online Edit is possible
RuntimeSystem supports "Download without Stop" & OnlineLinking = 0	Data are set to initial values, Online Edit is possible
RuntimeSystem doesn't support "Download without Stop" & OnlineLinking = 0	Online Edit not possible, data will be changed after rebuild and a new download.
RuntimeSystem doesn't support "Download without Stop" & OnlineLinking = 1	Online Edit not possible, data will be changed after rebuild and a new download.

Remark: OnlineLinking can be set in the hardware configuration file (.ini-File).

3.7.3 Save System

PLC -> Save System... writes the complete system persistent on the controller. This needs to be done if changes were made via [online edit](#).

Save System is an optional target system feature.

3.7.4 Error Logs

A detailed Error Log can be uploaded from the controller via **PLC -> Upload Error Log**. The uploaded file will be named `yymmdd_hhmmssErrorlog.txt` and will be stored in the current project directory.

Error Logs is an optional target system feature.

4 Reference

4.1 Keywords (by category)

4.1.1 IEC61131 Standard Function Blocks

OpenPCS implements the following function blocks of IEC61131-3:

[CTD](#)

[CTU](#)

[CTUD](#)

[F_TRIG](#)

[R_TRIG](#)

[RS](#)

[SR](#)

[TOF](#)

[TON](#)

[TP](#)

4.1.2 IEC61131-3 Standard Functions

OpenPCS implements the following functions of IEC61131-3:

[ABS](#)

[ACOS](#)

[AND](#)

[ASIN](#)

[ATAN](#)

[CONCAT](#)

[COS](#)

[DELETE](#)

[EQ](#)

[EXP](#)

[FIND](#)

[GE](#)

[GT](#)

[INSERT](#)

[LE](#)

[LEFT](#)

[LEN](#)

[LIMIT](#)

[LN](#)

[LOG](#)

[LT](#)

[MAX](#)

[MID](#)

[MIN](#)

[MOD](#)

[MUX](#)

[NE](#)

[NEG](#)

[OR](#)

[REAL TO *](#)

[RIGHT](#)

[ROL](#)

[ROR](#)

[SHL](#)

[SIN](#)

[SHR](#)

[SQRT](#)

[TAN](#)

[TIME TO *](#)

[TRUNC](#)

[XOR](#)

[RIGHT](#)

4.1.3 IEC61131-3 operations

OpenPCS implements the following operations of IEC61131-3:

[ADD](#)

[ADD \(time\)](#)

[DIV](#)

[DIV \(time\)](#)

[MUL](#)

[MUL \(time\)](#)

[SUB](#)

[SUB \(time\)](#)

4.1.4 OpenPCS Functions and Function Blocks

The following functions and function blocks are provided by OpenPCS in addition to IEC61131-3:

[GetTaskInfo](#)

[GetTime](#)

[GetTimeCS](#)

[GetVarData](#)

[GetVarFlatAddress](#)

The section [CANopen](#) gives an overview on functions and function blocks to use with CANopen

4.1.5 Data Types

The following elementary data types are defined by IEC61131-3:

[BOOL](#)

[BYTE](#)

[DATE AND TIME](#)

[DATE](#)

[DINT](#)

[DWORD](#)

[INT](#)

[REAL](#)

[SINT](#)

[STRING](#)

[TIME OF DAY](#)

[TIME](#)

[UDINT](#)

[UINT](#)

[WORD](#)

The following data types are defined by OpenPCS in addition to IEC61131-3:

[POINTER](#)

[VARINFO](#)

4.1.6 Declaration Keywords

[END_TYPE](#)

[END_VAR](#)

[RETAIN](#)

[TYPE](#)

[VAR_GLOBAL](#)

[VAR_IN_OUT](#)

[VAR_INPUT](#)

[VAR_OUTPUT](#)

[VAR_EXTERNAL](#)

[VAR](#)

4.1.7 Instruction List Instructions

Program Logic Instructions:

"(" (Right-parenthesis-operator)

[CAL](#) Instance name

[CALC](#) Instance name

[CALCN](#) Instance name

[JMP](#) Label

[JMPC](#) Label

[JMPCN](#) Label

[RET](#)

[RETC](#)

[RETCN](#)

Boolean Operations and Instructions:

[NOT](#)

[AND](#)

[ANDN](#)

[OR](#)

[ORN](#)

[XOR](#)

[XORN](#)

[S](#) BOOL

[R](#) BOOL

Mathematical Operations:

[ADD](#)

[SUB](#)

[MUL](#)

[DIV](#)

Load/Save Instructions:

[LD](#) ANY

[LDN](#) ANY_BIT

[ST](#) ANY

[STN](#) ANY_BIT

Logical Operators:

[GT](#)

[GE](#)

[EQ](#)

[NE](#)

[LE](#)

[LT](#)

4.1.8 Structured Text Keywords

OpenPCS uses the following keywords in Programming Language Structured Text:

:= (Assignment)

[BY](#)

[CASE](#)

[DO](#)

[ELSE](#)

[ELSIF](#)

[END_CASE](#)

[END_FOR](#)

[END_IF](#)

[END_REPEAT](#)

[END_WHILE](#)

[EXIT](#)

[FOR](#)

[IF](#)

[OF](#)

[REPEAT](#)

[RETURN](#)

[TO](#)

[UNTIL](#)

[WHILE](#)

4.1.9 CANopen

[CAN_RECV_EMCY](#)

[CAN_RECV_EMCY_DEV](#)

[CAN_NMT](#)

[CAN_GET_STATE](#)

[CAN_SDO_WRITE8](#)

[CAN_SDO_READ8](#)

[CAN_GET_CANOPEN_KERNEL_STATE](#)

[CAN_GET_LOCAL_NODE_ID](#)

[CAN_REGISTER_COBID](#)

[CAN_PDO_READ8](#)

[CAN_PDO_WRITE8](#)

[CAN_SDO_READ_STR](#)

[CAN_SDO_WRITE_STR](#)

[CAN_WRITE_EMCY](#)

[CAN_RECV_BOOTUP_DEV](#)

[CAN_RECV_BOOTUP](#)

[CAN_ENABLE_CYCLIC_SYNC](#)

[CAN_SEND_SYNC](#)

4.1.10 Others

[ACTION](#)

[ANY](#)

[ANY_BIT](#)

[ANY_DATE](#)

[ANY_INT](#)

[ANY_NUM](#)

[ANY_REAL](#)

[CD](#)

[CDT](#)

[CLK](#)

[CONFIGURATION](#)

[CU](#)

[CV](#)

[D\(DATE\)](#)

[D\(Action Qualifier\)](#)

[DS](#)

[DT](#)

[END_ACTION](#)

[END_CONFIGURATION](#)

[END_RESOURCE](#)

[END_STEP](#)

[END_STRUCT](#)

[END_TRANSITION](#)

[ET](#)

[EXPT](#)

[FROM](#)

[IN](#)

[INITIAL_STEP](#)

[Interval](#)

[L\(Action Qualifier\)](#)

[Lreal](#)

[Lword](#)

[N \(Action Qualifier\)](#)

[On](#)

[P\(Action Qualifier\)](#)

[Priority](#)

[PT](#)

[PV](#)

[Q\(Parameter\)](#)

[Q1](#)

[QD](#)

[QU](#)

[R\(Action Qualifier\)](#)

[R1](#)

[READ_ONLY](#)

[READ_WRITE](#)

[Release](#)

[Resource](#)

[RTC](#)

[S\(Action Qualifier\)](#)

[S1](#)

[SD](#)

[SEL](#)

[SEMA](#)

[Single](#)

[SL](#)

[STEP](#)

[Task](#)

[TOD](#)

[Transition](#)

[ULINT](#)

[USINT](#)

[VAR_ACCESS](#)

[WITH](#)

4.2 Keywords (A..Z)

4.2.1 ")" (Right-parenthesis-operator)

The right-parenthesis-operator executes an instruction, deferred by the left-parenthesis-modifier.

Example:

LD a

OR(b (* Execution of instruction "OR" is deferred *)

AND c

) (* "OR" will be executed now *)

OR(d

AND e

)

ST f

Notes:

This is an instruction in language Instruction List

It is defined by IEC61131-3

4.2.2 *_TO_BOOL

0 is converted to false, everthing else to true.

The conversions [String to bool](#) and [Real to bool](#) are described in the respective sections.

4.2.3 *_TO_STRING

Inputs

original data type *

Returns

converted data type string

The function block converts the first value of type * into the same value of type string. The following data types can be converted:

BOOL

true -> "true"

false -> "false"

DINT, INT und SINT

Siehe unter REAL

BYTE, DWORD, WORD und USINT, UINT, UDINT

The bitmask is directly converted into a string

Examples:

001101 -> "001101"

REAL

For converting string function `Sprintf(str, "%#g", value);` is used.

Examples:

0.0 -> "0.000000"

123.45678 -> " 123.456"

-12.345678 -> " -12.3456"

12345678.9 -> " 1.23457e+007"

0.000000123 -> " 1.23000e-007"

4.2.4 ABS

Input

In: ANY_NUM

Returns

ANY_NUM

Notes:

Returns the absolute value of the input.

Please note the following anomaly of the ABS function: The mathematical understanding of the ABS function is that it will never return a negative value. The signed integer data types in IEC61131-3 have a defined range of values which is asymmetric, e.g. SINT from -128..+127. As defined by IEC61131-3, the ABS function will return the same data type that it is provided as an input, e.g. when called with an SINT input, ABS will return an SINT output. The absolute value of -128 obviously is +128, but when passed to ABS for type SINT, exceeds the range of SINT and hence cannot be expressed. This overflow is, for performance reasons, silently ignored by OpenPCS, the result returned being undefined. If you need to rely on the negative maximum value to be properly handled, use a data type with a wider range, or check inputs.

This does not apply to the ABS function as called by the Ladder Diagram Editor, this ABS function will signal overflow via the ENO output.

4.2.5 ACOS

Input

In: REAL

Returns

REAL: arcus cosine of input

4.2.6 ACTION

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of [SFC](#), hence you will not be able to enter this keyword. You will see this when printing [SFC](#).

4.2.7 ADD

Inputs

In1: ANY_NUM

In2: ANY_NUM

Returns

ANY_NUM sum

Addition of two numbers. See [Table E.1: Error conditions](#) for result on overflow.

Notes:

Standardization: this is an operation defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.8 ADD (time)

Inputs

In1: TIME time duration value

In2: TIME

Returns

TIME Addition of the two time values provided

Addition of TIME values

Notes:

Standardization: this is an operation defined by IEC61131-3.

4.2.9 AND

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bit by bit AND of Input 1 and Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.10 ANDN

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bitwise AND of Input 1 and negated Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.11 ANY

ANY_BIT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [ANY_BIT](#), [ANY_DATE](#), [ANY_INT](#), [ANY_REAL](#)

4.2.12 ANY_BIT

ANY_BIT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [BOOL](#), [BYTE](#), [WORD](#), [DWORD](#), [LWORD](#).

4.2.13 ANY_DATE

ANY_DATE is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [DATE](#), [DATE AND TIME](#), [TIME OF DAY](#).

4.2.14 ANY_INT

ANY_INT is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [SINT](#), [USINT](#), [INT](#), [UINT](#), [DINT](#), [UDINT](#), [LINT](#), [ULINT](#).

4.2.15 ANY_NUM

ANY_NUM is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [ANY_INT](#), [ANY_REAL](#).

4.2.16 ANY_REAL

ANY_REAL is a "generic" data type defined by IEC61131-3. You are not allowed to use this data type to declare variables. Wherever this data type is used, it is understood to mean any one of the following: [REAL](#), [LREAL](#).

4.2.17 ARRAY

ARRAY is the keyword to declare arrays of elements, see [Derived Data Types](#)

Examples:

The following declares an array of five integers and assigns initial values:

```
VAR
```

```
x1: ARRAY[0..4] of INT := [1,2,3,4,5];
```

```
END_VAR
```

A three-dimensional array of 300 booleans:

```
VAR
```

```
x2: ARRAY[0..4, 15..20, 1..10] of BOOL;
```

```
END_VAR
```

An array of 100 structures:

```
TYPE
```

```
x3: STRUCT
```



```
member1: BOOL;  
  
member2: INT;  
  
END_STRUCT;  
  
END_TYPE  
  
VAR  
  
x4 : ARRAY[1..10,1..10] of x3;  
  
END_VAR
```

Initializing of multidimensional arrays:

To initialize arrays with more than one dimension, give a list of list of initial values, each dimension enclosed in brackets. The dimension given first in declaration will correspond to the outermost brackets.

```
VAR  
  
x2: ARRAY[0..4, 1..2] of INT := [[1,2], [3,4], [5,6], [7,8], [9,10]];  
  
x3: ARRAY[0..1, 0..2, 0..3] of INT :=  
[[[1,2,3,4],[5,6,7,8],[9,10,11,12]],[[13,14,15,16],[17,18,19,20],[21,22,23,24]]];  
  
END_VAR
```

Note: OpenPCS uses 16bit signed integers to represent array subscripts for performance reasons. Array bounds may not exceed the [-32768;32767]-range.

4.2.18 ASIN

Input

In: REAL

Returns

REAL: arcus sine of input

4.2.19 Assignment

An Assignment will assign the result of an expression to a variable.

Example

```
VAR
a: INT;
b: ARRAY [0..5] OF INT;
c: REAL;
e: INT;
END_VAR
a := 5;
(* assign 5 to a *)
b[1]:= a*2; e := a; (* two assignments *)
e:= REAL_TO_INT( c );
(* assignment with function call *)
```

The assignment instruction will evaluate the expression on the right side and assign the resulting value to the variable given on the left.

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.20 AT

AT is the keyword to define the memory location where OpenPCS should allocate memory for a given variable.

Very first input bit:

```
VAR
```

```
x1 at %ix0.0: bool;
```

```
END_VAR
```

Output word starting at second output byte:

```
VAR
```

x2 at %qw1.0: word;

END_VAR

4.2.21 ATAN

Input

In: REAL

Returns

REAL: arcus tangens of input

4.2.22 BOOL

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.23 BOOL_TO_*

Inputs

original data type bool

Returns

converted data type *

The function block converts the first value of type bool into the same value of type *.
The following data types can be converted:

DINT, INT und SINT

BYTE, DWORD, WORD und USINT, UINT, UDINT

true -> 1

false -> 0

REAL

true -> 1.0

false -> 0.0

STRING

true -> "true"

false -> "false"

4.2.24 BY

See [FOR](#)

4.2.25 BYTE

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.26 CAL

The program will be continued at the function block whose name is passed as operand. The unconditioned invocation may only be used as the end of a sequence and is not permitted within bracketing operations.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

See also [EN](#).

4.2.27 CALC

If the CR holds the value TRUE, the function block specified as operand will be called. If it holds the value "0", there is no invocation. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.28 CALCN

If the CR holds the value FALSE, the function block specified as operand will be called. If it holds the value "1", there is no invocation. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.29 CAN_ENABLE_CYCLIC_SYNC

Function block for enabling or locking cyclic SYNC messages.

Input

SYNC_MODE : BOOL Enables generation of cyclic SYNC messages

SYNC_TIME : TIME Time between two SYNC messages. 0 generates 1 SYNC after each PLC cycle.

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_ENABLE_CYCLIC_SYNC is used to enable cyclic SYNC messages. If SYNC_MODE is set to TRUE the function block will generate a SYNC message after a PLC cycle if SYNC_TIME has passed since the last SYNC.

This function block is only available on control units in "PLC with CANopen Master" mode.

4.2.30 CAN_GET_CANOPEN_KERNEL_STATE

Function block for state query of the CANopen kernel of the local PLC.

Input

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

CONFIRM : BOOL Output for signal service completion by the function block

STATE:: WORD State or error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

The function block CAN_GET_CANOPEN_KERNEL_STATE is used for a query about the state of the CANopen kernel of the local PLC.

4.2.31 CAN_GET_LOCAL_NODE_ID

Function block for a local node address query.

Input

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

DEVICE : USINT Local Node Address of the PLC

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_GET_LOCAL_NODE_ID is used for a query of the local node address of the PLC. The node address of a control unit has an influence over the availability of the various function blocks (PLC with and without CANopen Master)

4.2.32 CAN_GET_STATE

Function block for node state query of various devices.

Input

DEVICE : USINT Address of the node to be queried (1-127 or 0 for local node)

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

STATE : WORD Node state corresponding to the data type "CIA405_STATE"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_GET_STATE is used to enquire the node state for a specific device. The state query is based on monitoring by Heartbeat or Lifeguarding. The return values on the output STATE have the following meaning:

UNKNOWN: The CANopen device on the given address supports neither Heartbeat nor Lifeguarding, thus the state cannot be monitored. This state is also reported on an PLC without a CANopen Master if either no PLC with a CANopen Master is available in the network that supports a state transmission or if the Master PLC in question is in a stop state (PLC program has been halted).

NOT_AVAIL: The CANopen device on the given address no longer answers Heartbeat or Lifeguarding queries and is therefore no longer available to the system.

other: With the exception of the state values UNKNOWN and NOT_AVAIL the return values match the corresponding definitions of the CiA Draft Standard 30.

The call of the function block with DEVICE = 0 delivers the local node state of the local PLC.

4.2.33 CAN_NMT

Function block for sending NMT messages.

Input

DEVICE : USINT Address of the node to be controlled (1-127 or 0 for all nodes)

STATE : WORD Node state corresponding to the data type
"CIA405_TRANSITION_STATE"

ENABLE : BOOL Input for enabling or locking the function

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_NMT is used for controlling the state of a node (DEVICE = 1...127) or if DEVICE = 0 all the nodes in the network.

This function block is only available on a control unit in "PLC with CANopen Master" mode

4.2.34 CAN_PDO_READ8

Function block reading PDOs and CAN Layer 2 messages via the network layer.

Input

COBIB : UINT COBID (CAN-Identifier) of the message to be read

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

CONFIRM : BOOL Output for signal service completion by the function block

ERROR : WORD State or error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD Additional information on the error

DATA0 - DATA7 : BYTE Data bytes of the received message

DATALength : USINT Length of the received message

The function block CAN_PDO_READ8 reads a PDO or CAN Layer 2 message via the network layer. The message must be registered via [CAN_REGISTER_COBID](#) before. The receiving buffer stores only the most current message and the message will be erased from the networks layer's receiving buffer.

After a successful read operation, CONFIRM is set to TRUE and the elements DATA0 to DATA7 contain the individual bytes of the received message. DATALength reports the number of valid bytes. Empty messages are valid.

If no message with the given COBID was available, CONFIRM is set to false. If an error occurred the ERROR is set as well.

4.2.35 CAN_PDO_WRITE8

Function block sending PDOs and CAN Layer 2 messages via the network layer.

Input

COBIB : UINT COBID (CAN-Identifier) of the message to be read

ENABLE : BOOL Input for enabling or locking the function block

DATA0 - DATA7 : BYTE Data bytes of the received message

DATALength : USINT Length of the received message

NETNUMBER : USINT Network number

Output

CONFIRM : BOOL Output for signal service completion by the function block

ERROR : WORD State or error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD Additional information on the error

The function block CAN_PDO_WRITE8 sends a PDO or CAN Layer 2 message via the network layer. The elements DATA0 to DATA7 contain the individual bits of the message and DATALENGTH specifies the number valid bytes.

If the message is correctly stored in the send buffer of the CANopen kernel CONFIRM set to TRUE. However this does not indicate if the message is sent.

4.2.36 CAN_RECV_BOOTUP

Function block for reading Bootup messages of any node from the receiving buffer of the network layer.

Input

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_RECV_BOOTUP_DEV is used for reading Bootup messages of any node from the receiving buffer of the network layer. If a message is received successfully CONFIRM is set to TRUE, otherwise the buffer contains no message of any node. The function block always return the oldest message (FIFO-prinziple) und deletes the read message in the buffer.

This functiion block is only available on control units in "PLC with CANopen Master" mode.

4.2.37 CAN_RECV_BOOTUP_DEV

Function block for reading Bootup messages of a specific node from the receiving buffer of the network layer.

Input

DEVICE : USINT Additional information for diagnostic purpose

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_RECV_BOOTUP_DEV is used for reading Bootup messages of a specific node from the receiving buffer of the network layer. If a message is received successfully CONFIRM is set to TRUE, otherwise the buffer contains no message of the specific node. After a message is read it is deleted in the buffer.

This function block is only available on control units in "PLC with CANopen Master" mode.

4.2.38 CAN_RECV_EMCY

Function block for reading emergency messages of a node from the network layer's receiving buffer.

Input

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

DEVICE : USINT Address of the node (1-127) from which an emergency message was received

EMCY_ERR_CODE : WORD

EMCY_ERR_REGISTER : BYTE

EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5 : BYTE

Emergency error information corresponding to the CiA Draft Standard 301

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_RECV_EMCY is used for reading the emergency messages of any nodes from the network layer's receiving buffer. If upon the return of the function block the output CONFIRM is set to TRUE, the output DEVICE reports the node address from which the message was received. The elements EMCY_ERR contain the emergency error information of the node corresponding to the CiA Draft Standard 301. However, if the output CONFIRM is set to TRUE, the network layer's receiving buffer does not contain any emergency messages.

The function block always returns the first emergency message entered into the receiving buffer (= oldest message), the message is subsequently erased from the receiving buffer. Thus every emergency message can only be read one time by the PLC program. The function blocks CAN_RECV_EMCY_DEV and CAN_RECV_EMCY both access the same receiving buffer.

This function block is only available on control units in "PLC with CANopen Master" mode.

4.2.39 CAN_RECV_EMCY_DEV

Function block for reading emergency messages of a specific node from the receiving buffer of the network layer.

Input

DEVICE : USINT Address of the node (1-127), for which the receipt of emergency messages is to be tested

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

EMCY_ERR_CODE : WORD

EMCY_ERR_REGISTER : BYTE

EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5 : BYTE

Emergency error information corresponding to the CiA Draft Standard 301

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_RECV_EMCY_DEV is used to read the emergency messages of a specific node from the network layer's receiving buffer. If upon return of the function block the output CONFIRM is set to TRUE, the elements EMCY_ERR maintain the emergency error information of the node corresponding to the CiA Draft Standard 301. If however the output CONFIRM is set to FALSE, then the receiving buffer of the network layer does not contain any emergency messages for the node in question.

The function block always returns the first emergency message entered into the receiving buffer (= oldest message), the message is subsequently erased from the receiving buffer. Thus every emergency message can only be read one time by the PLC program. The function blocks CAN_RECV_EMCY_DEV and CAN_RECV_EMCY both access the same receiving buffer.

This function block is only available on one control unit in "PLC with CANopen Master" mode

4.2.40 CAN_REGISTER_COBID

Function block for registering or erasing the receipt of PDOs and CAN Layer 2 messages via the network layer.

Input

COBIB : UINT COBID (CAN-Identifier) of the message being newly registered or erased

REGISTER : BOOL TRUE : register COBID; FALSE : erase COBIS from registration

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

CONFIRM : BOOL Output for signal service completion by the function block

ERROR : WORD State or error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

The function block CAN_REGISTER_COBID registers a PDO or CAN Layer 2 message via the network layer or erases its registration based on the input parameter REGISTER. All registrations along with the messages in the network layer are erased via COBID = 0.

Messages must be registered in the network layer in order to be read by function blocks as [CAN_PDO_READ8](#)

4.2.41 CAN_SDO_READ8

Function block for reading a node's object entries by way of an SDO transfer.

Input

DEVICE : USINT Address of the node to be read (1-127 or 0 for local OD)

INDEX : WORD Number of the index entry to be read

SUBINDEX : BYTE Number of the sub index entry to be read

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

DATA0 - DATA7 : BYTE Data bytes of the entry that was read

DATALength : USINT Length of the entry that was read

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD SDO abort code of the communication partner corresponding to the data type "CIA405_SDO_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_SDO_READ8 is used to read the object entries of a node currently being used by the SDO transfer. The SDO transfer is always executed in the background.

If the output CONFIRM is set to TRUE upon the return of the function block, the elements DATA0 through DATA7 receive the individual bytes of the object entry that was read. The output DATALENGTH reports the number of valid data bytes (beginning at DATA0).

The network layer supports only a single SDO transfer through the PLC program at any one time. After the start of the SDO transfer by setting ENABLE to TRUE, the SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer.

A call of the function block with DEVICE = 0 leads to an access of the local Object Dictionary of the PLC. Thus values from the local Object Dictionary can be read as well.

4.2.42 CAN_SDO_READ_STR

Function block for reading strings from the object directory of a node via SDO transfer..

Input

DEVICE : USINT Address of the node to be read (1-127 or 0 for local OD)

INDEX : WORD Number of the index entry to be read

SUBINDEX : BYTE Number of the subindex entry to be read

SDOTYPE : SUBINDEX Type of the SDO transfer. Standard is "SDO_TYPE_AUTO_BEST_CASE"

RXDATA : STRING String variables for storing the read characters

MAXLENGTH : INT Limit the number of characters to be read.

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

RXDATA : STRING String variables for storing the read characters

RXLENGTH : INT Length of character sequence read

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD SDO abort code of the communication partner corresponding to the data type "CIA405_SDO_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_SDO_READ_STR is used to read strings from a node's Object Directory utilizing SDO transfer.

All SDO transfers are executed in the background. Synchronization can be handled via ENABLE and CONFIRM.

If CONFIRM is TRUE, the read string is stored in RXDATA and RXLENGTH contains its length.

4.2.43 CAN_SDO_WRITE8

Function block for writing object entries of a node by way of an SDO transfer.

Input

ENABLE : BOOL Input for enabling or locking the function block

DEVICE : USINT Address of the node to be written (1-127 or 0 for local Object Dictionary)

INDEX : WORD Number of the index entry to be written

SUBINDEX : BYTE Number of the sub index entry to be written

DATA0 - DATA7 : BYTE Data bytes of the entry to be written

DATALLENGTH : USINT Length of the entry to be written

NETNUMBER : USINT Network number

Output

CONFIRM : BOOL Output for signal service completion by the function block

ERROR : WORD Error code corresponds to the data type "CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD SDO abort code of the communication partner corresponding to the data type "CIA405_SDO_ERROR"

The function block CAN_SDO_WRITE8 is used to write the object entries of a node currently being used by the SDO transfer. The SDO transfer is always executed in the background.

The individual bytes of the object entry to be written are transferred to the elements DATA0 through DATA7. Whereby the input DATALENGTH specifies the number of valid data bytes (beginning with DATA0).

The network layer only supports a single SDO transfer through the PLC program at any one time. After the start of the SDO transfer by setting ENABLE to TRUE, this SDO channel is locked, preventing use by other components. The lock state is maintained until the SDO function block is called again by setting the ENABLE input to FALSE after completion of the data transfer

A call of the function block with DEVICE = 0 leads to access of the local Object Dictionary of the PLC. Thus values can be written to the local Object Dictionary as well.

4.2.44 CAN_SDO_WRITE_STR

Function block for reading strings from the object directory of a node via SDO transfer..

Input

DEVICE : USINT Address of the node to be read (1-127 or 0 for local OD)

INDEX : WORD Number of the index entry to be read

SUBINDEX : BYTE Number of the subindex entry to be read

SDOTYPE : SUBINDEX Type of the SDO transfer. Standard is "SDO_TYPE_AUTO_BEST_CASE"

TXDATA : STRING String variables for storing the characters to be written

TXLENGTH : INT Limit the number of characters to be written

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

TXDATA : STRING String variables for storing the read characters

ERROR : WORD Error code corresponding to the data type "CIA405_CANOPEN_KERNEL_ERROR"

ERRORINFO : DWORD SDO abort code of the communication partner corresponding to the data type "CIA405_SDO_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_SDO_READ_STR is used to write strings to a node's Object Directory utilizing SDO transfer.

All SDO transfers are executed in the background. Synchronization can be handled via ENABLE and CONFIRM.

If CONFIRM is TRUE, the read string is stored in RXDATA and RXLENGTH contains its length.

The characters to be written must be stored in TXDATA where TXLENGTH specifies the number of valid characters.

4.2.45 CAN_SEND_SYNC

Function block for enabling or locking cyclic SYNC messages.

Input

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type "CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL tput for signal service completion by the function block

The function block CAN_SEND_SYNC send one SYNC message when ENABLE is set to true.

This function block is only available on control units in "PLC with CANopen Master" mode.

4.2.46 CAN_WRITE_EMCY

Function block for sending application specific Emergency-Messages through the network layer.

Input

EMCY_ERR_CODE : WORD

EMCY_ERR_REGISTER : BYTE

EMCY_ERR_FIELD1 - EMCY_ERR_FIELD5 : BYTE

Emergency error information corresponding to the CiA Draft Standard 301

EMCY_ADD_INFO : WORD Additional information for diagnostic purpose

ENABLE : BOOL Input for enabling or locking the function block

NETNUMBER : USINT Network number

Output

ERROR : WORD Error code corresponding to the data type
"CIA405_CANOPEN_KERNEL_ERROR"

CONFIRM : BOOL Output for signal service completion by the function block

The function block CAN_WRITE_EMCY is used for sending application specific Emergency-Messages through the network layer. The EMCY_ERR members need to be filled with the respective emergency error information according to IEC61131-3.

If the message is stored in the CANopen buffer CONFIRM is set to true. However, the block does not indicate if the message was sent successfully.

4.2.47 CASE

Though IF instructions may be nested, checking for one of many conditions can look quite complicated using IF. CASE, instead, can check for more than one value with one instruction. The "expression" of the CASE-instruction is of type INT, and only the instruction will be executed that corresponds to this INT-value. After that the first instruction behind END_CASE will be executed.

If the expression does not match any of the case-values, the first instruction (block) behind the ELSE will be executed. This partial instruction is optional.

```
CASE expression OF
case_value1: { instructions; }
case_value2: { instructions; }
...
case_valueN: { instructions; }
[ ELSE instructions; ]
END_CASE;
```

Example:

```
VAR
number : INT:= 10;
amount : INT :=2;
END_VAR
CASE number OF
10: amount := amount +1;
11: amount := amount -1;
ELSE
amount := number;
END_CASE;
```

In this example, the value of "number" will be determined, and if it is equal to 10, "amount" will be incremented, if it is equal to "11", "amount" will be decreased. In any other case, "amount" will be set to equal "number".

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.48 CD

This is the name of a formal parameter of a standard function block ([CTD](#)), and as such defined to be a keyword.

4.2.49 CDT

This is the name of a formal parameter of a standard function block ([RTC](#)), and as such defined to be a keyword.

4.2.50 CLK

This is the name of a formal parameter of a standard function block ([R_TRIG](#)), and as such defined to be a keyword.

4.2.51 CONCAT

Inputs

IN1: STRING First String

IN2: STRING Second String

Returns

STRING Concatenation of both Strings

Description

The character strings "IN1" and "IN2" in the working register are chained to form one character string which is loaded into the working register. The strings IN1 to IN2 are written from the left to the right in ascending order.

The feature *Append Input Connector* is available for this function block.

4.2.52 Configuration

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.53 CONSTANT

CONSTANT is the keyword to declare variables that should not be modified by the application code. The OpenPCS compile will give an error message if you intent to write to such a variable:

```
VAR CONSTANT x1 : INT := 15; END_VAR
```

See [declaration sections](#).

4.2.54 COS

Input

In: REAL

Returns

REAL: cosine of input

4.2.55 CR

CR is the abbreviation of Current Result, the virtual accumulator used in IEC61131-3 programming languages.

4.2.56 CTD

The function block "CTD" serves for counting down impulses received from the input operand "CD". On initialization, the counter will be set to "0".

If the operand "LOAD" is "1", the value received by the operand "PV" will be taken over as a value into the counter.

Each rising edge at the input "CD" will decrease the counter by "1".

The output operand "CV" contains the current value of the counter. If the counter value is positive, the output operand "Q" will have the Boolean value "0". If the counter value reaches zero or becomes negative, the output "Q" will be set to "1".

Inputs

CD: bool counter pulse

LOAD: bool set initial value

PV: int reset value

Outputs

Q: bool signal when zero reached

CV: int counter value

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.57 CTU

The function block "CTU" serves for counting up impulses received from the input operand "CU". On initialization, the counter will be set to "0".

The counter value will be reset if the operand "RESET" receives the value "1".

Each rising edge at the input "CU" will increase the counter by "1".

The output operand "CV" contains the current value of the counter. If the counter value is below the margin value "PV", the output operand "Q" will have the Boolean value "0". If the counter value reaches or passes the margin, the output "Q" will be set to "1".

Inputs

CU: bool counter pulse

RESET: bool reset counter

PV: int counter upper limit

Outputs

Q: bool signals if counter has reached upper limit

CV: int current counter value

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.58 CTUD

The function block "CTUD" serves for counting up and down impulses. On initialization, the counter will be set to the value "0". Every rising edge at the input operand "CU". will increase the counter by "1", while every rising edge at the input "CD" will decrease it by "1".

If the operand "LOAD" is "1", the value received by the operand "PV" will be taken over as a value into the counter.

The counter value will be reset if the operand "RESET" receives the value "1". While the static state of the operand "RESET" remains unchanged, the counting conditions or the load condition will have no implication, independent of their value.

The output operand "CV" contains the current value of the counter. If the counter value is below the margin value "PV", the output operand "QD" will have the Boolean value "0". If the counter value reaches or passes the margin, the output "QD" will be set to "1". If the counter value is positive, the output operand "QD" will have the Boolean value "0". If the counter value reaches zero or becomes negative, the output "QD" will be set to "1".

Inputs

CU: bool counting impulses for counting up, rising edge

CD: bool counting impulses for counting down, rising edge

RESET: bool reset condition

LOAD: bool load condition

PV: int load value

Outputs

QU: bool signals whether counter state has reached PV

QD: bool signals whether counter state has reached "0"

CV: int counter state

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.59 CU

This is the name of a formal parameter of a standard function block ([CTU](#)), and as such defined to be a keyword.

4.2.60 CV

This is the name of a formal parameter of a standard function block ([CTD](#)), and as such defined to be a keyword.

4.2.61 D(Date)

nD can be used as an abbreviation to [DATE](#) when specifying the data type of a [literal constant](#). As data type DATE is not implemented in OpenPCS, you will not be able to use this keyword with OpenPCS.

4.2.62 D(Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.63 DATE

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.64 DATE_AND_TIME

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.65 DELETE

Inputs

IN1: STRING Basic character string in which a part should be deleted

L: ANY_INT (as supported) Length of the substring which should be deleted. L < 0 is invalid.

P: ANY_INT (as supported) Starting position of substring. P < 0 is invalid.

Returns

STRING Shortened string. IN1 for invalid parameters.

The function "DELETE" deletes a substring of length "L" starting at position "P" within the given string "IN1".

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.66 DINT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.67 DIV

Inputs

In1: ANY_NUM Value to be divided

In2: ANY_NUM Value to divide by

Returns

ANY_NUM quotient

Divides two numbers. See [Table E.1: Error conditions](#) for result if divisor is zero.

Notes:

Standardization: this is an operation defined by IEC61131-3.

4.2.68 DIV (time)

Inputs

In1: TIME time duration value

In2: ANY_NUM divisor

Returns

TIME divided time value

Division of TIME values

Notes:

Standardization: this is an operation defined by IEC61131-3.

4.2.69 DO

See [FOR](#) and [WHILE](#)

4.2.70 DS

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.71 DT

DT can be used as an abbreviation to [DATE_AND_TIME](#) when specifying the data type of a [literal constant](#). As data type DATE_AND_TIME is not implemented in OpenPCS, you will not be able to use this keyword with OpenPCS.

4.2.72 DWORD

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.73 ELSE

See [CASE](#) and [IF](#)

4.2.74 ELSIF

See [IF](#)

4.2.75 EN

Function Blocks may have an input variable of type BOOL named EN. If this is the case, an [invocation](#) of an instance of this function block is performed if and only if the value of the input variable EN of that instance is TRUE.

See also [CAL](#) and [ENO](#).

Notes:

1. "EN" is abbreviated for "Enable".
2. If input and/or output variables are assigned in the same statement as the CAL instruction, these assignments are performed even if the CAL is not taken due to EN=FALSE.
3. By default, EN is TRUE

4.2.76 END_ACTION

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.77 END_CASE

See [CASE](#)

4.2.78 END_CONFIGURATION

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.79 END_FOR

See [FOR](#)

4.2.80 END_FUNCTION

See [Function](#).

4.2.81 END_FUNCTION_BLOCK

See [Function Block](#).

4.2.82 END_IF

See [IF](#)

4.2.83 END_PROGRAM

See [PROGRAM](#)

4.2.84 END_REPEAT

See [REPEAT](#)

4.2.85 END_RESOURCE

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.86 END_STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.87 END_STRUCT

See [STRUCT](#).

4.2.88 END_TRANSITION

This keyword is defined by IEC61131-3 for the textual representation of programming language SFC. OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.89 END_TYPE

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POUs.

This is defined by IEC61131-3.

4.2.90 END_VAR

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POUs.

This is defined by IEC61131-3.

4.2.91 END_WHILE

See [WHILE](#)

4.2.92 ENO

Function Blocks may have an output variable of type BOOL named ENO. This typically is set to TRUE to signal correct execution and to FALSE to signal errors during execution. Typically, this ENO is wired to the [EN](#) input of another function block.

Notes:

1. "ENO" is abbreviated for "Enable Output"

4.2.93 EQ

Inputs

IN1: ANY input 1

IN2: ANY input 2

Returns

BOOL TRUE if Input 1 is equal to Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The function *Append Input Connector* is not available with this function block

4.2.94 ET

This is the name of a formal parameter of a standard function block ([TOF](#)), and as such defined to be a keyword.

4.2.95 ETRC

Generally an event task will be executed only once. Since the reaction on a special event can last longer than one cycle, it is necessary to restart the current task again. To perform this action the firmware function block ETRC (Event Task Run Control) can be used. It prolongs the execution of its own event task for another cycle. Additionally the function block provides at its outputs information like the cycle count or elapsed time since the first call on this the ETRC instance. With this information a reaction on errors, which would end up in an endless loop, could be handled.

Input:

IN : BOOL TRUE: The event task should be started for another cycle

FALSE: The event task should not be started again. The function block is called only to get the output information;

Output:

Q : BOOL TRUE: The event task will be executed for one cycle more

FALSE: the event task will be stopped after the current cycle

EVC : USINT The event code (EVC) describes the internal reason for the event task to be called.

ERT : TIME The elapsed runtime (ERT) returns the time since the first start of the current event task

CCV : UDINT The cycle counter value defines the count of event task cycles already executed

ERROR : USINT Return values of the ETRC execution.

0 : successful execution,

1 : execution not possible since function has been called out of a task (not a valid call)

Event Codes of the function block:

0	The called task is unknown
1	Coldstart executed
2	Warmstart executed
3	Hotstart executed
4	Single cycle start executed
5	PLC has been stopped by hardware RUN/STOP switch
6	PLC has been stopped by software stop
7	After executing a single cycle the PLC changes to status STOP
8	General error while PLC program execution
9	Division by zero
10	Invalid array index access
11	Error while executing a firmware function block

4.2.96 EXIT

Any of the loops can be "left" under program control before the loop condition dictates so. The EXIT instruction will jump to the first instruction after the innermost loop.

Example:

```
VAR
```

```
start: INT :=0;
```

```
summe: INT :=0;
```

```
ende : INT := 10;
```

```
END_VAR
```

```
FOR Start := 1 TO Ende BY 2 DO
Summe := Summe + 1;
IF Summe > 4 THEN
EXIT;
END_IF;
END_FOR;
(* Will continue here *)
```

As soon as "Summe" is greater than 4, the FOR loop will be left.

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.97 EXP

Input

In: REAL

Returns

REAL: $e^{**} \text{In}$

4.2.98 EXPT

Inputs :

In1 : ANY_REAL

In2 : ANY_NUM

Returns :

ANY_REAL: $\text{In1}^{**} \text{In2}$

4.2.99 F_EDGE

F_EDGE is used to indicate a falling edge detection function on Boolean inputs. This leads to an implicit declaration of a function block of type [F_TRIG](#) .

Example:

```
FUNCTION_BLOCK AND_EDGE
VAR_INPUT
X : BOOL R_EDGE;
Y : BOOL F_EDGE;
END_VAR

VAR_OUTPUT
Z : BOOL ;
END_VAR

Z := X AND Y ; (* ST language example *)
END_FUNCTION_BLOCK
```

4.2.100 F_TRIG

Inputs

CLK: bool input operand whose falling edge is detected

Outputs

Q: bool Output operand; indicates the falling edge of "CLK"

The function block "F_TRIG" detects the status of the input operand "CLK". The status change from "1" to "0" in a processing cycle is detected and indicated in the subsequent cycle with the Boolean value "1" via the output "Q". The output is "1" only in the processing cycle in which the change of the status of "CLK" is detected and a falling edge is indicated.

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.101 FALSE

Constant value of type [BOOL](#).

4.2.102 FBD

FBD is the abbreviation of Function Block Diagram, one of the programming languages of IEC61131-3.

4.2.103 FIND

Find one character string within another character string.

Inputs

IN1: STRING Basic character string in which a special character sequence is searched for; the string is made available via the working register

IN2: STRING Character sequence which is searched for in the "IN1" basic character string.

Returns

INT Position of first occurrence

A special character sequence is searched for in the "IN1" basic character string. If this string is found, the position of the first character of this sequence is entered into the working register or, otherwise, the value "0" is entered. If there are more than one in the basic character string, the string which was found first is entered.

Invocation of the FIND function in the program "search"

PROGRAM search

VAR

```
Basic_Text : STRING := "StartupCondition";  
Search_Text : STRING := "Switch";  
Position : INT;  
END_VAR  
LD Basic_Text  
FIND Search_Text  
ST Position (* Position: 4 *)  
END_PROGRAM
```

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.104 FOR

With the FOR loop, a loop control variable will be set to a specified starting value, then incremented (or decreased), and the loop will be terminated when a given end value is reached.

The syntax is:

```
FOR assignment TO Endvalue BY Increment DO  
Instructions;  
END_FOR;
```

Example

```
VAR  
Field : ARRAY[1..5] OF INT :=[2,14,8,12,5];  
Index : INT;  
MaxIndex : INT :=5;  
Maximum : INT :=0;
```

```
END_VAR
FOR Index :=1 TO MaxIndex BY 1 DO
IF Field[Index] > Maximum THEN
Maximum := Field[Index];
END_IF;
END_FOR;
```

The loop control variable "Index" will start with "1", and will be incremented "BY 1" on each execution of the loop. This will be done until the end value "MaxIndex" (=5) will be reached.

Note: the BY-term is optional and can be omitted. Default then is to increment by 1.

Execution of the FOR-loop:

Initializing of the control variables.

Check of the termination criterion and termination if necessary.

Execution of the instruction block.

Increase/decrease of the control variable about the step size.

Go to step 2.

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.105 FROM

See [Transition](#).

4.2.106 Function

IEC61131-3 defines three block types: [PROGRAM](#), [FUNCTION](#) and [FUNCTION BLOCK](#). See [block types](#) under "Advanced Topics" for more details.

Functions return values by assignment to a variable having the same name and type as the function, e.g.

```
FUNCTION MyFun : INT
```

```
...
```

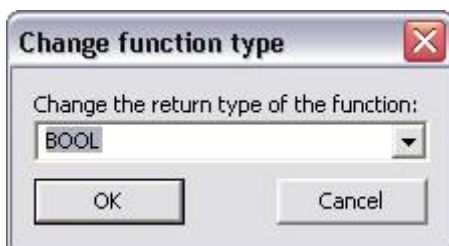
```
MyFun := 999;
```

```
END_FUNCTION
```

Note:

1. Some IEC61131 dialects take the current result at the END_FUNCTION or RETURN as the value to be returned by the function. OpenPCS will ignore this value and only use the value assigned to the function name.
2. The keywords FUNCTION and END_FUNCTION are typically invisible within OpenPCS, as they are maintained by the Editors internally.
3. The function return type (INT in the example shown above) is selected in the same dialog box where you specify the function name, at the very bottom. The default is BOOL.
4. You can also enter user-defined data types ([STRUCTs](#), [ARRAYs](#), etc.) by entering the name of the data type manually into the input-field.
5. To change a return type of a function, open the file in the project browser. Open the change return type dialog by selecting **Edit->Change Return Type...**

The following dialog will pop up:



You can choose one of the given types or type in a user specific one.

4.2.107 FUNCTION BLOCK

IEC61131-3 defines three block types: [PROGRAM](#), [FUNCTION](#) and FUNCTION_BLOCK. See [block types](#) under "Advanced Topics" for more details.

The keywords FUNCTION_BLOCK and END_FUNCTION_BLOCK are typically invisible within OpenPCS, as they are maintained by the editors internally.

4.2.108 GE

Inputs

IN1: ANY input 1

IN2: ANY input 2

Returns

BOOL TRUE if Input 1 is greater or equal than Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.109 GetDateStruct

Input

IN: Date date to convert

InOut

DATESTRUCT_INOUT : DateStruct output of date as a struct

whereas

DateStruct: STRUCT

YEAR : UINT;

MONTH : USINT;

DAY : USINT;

END_STRUCT

GETDATESTRUCT will convert the given date to a DateStruct, providing the date information separated in single integer variables for day, month and year.

4.2.110 GETSYSTEMDATEANDTIME

Inputs

[EN](#): BOOL

Outputs

[ENO](#): BOOL

ODT: DATE_AND_TIME

The function "GetSystemDateAndTime" returns the actual system time in ODT.

Notes:

Standardization: this function block is *not* defined by IEC61131-3

4.2.111 GetTaskInfo

Output

Count: DWORD; (*number of cycles this task is executed *)

LastCT: TIME; (*time needed for last cycle*)

AverageCT: TIME; (*average time needed for execution*)

MinCT: TIME; (*minimum time needed for execution*)

MaxCT: TIME; (*maximum time needed for execution*)

State: DWORD; (*not yet used

GetTaskInfo returns information about the execution time of the last cycle of the current task. This function block has no input parameters.

4.2.112 GetTime

Input

IN1: TIME previous time

Returns

TIME: time elapsed since power on, minus IN1

GETTIME will retrieve the time elapsed since the controller has last been switched on, less the time value supplied as an input. This can be used to easily measure time spans.

Example "Stop Watch"

```
PROGRAM StopW
```

```
VAR
```

```
begin, result : TIME;
```

```
END_VAR
```

```
start:
```

```
LD t#0ms
```

```
GETTIME
```

```
ST begin
```

```
...
```

```
stop:
```

```
LD begin
```

```
GETTIME
```

```
ST result
```

```
END_PROGRAM
```

4.2.113 GetTimeCS

Get current system time

Input

IN1: TIME previous time

Returns

TIME: time elapsed since power on, minus IN1

GETTIME will retrieve the time elapsed at the last system control point since the controller has last been switched on, less the time value supplied as an input. This can be used to easily measure time spans. Compared to GETTIME, GETTIMECS will return the same value when called multiple times within the same cycle.

Example "Stop Watch"

```
PROGRAM StopW
```

```
VAR
```

```
begin, result : TIME;
```

```
END_VAR
```

```
...
```

```
start:
```

```
LD t#0ms
```

```
GETTIMECS
```

```
ST begin
```

```
...
```

```
stop:
```

```
LD begin
```

```
GETTIMECS
```

```
ST result
```

```
END_PROGRAM
```

4.2.114 GetVarData

InOut

VarName: STRING Name of variable requested

Output

Q: bool TRUE if VarInfo is valid

VarData: VarInfo information on variable

The variable specified as input is located within the memory address space and information on that variable is returned. If the variable cannot be located, Q is returned as FALSE.

Please note:

for OpenPCS to be able to locate variables by name, a MAP file has to be generated (resource options)

for definition of VARINFO, see [VARINFO](#) under "keywords".

4.2.115 GetVarFlatAddress

InOut

VarName: STRING Name of variable requested

Output

Q: bool TRUE if VarInfo is valid

Address: DWORD flat memory address of specified variable

The variable specified as input is located within the memory address space and the address of its location is returned. If the variable cannot be located, Q is returned as FALSE.

Please note:

for OpenPCS to be able to locate variables by name, a MAP file has to be generated (resource options)

the memory location returned must not be stored and used in another but the current execution cycle.

4.2.116 GT

Inputs

IN1: ANY Input 1

IN2: ANY Input 2

Returns

BOOL TRUE if Input 1 is greater than Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.117 IF

The IF-instruction has following syntax:

```
IF expression THEN Block  
{ ELSIF expression THEN Block}  
[ ELSE Block ]  
END_IF;
```

If the expression after IF evaluates to "true", the instructions given after THEN will be executed. If the expression after IF evaluates to "false", the instructions after ELSE will be executed or the ELSEIF-condition will be checked. In any case, execution will then continue with the next instruction after END_IF.

Remark:

It is recommended to use the absolute value ABS() of a floating point number if a comparison with 0.0 is to be done since $-0.0 == 0.0$ will not return true.

he following IF instruction will compute the maximum of two numbers:

```
IF a>b THEN  
maximum := a;
```

```
ELSE  
maximum := b;
```

```
END_IF;
```

IF instructions may be nested, i.e. the THEN-part as well as the ELSE-part may contain other IF instructions.

Example:

The following program will again compute the maximum of two numbers, but if this maximum is "a" and "a" is greater than 10, it will be reduced by 1:

```
VAR  
a: INT :=12;  
b: INT :=5;  
maximum: INT;
```

```
END_VAR  
IF a>b THEN  
maximum :=a;  
IF (a>10) THEN  
a:=a-1;  
ELSE  
a:=a+1;  
END_IF;  
ELSE  
maximum :=b;  
END_IF;
```

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.118 IL

IL is the abbreviation of [Instruction List](#), one of the programming languages of IEC61131-3.

4.2.119 IN

This is the name of a formal parameter of a standard function block ([TOE](#)), and as such defined to be a keyword.

4.2.120 INITIAL_STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.121 INSERT

Inputs

IN1: STRING character string

IN2: STRING character string to be inserted

P: ANY_INT (as supported) Starting position. $P < 0$ and $P > \text{LEN}(\text{IN1})$ is invalid.

Returns

STRING Composed string. IN1 for invalid parameters.

The "INSERT" function inserts the string "IN2" into "IN1". The concatenated string consists of the first "P" characters of "IN1", the completed string "IN2" and the rest of "IN1".

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.122 INT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.123 Interval

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.124 JMP

The program flow continues at the position specified by the jump target. The jump target must be a sequence start uniquely identified by a label.

A jump is possible only within a POU.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.125 JMPC

If the CR holds the value TRUE, the program flow continues at the position specified by the jump target. If it holds the value "0", there is no jump. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.126 JMPCN

If the CR holds the value FALSE, the program flow continues at the position specified by the jump target. If it holds the value "1", there is no jump. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.127 L(Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.128 LD

The value of the operand is evaluated and loaded into the current result. This overwrites data stored in CR. The operand is not modified. The data type of the operand determines the permissible data type for consecutive operands.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.129 LD (Ladder Diagram)

LD is the abbreviation of [Ladder Diagram](#), one of the programming languages of IEC61131-3.

4.2.130 LDN

The operand is evaluated, and the current result is loaded with the negated value. The operand is not modified. The data type of the operand determines the permissible data type for consecutive operands.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.131 LEFT

Inputs

IN: STRING character string

L: ANY_INT (as supported) Number of characters to retrieve. L < 0 is invalid.

Returns

STRING the "L" leftmost characters of IN. IN for invalid parameters.

The "LEFT" function enters the left part of the currently loaded character string into the working register. The input operand "L" defines the number of characters to be entered.

4.2.132 LE

Inputs

IN1: ANY input 1

IN2: ANY input 2

Returns

BOOL RUE if Input 1 is less or equal than Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.133 LEN

Inputs

In: STRING character string

Returns

INT length of IN

The function "LEN" determines the length of the character string in the working register (input operand of data type "STRING") and enters the determined value as INT number into the working register.

4.2.134 LIMIT

Inputs

MN: Any_Num lower limit

IN: Any_Num Test value

MX: Any_Num Upper Limit

Returns

Any_Num One of the input values, see description

The "MN" and "MX" values define the lowest and highest limit value. The function compares the test value "IN" with "MN" and "MX". If "IN" is between the two limit values, it is loaded into the working register. If "IN" is smaller than "MN", the "MN" value is output. If "IN" is greater than "MX", the "MX" value is loaded.

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.135 LINT

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by OpenPCS. See [Table 10](#) in the compliance statement.

4.2.136 LN

Input

In: REAL

Returns

REAL: logarithm to the base of e

4.2.137 LOG

Input

In: REAL

Returns

REAL: logarithm to the base of 10

4.2.138 LREAL

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.139 LT

Inputs

IN1: ANY input 1

IN2: ANY Input 2

Returns

BOOL RUE if Input 1 is less than Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.140 Lword

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by OpenPCS. See [Table 10](#) in the compliance statement.

4.2.141 MUX

OpenPCS does not implement the MUX function.

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.142 MAX

Inputs

In1: Any_Num Input Value1

In2: Any_Num Input Value2

...

InN: Any_Num Input ValueN

Returns

Any_Num Maximum of all input values

The "MAX" function determines which input operand has the highest value. The selected operand is loaded into the working register.

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.143 MID

Inputs

IN: STRING character string

L: ANY_INT (as supported) number of characters to retrieve. L < 0 is invalid.

P: ANY_INT (as supported) starting position. P ≤ 0 and P > LEN(IN) is invalid.

Returns

STRING the next " L" characters of IN, starting at the P-th character. IN for invalid parameters.

The "MID" function enters a middle part of the currently loaded character string into the working register. The input operand "P" defines the first character to be entered, "L" defines the number of characters to be entered

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.144 MIN

Inputs

In1: Any_Num Input Value1

In2: Any_Num Input Value2

...

InN: Any_Num Input ValueN

Returns

Any_Num Minimum of all input values

The "MIN" function determines which input operand has the smallest value. The selected operand is loaded into the working register.

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.145 MOD

Input

In1: ANY_INT

In2: ANY_INT

Returns

ANY_INT

The first input will be divided by the second input. MOD delivers the residue to current result.

4.2.146 MOVE

Inputs

In: ANY

Outputs

Out: ANY

The function "MOVE" is an arithmetic function that serves for assigning a value.

4.2.147 MUL

Inputs

In1: ANY_NUM Value to be multiplied

In2: ANY_NUM Value to multiply with

Returns

ANY_NUM product

Multiplies two numbers. See [Table E.1: Error conditions](#) for result on overflow.

Notes:

Standardization: this is an operation defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.148 MUL (time)

Inputs

In1: TIME time duration value

In2: ANY_NUM multiplicand

Returns

TIME multiplied time value

Multiplication of TIME values

Notes:

Standardization: this is an operation defined by IEC61131-3.

4.2.149 N (Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.150 NCC

NCC is an acronym for [native code compiler](#).

4.2.151 NE

Inputs

IN1: ANY input 1

IN2: ANY input 2

Returns

BOOL TRUE if Input 1 is not equal to Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.152 NEG

Input

In: ANY_NUM

Returns

ANY_NUM: negated numeric value of input

4.2.153 NOT

Inputs

IN1: ANYBIT Input

Returns

ANYBIT logical negation (1-complement) of Input

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.154 OF

See [CASE](#)

4.2.155 On

See [RESOURCE](#).

4.2.156 OPC

The var qualifier OPC allows a user, to mark dedicated variables, to become part of the variable table, already within the declaration editor of OpenPCS.

See [Declaration Sections](#)

4.2.157 OR

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bit by bit OR of Input 1 and Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.158 ORN

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bitwise OR of Input 1 and negated Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.159 P(Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.160 POINTER

The data type pointer is defined by OpenPCS in addition to IEC61131-3. Using this data type, it is now possible to call Functions or Functionblocks with arrays of different sizes. A pointer must be declared as follows:

VAR

IntVar : INT;

 pInt : POINTER;

END_VAR

To access the address of a variable, the address operator("&") must be written in front of the variable's name.

Example IL: LD &IntVar

Example ST: pInt := &IntVar;

4.2.161 POU

POU is the abbreviation of Program Organization Unit, meaning a Program, Function or Function Block written in one of the programming languages of IEC61131-3.

4.2.162 Priority

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.163 PROGRAM

IEC61131-3 defines three block types: [PROGRAM](#), [FUNCTION](#) and [FUNCTION BLOCK](#). See [block types](#) under "Advanced Topics" for more details.

The keywords PROGRAM and END_PROGRAM are typically invisible within OpenPCS, as they are maintained by the editors internally.

4.2.164 PT

This is the name of a formal parameter of a standard function block ([TOF](#)), and as such defined to be a keyword.

4.2.165 PV

This is the name of a formal parameter of a standard function block ([CTD](#)), and as such defined to be a keyword.

4.2.166 Q(Parameter)

This is the name of a formal parameter of a standard function block ([CTD](#)), and as such defined to be a keyword.

4.2.167 Q1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

4.2.168 QD

This is the name of a formal parameter of a standard function block ([CTUD](#)), and as such defined to be a keyword.

4.2.169 QU

This is the name of a formal parameter of a standard function block ([CTUD](#)), and as such defined to be a keyword.

4.2.170 R(Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.171 R(eset)

The operand is reset, if the content of the CR equals "1". If this precondition is not met, operands will not be changed. The CR is not modified.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.172 R_EDGE

R_EDGE is used to indicate a rising edge detection function on Boolean inputs. This leads to an implicit declaration of a function block of type [R_TRIG](#).

Example:

```
FUNCTION_BLOCK AND_EDGE
```

```
VAR_INPUT
```

```
X : BOOL R_EDGE;
```

```
Y : BOOL F_EDGE;
```

```
END_VAR
```

```
VAR_OUTPUT
```

```
Z : BOOL ;
```

END_VAR

Z := X AND Y ; (* ST language example *)

END_FUNCTION_BLOCK

4.2.173 R_TRIG

Inputs

CLK: bool Input operand whose rising edge is detected

Outputs

Q: bool Output operand; indicates the rising edge of "CLK"

The function block "R_TRIG" detects the status of the input operand "CLK". The status change from "0" to "1" in a processing cycle is detected and indicated with the Boolean value "1" via the output "Q". The output is "1" only in the processing cycle in which the change of the status of "CLK" is detected and a rising edge is indicated.

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.174 R1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

4.2.175 READ_ONLY

This keyword is defined by IEC61131-3 for the definition of Access Paths. OpenPCS does not support Access Paths, hence you will not be able to use this keyword with OpenPCS.

4.2.176 READ_WRITE

This keyword is defined by IEC61131-3 for the definition of Access Paths. OpenPCS does not support Access Paths, hence you will not be able to use this keyword with OpenPCS.

4.2.177 Refresh project information

Project -> Refresh project information refreshes the project information and writes the project internal newly. Thus e.g., prototypes are newly read in and libraries are re-freshed.

4.2.178 REAL

See [Elementary Data Types](#)

Hint: The (internal) binary representation of floating point numbers is hardware-dependent and some values cannot even be stored correctly. So comparisons of value equality may fail. Therefore it's a better way to use intervals with relative or absolute error tolerance.

Using floating numbers may run out of precision under certain circumstances. For instance a float type is 32 bits wide. Twenty four of these bits are devoted to the significand (what used to be called the mantissa) and the rest to the exponent. The number 16777216 is 2^{24} and so there is no precision left to represent $16777216+1$. If floating number is used in a counter like $x = x + 1.0$, the counter will stay at 16777216 once this value is reached.

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.179 REAL_TO_*

Inputs

original data type real

Returns

converted data type *

The function block converts the first value of type real into the same value of type *.
The following data types can be converted:

BOOL

Values within the interval $\pm 1,175494351e-38$ are cast to false all other values to true.

Examples:

1.1 -> true
-22.33 -> true
1.1e-39 -> false

DINT, INT und SINT

Values are rounded off, therefore values smaller than x.5 are rounded to the absolute smaller number else to the next larger one.

Examples:

0.3 -> 0
-0.6 -> -1
-1.5 -> -2

BYTE, DWORD, WORD und USINT, UINT, UDINT

The conversion is analog to an integer-conversion for positive values

Negative values are cast to the new size and the generated bit pattern is interpreted as a positive number

Examples:

-1.6 -> 254 (USINT), 65534 (UINT), 4294967294 (UDINT); (A sint -2 has the bit pattern: 1111 1110 which is interpreted as 254)
33.3 -> 33

STRING

For converting string function `Sprintf(str, "%#g", value);` is used.

Examples:

0.0 -> "0.000000"
123.45678 -> " 123.456"
-12.345678 -> " -12.3456"
12345678.9 -> " 1.23457e+007"
0.000000123 -> " 1.23000e-007"

4.2.180 Release

This is the name of a formal parameter of a standard function block ([SEMA](#)), and as such defined to be a keyword.

4.2.181 REPEAT

In contrast to the other loop types, REPEAT will check the loop expression after execution of the loop. The syntax is:

```
REPEAT  
instructions;  
UNTIL expression  
END_REPEAT;
```

So, the REPEAT loop will always be executed at least once. Example:

```
VAR  
i : INT := -1;  
END_VAR  
REPEAT  
i:=i-1;  
UNTIL i < 0  
END_REPEAT;  
(* now, i = -2 *)
```

Although "i" will meet the loop condition from the beginning, the REPEAT loop will be executed once anyway.

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.182 REPLACE

Inputs

IN1: STRING Basic character string in which a part should be replaced

IN2: STRING New character string

L: ANY_INT (as supported) Length of the substring which should be cut out off "IN1". L < 0 invalid.

P: ANY_INT (as supported) Starting position of the inserted string. P < 0 and P > LEN(IN1) invalid.

Returns

STRING New composited. IN1 for invalid parameters.

The function "REPLACE" replaces a substring of length "L" starting at position "P" within the given string "IN1" by the string "IN2".

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.183 Resource

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.184 RESUME

The Resume function block enables to resume the execution after its has been stopped, e.g. in an interrupt task for error handling.

Outputs:

Q: BOOL TRUE if succeeded

4.2.185 RET

The "RET" instruction causes an unconditioned return jump to the calling POU - if this POU is the program POU, a return jump to the system program. When jumping back, the calling POU is resumed at the point of interruption. Delayed operations will be executed.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.186 RETAIN

RETAIN is the keyword to declare variables as retentive, and is optional after VAR, VAR_GLOBAL. Implementation of retentiveness depends on your controller. See [declaration sections](#).

4.2.187 RETC

Conditional Return

Instruction does not take any operands.

If the CR holds the value "1", a return jump to the calling POU is performed - i.e. to the system program if calling POU is of type "program". If the CR holds the value "0", there is no return jump. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.188 RETCN

Conditional Return

Instruction does not take any operands.

Conditioned return jump depending on the Boolean content of the CR.

If the CR holds the value "0", a return jump to the calling POU is performed - i.e. to the system program if calling POU is of type "program". If the CR holds the value "1", there is no return jump. The program flow continues with the instruction following the jump instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.189 RETURN

The RETURN instruction will cause the current POU to be left, transferring control back to the caller of the current POU. Note that on working with functions, the function value (variable with the name of the function) must be assigned. If output values of function blocks aren't assigned by local values of the function block, they have the predefined values of their data types.

Example:

```
IF a<b THEN
RETURN;
END_IF;
```

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.190 RIGHT

Inputs

IN: STRING character string

L: ANY_INT (as supported) Number of characters to retrieve. L < 0 is invalid.

Returns

STRING the "L" rightmost characters of IN. IN for invalid parameters.

The "RIGHT" function enters the right part of the currently loaded character string into the working register. The input operand "L" defines the number of characters to be entered.

4.2.191 ROL

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, rotated left N bits

The leftmost bits will be rotated in from right

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.192 ROR

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, rotated right N bits

The rightmost bits will be rotated in from left.

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.193 RS

Inputs

Set: bool Set condition

Reset1: bool Reset condition

Outputs

Q1: bool Output state of the bistable element

The characteristic feature of the "RS" function module is to have a state corresponding to its output variable Q1 and to have a dominant input Reset1.

1. If Reset1 is true: Q1 is always false.
2. If Reset1 is false: Q1 is true, if it was true before or Set is true.

Q1 is initially false.

Formula

$Q1 \text{ } \beta \text{ } \text{NOT} (\text{Reset1}) \text{ AND } (Q1 \text{ OR } \text{Set})$

Table (Karnough Map)

RS		Set, Reset1			
		00	01	11	10
Q1	0	0	0	0	1
	1	1	0	0	1

	1				
--	---	--	--	--	--

Notes

Standardization: this function block is defined by IEC61131-3.

4.2.194 RTC

The RTC function block sets the output CDT to the input PDT if IN=1. Otherwise CDT is invalid

Inputs:

IN: BOOL

PDT: DATE_AND_TIME Present date and time

Outputs

Q: BOOL copy of IN

CDT: DATE_AND_TIME Current date and time, valid when IN=1

Notes:

Standardization: this function block is defined by IEC61131-3

4.2.195 S(Action Qualifier)

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.196 S(et)

The operand is set, if the content of the CR equals "1". If this precondition is not met, operands will not be changed. The CR is not modified.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.197 S1

This is the name of a formal parameter of a standard function block, and as such defined to be a keyword.

4.2.198 SD

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.199 SEL

This is the name of a standard function block, which is defined in IEC61131-3, but not provided by OpenPCS. See [Table 31](#) in the compliance statement.

4.2.200 SEMA

This is the name of a standard function block, which is defined in IEC61131-3, but not provided by OpenPCS. See [Table 34](#) in the compliance statement.

4.2.201 SETSYSTEMDATEANDTIME

Inputs

[EN](#): BOOL

[IDT](#): DATE_AND_TIME

Outputs

[ENO](#): BOOL

The function "SetSystemDateAndTime" sets the actual system time in IDT.

Notes:

Standardization: this function block is *not* defined by IEC61131-3.

4.2.202 SFC

SFC is the abbreviation of [Sequential Function Chart](#), one of the programming languages of IEC61131-3.

4.2.203 SHL

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, shifted left N bits

Rightmost bits will be filled with zeros

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.204 SHR

Inputs

IN: ANY_BIT Bit Pattern

N: UINT Number of bits to shift

Returns

ANY_BIT IN, shifted right N bits

Leftmost bits will be filled with zeros

Notes:

Standardization: this function is defined by IEC61131-3

4.2.205 SIN

Input

In: REAL

Returns

REAL: sine of input

4.2.206 Single

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.207 SINT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.208 SL

This is an Action qualifier, see [Table 45](#) in the compliance statement. As OpenPCS only supports actions of type N, you will not need to use this keyword with OpenPCS.

4.2.209 SQRT

Input

In: REAL

Returns

REAL: square root of input

SQRT will compute the square root of the input

4.2.210 SR

Inputs

Set1: bool Set condition

Reset: bool Reset condition

Outputs

Q1: bool Output state of the bistable element

The characteristic feature of the "SR" function module is to have a state corresponding to its output variable Q1 and to have a dominant input Set1.

1. If Set1 is true: Q1 is always true.
2. If Set1 is false: Q1 is true, if it was true before and Reset is false.

Q1 is initially false.

Formula

$Q1 \beta \text{ Set1 OR } (Q1 \text{ AND NOT Reset })$

Table (Karnough Map)

SR	Set1, Reset		
	01	11	10
00			

Q1	0	0	0	1	1
		1	0	1	1
	1				

Notes

Standardization: this function block is defined by IEC61131-3.

4.2.211 ST

The content of the CR register is assigned to the operand. This overwrites the value of the operand. The data type of the operand must match the data type of the data element in the register. The data type of the CR is determined by the data type of the variable first assigned a value. Further assignments will then be possible only if the types of further variables match. An assignment may be followed by another assignment.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3,

4.2.212 ST (Structured Text)

ST is the abbreviation [Structured Text](#), one of the programming languages of IEC61131-3.

4.2.213 STEP

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.214 STN

The negated content of the CR register is assigned to the operand. This overwrites the value of the operand. The data type of the operand must match the data type of the data

element in the register. The CR register is not modified by this operation. An assignment "STN" may be followed by another "ST" or "STN" instruction.

Notes:

This is a keyword in language Instruction List.

This is defined by IEC61131-3.

4.2.215 STRING

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.216 STRING_TO_*

Inputs

original data type string

Returns

converted data type *

The function block converts the first value of type string into the same value of type *. The following data types can be converted:

BOOL

The strings "1" and "true" are converted to true, the rest to false.

DINT, INT und SINT

The string is read from left to right until an illegal character or the word is finished.

Examples:

"-1" -> -1

"213hallo" -> 213

"23.5" -> 23

BYTE, DWORD, WORD und USINT, UINT, UDINT

The conversion is analog to an integer-conversion for positive values

Negative values are cast to the new size and the generated bit pattern is interpreted as a positive number

Examples:

"-1.6" -> 254 (USINT), 65534 (UINT), 4294967294 (UDINT); (A sint -2 has the bit pattern: 1111 1110 which is interpreted as 254)

"33.3" -> 33

REAL

Analog the above conversion. The e-Notation is permitted

Examples:

"-123.456" -> -123.456

"0.23" -> 0.23

"-1.2e-2" -> -0.012

4.2.217 STRUCT

STRUCT is the keyword to define structured data types, see and [Derived Data Types](#)

A variable consisting of two members:

VAR

x1: STRUCT

x2: INT;

```
x3: BOOL;  
  
END_STRUCT;  
  
END_VAR
```

A variable of user defined type:

```
TYPE  
  
x4: STRUCT  
  
x5: REAL;  
  
x6 : BOOL;  
  
END_STRUCT;  
  
END_TYPE  
  
VAR  
  
x7: x4;  
  
END_VAR
```

4.2.218 SUB

Inputs

In1: ANY_NUM

In2: ANY_NUM

Returns

ANY_NUM Difference In1-In2

Subtraction of two numbers.

Notes:

Standardization: this is an operation defined by IEC61131-3.

The function *Append Input Connector* is not available with this function block

4.2.219 SUB (time)

Inputs

In1: TIME time duration value

In2: TIME

Returns

TIME difference between the two time values provided

Subtraction of TIME values

Notes:

Standardization: this is an operation defined by IEC61131-3.

4.2.220 TAN

Input

In: REAL

Returns

REAL: tangent of input

4.2.221 Task

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.222 THEN

See [IF](#)

4.2.223 TIME

See [Elementary Data Types](#)

See also [Constants](#) on how to create TIME-constants.

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.224 TIME_OF_DAY

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.225 TIME_TO_*

Inputs

original data type time

Returns

converted data type *

The function block oververts the first value of type time into the same value of type *.

The following data types can be converted:

BOOL

BYTE

DINT

DWORD

INT

REAL

SINT

STRING

UDINT

UINT

USINT

WORD

Notes:

Standardization: this function is defined by IEC61131-3.

Except TIME_TO_DINT and TIME_TO_REAL, all TIME convert functions are only available within the Ladder-Diagram-Editor.

4.2.226 TO

See [FOR](#)

4.2.227 TOD

TOD can be used as an abbreviation to [TIME_OF_DAY](#) when specifying the data type of a [literal constant](#). As data type TIME_OF_DAY is not implemented in OpenPCS, you will not be able to use this keyword with OpenPCS.

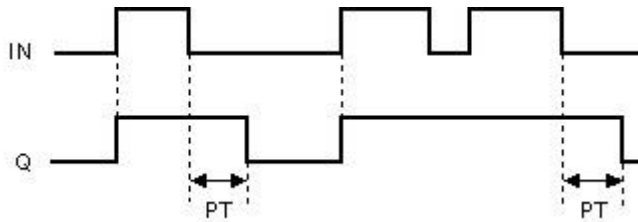
4.2.228 TOF

If the state of the input operand "IN" is "1", this will be passed to the output operand "Q" without any delay. If there is a falling edge, a timer function will be started lasting as long an interval as specified by the operand "PT"

It is after the time is up that the operand "Q" will change to the state "0". If the "PT" value changes after the start, it will have no implications until there is the next rising edge of the operand "IN".

The operand "ET" contains the current timer value. If the time is up, the operand "ET" will keep its value as long as the operand "IN" has the value "0". If the state of the "IN" operand changes to "1", the value of "ET" will switch to "0".

If the input "IN" is switched off, this will switch off the output "Q" after an interval specified by the delay value.



Inputs:

IN: Start condition

PT: time Initial time value

Outputs

Q: bool binary state of the timer

ET: time current time value

Notes:

Standardization: this function block is defined by IEC61131-3

4.2.229 TON

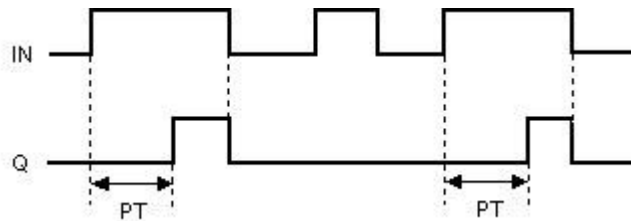
The rising edge of the input operand "IN" will start the timer "ON", and it will run as long a time interval as specified by the operand "PT".

While the timer is running, the output operand "Q" will have the value "0". If the time is up, the state will change to "1" and keep this value until the operand "IN" changes to "0".

If the "PT" value changes after the timer has been started, this will have no implications until the next rising edge of the operand "IN".

The output operand "ET" contains the current timer value. If the time is up, the operand "ET" will keep its value as long as the operand "IN" has the value "1". If the state of the "IN" operand changes to "0", the value of "ET" will switch to "0".

If the input "IN" is switched on, this will switch on the output "Q" after an interval specified by the delay value.



Inputs:

IN: Start condition

PT: time Initial time value

Outputs

Q: bool binary state of the timer

ET: time current time value

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.230 TP

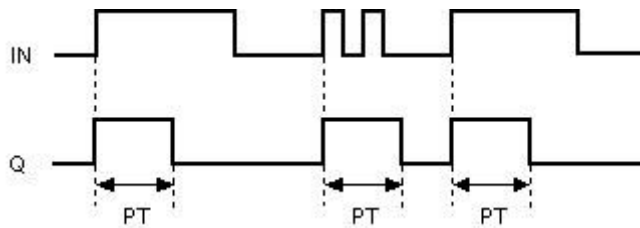
A rising edge of the input operand "IN" will start the timing function of the timer "TP", and it will run as long an interval as specified by the operand "PT".

While the timer is running, the output operand "Q" will have the state "1". Any changes of state at the input "IN" will have no implication on the procedure.

If the "PT" value changes after the start, this will not have any implication before the next rising edge of the "IN" operand.

The output operand "ET" contains the current timer value. If the operand "IN" has the state "1" after the time is up, the operand "ET" will keep its value.

Every edge occurring while the timer is not running will cause an impulse at the output Q that lasts as long as specified.



Inputs

IN: bool start timer

PT: time initial time value

Outputs

Q: bool binary state of timer

ET: time elapsed time

Notes:

Standardization: this function block is defined by IEC61131-3.

4.2.231 Transition

This keyword is defined by IEC61131-3 for the textual representation of programming language [SFC](#). OpenPCS does not support the textual representation of SFC, hence you will not be able to enter this keyword. You will see this when printing SFC.

4.2.232 TRUE

Constant value of type [BOOL](#).

4.2.233 TRUNC

Inputs

In: REAL

Returns

ANY_INT

Returns the integer part of the supplied real value.

Notes:

Standardization: this function is defined by IEC61131-3.

4.2.234 TYPE

See [Declaration Sections](#) and [Derived Data Types](#)

Notes:

This is a keyword only for declaration parts of POUs.

This is defined by IEC61131-3.

Keywords TYPE .. END_TYPE should not be nested within a VAR..END_VAR block, but rather be on top level in the declaration section, or in a type declaration file on project level.

4.2.235 UDINT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.236 UINT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.237 ULINT

This is the name of an elementary data type, which is defined by IEC61131-3, but not supported by OpenPCS. See [Table 10](#) in the compliance statement.

4.2.238 UNTIL

See [REPEAT](#)

4.2.239 USINT

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3

4.2.240 VAR

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POUs.

This is defined by IEC61131-3.

4.2.241 VAR_ACCESS

This keyword is defined by IEC61131-3 for the definition of Access Paths. OpenPCS does not support Access Paths, hence you will not be able to use this keyword with OpenPCS.

4.2.242 VAR_INPUT

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POU's.

This is defined by IEC61131-3.

4.2.243 VAR_OUTPUT

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POU's.

This is defined by IEC61131-3.

4.2.244 VAR_IN_OUT

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POU's.

This is defined by IEC61131-3.

4.2.245 VAR_GLOBAL

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POU's.

This is defined by IEC61131-3.

4.2.246 VAR_EXTERNAL

See [Declaration Sections](#)

Notes:

This is a keyword only for declaration parts of POUs.

This is defined by IEC61131-3.

4.2.247 VARINFO

VARINFO is defined as

VARINFO : Struct

TYP : UINT;

SIZE : UINT;

PROG : UINT;

SEG : UINT;

OFFSET: UINT;

BIT: UINT;

SCOPE: UINT;

end_struct;

4.2.248 WHILE

The WHILE loop will execute the loop body as long as the given expression evaluates to "true". Syntax:

```
WHILE expression DO  
instructions;  
END_WHILE;
```

The expression given after the keyword **WHILE** will be evaluated before entering the loop. If it is true, the loop body will be executed. This will terminate only when the expression evaluates to "false".

Example

```
VAR  
i : INT := 3;  
END_VAR  
WHILE i > 0 DO  
i:=i-1;  
END_WHILE;
```

Initially, "i" equals 3. 3 is greater than 0, so the expression after WHILE is true and the loop body executed. This will decrement the value of "i" to 2. 2 is still greater than 0, so the loop body will be executed again. Sometime later, the loop body will decrement "i" from 1 to 0. On the next check, the expression after WHILE will be false, hence the loop body will not be executed again.

Notes:

This is a keyword only for language ST.

This is defined by IEC61131-3.

4.2.249 WITH

This keyword is defined by IEC61131-3 for the textual definition of configurations, resources and tasks. With OpenPCS, these are defined and configured using [property-dialog boxes](#). You will see this keyword in OpenPCS only when [printing](#) the definition of a configuration.

4.2.250 WORD

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.251 WSTRING

See [Elementary Data Types](#)

Notes:

Standardization: this is a data type defined by IEC61131-3.

4.2.252 XOR

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bitwise XOR of Input 1 and Input 2

Notes:

Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.2.253 XORN

Inputs

IN1: ANY_BIT Input 1

IN2: ANY_BIT Input 2

Returns

ANY_BIT logical, bitwise XOR of Input 1 and inverted Input 2

Notes:

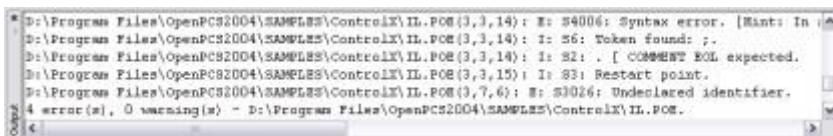
Standardization: this function is defined by IEC61131-3.

The feature *Append Input Connector* is available for this function block

4.3 Errors and Warnings

4.3.1 How to Read Error Message

In the [Output Window](#) you will find any error messages from the compiler.



```

D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC(3,3,14): E: S4004: Syntax error. [Hint: In
D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC(3,3,14): I: S56: Token found: ;.
D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC(3,3,14): I: S2: . [ COMMENT EOL expected.
D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC(3,3,15): I: S3: Restart point.
D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC(3,7,6): E: S3026: Undeclared identifier.
4 error(s), 0 warning(s) - D:\Program Files\OpenPCS2004\SAMPLES\ControlX\IL.POC.
  
```

Each error message line fits the following style:

1. The file name including path of the source code that caused the error message.
2. A triple of numbers where the first number indicates the section the error occurred ("2" for "Declaration" and "3" for "Instruction"), the second is the line and the last the column (within the section mentioned before).
3. A capital letter indicates the type of message:
 letter stands for
 I Info
 E Error
 W Warning
 F Fatal Error
4. The error number code that allows you to find a detailed error description here in the documentation.
5. A short description of the error.

4.3.2 General Errors

4.3.2.1 G10001

Warning G10001: The file [file name] is inconsistent. You should not use it.

The File is inconsistent. A reason might be that the file name is different from the POU name within the file. This is normally caused by renaming files outside of OpenPCS. POUs should always be renamed by using the OpenPCS function **File->File->Rename**.

4.3.3 Syntax Errors

4.3.3.1 S1000

Nested comments are not allowed.

You are using an IEC 61131-3 compatible version. In this version nested comments are not allowed.

4.3.3.2 S1001

Invalid character.

An unsupported character was used. See also [Table 1: Character set features](#)

4.3.3.3 S1002

End of file found in comment.

The end of the file was reached before an open comment has been closed. Please close the comment before calling the syntax check.

4.3.3.4 S1003

Reserved keyword.

A reserved keyword was used as an identifier.

4.3.3.5 S1004

Invalid value for hour.

The numeric value for the hour unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [0, 23].

4.3.3.6 S1005

Invalid value for minute.

The numeric value for the minute unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [0, 59].

4.3.3.7 S1006

Invalid value for second.

The numeric value for the seconds unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be a fixed point number in the range [0, 60).

4.3.3.8 S1008

Invalid value for month.

The numeric value for the month unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [1, 12].

4.3.3.9 S1009

Invalid day range.

The numeric value for the day unit of a TIME_OF_DAY or a DATE_AND_TIME literal must be an integer in the range [1, 31], giving the day of the month. I. e. if the respective month has less than 31 days, the maximum number of days in the month is the greatest valid value for the day literal.

4.3.3.10 S1010

Exponent too large.

The numeric value for the exponent of a real literal must be an integer in the range [-37, 38] and for a LREAL literal an INT in the range [-307, 308].

4.3.3.11 S1011

Incorrect direct address.

The numeric value for a location field in the hierarchical address of a directly represented variable is hardware dependent integer, but must not exceed 4294967295. Please consult your hardware documentation to determine the maximum value for each field in the address hierarchy.

4.3.3.12 S1012

Invalid day entry.

The numeric value for the day unit of a TIME literal must be a fixed point number in the range [0, 255].

4.3.3.13 S1013

Invalid hour entry.

The numeric value for the hour unit of a TIME literal must be a fixed point number in the range [0, 24] if the hour is not the most significant unit of the duration literal. An overflow is only permitted if the hour unit is the most significant unit of the TIME literal.

Example:

T#25h_15m is permitted.

T#1d_25h_15m is not allowed. The correct representation of this duration literal is:
T#2d_1h_15m.

4.3.3.14 S1014

Invalid minutes entry.

The numeric value for the minute unit of a TIME literal must be a fixed point number in the range [0, 60] if minute is not the most significant unit of the duration literal. An overflow is only permitted if the minute unit is the most significant unit of the TIME literal.

Example:

T#75m is permitted.

T#5h_75m is not allowed. The correct representation of this duration literal is:
T#6h_15m.

4.3.3.15 S1015

Invalid seconds entry.

The numeric value for the seconds unit of a TIME literal must be a fixed point number in the range [0, 60] if seconds are not the most significant unit of the duration literal. An overflow is only permitted if the seconds unit is the most significant unit of the TIME literal.

Example:

T#75s is permitted.

T#5m_75s is not allowed. The correct representation of this duration literal is:
T#6m_15s.

4.3.3.16 S1016

Invalid milliseconds entry.

The numeric value for the milliseconds unit of a TIME literal must be a fixed point number in the range [0, 1000] if the milliseconds are not the most significant unit of the duration literal. An overflow is only permitted if the milliseconds unit is the only unit of the TIME literal.

Example:

T#1200s is permitted.

T#1s_1200ms is not allowed. The correct representation of this duration literal is:
T#2s_200ms.

4.3.3.17 S1017

Direct address too complex.

The maximum number of location fields in the address hierarchy of a directly represented variable is hardware dependent but must not exceed 8. Please consult your hardware documentation to determine the maximum depth of the address hierarchy.

4.3.3.18 S1018

Integer constant too large/small.

A constant's value must be in the range of representable values for its type. The type of an integer constant depends on the type of the variable the constant is assigned to but must not exceed the range of a LINT/ULINT (8 byte integer/unsigned integer) constant.

4.3.3.19 S1019

Integer constant too large/small (does not fit into 32 bits).

The numeric value of the given constant exceeds the range of values of type DINT/UDINT.

4.3.3.20 S1020

Numeric value too large/small.

A constant's value must be in the range of representable values for its type. The type of a signed integer constant depends on the type of the variable the constant is assigned to but must not exceed the range of a LINT (8 byte integer) constant.

4.3.3.21 S1021

Error while processing a floating-point function of the math library.

4.3.3.22 S1022

Invalid string constant.

The given string constant contains an invalid character. A character string literal is a sequence of zero or more characters prefixed and terminated by the single quote character ('). Valid characters are any printable character except "\$". The three-character combination of the dollar sign (\$) followed by two hexadecimal digits shall be interpreted as an hexadecimal representation of the eight bit character code as shown in table [Character string literal feature](#).

Additionally, two-character combinations beginning with the dollar sign shall be interpreted as shown in table [Two-character combinations in character strings](#) when they occur in character strings.

4.3.3.23 S1023

Invalid number (i.e., numerical constant).

The given numeric constant contains an invalid character. See table [Numeric literals](#) for examples of valid numeric literals.

4.3.3.24 S1024

Invalid constant.

The given constant contains invalid characters.

For a list of valid constant representations see [Table 53: Function block invocation features for IL language](#).

4.3.3.25 S1025

Invalid direct address.

A directly represented variable contains invalid characters.

The direct representation of a variable shall be provided by the concatenation of the percent sign "%", a [location prefix](#), an optional [size prefix](#) and one or more unsigned integers separated by periods (.)

The manufacturer shall specify the correspondence between the direct representation of a variable and the physical or logical location of the addressed item in memory, input or output. When a direct representation is extended with additional integer fields separated by periods, it shall be interpreted as a hierarchical physical or logical address with the leftmost field representing the highest level of the hierarchy, with successively lower levels appearing to the right. For instance, the variable %IW2.5.7.1 may represent the first "channel" (word) of the seventh "module" in the fifth "rack" of the second "I/O bus" of a programmable controller system.

The use of directly represented variables is only permitted in programs. The maximum number of levels of hierarchical addressing is hardware dependent and must not exceed 8.

Please consult your hardware documentation to determine the maximum levels of hierarchical addressing.

4.3.3.26 S1026

Invalid identifier (name, variable, parameter,...)

An identifier contains one or more invalid characters.

An identifier is a string of letters, digits, and underline characters which shall begin with a letter or underline character. The letters can be upper or lower case. Multiple leading or multiple embedded underlines are not allowed.

Imbedded space characters are not allowed.

4.3.3.27 S1027

End of file found in file header.

An error occurred while reading the file header. You can fix this error, by opening the file with a text editor and removing all lines preceding the PROGRAM, FUNCTION or FUNCTION_BLOCK keyword. If this error occurs more often, please contact your manufacturer.

4.3.3.28 S1028

This identifier is too long (> 64 characters).

The length of an identifier is greater than the maximum supported length. In this implementation only identifiers up to 64 characters are supported.

4.3.3.29 S1029

This word (identifier, constant literal, string, comment) is too long (> 1024 characters).

A token (identifier, constant literal, string, comment) exceeds 1024 characters. In this implementation only tokens up to 1024 characters are supported.

4.3.3.30 S1030

Too many identifiers.

The maximum number of identifiers has been exceeded. Maximum 65535 identifiers are supported.

4.3.3.31 S1031

Unallowed usage of EN. Just allowed as an identifier for a bool variable in input section.

A variable with the name "EN" has been declared in the wrong variable section or with incorrect type.

The name "EN" (enable) is reserved for Boolean input variables.

If the value of EN is FALSE when the function or function block is invoked the operations defined by the function/function block shall not be executed. If the Boolean output parameter ENO has been defined too than the value of ENO is reset to FALSE.

If the value of EN is TRUE when the function or function block is invoked the operations defined by the function/function block are executed. These operations can include the assignment of a Boolean value to the Boolean output parameter ENO, if this parameter has been defined too.

4.3.3.32 S1032

Unallowed usage of ENO. Just allowed as an identifier for a bool variable in output section.

A variable with the name "ENO" has been declared in the wrong variable section or with incorrect type.

The name "ENO" (Enable Out) is reserved for Boolean output variables. The variable "ENO" requires the Boolean input variable "EN".

If the value of EN is FALSE when the function or function block is invoked the operations defined by the function/function block shall not be executed and the output parameter ENO is reset to FALSE.

If the value of EN is TRUE when the function or function block is invoked the operations defined by the function/function block are executed. These operations can include the assignment of a Boolean value to ENO.

4.3.3.33 S3000

Function block not declared.

A CAL to an unknown function block instance has been found.

An instance of a function block must be declared before it can be used.

Tips

Make sure that an instance of the requested function block is declared in one of the variable declaration sections.

Make sure the name of the name of the function block instance is spelled correctly.

4.3.3.34 S3001

Function not present.

A call to an unknown function has been found.

A function must be declared before it can be used. The parameters that a function uses must be specified in a declaration, or prototype, before the function can be used.

Tips

Make sure that the file containing the declaration or prototype of the function is in the scope of the project or that the function is part of the firmware.

Make sure the name of the name of the function is spelled correctly.

4.3.3.35 S3002

Incorrect parameter.

The requested parameter was not found in the formal parameter list of the function block.

Tips

Make sure the name of the name of the parameter is spelled correctly.

Make sure that the parameter list of the function block-definition contains a parameter with the name used in the assignment.

4.3.3.36 S3003

Jump label not present.

A JMP instruction to an unknown label has been found.

A label has to be defined in the instruction part of the program unit in which it is used.

Tips

Make sure that a the label is defined in the same program unit.

Make sure the name of the name of the label is spelled correctly.

4.3.3.37 S3004

Multiple assignment of a variable/name.

The given identifier was defined more than once.

Tips

Make sure the identifier has not been defined twice in the same program unit.

Make sure the identifier has not been used in a user type declaration, a global type declaration or as a function, function block or program name.

4.3.3.38 S3005

This is not a function block instance.

A variable with the name used in a CAL-statement has been found but is not an instance of a function block.

Tips

Make sure that the identifier is spelled correctly.

Make sure that a function block instance with the specified name has been declared either in the scope of the program unit or in the global scope.

4.3.3.39 S3006

This is not a struct variable or a function block instance.

An access to a member of a struct or function block variable has been attempted, but the variable specified by the identifier is not a function block or a struct.

Tips

Make sure that the identifier is spelled correctly.

Make sure that the variable with the given name is a struct or a function block.

4.3.3.40 S3007

This is not a FUNCTION-POU.

An identifier used as a function name has been defined but is not a function name.

Tips

Make sure that the identifier is spelled correctly.

Make sure that the identifier is the name of a function and not the name of a function block.

Make sure that a function invocation and not a call of a function block instance has been desired on the specified position.

4.3.3.41 S3008

No structure element or block parameter.

An access to a member of a struct or function block variable has been attempted, but the member specified by the identifier is not a parameter of the accessed function block or struct instance.

Tips

Make sure that the identifier is spelled correctly.

Make sure that the right function block or struct instance is used.

If the accessed variable is an instance of a function block make sure that the function block has a parameter with the name given by the identifier.

If the accessed variable is an instance of a struct, make sure that the struct has a member with the name given by the identifier.

4.3.3.42 S3009

No jump label.

The identifier used in the JMP/JMPC/JMPCN-statement at the given position has been found but is not a label name.

Tips

Make sure that the identifier is spelled correctly.

Make sure that identifier used after the JMP/JMPC/JMPCN-statement is a label name.

4.3.3.43 S3010

Type or function block name expected.

A type or a function block name has been expected. The identifier has been found in the current scope but is neither a type nor a function block name.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not a variable name (e. g. a function block name).

4.3.3.44 S3011

Identifier is not a variable or type name.

A variable or a function block instance has been expected. The identifier has been found in the current scope but is neither a variable nor a function block instance.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not a type name (e. g. a function block name).

4.3.3.45 S3012

Variable name or constant expected.

This error occurs, if an identifier, which is not a variable name or an enum constant, is used where a variable name or a constant is expected.

Example:

TYPE

Colors : (red, yellow, blue) := red;

END_TYPE

VAR

Color : Colors := Colors; (* Error: Enum constant expected.

EnumType is a type name *)

END_VAR

LD Colors (* Error: constant or variable name expected. EnumType is a type name *)

ST Color

4.3.3.46 S3014

Numeric data type expected.

Operator and operand type are incompatible. An operand of an ANYNUM type has been expected.

4.3.3.47 S3016

Bit data type expected.

Operator and operand type are incompatible. An operand of an ANYBIT type has been expected.

4.3.3.48 S3017

Boolean value expected.

Operator and operand type are incompatible. An operand of type BOOL has been expected.

4.3.3.49 S3018

Numeric data type expected.

Illegal operand type. Operand of an ANYNUM type expected.

4.3.3.50 S3019

Operators of type incompatible.

Operand and result type are incompatible.

4.3.3.51 S3020

Operand types incompatible.

This error occurs if an illegal combination of time and date data types is used for the input parameters of a SUB operation. For allowed combination of the input and output data types for this operation see [Table 30 - Functions of time data types](#) in the IEC 1131-3 Compliance Statement.

Example:

VAR

TimeVar : TIME;

DateVar : DATE;

END_VAR

LD DateVar

SUB TimeVar

(* Error: SUB is not defined for the this combination of input parameters *)

ST DateVar

4.3.3.52 S3022

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of type TIME or of an ANYNUM type has been expected.

4.3.3.53 S3023

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of type TIME, TIME_OF_DAY, DATE_AND_TIME or of an ANYNUM type has been expected.

4.3.3.54 S3024

Invalid operand type for this operation.

Invalid operand type for the operation on the specified position. An operand of an ANYBIT type has been expected.

4.3.3.55 S3025

Boolean result required.

Incompatible result type. Result should be of type BOOL.

4.3.3.56 S3026

Undeclared identifier.

This error occurs, if the identifier at the given position, has not been defined in the scope valid for the compiled program organization unit.

Example:

TYPE

Colors : (red, yellow, blue) := red;

END_TYPE

VAR

Color : Colors := green;

(* Error: green has not been declared as an enum constant *)

END_VAR

LD IntVar (* Error: IntVar has not been declared. *)

ADD 5

ST IntVar

4.3.3.57 S3028

Comparison not defined for the data type of the current result.

The comparison on the given position is not defined for the type of the current result. I. e. the type of the actual parameter is incompatible with the type of the first formal parameter. For more information see [Table 28 - Standard comparison functions](#) in the IEC 1131-3 Compliance Statement.

Example:

TYPE

Day_of_Week : STRUCT

Name : String;

DayNo : INT(1..7);

END_STRUCT;

END_TYPE

VAR

DayVar1 : Day_of_Week;

DayVar2 : Day_of_Week;

BoolVar : BOOL;

END_VAR

LD DayVar1

GT DayVar2 (* Error: comparisons on structured variables are not allowed *)

ST boolVar

4.3.3.58 S3030

Comparison not defined for this type.

The type of the operand at the given position is not allowed for comparisons. I. e. the type of the actual parameter is incompatible with the type of the formal parameter. For more information see [Table 28 - Standard comparison functions](#) in the 1131-3 Compliance Statement.

Example:

TYPE

Day_of_Week : STRUCT

 Name : String;

 DayNo : INT(1..7);

END_STRUCT;

END_TYPE

VAR

DayVar1 : Day_of_Week;

```
DayVar2 : Day_of_Week;
```

```
BoolVar : BOOL;
```

```
END_VAR
```

```
LD DayVar1
```

```
GT DayVar2 (* Error: comparisons on structured variables are not allowed *)
```

```
ST BoolVar
```

4.3.3.59 S3032

Self-referencing (i.e., recursive) declarations are not allowed.

Recursion detected. A function cannot invoke itself recursively, neither directly nor indirectly (i.e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i.e. by calling an instance of another function block that declares an instance of a function block type in the calling hierarchy).

4.3.3.60 S3033

Operand of type TIME expected.

A constant or a variable of type TIME was expected and the operand at the given position is of another type.

Example:

```
VAR
```

```
StartTime : TIME_OF_DAY;
```

```
StopTime : TIME_OF_DAY;
```

```
RunTime : TIME := T#10s;
```

```
END_VAR
```

```
LD StartTime
```

```
ADD 10000 (* Error: operand must be of type TIME *)
```

```
ST StopTime
```

LD StartTime

ADD RunTime (* Correct *)

ST Stop Time

4.3.3.61 S3034

String too long for variable.

A string literal has been assigned to a string variable but the string literal does not fit in the string variable. I. e. the length of the string literal is greater than the allocated length of the string variable.

4.3.3.62 S3035

Unallowed operand type for this function! Numeric operand or operand of date or time type expected.

The operation at the given position is not defined for the type of the current result (i.e. the first actual parameter).

Example:

VAR

BitMask: WORD;

END_VAR

LD BitMask (* Error: operand must be of type TIME, ANY_DATE or ANY_NUM *)

SUB 3

ST BitMask

4.3.3.63 S3036

Integer constant is out of range.

The integer constant at the given position is not in the range of the associated data type.

Example:

VAR

Range1 : UINT(-1..1000);

(* Error: Sign mismatch. Values for UINT must not be negative *)

Range2 : INT(-1..36000);

(* Error: Overflow: the upper range is greater as the
maximum valid INT value *)

END_VAR

4.3.3.64 S3037

The lower bound of the sub range must not be greater than the upper bound.

The value of the upper bound in the sub range declaration on the specified position is lower than the value of the lower bound. A sub range declaration restricts the range of an integer type to values between and including the specified upper and lower limits, where the upper limit has to be greater than the lower limit.

4.3.3.65 S3038

Initialization is out of bounds of sub range (Data type is a sub range type).

A variable of a sub range type has been initialized with a value that is out of the range of this sub range type. A sub range declaration specifies that the value of any data element of this type can only take on values between and including the specified upper and lower limits.

4.3.3.66 S3039

Index is out of bounds.

An access to a variable of an array type has been attempted with an index whose value is out of the range specified in the type or variable declaration.

4.3.3.67 S3040

Invalid data type. ANY_NUM required.

The operation at the given position is not defined for the type of the current result (i.e. the first actual parameter).

Example:

VAR

BitMake: WORD;

END_VAR

LD BitMask (* Error: operand must be of type TIME, ANY_DATE or ANY_NUM *)

NEG

ST BitMask

4.3.3.68 S3041

Unallowed EN/ENO type. Must be of type bool. Must not be RETAIN.

An input variable with the name EN or an output variable with name ENO has been declared with an illegal type or with the RETAIN qualifier.

The identifier "EN" is reserved for input variables of type BOOL

The identifier "ENO" is reserved for output variables of type BOOL This variable must not be declared with RETAIN qualifier.

4.3.3.69 S3042

Missing EN. Use of ENO allowed only in combination with EN.

An output variable with the name "ENO" has been defined but no input variable with name "EN" has been found. The output variable "ENO" can only be used in combination with "EN".

4.3.3.70 S3044

Data missing. You either need a load or an expression.

The current result is undefined. Either a LD instruction or an expression must precede the instruction on the current position. This error occurs as a consequence of error Syntax Error [S5010](#) . Please move the instruction out of the parenthesis.

4.3.3.71 S3046

Type names cannot be used as an instance names.

A type name or the name of a program organization unit has been used in a declaration as a variable name. Program organization units and types defined on project level are known in the whole project scope and their names cannot be used as identifiers for local variables.

Example:

```
FUNCTION Power
```

```
(* function block declarations *)
```

```
(* statements *)
```

```
END_FUNCTION
```

```
PROGRAM main
```

```
VAR
```

```
Power : REAL; (* Error: Power is not allowed as a variable name, because it already has been used as a function name *)
```

```
END_VAR
```

```
(* Code *)
```

```
END_PROGRAM
```

4.3.3.72 S3047

Function parameters must be specified in the order as defined in the Function prototype. Permuted parameter sequences will lead to incorrect code even if parameter names are specified.

If a function block is called in ST, the ST compiler translates the given calling parameter list directly to IL code since it has no knowledge of the function block's declaration. Because of this, the specified order must match the declaration order of the function blocks Input and Output variables.

Example:

FUNCTION_BLOCK Example

VAR_INPUT

In1 : int;

In2 : int;

END_VAR

FUNCTION_BLOCK_END

Program:

VAR

Instance : Example;

Local1 : int;

Local2 : int;

END_VAR

(* correct: parameter order matches declaration order *)

Example(In1 := Local1, In2 := Local2);

(* WRONG: does not match declaration order *)

Example(In2 := Local2, In1 := Local1);

4.3.3.73 S3048

Possible string truncation in assignment.

This warning is issued if the destination string in a string assignment has a shorter overall length than the source string. This check is done at compile time based on the *declared* lengths of both strings.

Example:

VAR

```
    strDestination  : string[10];
```

```
    strSource       : string[40];
```

END_VAR

```
strDestination := strSource;
```

4.3.3.74 S3049

Error in ST syntax (double click to get the ST error)

Compiling an ST POU containing a syntax error raises this error. A double click on the message jumps to source of error and shows the corresponding syntax-error message.

4.3.3.75 S3050

Array range mismatch.

Warning if assigning two arrays with different range but same size and type. The compiler allows array assignments, if both arrays are from the same type and size. E.g. an assignment of two int-arrays with range [0..9] and [1..10] is possible, but causes this warning.

4.3.3.76 S4000

"AT%": Simultaneous declaration of several direct variables is invalid.

A list of identifiers has been used in a located variable declaration. Direct representations can only be associated to a single identifier.

Example:

The following declaration is not allowed:

```
VAR
```

```
dirVar1, dirVar2, dirVar3 : at%I0.0;
```

```
END_VAR
```

4.3.3.77 S4001

Too many variables (identifiers). Maximum is 60 identifiers.

Too many identifiers in the identifier list of a variable declaration. Identifier lists with maximum 60 identifiers are supported.

4.3.3.78 S4003

Array too big.

The element count of a dimension in an array declaration exceeds the maximum number of elements supported by OpenPCS. The maximum element count is determined by the supported index range.

4.3.3.79 S4005

Upper bound must be greater or equal than lower bound.

The value of the upper bound index in the array declaration on the specified position is lower than the value of the lower bound index of the same dimension. The upper bound index of a dimension must be greater or equal than the associated lower bound index.

4.3.3.80 S4006

Syntax error. [Hint: In some cases, the actual error is located in a previous line (";" missing etc.)].

4.3.3.81 S4007

Self-referencing (i.e., recursive) declarations are invalid.

Recursion detected. A function cannot invoke itself recursively, neither directly nor indirectly (i.e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i.e. by calling an instance of another function block that declares an instance of a function block type in the calling hierarchy).

4.3.3.82 S4008

Too many attributes "RETAIN" or "CONSTANT". You may use only one.

Too many qualifiers used in a variable declaration part.

4.3.3.83 S4009

A STRUCTure must contain at least one structure element (variable declaration).

An empty structure has been declared. This is not allowed. A structure must contain at least one member variable.

Example:

Not allowed:

```
TYPE
```

```
Mystruct : struct end_struct;
```

```
END_TYPE
```

Allowed:

```
TYPE
```

```
Mystruct : STRUCT
```

```
  M1 : int;
```

```
END_STRUCT
```

```
END_TYPE
```

4.3.3.84 S4010

Simultaneous type declarations are not allowed.

The type declaration on the specified position contains a list of identifiers. This is not allowed. Please write a declaration for any new type.

Example:

Not allowed:

TYPE

MyInt1, MyInt2, MyInt3 : int;

END_TYPE

Allowed:

TYPE

MyInt1 : int;

MyInt2 : int;

MyInt3 : int;

END_TYPE

4.3.3.85 S4011

Valid only in PROGRAMs and there within VAR- und VAR_GLOBAL-Sections.

A directly represented variable has been declared in a program organization unit or a variable declaration part in which it is not supported. Located variable declarations are supported only in VAR- or VAR_GLOBAL-declaration-parts of PROGRAMs.

4.3.3.86 S4012

Valid only in PROGRAMs, FUNCTION_BLOCKS, and in FUNCTIONs.

A variable declaration part (VAR <declarations> END_VAR) was found in a unit where it is not supported. Variable declaration parts are allowed in programs, functions and function blocks.

4.3.3.87 S4013

Valid only in PROGRAMs, FUNCTION_BLOCKS, and in FUNCTIONs.

An input variable declaration (VAR_INPUT <declarations> END_VAR) part was found in a program organization unit where it is not supported.

4.3.3.88 S4014

Valid only in PROGRAMs and in FUNCTION_BLOCKS.

An in/out variable declaration part (VAR_IN_OUT <declarations> END_VAR) was found in a program organization unit where it is not supported.

4.3.3.89 S4015

Valid only in PROGRAMs and in FUNCTION_BLOCKS.

An output variable declaration part (VAR_OUTPUT <declarations> END_VAR) was found in a program organization unit where it is not supported.

4.3.3.90 S4016

Valid only in PROGRAMs and in FUNCTION_BLOCKS.

An external variable declaration part (VAR_EXTERNAL <declarations> END_VAR) was found in a program organization unit where it is not supported. External variable declarations are supported in PROGRAMs and FUNCTION_BLOCKS.

4.3.3.91 S4017

Valid only in PROGRAMs.

A global variable declaration part (VAR_GLOBAL <declarations> END_VAR) was found in a program organization unit where it is not supported. Global variable declarations are allowed in PROGRAMs only.

4.3.3.92 S4018

Valid only in VAR- and in VAR_GLOBAL-Sections.

The qualifier "CONSTANT" has been used in a variable declaration part in which it is not supported.

4.3.3.93 S4019

Valid only in PROGRAMs or in FUNCTION_BLOCKS and there within VAR-, VAR_OUTPUT-, or VAR_GLOBAL-Sections).

The qualifier "RETAIN" has been used in a variable declaration part in which it is not supported.

4.3.3.94 S4020

Valid only in PROGRAMs or in FUNCTION_BLOCKS and there within VAR_INPUT-Sections with Type "BOOL" without Initialization.

A variable has been declared with an edge qualifier in a program organization unit or variable declaration part where this is not supported.

4.3.3.95 S4021

Valid only within VAR_INPUT, VAR_OUTPUT, and VAR_IN_OUT-Sections.

A variable has been declared with the ADDRESS qualifier in a program organization unit or variable declaration part where this is not supported.

4.3.3.96 S4022

Valid only in FUNCTION_BLOCKS or FUNCTIONS and there within VAR..END_VAR-Sections without CONSTANT/RETAIN-Modifiers.

A variable has been declared with the ATTRIBUTES qualifier in a program organization unit or variable declaration part where this is not supported. This attribute is supported only in VAR-Sections without CONSTANT or RETAIN qualifiers of FUNCTIONS and FUNCTION_BLOCKS.

Note: Keyword ATTRIBUTES is supported by OpenPCS only in custom versions to define additional attributes for variables in extension to IEC61131-3. You should not see this message in standard OpenPCS.

4.3.3.97 S4023

Valid only in TYPE..END_TYPE-Sections.

A struct declaration was found in a declaration part where this is not supported. Struct declarations are supported only in TYPE declaration parts.

4.3.3.98 S4024

Valid not within VAR_EXTERNAL-Sections.

A variable has been declared in an EXTERNAL declaration section with an initial value. This is not allowed. Please assign the initial value in the respective GLOBAL variable declaration.

Example:

```
VAR_EXTERNAL
```

```
A : INT := 5;
```

```
END_VAR
```

```
VAR_EXTERNAL
```

```
A : INT;
```

```
END_VAR
```

```
VAR_GLOBAL
```

```
A : INT := 5
```

```
END_VAR
```

4.3.3.99 S4033

Multiple initialization.

A member of a struct variable has been initialized more than once. This error occurs when both an explicit struct initialization and a per element initialization are made.

Example:

The following initialization is not allowed:

TYPE

StructType : Struct

Member1 : int := 5;

Member2 : bool;

END_STRUCT := (Member1 := 4, Member2 := true);

END_TYPE

Use one of the following initializations instead:

TYPE

StructType : Struct

Member1 : int ;

Member2 : bool;

END_STRUCT := (Member1 := 4, Member2 := true);

END_TYPE

or

TYPE

StructType : Struct

Member1 : int := 5;

```
Member2 : bool := true;
```

```
END_STRUCT;
```

```
END_TYPE
```

4.3.3.100 S4034

Invalid POU name.

This error occurs when a keyword has been used as a POU name or if no name has been defined.

4.3.3.101 S4035

Invalid type for function.

The function type must be a predefined type or an identifier. This error occurs most commonly, when a reserved keyword, a IEC61131-3 character string or a number is used as a function type or if no function type has been defined.

4.3.3.102 S4036

FUNCTIONs need at least one input parameter VAR_INPUT.

A function has been defined without an input parameter. In IEC61131-3 a function needs at least one input-parameter.

4.3.3.103 S5000

Wrong parameter type.

The type of an actual parameter of a function or a function block instance is incompatible with the type of the formal parameter it has been assigned to.

4.3.3.104 S5001

Array expected. This is not an array.

An indexed access has been attempted to a variable which is not an array.

Example:

PROGRAM

VAR

x : INT;

y : INT;

END_VAR

LD x[3] (* not allowed if the variable is not an array *)

ST y

END_VAR

4.3.3.105 S5002

This FUNCTION_BLOCK is called by CAL if EN=TRUE. CALC/CALCN are both invalid.

An instance of a function block with an "EN" input parameter has been called via CALC/CALCN. This is not allowed. Use the CAL-statement instead. The code of a function block with an "EN" parameter is invoked if the value of this parameter is TRUE.

4.3.3.106 S5003

Function block instances may not be "CONSTANT".

An instance of a function block has been defined in a variable section with CONSTANT attribute. This is not allowed. Please remove the attribute or move the instance declaration in another variable section, which has no CONSTANT attribute.

4.3.3.107 S5004

Function blocks instances are invalid in "FUNCTION"-POUs, STRUCTs, and in ARRAYs.

An instance of a function block has been defined in a variable section of a function or as a member of a STRUCT or an ARRAY type. IEC61131-3 doesn't allow declarations of function block instances in functions. Function block instances as members of STRUCT and ARRAY types are not supported by OpenPCS.

4.3.3.108 S5005

Function block instances as function results are not supported.

Function block instances as result type of a function are not supported in OpenPCS.

4.3.3.109 S5006

Function block instances as parameters are not supported.

Parameters of a function block type are not supported in OpenPCS.

4.3.3.110 S5008

Expected an integer or an enum. Invalid array index.

The type variable or constant used as an index in an indexed variable access is invalid. An index must be of type INT or of an enumeration type.

4.3.3.111 S5009

Invalid sequence beginning. Current result is empty. Use "LD" to initialize current result.

This error occurs when a sequence of statements starts with an instruction that uses the current result. The first instruction usually is a load statement. This error can also occur, if the current result is used in the first instruction after a CAL, a JMP or a label.

Example:

```
PROGRAM main
```

```
VAR
```

```
Switch : BOOL;
```

```
END_VAR
```

```
ST Switch (* Error: Current result is undefined. *)
```

LD Switch

EQ TRUE

JMPC NextStep

LD TRUE

JMP End (* The value loaded in the previous statement will be lost after the JMP-statement *)

NextStep:

LD FALSE

END:

ST Switch (* Error: Current result is undefined after a label *)

(* Code *)

END_PROGRAM

4.3.3.112 S5010

Invalid instruction within a parentheses computation.

The instruction at the given position is not allowed between parentheses. Please replace the instruction or move it out of the parentheses.

Example:

FUNCTION_BLOCK Count

VAR_INPUT

StartValue : DINT;

```
FReset : BOOL;

END_VAR

VAR_OUTPUT

CurrentCountValue : DINT;

END_VAR

VAR

CountValue : DINT;

END_VAR

LD fReset

EQ TRUE

JMPCN Continue

LD StarValue

ST CountValue

Continue:

LD CountValue

ADD 1

ST CountValue

ST CurrentCountValue

END_FUNCTION_BLOCK

PROGRAM main

VAR

Counter : Count;

StartValue : DINT;
```


Result : DINT;

END_VAR

LD 5

ADD (StartValue

ST Counter.StartValue

EQ 1000

ST Counter.fReset

CAL Counter (* Error: CAL is not allowed between parentheses *)

LD Counter.CurrentCounter (* Error: Load is not allowed between parentheses *)

)

ST Result

END_PROGRAM

4.3.3.113 S5011

ARRAYs of function block instances are invalid.

Arrays of function blocks are not supported.

4.3.3.114 S5012

Result type and operand type are incompatible.

The result type of the preceding operation and the type of the variable in which this result is stored are incompatible.

Example:

VAR

X : INT;

END_VAR

LD 65000

ST x (* 65000 is not of type INT *)

4.3.3.115 S5013

Result type and type of the first formal input parameter are incompatible.

The result type of the preceding operation and the type of the first input parameter in a function or function block call are incompatible.

Example:

FUNCTION Fun1

VAR

InVar : INT;

END_VAR

(* Code *)

END_FUNCTION

PROGRAM main

VAR

X : DINT;

END_VAR

LD x

ADD 1000

Fun1 (* Error: result type of the preceding operation is DINT, the type of the first input parameter of Fun1 is INT *)

ST x

END_PROGRAM

4.3.3.116 S5014

Wrong number of parameters.

Too many parameters found in a call of a function or a function block.

4.3.3.117 S5015

Invalid type for direct address.

A located variable has been declared with an unsupported type. Only located variables of type ANY_NUM or ANY_BIT are supported.

4.3.3.118 S5016

Variable is read-only. Write-access invalid.

A write access has been attempted to a variable, that has only read access.

4.3.3.119 S5017

Variable is not a STRUCTure.

A initialization value for a structure has been assigned to a variable which is not of a structured type.

Example.

VAR

A : INT := (m1 := 5, m2 := TRUE); (* not allowed *)

END_VAR

4.3.3.120 S5018

Variable is no array.

An array initialization has been assigned to a variable which is not of an array type.

Example.

VAR

A : INT := [4]; (* not allowed *)

END_VAR

4.3.3.121 S5019

Initialization value and variable type incompatible.

The type of the initialization value and the type of the variable are incompatible.

Example:

VAR

X : INT := 65000;

END_VAR

4.3.3.122 S5020

Too many initialization values.

The initialization value for an array type or variable has more elements as provided by the array declaration.

Example:

VAR

A : ARRAY [1..5] OF INT := [1, 2, 3, 4, 5, 6];

(* too much initialization values, array has only 5 elements *)

END_VAR

4.3.3.123 S5021

Formal parameter incorrectly declared.

The name of an output parameter has been expected. The identifier has been found in the current scope but is not the name of an output parameter.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not an input or in/out parameter.

4.3.3.124 S5022

Multiple assignments to a parameter in a call of a function block instance.

This error occurs, when in a call of a function block instance a parameter is initialized twice.

Example:

```
FUNCTION_BLOCK Fb1
```

```
VAR_INPUT
```

```
InParam1 : int;
```

```
InParam2 : int;
```

```
InParam3 : bool;
```

```
END_VAR
```

```
(* Code *)
```

```
END_FUNCTION_BLOCK
```

```
PROGRAM main
```

```
VAR
```

```
fbInst : fb1;

END_VAR

(* Code *)

cal fbInst( InParam1 := 1,

InParam1 := 2,

InParam3 := true

)

(* Code *)

END_PROGRAM
```

4.3.3.125 S5023

Too much initialization data.

This error occurs, when a member of a struct type or instance is initialized twice in an explicit structure initialization.

Example:

```
TYPE

StructType : STRUCT

Member1 : int;

Member2 : int;

Member3 : bool;

END_STRUCT;

END_TYPE

VAR

StructVar : StructType := (Member1 := 1, Member1 := 2, Member3 := FALSE);

END_VAR
```

4.3.3.126 S5024

Unallowed type for this operation.

The operation on the given position is not defined for the type of the current result. I. e. the type of the actual parameter is incompatible with the type of the first formal parameter.

Example:

VAR

X : REAL;

END_VAR

LD 1 (* The constant 1 can be converted implicitly to any integer or any bit type *)

LN (* Error: LN is only defined for ANY_REAL types *)

ST X

4.3.3.127 S5025

Unallowed parameter type for this function.

The type of the actual parameter is incompatible with any type allowed for the parameter at the given position.

Example:

VAR

X : STRING;

END_VAR

LD "EXAMPLE"

LEFT 3.0 (* Error: the second parameter of LEFT has type UINT *)

ST X

4.3.3.128 S5026

Invalid formal parameter type.

The name of an input or an in/out parameter has been expected. The identifier has been found in the current scope but is neither the name of an input nor of an output parameter.

Tips:

Check if the name is spelled correctly.

Make sure that the identifier is not an output parameter.

4.3.3.129 S5027

Incompatible operand types.

The operands for the operation at the given position must be compatible. I. e. they must have the same type or, if at least one of the parameter is a constant an implicit cast to the type of the other operand has be possible.

Example:

VAR

X : REAL;

END_VAR

LD 1

(* The constant 1 can be converted implicitly to any integer or any bit type *)

MAX X (* Error: X is of type REAL *)

ST X

4.3.3.130 S5028

Data type not allowed for this operation.

This error occurs, if the type of an actual parameter is not allowed for the operation at the given position.

Example:

```
VAR
```

```
StringVar : STRING;
```

```
END_VAR
```

```
LD 1
```

```
CONCAT "EXAMPLE"
```

(* Error: CONCAT expects a STRING operand as first input parameter *)

```
ST StringVar
```

4.3.3.131 S5029

Invalid function block call.

This error occurs, if a call to a function block instance is attempted and this instance is an input parameter of the calling function block or program.

Example:

```
FUNCTION_BLOCK Fb1
```

```
VAR_INPUT
```

```
InParam1 : int;
```

```
InParam2 : int;
```

```
InParam3 : bool;
```

```
END_VAR
```

(* Code *)

```
END_FUNCTION_BLOCK
```

```
FUNCTION_BLOCK Fb2
```

```
VAR_INPUT  
  
fbInstInput : Fb1;  
  
(* other input declarations *)  
  
END_VAR  
  
VAR  
  
(* local variable declarations *)  
  
END_VAR  
  
(* Code *)  
  
cal fbInstInput( InParam1 := 1,  
InParam2 := 2,  
InParam3 := true  
)  
  
(* Code *)  
  
END_PROGRAM
```

4.3.3.132 S5030

Variable is write-only. Read-access invalid.

A read access has been attempted to a variable, that has only write access.

4.3.3.133 S5031

Bit access allowed only on bit data types.

This error occurs if a bit selection is attempted on a variable that is not of a bit data type or of type BOOL.

Example:

```
VAR  
  
DintVar : DINT;
```

```
BoolVar : BOOL;
```

```
END_VAR
```

```
LD DintVar.4
```

(* Error: bit selection allowed only on variables of type ANY_BIT except BOOL *)

```
ST BoolVar
```

4.3.3.134 S5032

Bit position is greater than the number of bits in the selected variable.

This error occurs, when the bit position given in a bit selection is greater than the number of the most significant bit of the selected variable. The number of bits accessible in a bit selection depends on the variables data type. The bit positions are counted from the least significant bit at position 0 to the most significant bit at position $n - 1$, where n is the number of bits in the data type.

Example:

```
VAR
```

```
wVar : WORD := 5;
```

```
fVar : BOOL := FALSE;
```

```
END_VAR
```

```
(* Code *)
```

```
LD wVar.16 (* The selected variable is of type WORD. I. e. it has 16 bits with bit positions from 0 to 15. *)
```

```
ST fVar
```

```
(* Code *)
```

4.3.3.135 S5033

IN_OUT parameter missing. Please supply every formal IN_OUT parameter with a an actual parameter.

This error occurs, if at least one of the IN_OUT parameters of a function block is not supplied with an actual parameter, when calling an instance of the respective function block. IN_OUT parameters are references and have to be supplied with an actual parameter in every call of a function block instance.

Example:

```
FUNCTION_BLOCK Fb1
```

```
VAR_IN_OUT
```

```
InOutParam1 : INT;
```

```
InOutParam2 : BOOL;
```

```
END_VAR
```

```
(* Code *)
```

```
END_FUNCTION_BLOCK
```

```
PROGRAM main
```

```
VAR
```

```
fbInst : fb1;
```

```
IntVar1 : INT;
```

```
IntVar2 : INT;
```

```
END_VAR
```

```
(* Code *)
```

```
cal fbInst() (* Error: none of the IN_OUT variables of FB1 is supplied with an actual parameter *)
```

```
cal fbInst( InOutParam1 := IntVar1
```

```
) (* Error: the actual parameter for the second IN_OUT parameter is missing *)
```

```
cal fbInst ( InOutParam1 := IntVar1,
```

```
InOutParam2 := IntVar2
```

```
) (* Correct: every formal IN_OUT parameter of FB1 is supplied with an actual parameter *)
```

```
(* Code *)
```

END_PROGRAM

4.3.3.136 S5034

Invalid IN_OUT parameter. IN_OUT parameters must not be expressions or constants.

This error occurs, if an IN_OUT parameter is supplied with an expression or a constant value. This is not allowed because IN_OUT parameters are references.

Example:

```
FUNCTION_BLOCK Fb1
```

```
VAR_IN_OUT
```

```
InOutParam1 : INT;
```

```
InOutParam2 : BOOL;
```

```
END_VAR
```

```
(* Code *)
```

```
END_FUNCTION_BLOCK
```

```
PROGRAM main
```

```
VAR
```

```
fbInst : fb1;
```

```
IntVar1 : INT;
```

```
IntVar2 : INT;
```

```
END_VAR
```

```
(* Code *)
```

```
cal fbInst( InOutParam1 := IntVar1,
```

```
InOutParam2 := 5
```

```
)( * Error: the actual parameter for the second IN_OUT parameter is a constant. *)
```

```
cal fbInst( InOutParam1 := IntVar1,  
InOutParam2 := (IntVar1  
  ADD IntVar2)  
) (* Error: the actual parameter for the second IN_OUT parameter is an expression. *)
```

```
cal fbInst ( InOutParam1 := IntVar1,  
InOutParam2 := IntVar2  
  ) (* Correct: Both IN_OUT parameters of FB1 are supplied with variables. *)  
(* Code *)  
END_PROGRAM
```

4.3.3.137 S5035

Generic data types are not allowed.

This error occurs, if an ANY data type is used in a variable or parameter declaration. The use of generic data types is allowed only for function overloading and type conversion in standard function or functions provided by the manufacturer.

Example:

```
FUNCTION IntegerToString : STRING  
VAR_INPUT  
InVar : ANY_INT; (* Error: User-defined functions cannot be overloaded *)  
END_VAR  
(* Code *)  
END_FUNCTION
```

4.3.3.138 S5036

Local types are not allowed in this variable section.

This error occurs, if a local user defined type is used in the declaration of a global or external variable or in the declaration of a parameter. Global and external variables as well as parameters have to be of a predefined type or of a global type. Global types are either hardware dependent types, provided by the firmware or project global user defined types.

Example:

```
PROGRAM main
```

```
TYPE
```

```
StructType : STRUCT
```

```
Member1 : BOOL;
```

```
Member2 : STRING;
```

```
END_STRUCT;
```

```
(* Other type definitions *)
```

```
END_TYPE
```

```
VAR_GLOBAL
```

```
GlobVar : StructType; (* Not allowed because StructType is not known in other POU's *)
```

```
(* Other global variable definitions *)
```

```
END_VAR
```

```
VAR
```

```
(* Local variable definitions *)
```

```
END_VAR
```

```
(* Code *)
```

```
END_PROGRAM
```

```
FUNCTION_BLOCK Fb1
```

TYPE

StructType : STRUCT

Member1 : BOOL;

Member2 : STRING;

END_STRUCT;

END_TYPE

VAR_EXTERNAL

GlobVar : StructType; (* Not allowed because StructType is not known in other POU's *)

(* Other external declarations *)

END_VAR

VAR_INPUT

InVar : StructType; (* Not allowed because StructType is not known in other POU's *)

(* Other input declarations *)

END_VAR

(* Code *)

END_FUNCTION_BLOCK

4.3.3.139 S5037

Too many indices within the braces [...] of an array-access.

This error occurs, if an access to an array element is attempted with more indices as dimensions provided in the type definition of the elements data type.

Example:

PROGRAM main

TYPE

ArrayType : Array[1..5, 1..20] of INT;

(* Other type definitions *)

END_TYPE

VAR

ArrayVar : ArrayType;

IntVar : INT;

(* Other variable definitions *)

END_VAR

LD ArrayVar[1, 2, 3] (* Error: Variables of type ArrayType have only 2 dimensions *)

ST IntVar

(* Code *)

END_PROGRAM

4.3.3.140 S5038

Directly represented variables are only allowed as parameters in prototypes.

A directly represented variable has been declared in the VAR_INPUT, VAR_OUTPUT or VAR_IN_OUT section of a program organization unit. This is not allowed. Directly represented variables are not allowed in functions and function blocks. VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT variables are not supported in programs.

If you want to access a directly represented variable from a function block, declare the variable with a symbolic name in the VAR_GLOBAL section of a program and use this symbolic name in a declaration in the VAR_EXTERNAL section of the function block.

Functions cannot access directly represented variables.

Example:

FUNCTION_BLOCK SetOutput

VAR_EXTERNAL

```
OutputLocation : BOOL;
```

```
END_VAR
```

```
VAR_INPUT
```

```
Value : BOOL;
```

```
END_VAR
```

```
LD Value
```

```
ST OutputLocation
```

```
END_FUNCTION_BLOCK
```

```
PROGRAM main
```

```
VAR_GLOBAL
```

```
OutputLocation AT%Q0.0 : BOOL;
```

```
END_VAR
```

```
VAR
```

```
Switch : SetOutput;
```

```
CurrentValue : BOOL;
```

```
END_VAR
```

```
LD CurrentValue
```

```
NOT
```

```
CAL Switch(Value := CurrentValue)
```

```
END_PROGRAM
```

4.3.3.141 S5039

"&x" is only allowed if x is a direct variable.

The identifier preceded by the &-operator is not the name of a directly represented variable.

Tips:

Make sure that the name is spelled correctly.

Make sure that the variable is a directly represented variable.

4.3.3.142 S5040

Too few indices within the braces [...] of an array access.

This error occurs, if an access to an array element is attempted with less indices as dimensions provided in the type definition of the elements data type.

Example:

```
PROGRAM main
```

```
TYPE
```

```
ArrayType : Array[1..5, 1..10, 1..20] of INT;
```

```
(* Other type definitions *)
```

```
END_TYPE
```

```
VAR
```

```
  ArrayVar : ArrayType;
```

```
  IntVar : INT;
```

```
(* Other variable definitions *)
```

```
END_VAR
```

```
LD ArrayVar[1, 2] (* Error: Variables of type ArrayType have 3 dimensions *)
```

```
ST IntVar
```

```
(* Code *)
```

```
END_PROGRAM
```

4.3.3.143 S5041

Values of type INT24 or REAL48 are invalid in this context.

Operation not supported for this type.

4.3.3.144 S5042

Function block instances may not be "RETAIN".

An instance of a function block has been defined in a variable section with RETAIN attribute. This is not supported. Please remove the attribute or move the instance declaration in another variable section, which has no RETAIN attribute.

4.3.3.145 S5043

Variables, constants and parameters are not allowed as initialization values in declarations. Please use a literal or enumeration value.

In declarations variables, constants or parameters cannot be used to initialize values.

4.3.3.146 S6002

No prototype.

An unknown type name has been used in a variable declaration or a function call.

Tips

Make sure that a type a function or function block with this name is declared in the context of the active project.

Make sure the name of the type, function or function block is spelled correctly.

Recompile the whole project.

Please consult your hardware documentation if none of the above actions eliminates the problem.

4.3.3.147 S6004

Recursion (i.e., direct or indirect self-reference) detected.

Recursion detected. A function cannot invoke itself recursively, neither directly nor indirectly (i.e. by invoking another function, that invokes one of the functions in the calling hierarchy). Function blocks and programs can not declare instances of themselves, neither directly nor indirectly (i.e. by calling an instance of another function block that declares an instance of a function block type already used in the calling hierarchy).

4.3.3.148 S6005

Too many types and function blocks. For the maximum number of type definitions please consult your hardware documentation.

This error occurs, if too many types functions or function blocks have been used in the calling hierarchy of a program organization unit. For the maximum number of types, functions and function blocks supported see the Table D.1: Implementation-dependent parameters

4.3.4 Linker Messages

4.3.4.1 L10001

Variable declared twice: <Variable name>.

The variable with the specified name has been declared twice.

Tips:

If the variable is declared in a PROGRAM POU, check if a resource global variable with the same name has been declared.

If the variable is a resource global variable check if a global variable with the same name has been declared in a PROGRAM POU of the resource.

If one of the above cases is true, change the name of one of the variables or move the variable declaration in the PROGRAM POU in a VAR_EXTERNAL section. Attention: if you move the variable into the external section, every access to the external variable accesses the resource-global variable with the same name.

4.3.4.2 L10004

Unresolved external: <Variable name>.

Either a global variable with the specified name has not been found, or a function block type with the specified name has not been found.

Tips:

Make sure that the variable name is spelled correctly.

If the variable is not a function block instance, make sure that a variable with this name is declared in the VAR_GLOBAL section of the calling program or in a file with resource-global variable declarations.

If the variable is a function block instance, make sure that the function block has been compiled successfully, i.e. an object file for this function block exists.

4.3.4.3 L10026

Unsupported address: <AddressDescription>.

The address <AddressDescription> is not supported by this hardware.

Tips:

Check if the address is spelled correctly.

Check if the syntax of the address description is correct. The syntax of the address description is hardware dependent, but must be a string formed of the percent sign "%" followed by a location prefix, a size prefix and one or more unsigned integers, separated by periods (.). The size prefix may be empty. For valid location and size prefixes consult your hardware documentation.

4.3.4.4 L10027

Invalid hardware description: %1..

The hardware description file for the hardware with name <hardware name> has not been found.

Tips:

Check if the resource specification contains a valid hardware module name.

Reinstall OpenPCS. If this doesn't remove your error, consult your hardware documentation or refer to your hardware manufacturer.

4.3.4.5 L10029

Hardware configuration error.

An error occurred while getting firmware information. Please check if the hardware configuration file is correct or if the DLL for the specified firmware is installed in your OpenPCS directory.

ATTENTION: This file should be altered only by the manufacturer.

4.3.4.6 L10030

Invalid type for variable: %1.

A directly represented variable of a complex type (array, struct, string) has been found. This is not supported by the hardware.

4.3.4.7 L10031

Initializations of directly represented variables are not allowed.

An initialization of a directly represented variable has been found. This is not supported by the hardware. Please remove the initialization.

4.3.4.8 L10032

Address <AddressDescription> invalid in this context.

The address with the specified description is a valid address but not allowed in this context (Task, POU, Resource, Configuration).

4.3.4.9 L10033

Attribute RETAIN not supported for directly represented variables.

A directly represented variable with RETAIN attribute has been found. This is not supported by the hardware. Please move the variable declaration in another section or remove the attribute from the section.

4.3.4.10 L10034

Attribute CONST not supported for directly represented variables.

A directly represented variable with CONST attribute has been found. This is not supported by the hardware. Please move the variable declaration in another section or remove the attribute from the section.

4.3.4.11 L10035

Instance limit for function block <FunctionBlockName> reached.

The maximum number of instances of the specified function block has already been exceeded. The maximum number of instances of a firmware function block is hardware dependent and can be changed by the hardware manufacturer by setting or changing the "MaxInstances" entry in the specification section of the function block in the hardware description file. Please consult your hardware documentation, for the maximum number of instances of a firmware function block.

4.3.4.12 L10036

Invalid process image description. Please contact your manufacturer.

The description of the process image in the hardware configuration file is invalid. Please check if the sizes for the input, output and marker sections are correct and if all size entries are of the same unit. They should be specified either in bits or bytes.

ATTENTION: This file should be altered only by the manufacturer.

4.3.4.13 L10063

An error occurred while opening a file: %1.

4.3.4.14 L10105

Internal error while loading function or DLL: <DLL/Function-Name>.

The specified DLL or function could not be loaded. Either your OpenPCS directory does not contain a DLL with the specified name, or your DLL has an invalid version. Please reinstall your system or consult your hardware description.

4.3.4.15 L10106

Native code compiler needed for selected optimization. Please choose another optimization or install a native code compiler.

"Speed only" optimization is activated but no native code compiler is defined for this hardware. "Speed only" optimization is only valid, if a native code compiler is installed. If you do not have a native code compiler please select another optimization in the "Edit Resource Specifications" dialog. For a native code compiler for your hardware please refer to your manufacturer.

4.3.4.16 L12001

Type conflict. Type of external the variable doesn't match with type of the global variable with the same name.

A global variable with the same name as the external variable has been found, but the types of the global and the external variable are different.

Tips:

Make sure that the external variable name is spelled correctly.

Make sure that the type of the external variable is spelled correctly.

Make sure that the global variable is the requested variable.

Change the type of the external or the global variable.

4.3.4.17 L12002

Readable access to this variable is not allowed: <Variable name>.

A read access to a variable that has only write access has been attempted.

Tips:

Make sure that the specified variable name is spelled correctly.

The specified variable is an output location. A read access to output locations is not allowed.

4.3.4.18 L12003

Writable access to this variable is not allowed: <Variable name>.

A write access to a variable that has only read access has been attempted.

Tips:

Make sure that the specified variable name is spelled correctly

The specified variable is a constant. Write access to a constant variable is not allowed. Check if the CONSTANT attribute can be removed from the variable.

The specified variable is an input location. A write access to input locations is not allowed.

4.3.4.19 L12005

Internal linker error no.: <errorno>. Please contact your manufacturer.

4.3.4.20 L12006

Memory allocation failure. Not enough memory to perform operation.

4.3.4.21 L12007

No object information found for task <TaskName>. Please rebuild all.

The object file (<TaskName>.crd) for the specified task has not been found. Please rebuild the whole resource.

4.3.4.22 L12008

Interpreter stack overflow in task <TaskName>.

Interpreter call-stack-overflow. Please reduce the depth of the calling hierarchy of <TaskName>.

4.3.4.23 L12064

Error exporting OPC variables to OPC server configuration. Error code: %1.

An OPC variable is erroneous. Please use a proper one.

4.3.4.24 L12065

Error initializing ConfOPC.DLL. Please contact your manufacturer.

The DLL could not be initialized. Please ask the hardware manufacturer.

4.3.4.25 L12066

Incorrect alignment for address <address>: variable must be placed at an alignment border."

The direct variable should be moved to a properly aligned address, in order to avoid potential erroneous behavior on some controllers that have an alignment of 2 or 4. With alignment 2, all variables having the size of a WORD (W) or a DWORD (D) should be moved to even addresses. With alignment 4, all variables having the size of a WORD (W) should be moved to even addresses and all variables having the size of a DWORD (D) should be moved to addresses divisible by 4.

4.3.4.26 L12996

Unknown command: <Command>.

An unknown command line argument has been used with [ITLINK](#).

4.3.4.27 L12997

Unknown object kind: <ObjectKindSpecification>.

An invalid object file has been found. Please rebuild the whole resource.

4.3.4.28 L12998

Invalid object kind. Kind found/requested: <ObjectKind>.

An invalid object file has been found. Please rebuild the whole resource.

4.3.4.29 L12999

Invalid object version found. Object version found/expected: <ObjectVersion>.

The object file version and the compiler object version are different. The object file has been created with a different compiler version. Please recompile the whole resource.

4.3.4.30 L13000

Load of resource global variable information failed.

The object file with the resource global information has not been found. Please rebuild the whole resource.

4.3.4.31 L13001

No object information found for pou <pouname>

The object file (<pouname>.obj) for the specified POU has not been found. Please rebuild the whole resource.

4.3.4.32 L14009

Resource size exceeds size of PLC memory.

The size of the resource exceeds the PLC memory limit. Calculation can differ from real size.

4.3.4.33 L14010

Resource size warning limit reached. Used X of Y bytes.

The size of the resource has reached the configured warning limit. Set the size within the [browser options](#) dialog.

4.3.4.34 L15001

An undefined task type has been used or no task type has been defined for task %1.

Check the configuration parameters of the properties of the task type. You may also ask your hardware manufacturer.

4.3.4.35 L20012

Persistency file creation disabled due to online linking

If your hardware uses online linking the creation of the persistency file is disabled.

4.3.5 Compiler Messages

4.3.5.1 C10006

Data type "REAL" is not supported.

Data type "REAL" is not supported by the active hardware. For a list of data types supported by OpenPCS see the IEC 1131-3 Compliance statement Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.2 C10007

Data type "DATE" is not supported.

Data type "DATE" is not supported. For a list of data types supported by OpenPCS see IEC 1131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.3 C10008

Data type "TIME_OF_DAY" is not supported.

Data type "TIME_OF_DAY" is not supported. For a list of data types supported by OpenPCS see IEC 1131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.4 C10009

Data type "STRING" is not supported.

Data type "STRING" is not supported by the active hardware. For a list of data types supported by OpenPCS see the IEC 1131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.5 C10010

Data type "DATE_AND_TIME" is not supported.

Data type "DATE_AND_TIME" is not supported. For a list of data types supported by OpenPCS see the IEC 1131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.6 C10012

Data type "TIME" is not supported.

Data type "TIME" is not supported by the active hardware. For a list of data types supported by OpenPCS see the IEC 1131-3 Compliance statement. Please consult your hardware documentation for a list of data types supported by your hardware.

4.3.5.7 C10017

The sections "VAR_INPUT", "VAR_OUTPUT" and "VAR_IN_OUT" are not supported in programs.

VAR_INPUT, VAR_OUTPUT and VAR_IN_OUT sections in programs are not supported. For more information about supported variable types see the IEC 1131-3 Compliance statement.

4.3.5.8 C10019

Directly represented variables are not allowed in this POU.

Either the program organization unit is a function or a function block or a file with global symbolic variable definitions. Directly represented variables are not allowed in functions or function blocks. If you want to access a directly represented variable from a function block, declare the variable with a symbolic name in the VAR_GLOBAL section of a program and use this symbolic name in a declaration in the VAR_EXTERNAL section of the function block. Functions cannot access directly represented variables.

Directly represented resource global variables have to be declared in a specific file.

4.3.5.9 C10020

Bit access not allowed for this variable/parameter.

Variable or parameter has to be of the ANY BIT type.

4.3.5.10 C10021

Constant must not be negative.

A negative constant has been found where an unsigned operand has been expected. Please change the constant value or the variable type (if possible).

4.3.5.11 C10024

Constant is out of range.

The constant at the given position is not in the range of the associated data type.

4.3.5.12 C10025

IN/OUT parameters must always be supplied with actual parameters.

A formal in/out parameter has been declared in a function block, but not supplied with an actual parameter in the CAL statement of an instance. In/out parameters are references and must be supplied with an actual parameter.

4.3.5.13 C10026

Unsupported address.

The address at the given position is not supported by the active hardware. Please consult your hardware documentation for a list of addresses supported by the hardware.

4.3.5.14 C10028

Inout-parameters of type struct are not supported.

Structured in/out-parameters are not supported. Please define an input parameter and an output parameter of this kind.

4.3.5.15 C10030

Value of hardware configuration file entry "Alignment" must be greater than 0.

The value of the entry **Alignment** in your hardware configuration file is set to 0. Please set it at least to 1.

4.3.5.16 C10031

RETAIN-variables are not supported by this hardware.

Your hardware doesn't support RETAIN variables. Please remove the attribute. For a list of supported variable types consult your hardware documentation.

4.3.5.17 C10034

Invalid command for this hardware.

The command at the given position is not supported by this hardware. For a list of unsupported commands p consult your hardware documentation. For a list of commands not supported by OpenPCS see the IEC 1131-3 Compliance statement.

4.3.5.18 C10035

The operand/parameter must be of type "UINT".

An actual parameter of type UINT has been expected in a function call (operation), but the actual parameter is not of this type.

Example:

VAR

StringVariable : STRING;

Length : INT := 32;

END_VAR

LD "EXAMPLE"

LEFT length (* Error: this parameter must be of type UINT *)

ST StringVariable

4.3.5.19 C10036

Structs and arrays of complex data types are not supported by this hardware.

An array of a structured type, an array of an array type, a structure with a structured member or a structure with an array member has been declared. This is not supported by the hardware. For more information about supported data types for your hardware, consult your hardware documentation.

Example:

TYPE

DayOfWeek : STRUCT

Name : STRING;

DayNumber : UINT;

END_STRUCT;

DayDescriptions : ARRAY[1..100] OF DayOfWeek;

(* Error: Day of Week is a complex data type. Arrays of complex data types are not supported by the hardware. *)

Presence : STRUCT

Name : STRING;

OursPerDay : ARRAY[1..31] OF UINT;

(* Error: ARRAY is a complex data type. Structs of complex data types are not supported by the hardware *)

END_STRUCT;

END_TYPE

4.3.5.20 C10038

Couldn't detect the type of the constant.

The type of a constant could not be determined. Please initialize a variable of the desired type with this constant and use the variable instead of the constant.

4.3.5.21 C10043

Implementation code is not allowed.

Implementation code has been found in a file with resource global variable declarations. This is not allowed. Please declare the requested variable in another program organization unit as an external variable and move the code in the respective file.

4.3.5.22 C10045

Function blocks instances are not allowed in this section.

An instance declaration of a function block has been found in a section where this is not allowed. Please move the declaration in a section, where function block instances are supported.

4.3.5.23 C10046

"VAR_GLOBAL" is not allowed.

A VAR_GLOBAL section has been found in a program organization unit where this section kind is not supported. Please change the section kind or move the variable declaration in a file, where global variables are supported.

According to the IEC 61131-3 VAR_GLOBAL sections are supported only in PROGRAMs. However the hardware manufacturer may restrict the declaration of global variables to resource global variable files. I. e. global variables are allowed only in specific files which contain only global variable declarations.

4.3.5.24 C10047

Only "VAR_GLOBAL" allowed.

A variable declaration section, which is not a VAR_GLOBAL section, has been found in a file for resource global variable declaration. This is not allowed. Please change the section kind or move the variable declaration in another file, where this kind of declarations are supported.

4.3.5.25 C10049

String too long.

A string has been declared with a length specification, which exceeds the maximum string length supported by the hardware.

For the maximum string length supported by OpenPCS see the IEC 1131-3 Compliance statement. However, the hardware-manufacturer can restrict the maximum string length by changing the value of the "MaxStringLength" entry in the [MODULE] section of the hardware description file.

4.3.5.26 C10055

This variable cannot be initialized.

Either an initialization of a directly represented variable has been found or the hardware doesn't support variable initializations. The initialization of directly represented variables is not supported by OpenPCS. The initialization of symbolic variables can be forbidden by the manufacturer by changing the value for the "InitVariables" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out, if variable initialization is supported by your hardware.

4.3.5.27 C10057

Data type is not supported.

The data type at the given position is not supported. For a list of data types supported by OpenPCS see the IEC 1131-3 Compliance statement. For a list of data types supported by your hardware, please consult your hardware documentation.

4.3.5.28 C10060

LD/ST of function block instances is not allowed.

A LD or ST instruction with a function block instance as an operand has been found. This is not allowed.

4.3.5.29 C10063

An error occurred while opening a file.

4.3.5.30 C10064

Internal Compiler Error No. %1. Please contact your manufacturer.

An internal compiler error occurred. Please contact your manufacturer.

4.3.5.31 C10067

Struct declarations are not supported.

A struct declaration has been detected, but is not supported by the hardware. Struct declarations are supported by OpenPCS. The hardware manufacturer however, can forbid struct declarations by setting the value of the "StructAllowed" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out if struct declarations are supported by your hardware.

4.3.5.32 C10068

Array declarations are not supported.

An array declaration has been detected, but is not supported by the hardware. Array declarations are supported by OpenPCS. The hardware manufacturer however, can forbid array declarations by setting the value of the "ArrayAllowed" entry in the [MODULE] section of the hardware description file to 0. Please consult your hardware documentation to find out if array declarations are supported by your hardware.

4.3.5.33 C10069

Enumerated data type declarations are not supported.

A enumerated data type declaration has been detected, but is not supported by the hardware. Enumerated data type declarations are supported by OpenPCS. The hardware manufacturer however, can forbid this declarations by setting the value of the "EnumAllowed" entry in the [MODULE] section of the hardware description file to 0. consult your hardware documentation to find out if enumerated data type declarations are supported by your hardware.

4.3.5.34 C10075

Invalid array index. It has to range between -32767 and 32767.

An array index is out of the supported range [-32767, 32767].

4.3.5.35 C10076

Lower array bound exceeds minimum array bound limit (dimension #).

An lower array bound is out of the supported range [-32767, 32767]. dimension # is the erroneous dimension of the array, if multidimensional.

4.3.5.36 C10078

Invalid type of a global or directly represented variable.

A directly represented variable of a complex or an user defined type has been declared. This is not supported. Global variable of structured types are also not supported.

4.3.5.37 C10083

Only directly represented variables are allowed in this POU.

Resource global variables are separated in two kind of files. Files which contain only symbolic variables and files which contain the directly represented variables. In these files symbolic and directly represented variables must not be mixed up.

4.3.5.38 C10084

Global structs are not supported.

Please declare this variable in a local section and use input and output parameters, if the value should be changed by a function or function block. The type declaration for the desired structure must be done on project level.

Example:

(The following structure has to be declared as a project global type*)*

TYPE

DayOfWeek : STRUCT

Name : STRING;

DayNumber : UINT;

END_STRUCT;

END_TYPE

FUNCTION_BLOCK AdjustDayName

VAR_INPUT

DayIn : DayOfWeek;

END_VAR

VAR_OUTPUT

DayOut : DayOfWeek;

END_VAR

LD DayIn

ST DayOut

LD DayIn.DayNumber

EQ 1

LD "MONDAY"

ST DayOut.Name

LD DayIn.DayNumber

EQ 2

LD "TUESDAY"

ST DayOut.Name

END_FUNCTION_BLOCK

PROGRAM main

VAR

Day : DayOfWeek;

DayNumber : UINT;

END_VAR

LD DayNumber

ST Day.DayNumber

CAL AdjustDayName(DayIn := Day | Day := DayOut)

END_PROGRAM

4.3.5.39 C10092

Memory allocation failure.

4.3.5.40 C10093

Data Segment Out Of Memory

To much data (e.g. variables) for program or function block so the data doesn't fit into a 64 kB segment. Segments are restricted to 64 kB.

Remark:

If this error occurs, try to restruct the program/function block and put some variables into other function blocks (FBs can be used as data containers) or use resource global variables.

4.3.5.41 C10094

Initial Data Segment Out Of Memory

To much data (e.g. variables) for program or function block so the data doesn't fit into a 64 kB segment. Segments are restricted to 64 kB.

Remark:

If this error occurs, try to restruct the program/function block and put some variables into other function blocks (FBs can be used as data containers) or use resource global variables.

4.3.5.42 C10095

Code Segment Memory Allocation Failure

This error occurs if the program code (UCode/Native Code) doesn't fit into a 64 kB segment. The size for a segment is restricted to 64 kB.

Remark:

If this error occurs, it is possible to restruct the program (e.g. putting some parts of the code into Function Blocks) so that the program decreases down to 64 kB.

4.3.5.43 C10096

Data Segment size warning limit reached. Used X of Y bytes.

The size of the corresponding data segment has reached the configured warning limit. Set the size within the [browser options](#) dialog.

4.3.5.44 C10097

Initial Data Segment size warning limit reached. Used X of Y bytes.

The size of the corresponding initial data segment has reached the configured warning limit. Set the size within the [browser options](#) dialog.

4.3.5.45 C10098

Code Segment size warning limit reached. Used X of Y bytes.

The size of the corresponding code segment has reached the configured warning limit. Set the size within the [browser options](#) dialog.

4.3.5.46 C10100

Invalid expression for parameter.

An invalid expression has been passed as an actual parameter in a call of a function or a function block instance.

4.3.5.47 C10108

Constant of type TIME is out of range.

For the range of TIME constants supported by OpenPCS see the IEC 1131-3 Compliance statement.

4.3.5.48 C10109

Invalid data type for this operation. Integer or real type expected.

The operation at the given position is only supported for integer and real operands.

4.3.5.49 C10110

Nested functions are not supported.

A function call has been passed as an actual parameter in the call of a function or a function block instance. This is not supported. Please save the return value of the function in a variable and pass this variable as an actual parameter to the called program organization unit.

4.3.5.50 C10112

Type conflict.

Either the current result is incompatible with the expected data type or the type of an actual parameter is incompatible with the type of the respective formal parameter.

4.3.5.51 C10113

Operation not supported for this data type.

The data type of an operand is not allowed for the operation at the given position. For more information about allowed data types for this operation see IEC 61131-3 and the IEC 1131-3 Compliance statement.

4.3.5.52 C10114

Parameter expressions are not supported for this operation.

An expression has been used as an actual parameter. This is not supported. Please store the result of the expression in a variable and pass this variable to the called function or function block.

4.3.5.53 C10115

Retain attribute for FB instances forbidden.

RETAIN function block instances are not supported. Please remove the attribute or move the instance declaration out of this section.

4.3.5.54 C10777

Upper array bound exceeds maximum array bound limit (dimension #).

An upper array bound is out of the supported range [-32767, 32767]. dimension # is the erroneous dimension of the array, if multidimensional.

4.3.5.55 C11001

Can't determine unambiguously the type of constant -> take %1.

The type of a numeric constant couldn't be determined unambiguously. In this case usually the biggest supported data type of the expected data type class (ANY_INT, ANY_REAL, ANY_BIT) is presumed.

4.3.5.56 C11007

Function has no input parameter. Is this intended?

A function call to a function which has no parameters has been detected. Was this the intend? Functions do not contain internal state information and can be supplied only with input parameters. Generally the return value is computed by using the input parameters. Because of this reasons a function without input parameters usually doesn't make sense. Please check if the called function makes sense.

4.3.6 Make Messages

4.3.6.1 M21004

Unknown command: %1.

An unknown command line argument has been used with [ITMAKE](#).

4.4 Shortcuts

4.4.1 Common Shortcuts

File Submenu

CTRL+N:	New File
CTRL+F4:	Close
CTRL+S:	Save
ALT+F10:	Syntax Check
CTRL+P:	Print
CTRL+O:	Open Project
ALT+F4:	Exit

Edit Submenu

CTRL+Z:	Undo
	Redo
CTRL+Y:	Cut
CTRL+X/SHIFT+DEL:	Copy
CTRL+C/CTRL+INS:	Paste
CTRL+V/SHIFT+INS:	Delete
DEL:	Next Error
F4:	Previous Error
SHIFT+F4:	Find
CTRL+F:	Replace
CTRL+H:	Goto IL Line (SFC)
CTRL+G:	Select All
CTRL+A:	Properties
ALT+RETURN:	

PLC Submenu

F7:	Build Active Resource
CTRL+F7:	Rebuild Active Resource
F9:	Toggle Breakpoint
F5:	Go
F11:	Step Into
F10:	Step Over
SHIFT+F11:	Step Out
ALT+ENTER:	Resource Properties

Window Submenu

F6:	Next Pane
ALT+1:	Project
ALT+2:	Document
ALT+3:	Test and Commissioning
ALT+4:	Output
Ctrl+Enter:	Fullscreen

Insert->Variable Submenu

ALT+SHIFT+V:	All Variables
ALT+SHIFT+I:	Input Variables
ALT+SHIFT+O:	Output Variables
ALT+SHIFT+N:	In/Out Variables
ALT+SHIFT+L:	Local Variables
ALT+SHIFT+G:	Global Variables
ALT+SHIFT+E:	External Variables
ALT+SHIFT+F:	FB-Instance Variables

4.4.2 Editor depending Shortcuts

IL/ST Editor

CTRL+ALT+F:	Insert Function
CTRL+ALT+B:	Insert Functionblock

LADDER Editor

F12:	Insert Network
CTRL+ALT+F:	Insert Function
CTRL+ALT+B:	Insert Functionblock
SHIFT+RETURN:	Insert New Line in Comment

SFC Editor

CTRL+ALT+S:	Insert Step/Transition
CTRL+ALT+L:	Insert Step/Transition left
CTRL+ALT+R:	Insert Step/Transition right
CTRL+ALT+J:	Insert Jump
CTRL+ALT+B:	Insert Functionblock

CTRL+ALT+F: Insert Function

CFC/FBD Editor

CTRL+B: Insert Connection

CTRL+SHIFT+V: Switches between variable value and variable name at the margins in onlinemode

5 Index

)	
) (Right-paranthesis-operator).....		202
	*	
*_TO_**.....		203, 289
*_to_bool		203
*_TO_STRING		203
	A	
About OPC		36
About OPC Server		113
About this manual.....		24
ABS.....		204
ABS_DINT.....		204
ABS_DINT_FBD		204
ABS_INT.....		204
ABS_INT_FBD		204
ABS_REAL.....		204
ABS_REAL_FBD		204
ABS_SINT.....		204
ABS_SINT_FBD.....		204
ABS_UDINT_FBD		204
ABS_UINT_FBD		204
ABS_USINT_FBD		204
ACOS		205
ACOS_REAL		205
ACOS_REAL_FBD		205
ACTION		205
Active Document Server		137
Active Resource		35
ADD		205
ADD (time)		206
Add files		40
Add Task		35
ADD_DINT		205
ADD_DINT_FBD.....		205
ADD_INT.....		205
ADD_INT_FBD		205
ADD_REAL.....		205
ADD_REAL_FBD		205
ADD_SINT		205
ADD_SINT_FBD.....		205
ADD_TIME.....		205, 206
ADD_TIME_FBD		205, 206
ADD_UDINT.....		205
ADD_UDINT_FBD		205
ADD_UINT.....		205
ADD_UINT_FBD		205
ADD_USINT		205
ADD_USINT_FBD		205
AddHW.....		119
Adding a Library to a project.....		140
Adding Hardware Support.....		22
Adding input or output to compound block.		87
Adjusting order of cyclic tasks.....		128
Alias names		74
AND.....		206
AND_BOOL_EN		206
AND_BOOL_FBD.....		206
AND_BYTE_FBD		206
AND_DWORD_EN		206
AND_DWORD_FBD		206
AND_WORD_EN		206
AND_WORD_FBD		206
ANDN.....		206
ANDN_BOOL_FBD.....		206
ANDN_BYTE_FBD		206
ANDN_DWORD_FBD		206
ANDN_WORD_FBD		206
ANY		207
ANY_BIT.....		207
ANY_DATE		207
ANY_INT		207
ANY_NUM.....		208
ANY_REAL		208

ARRAY	208
ASIN	209
ASIN_REAL	209
ASIN_REAL_FBD	209
Assignment	210
Assignment Editor Introduction.....	54
AT.....	210
ATAN.....	211
ATAN_REAL.....	211
ATAN_REAL_FBD	211
AutoComplete.....	60, 66
AutoDeclare	60, 66
Automatic positioning of the caret.....	77

BYTE_TO_REAL_EN	212
BYTE_TO_sint	212
BYTE_TO_SINT_EN.....	212
BYTE_TO_STRING_EN.....	203, 212
BYTE_TO_TIME_EN.....	212
BYTE_TO_udint	212
BYTE_TO_UDINT_EN	212
BYTE_TO_uint.....	212
BYTE_TO_UINT_EN.....	212
BYTE_TO_usint	212
BYTE_TO_USINT_EN.....	212
BYTE_TO_WORD	212
BYTE_TO_WORD_EN.....	212

B

Block specific help.....	71
Block Type Program Function Function Block	156
BOOL	211
Bool_to_*	211
BOOL_TO_BYTE.....	211
BOOL_TO_BYTE_EN	211
BOOL_TO_dint.....	211
BOOL_TO_DINT_EN	211
BOOL_TO_DWORD.....	211
BOOL_TO_DWORD_EN	211
BOOL_TO_int	211
BOOL_TO_INT_EN	211
BOOL_TO_REAL.....	211
BOOL_TO_REAL_EN	211
BOOL_TO_sint.....	211
BOOL_TO_SINT_EN.....	211
BOOL_TO_STRING_EN	203, 211
BOOL_TO_TIME_EN	211
BOOL_TO_udint.....	211
BOOL_TO_UDINT_EN	211
BOOL_TO_uint.....	211
BOOL_TO_UINT_EN	211
BOOL_TO_usint	211
BOOL_TO_USINT_EN.....	211
BOOL_TO_WORD	211
BOOL_TO_WORD_EN	211
Breakpoints.....	189
Browser Introduction.....	27
Browser Options	40
Build active resource	36
BY.....	212
BYTE	212
BYTE_TO_BOOL	203, 212
BYTE_TO_BOOL_EN	203, 212
BYTE_TO_dint	212
BYTE_TO_DINT_EN	212
BYTE_TO_DWORD.....	212
BYTE_TO_DWORD_EN	212
BYTE_TO_int	212
BYTE_TO_INT_EN	212
BYTE_TO_REAL.....	212

C

C10006.....	365
C10007.....	365
C10008.....	365
C10009.....	365
C10010.....	365
C10012.....	366
C10017.....	366
C10019.....	366
C10020.....	366
C10021.....	366
C10024.....	367
C10025.....	367
C10026.....	367
C10028.....	367
C10030.....	367
C10031.....	367
C10034.....	368
C10035.....	368
C10036.....	368
C10038.....	369
C10043.....	369
C10045.....	370
C10046.....	370
C10047.....	370
C10049.....	370
C10055.....	371
C10057.....	371
C10060.....	371
C10063.....	371
C10064.....	371
C10067.....	371
C10068.....	372
C10069.....	372
C10075.....	372
C10076.....	372
C10078.....	372
C10083.....	373
C10084.....	373
C10092.....	375
C10093.....	375
C10094.....	375
C10095.....	375

EXP_REAL	242
Expressions in ST	58
EXPT	242
EXPT_DINT	242
EXPT_INT	242
EXPT_REAL	242
EXPT_SINT	242
EXPT_UDINT	242
EXPT_USINT	242
Extensible inputs	72

F

F_EDGE	243
F_TRIG	243
FALSE	244
Fast navigation with the caret	83
FBD	244
FBD Editor Online	105
FBD language Elements	184
File	33
File Operations	33
File-Pane	27
FIND	244
FIND_STRING	244
FIND_STRING_FBD	244
Finding error position	98
Finding Errors in CFC	71
Finding Errors in FBD	105
First Program	10
FOR	245
Force Variables	106
FROM	246
Function	64, 246
FUNCTION BLOCK	247
Functionblock	64
Functionblocks	64
Functionblocks and Functions	64
Functions	64
Functions with negatable inputs	72
Fundamentals for keyboard usage	76

G

G10001	301
GE	248
GE_BOOL_FBD	248
GE_BYTE_FBD	248
GE_DINT_FBD	248
GE_DWORD_FBD	248
GE_INT_FBD	248
GE_REAL_FBD	248
GE_SINT_FBD	248
GE_STRING_FBD	248
GE_TIME_FBD	248
GE_UDINT_FBD	248
GE_UINT_FBD	248
GE_USINT_FBD	248

GE_WORD_FBD	248
GetDateStruct	248
GETSYSTEMDATEANDTIME	249
GetTaskInfo	249
GetTime	249
GetTimeCS	250
GetVarData	251
GetVarFlatAddress	252
Global Id	76
Going Online	36
GT	253
GT_BOOL_FBD	253
GT_BYTE_FBD	253
GT_DINT_FBD	253
GT_DWORD_FBD	253
GT_INT_FBD	253
GT_REAL_FBD	253
GT_SINT_FBD	253
GT_STRING_FBD	253
GT_TIME_FBD	253
GT_UDINT_FBD	253
GT_UINT_FBD	253
GT_USINT_FBD	253
GT_WORD_FBD	253

H

Hardware	119
Hardware and Software Requirements	8
Hardware information	38
Help-Pane	30
How to Read Error Message	300

I

IEC61131 Standard Function Blocks	192
IEC61131-3 operations	194
IEC61131-3 Standard Functions	192
IF253	
IL 254	
IL Editor Introduction	55
IL Editor Online	57
Import/Export	32
IN	255
INITIAL_STEP	255
Inline edit at the caret position	84
Input and Output Variables	101
INSERT	255
Insert a DCF-file into OpenPCS	146
Insert connections by keyboard	85
Insertion of blocks by keyboard usage	84
Install a Library	139
Installation	8
Instruction List Instructions	196
Instructions in IL	56
Instructions in ST	58
INT	255
int_TO_BOOL	203, 255

INT_TO_BOOL_EN	203, 255	L12002	361
INT_TO_BYTE_EN	255	L12003	362
int_TO_DINT	255	L12005	362
INT_TO_DINT_EN	255	L12006	362
int_TO_DWORD	255	L12007	362
INT_TO_DWORD_EN	255	L12008	362
int_TO_REAL	255	L12064	362
INT_TO_REAL_EN	255	L12065	363
int_TO_sint	255	L12066	363
INT_TO_SINT_EN	255	L12996	363
INT_TO_STRING_EN	203, 255	L12997	363
INT_TO_TIME_EN	255	L12998	363
int_TO_udint	255	L12999	363
INT_TO_UDINT_EN	255	L13000	364
INT_TO_UINT_EN	255	L13001	364
int_TO_usint	255	L14009	364
INT_TO_USINT_EN	255	L14010	364
int_TO_WORD	255	L15001	364
INT_TO_WORD_EN	255	L20012	364
Intellisense	60, 66	Ladder Editor introduction	61
Interrupt Tasks	112	Ladder Editor Online	65
Interrupts	126	Ladder Logic introduction	61
Interval	256	LD	257
Introduction CFC Editor	67	LD (Ladder Diagram)	257
Introduction FBD Editor	98	LDN	257
J			
JMP	256	LE	258
JMPC	256	LE_BOOL_FBD	258
JMPCN	256	LE_BYTE_FBD	258
Jumps	92	LE_DINT_FBD	258
K			
Keyboard combinations for navigating the caret	85	LE_DWORD_FBD	258
Keyboard handling for CFC and FBD	105	LE_INT_FBD	258
L			
L 257		LE_REAL_FBD	258
L10001	357	LE_SINT_FBD	258
L10004	358	LE_STRING_FBD	258
L10026	358	LE_TIME_FBD	258
L10027	358	LE_UDINT_FBD	258
L10029	359	LE_UINT_FBD	258
L10030	359	LE_USINT_FBD	258
L10031	359	LE_WORD_FBD	258
L10032	359	LEFT	258
L10033	359	LEFT_DINT	258
L10034	360	LEFT_INT	258
L10035	360	LEFT_SINT	258
L10036	360	LEFT_STRING_FBD	258
L10063	360	LEFT_UDINT	258
L10105	360	LEFT_UINT	258
L10106	361	LEFT_USINT	258
L12001	361	LEN	258
		LEN_STRING	258
		LEN_STRING_FBD	258
		Lib-Pane	30
		Library Overview	137
		Library-Pane	30
		License Editor Overview	123
		LIMIT	259
		LIMIT_BOOL	259
		LIMIT_BYTE	259
		LIMIT_DINT	259
		LIMIT_DWORD	259

LIMIT_INT.....	259	MAX_UINT_FBD	261
LIMIT_REAL	259	MAX_USINT	261
LIMIT_SINT.....	259	MAX_USINT_FBD.....	261
LIMIT_STRING.....	259	MAX_WORD	261
LIMIT_TIME.....	259	Maximum String Length	153
LIMIT_UDINT	259	MID	261
LIMIT_UINT.....	259	MIN	262
LIMIT_USINT.....	259	MIN_BOOL.....	262
LIMIT_WORD.....	259	MIN_BYTE	262
Linker Command Line.....	121	MIN_DINT	262
LINT	259	MIN_DINT_FBD.....	262
LN.....	259	MIN_DWORD	262
LN_REAL.....	259	MIN_INT	262
LN_REAL_FBD	259	MIN_INT_FBD	262
LOG	259	MIN_REAL	262
LOG_REAL	259	MIN_REAL_FBD.....	262
LOG_REAL_FBD	259	MIN_SINT.....	262
Lreal	260	MIN_SINT_FBD	262
LT	260	MIN_STRING	262
LT_BOOL_FBD	260	MIN_TIME	262
LT_BYTE_FBD.....	260	MIN_UDINT	262
LT_DINT_FBD.....	260	MIN_UDINT_FBD.....	262
LT_DWORD_FBD.....	260	MIN_UINT_FBD	262
LT_INT_FBD.....	260	MIN_USINT	262
LT_REAL_FBD.....	260	MIN_USINT_FBD	262
LT_SINT_FBD	260	MIN_WORD	262
LT_STRING_FBD	260	MOD	262
LT_TIME_FBD.....	260	MOD_DINT	262
LT_UDINT_FBD.....	260	MOD_DINT_FBD.....	262
LT_UINT_FBD.....	260	MOD_INT	262
LT_USINT_FBD.....	260	MOD_INT_FBD	262
LT_WORD_FBD.....	260	MOD_SINT	262
Lword.....	260	MOD_SINT_FBD	262
		MOD_UDINT	262
		MOD_UDINT_FBD.....	262
		MOD_UINT	262
		MOD_UINT_FBD.....	262
		MOD_USINT	262
		MOD_USINT_FBD.....	262
		Monitoring code	17
		More Information	24
		MOVE.....	263
		MOVE_DINT.....	263
		MOVE_INT.....	263
		MOVE_REAL.....	263
		MOVE_SINT.....	263
		MOVE_UDINT.....	263
		MOVE_UINT.....	263
		MOVE_USINT.....	263
		Moving/copying blocks and margin connectors by keyboard	84
		MUL.....	263
		MUL (time)	264
		MUL_DINT.....	263
		MUL_DINT_FBD.....	263
		MUL_INT	263
		MUL_INT_FBD.....	263
		MUL_REAL.....	263
		MUL_REAL_FBD.....	263

M

M21004	378
Make Command Line	122
Margin Bars.....	68
Marking a single element	96
Marking several elements	97
Masking of unused connectors	74
MAX	261
MAX_BOOL.....	261
MAX_DINT	261
MAX_DINT_FBD.....	261
MAX_DWORD	261
MAX_INT	261
MAX_INT_FBD	261
MAX_REAL	261
MAX_REAL_FBD.....	261
MAX_SINT	261
MAX_SINT_FBD	261
MAX_STRING	261
MAX_TIME	261
MAX_UDINT	261
MAX_UDINT_FBD.....	261
MAX_UINT	261

MUL_SINT	263
MUL_SINT_FBD	263
MUL_UDINT	263
MUL_UDINT_FBD	263
MUL_UINT	263
MUL_UINT_FBD	263
MUL_USINT	263
MUL_USINT_FBD	263
Multiple Connections	70
Multiple Resources	127
Multitasking	125
MUX	261

N

N (Action Qualifier)	264
Native Code	128
NCC	264
NCC ARM ARM Mode	131
NCC ARM THUMB Mode	131
NCC Hitachi H8/300H	131
NCC Infineon C16x (huge model)	130
NCC Intel Protected Mode	130
NCC Intel Real Mode	131
NCC Motorola 68K	131
NCC Motorola DSP563xx	131
NCC Motorola PowerPC 8x	131
NE	264
NE_BOOL_FBD	264
NE_BYTE_FBD	264
NE_DINT_FBD	264
NE_DWORD_FBD	264
NE_INT_FBD	264
NE_REAL_FBD	264
NE_SINT_FBD	264
NE_STRING_FBD	264
NE_TIME_FBD	264
NE_UINT_FBD	264
NE_USINT_FBD	264
NE_WORD_FBD	264
NEG	265
Nested Comments	156
Network	61
NOT	265
NOT_BOOL_FBD	265
NOT_BYTE_FBD	265
NOT_DWORD_FBD	265
NOT_WORD_FBD	265

O

OF	265
On	265
Online Change	190
Online connections introduction	116
Online Edit	20, 191
Online Server Overview	116
OPC	266

OPC - I/O Introduction	35
OPC - I/O-Pane	29
Open Project	32
OpenPCS Framework Introduction	25
OpenPCS Function Blocks	194
OpenPCS Samples	10
Operators	62
Optimisation Settings	126
OR	266
OR_BOOL	266
OR_BOOL_FBD	266
OR_BYTE	266
OR_BYTE_FBD	266
OR_DWORD	266
OR_DWORD_FBD	266
OR_WORD	266
OR_WORD_FBD	266
ORN	266
ORN_BOOL_FBD	266
ORN_BYTE_FBD	266
ORN_DWORD_FBD	266
ORN_WORD_FBD	266
Oscilloscope	110
Others	200
Output Window	26
Overview SmartSIM	112

P

P 267	
Passing Output Parameters	155
performance	127
POINTER	267
Positioning of the caret	77
POU	267
Print Form	136
Print IEC61131 Configuration	132
Priority	267
PROGRAM	268
PT	268
PV	268

Q

Q 268	
Q1	268
QD	268
QU	268

R

R(Action Qualifier)	269
R(reset)	269
R_EDGE	269
R_TRIG	270
R1	270
READ_ONLY	270
READ_WRITE	271

REAL	271	S(et)	279
Real_to_*	271	S1	280
REAL_TO_BOOL	203	S1000	301
REAL_TO_BOOL_EN	203	S1001	301
REAL_TO_STRING_EN	203	S1002	301
Rebuild active resource	36	S1003	301
Rebuild all resources	36	S1004	301
Refresh project information	32, 271	S1005	302
Region marks	96	S1006	302
Release	273	S1008	302
Remote OPC Server	113	S1009	302
REPEAT	273	S1010	302
REPLACE	274	S1011	303
Replacement of Blocks	71, 104	S1012	303
Representation of the caret	76	S1013	303
Resource	274	S1014	303
Resource global variables	40	S1015	304
Resource information	38	S1016	304
Resource-Pane	28	S1017	304
Resources introduction	33	S1018	305
RESUME	274	S1019	305
RET	275	S1020	305
RETAIN	275	S1021	305
RETC	275	S1022	305
RETCN	275	S1023	306
RETURN	276	S1024	306
RIGHT	276	S1025	306
RIGHT_DINT	276	S1026	306
RIGHT_INT	276	S1027	307
RIGHT_SINT	276	S1028	307
RIGHT_STRING_FBD	276	S1029	307
RIGHT_UDINT	276	S1030	307
RIGHT_UINT	276	S1031	307
RIGHT_USINT	276	S1032	308
ROL	277	S3000	308
ROL_BOOL	277	S3001	309
ROL_BOOL_FBD	277	S3002	309
ROL_BYTE	277	S3003	310
ROL_BYTE_FBD	277	S3004	310
ROL_DWORD	277	S3005	310
ROL_DWORD_FBD	277	S3006	311
ROL_WORD	277	S3007	311
ROL_WORD_FBD	277	S3008	311
ROR	277	S3009	312
ROR_BOOL	277	S3010	312
ROR_BOOL_FBD	277	S3011	312
ROR_BYTE	277	S3012	313
ROR_BYTE_FBD	277	S3014	313
ROR_DWORD	277	S3016	314
ROR_DWORD_FBD	277	S3017	314
ROR_WORD	277	S3018	314
ROR_WORD_FBD	277	S3019	314
RS	278	S3020	314
RTC	279	S3022	315
		S3023	315
		S3024	315
		S3025	315
		S3026	315
		S3028	316

S

S(Action Qualifier)	279
---------------------	-----

S3030	317	S5015.....	339
S3032	318	S5016.....	339
S3033	318	S5017.....	339
S3034	319	S5018.....	339
S3035	319	S5019.....	340
S3036	319	S5020.....	340
S3037	320	S5021.....	341
S3038	320	S5022.....	341
S3039	320	S5023.....	342
S3040	320	S5024.....	343
S3041	321	S5025.....	343
S3042	321	S5026.....	344
S3044	321	S5027.....	344
S3046	322	S5028.....	344
S3047	322	S5029.....	345
S3048	324	S5030.....	346
S3049	324	S5031.....	346
S3050	324	S5032.....	347
S4000	324	S5033.....	347
S4001	325	S5034.....	349
S4003	325	S5035.....	350
S4005	325	S5036.....	350
S4006	325	S5037.....	352
S4007	326	S5038.....	353
S4008	326	S5039.....	354
S4009	326	S5040.....	355
S4010	327	S5041.....	356
S4011	327	S5042.....	356
S4012	327	S5043.....	356
S4013	328	S6002.....	356
S4014	328	S6004.....	357
S4015	328	S6005.....	357
S4016	328	Sample Program.....	11
S4017	328	Save System	191
S4018	328	SaveSystemCmd	191
S4019	329	SD.....	280
S4020	329	Search within project	32
S4021	329	SEL.....	280
S4022	329	Select Connection.....	118
S4023	330	SEMA	280
S4024	330	Set variables.....	106
S4033	330	SETSYSTEMDATEANDTIME	280
S4034	332	Setting fonts and color	42
S4035	332	SFC	281
S4036	332	SFC Editor Online	92
S5000	332	SFC introduction.....	87
S5001	332	SHL	281
S5002	333	SHL_BOOL.....	281
S5003	333	SHL_BOOL_FBD	281
S5004	333	SHL_BYTE	281
S5005	334	SHL_BYTE_FBD	281
S5006	334	SHL_DWORD	281
S5008	334	SHL_DWORD_FBD	281
S5009	334	SHL_WORD	281
S5010	335	SHL_WORD_FBD	281
S5011	337	SHR	281
S5012	337	SHR_BOOL	281
S5013	338	SHR_BOOL_FBD.....	281
S5014	339	SHR_BYTE	281

Table 27 Standard selection functions.....	168
Table 28 Standard comparison functions..	168
Table 29 Standard character string functions	
.....	169
Table 3 Comment features	157
Table 30 Functions of time data types	169
Table 31 Functions of enumerated data types	
.....	170
Table 33 Function block declaration features	
.....	170
Table 34 Standard bistable function blocks	
.....	171
Table 35 Standard edge detection function	
blocks	172
Table 36 Standard counter function blocks	
.....	172
Table 37 Standard timer function blocks ..	172
Table 39 Program declaration features.....	172
Table 4 Numeric Literals	158
Table 40 Step features	174
Table 41 Transitions and Transition conditions	
.....	174
Table 42 Declaration of actions.....	175
Table 43 Step/action association.....	176
Table 44 Action block features	176
Table 45 Action qualifiers	176
Table 46 Sequence evolution	177
Table 5 Character string literal features ...	158
Table 52 Instruction list (IL) operators.....	178
Table 53 Function block invocation features	
for IL language	179
Table 55 Operators of the ST language	179
Table 56 ST language statements	180
Table 57 Representation of lines and block	
.....	181
Table 58 Graphic execution control elements	
.....	182
Table 59 Power rails	182
Table 6 Two character combinations in	
character strings.....	159
Table 60 Link Elements.....	182
Table 61 Contacts	183
Table 62 Coils.....	183
Table 63 Reserved Names.....	184
Table 7 Duration literal features.....	159
Table 8 Date and time of day literals	159
Table D.1 Implementation-dependent	
parameters	184
Table E.1 Error conditions	187
TAN.....	288
TAN_REAL.....	288
TAN_REAL_FBD	288
Task.....	288
Templates	22
Test and Commissioning Introduction	105
Text Block.....	69
THEN.....	288
TIME	289
TIME_OF_DAY	289
TIME_TO_BOOL_EN	289
TIME_TO_BYTE_EN	289
TIME_TO_DINT_EN.....	289
TIME_TO_DWORD_EN.....	289
TIME_TO_INT_EN.....	289
TIME_TO_REAL_EN.....	289
TIME_TO_SINT_EN	289
TIME_TO_STRING_EN.....	203, 289
TIME_TO_UDINT_EN.....	289
TIME_TO_UINT_EN.....	289
TIME_TO_USINT_EN.....	289
TIME_TO_WORD_EN.....	289
TO.....	245, 290
TO_STRING	165
TOD.....	290
TOF	290
TON.....	291
Tooltips for structs and elements of structs	60
TP	292
Transition	293
Transitions	91
Trend View	19, 108
Trigger.....	111
TRUE	293
TRUNC.....	293
TYPE.....	294
Type definitions.....	40
U	
UCODE.....	128
UDINT.....	294
udint_TO_BOOL	203, 294
UDINT_TO_BOOL_EN.....	203, 294
udint_TO_BYTE	294
UDINT_TO_BYTE_EN	294
udint_TO_dint.....	294
UDINT_TO_DINT_EN	294
udint_TO_DWORD	294
UDINT_TO_DWORD_EN	294
udint_TO_int.....	294
UDINT_TO_INT_EN.....	294
udint_TO_REAL	294
UDINT_TO_REAL_EN	294
udint_TO_sint	294
UDINT_TO_SINT_EN.....	294
UDINT_TO_STRING_EN.....	203, 294
UDINT_TO_TIME_EN.....	294
udint_TO_uint.....	294
UDINT_TO_UINT_EN.....	294
udint_TO_usint	294
UDINT_TO_USINT_EN.....	294
udint_TO_WORD	294
UDINT_TO_WORD_EN.....	294
UINT.....	294
uint_TO_BOOL	203, 294
UINT_TO_BOOL_EN	203, 294
uint_TO_BYTE.....	294

UINT_TO_BYTE_EN	294
uint_TO_dint	294
UINT_TO_DINT_EN	294
uint_TO_DWORD	294
UINT_TO_DWORD_EN	294
uint_TO_int	294
UINT_TO_INT_EN	294
uint_TO_REAL	294
UINT_TO_REAL_EN	294
uint_TO_sint	294
UINT_TO_SINT_EN	294
UINT_TO_STRING_EN	203, 294
UINT_TO_TIME_EN	294
uint_TO_udint	294
UINT_TO_UDINT_EN	294
uint_TO_usint	294
UINT_TO_USINT_EN	294
uint_TO_WORD	294
UINT_TO_WORD_EN	294
ULINT	295
Uninstall Library	141
Unknown instructions	130
UNTIL	295
Upload	38
Usage without License Key	124
Using constants as inputs	69
Using languages other than IL	98
USINT	295
usint_TO_BOOL	203, 295
USINT_TO_BOOL_EN	203, 295
usint_TO_BYTE	295
USINT_TO_BYTE_EN	295
usint_TO_dint	295
USINT_TO_DINT_EN	295
usint_TO_DWORD	295
USINT_TO_DWORD_EN	295
usint_TO_int	295
USINT_TO_INT_EN	295
usint_TO_REAL	295
USINT_TO_REAL_EN	295
usint_TO_sint	295
USINT_TO_SINT_EN	295
USINT_TO_STRING_EN	203, 295
USINT_TO_TIME_EN	295
usint_TO_udint	295
USINT_TO_UDINT_EN	295
usint_TO_uint	295
USINT_TO_UINT_EN	295
usint_TO_WORD	295
USINT_TO_WORD_EN	295

V

VAR	295
VAR_ACCESS	295
VAR_EXTERNAL	297
VAR_GLOBAL	296
VAR_IN_OUT	296

VAR_INPUT	296
VAR_OUTPUT	296
Variable Address	127
Variablecatalog	43
Variablegrid	43
Variabletable	43
VARINFO	297

W

Watch variables	106
Watching variables	37
Watchlist	107
WHILE	297
Wiring	100
WITH	298
WORD	298
WORD_TO_BOOL	203, 298
WORD_TO_BOOL_EN	203, 298
WORD_TO_BYTE	298
WORD_TO_BYTE_EN	298
WORD_TO_dint	298
WORD_TO_DINT_EN	298
WORD_TO_DWORD	298
WORD_TO_DWORD_EN	298
WORD_TO_int	298
WORD_TO_INT_EN	298
WORD_TO_REAL	298
WORD_TO_REAL_EN	298
WORD_TO_sint	298
WORD_TO_SINT_EN	203, 298
WORD_TO_STRING_EN	203, 298
WORD_TO_TIME_EN	298
WORD_TO_udint	298
WORD_TO_UDINT_EN	298
WORD_TO_uint	298
WORD_TO_UINT_EN	298
WORD_TO_usint	298
WORD_TO_USINT_EN	298
Working with Blocks	67, 99
Working with Networks	103
Working with watchlists	107
WSTRING	299

X

XML-Import/Export	23
XOR	299
XOR_BOOL_EN	299
XOR_BOOL_FBD	299
XOR_BYTE_EN	299
XOR_BYTE_FBD	299
XOR_DWORD_EN	299
XOR_DWORD_FBD	299
XOR_WORD_EN	299
XOR_WORD_FBD	299
XORN	299
XORN_BOOL_FBD	299

XORN_BYTE_FBD299
XORN_DWORD_FBD299

XORN_WORD_FBD 299



Contact

Supportline

Tel: +49 (0) 9131 / 78 00 - 99
Fax: +49 (0) 9131 / 78 00 - 50
supportline@infoteam.de

infoteam Software AG
Am Bauhof 9
D-91088 Bubenreuth

All used hard- and software names are trademarks and/or registered labels of the respective manufacturers.

© 2015,
infoteam Software AG.

Subject to change without prior notice.