PC WORX 6

IEC 61131-Programming

PC WORX 6

IEC 61131-Programming

Your trainer:

CCAX
Competence Center AUTOMATIONWORX

Any questions or suggestions?
Give us a call:
+49 (0) 52 81 – 9 46 – 21 01

Notes:

Software Architecture

Chapter 1

Notes:

Contents

This chapter offers an initial overview of the PC WORX interface.

As a container, PC WORX offers five freely configurable workspaces which display the dockable windows that can be shown/hidden as desired. With their preassignment, these five workspaces allow for the access to the main PC WORX functions (bus configuration, programming, process data linkage, project comparison and field device configuration).

ℹ The workspaces can be reset via the *Extras* → *Options* Menu, selecting the *General* tab.

💡 Even though the workspaces are freely configurable, you should organize them clearly, you should not overload them with shown windows and for efficient use, you should change between the workspaces according to the current application task.

⚠ Attention!

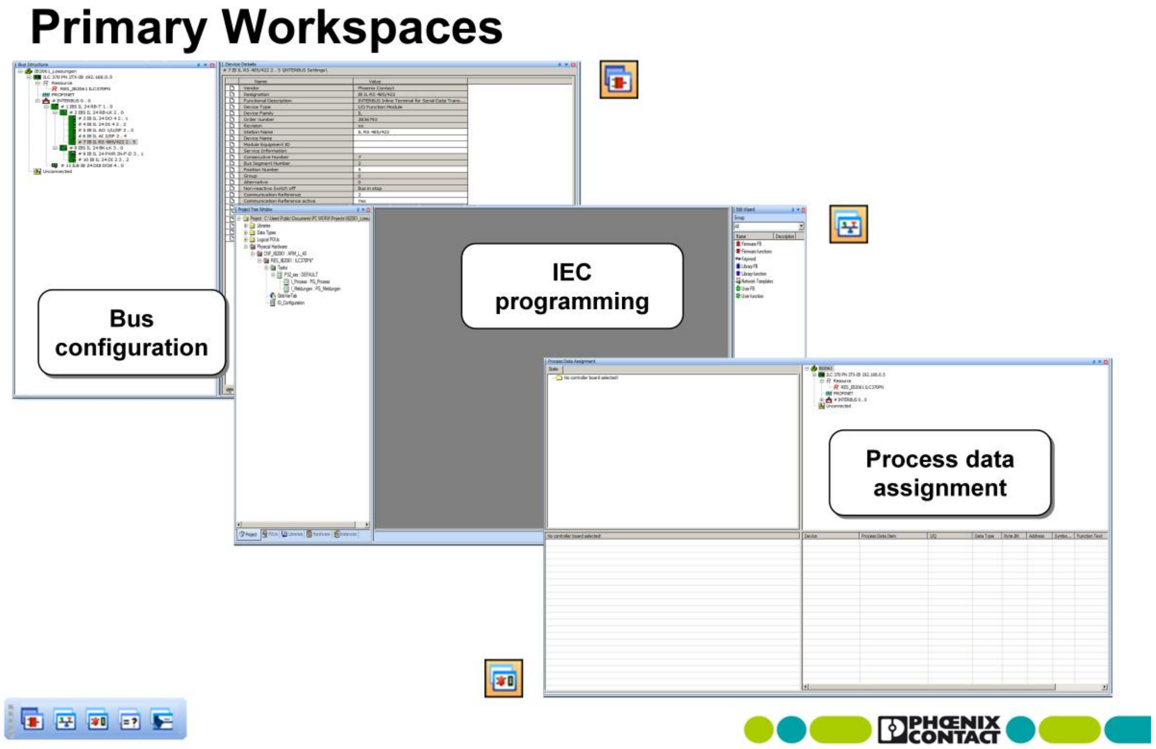
ℹ Information

💡 Tip

Notes:

Workspaces

Notes:



Primary Workspaces

Bus configuration

IEC programming

Process data assignment

The designations and windows that are shown as a standard are similar to the main functions of PC WORX. This applies to the three primary workspaces. The menu bar ensures fast switch-over between the different areas.

Bus configuration

Creation of the bus systems supported by the controller (INTERBUS and PROFINET), editing the devices used and general device data management.

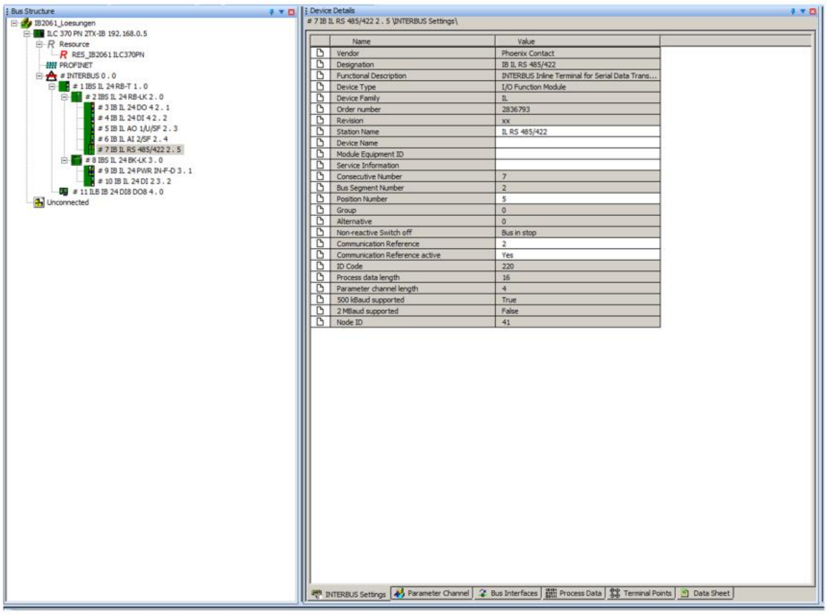
Process data assignment

Connecting the process data items made available by the bus modules to the global variables of the programming, or creating global variables based on above-mentioned process data objects according to syntax standard.

IEC programming

IEC-based programming (programs, function blocks and functions) for the creation of PLC hardware and tasks, declaring user-definable data types, integrating libraries and further software-related functions.

Bus Structure Workspace



7 IB IL RS 485/422 2 . 5 \INTERBUS Settings\

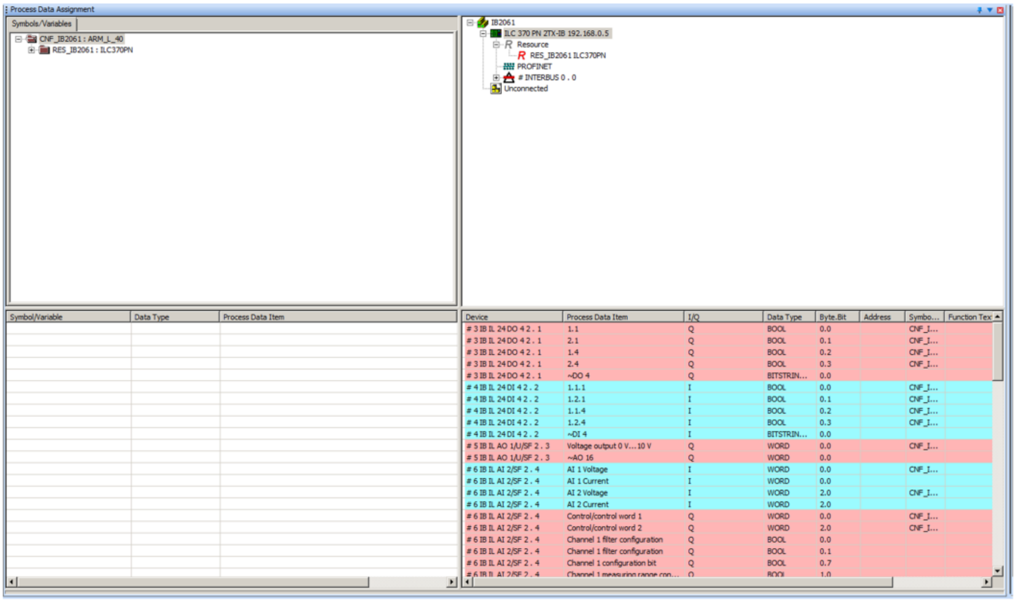
Name	Value
Vendor	Phoenix Contact
Designation	IB IL RS 485/422
Functional Description	INTERBUS Inline Terminal for Serial Data Trans...
Device Type	I/O Function Module
Device Family	IL
Order number	2836793
Revision	xx
Station Name	IL RS 485/422
Device Name	
Module Equipment ID	
Service Information	
Consecutive Number	7
Bus Segment Number	2
Position Number	5
Group	0
Alternative	0
Non-reactive Switch off	Bus in stop
Communication Reference	2
Communication Reference active	Yes
ID Code	220
Process data length	16
Parameter channel length	4
500 kBaud supported	True
2 MBaud supported	False
Node ID	41

As usually in a project first of all the hardware configuration is implemented, the *Bus configuration* workspace is the initial working environment for the user. As a standard, the windows *Bus configuration*, *Device details* and *Device catalog* are shown. Here, the latter has been consciously taken out of the view as it is only needed for the editing phase.

The device details are directly related to the bus configuration. The element selected in the bus configuration is displayed with its details in the device details. Depending on the element type (project, bus directory, bus device), the number and types of tabs are adjusted in this window. The information on which pages have to be displayed is read out of the device or system file. This explains the short time delay of the display. The text-based file is "translated" and displayed in a user-friendly way in the device details.

After configuration of the bus system, the complete documentation for the hardware can be entered in the device details (equipment identification, station name etc.).

Process Data Assignment Workspace



Device	Process Data Item	I/Q	Data Type	Byte.Bit	Address	Symbo...	Function Tex
# 3 IB IL 24 DO 4 2 . 1	1.1	Q	BOOL	0.0		CNF_I...	
# 3 IB IL 24 DO 4 2 . 1	2.1	Q	BOOL	0.1		CNF_I...	
# 3 IB IL 24 DO 4 2 . 1	1.4	Q	BOOL	0.2		CNF_I...	
# 3 IB IL 24 DO 4 2 . 1	2.4	Q	BOOL	0.3		CNF_I...	
# 3 IB IL 24 DO 4 2 . 1	~DO 4	Q	BITSTRIN...	0.0			
# 4 IB IL 24 DI 4 2 . 2	1.1.1	I	BOOL	0.0		CNF_I...	
# 4 IB IL 24 DI 4 2 . 2	1.2.1	I	BOOL	0.1		CNF_I...	
# 4 IB IL 24 DI 4 2 . 2	1.1.4	I	BOOL	0.2		CNF_I...	
# 4 IB IL 24 DI 4 2 . 2	1.2.4	I	BOOL	0.3		CNF_I...	
# 4 IB IL 24 DI 4 2 . 2	~DI 4	I	BITSTRIN...	0.0			
# 5 IB IL AO 1/U/SF 2 . 3	Voltage output 0 V... 10 V	Q	WORD	0.0		CNF_I...	
# 5 IB IL AO 1/U/SF 2 . 3	~AO 16	Q	WORD	0.0			
# 6 IB IL AI 2/SF 2 . 4	AI 1 Voltage	I	WORD	0.0		CNF_I...	
# 6 IB IL AI 2/SF 2 . 4	AI 1 Current	I	WORD	0.0			
# 6 IB IL AI 2/SF 2 . 4	AI 2 Voltage	I	WORD	2.0		CNF_I...	
# 6 IB IL AI 2/SF 2 . 4	AI 2 Current	I	WORD	2.0			
# 6 IB IL AI 2/SF 2 . 4	Control/control word 1	Q	WORD	0.0		CNF_I...	
# 6 IB IL AI 2/SF 2 . 4	Control/control word 2	Q	WORD	2.0		CNF_I...	
# 6 IB IL AI 2/SF 2 . 4	Channel 1 filter configuration	Q	BOOL	0.0			
# 6 IB IL AI 2/SF 2 . 4	Channel 1 filter configuration	Q	BOOL	0.1			
# 6 IB IL AI 2/SF 2 . 4	Channel 1 configuration bit	Q	BOOL	0.7			
# 6 IB IL AI 2/SF 2 . 4	Channel 1 measuring range con...	Q	BOOL	1.0			

The process data assignment workspace is divided into four quadrants. The left-hand side shows the PLC and its structure. Depending on the selected element, the corresponding global variables are displayed in the table below.

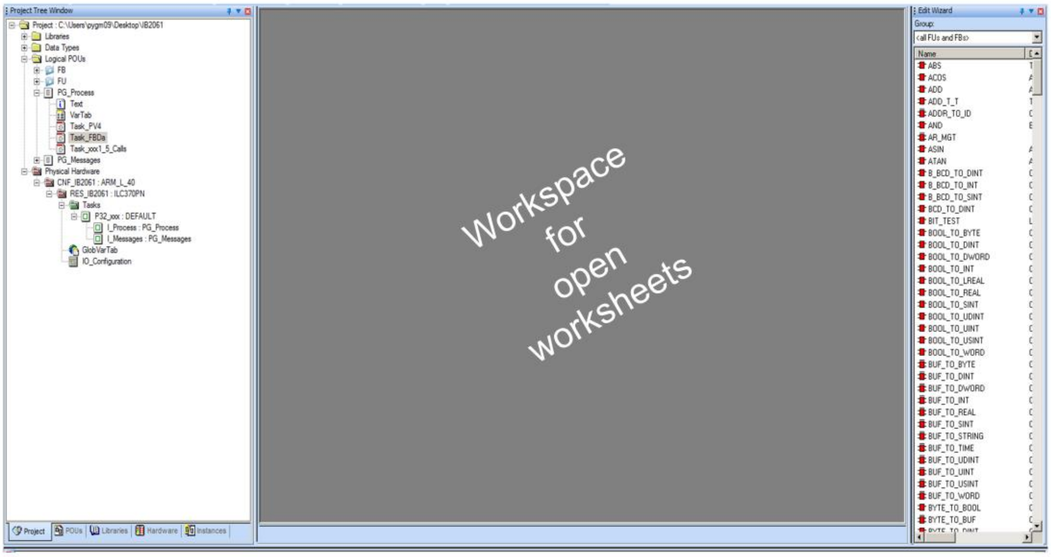
The right-hand side shows the bus system hardware. Depending on the selected element, here the corresponding process data items are displayed:

- For single devices: only the objects of the selected device
- For bus terminals: all process data items available in the respective branch
- For control systems: all objects dependent on this control system

In this window, the programming variables are connected to the process data object items in the connected bus systems.

Notes:

IEC Programming Workspace



The workspace for the IEC-based programming of a PLC is divided into the project tree and the edit wizard.

Besides a complete display of the project, the project tree offers specific views for the program organization units (POU), libraries, hardware and tasks as well as the function call structure of the programs (instances).

The edit wizard is context-sensitive. This means that, depending on the element worked on, the wizard offers help for the creation of user-defined data types or the creation of a program.

The gray workspace in the screenshot is used for the open worksheets (code worksheets and variable tables). Just as in many other applications, these windows can be arranged in different styles (see *Window* menu).

For maximizing the worksheet, it makes sense to hide the project tree, the edit wizard and the message window that automatically pops up during compilation processes. The pre-defined shortcuts are:

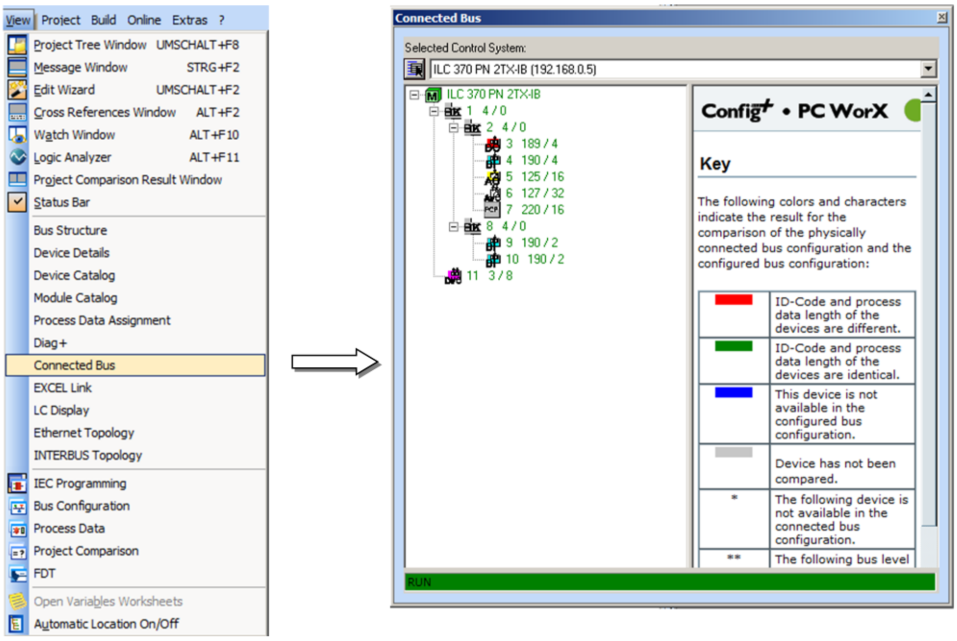
Shift+F8	Show/hide project tree
Ctrl+F2	Show/hide message window
Shift+F2	Show/hide edit wizard

Shortcuts can be adjusted via the *Tools* → *Shortcuts* menu.

Notes:

Windows (selection)

Connected Bus



Key

The following colors and characters indicate the result for the comparison of the physically connected bus configuration and the configured bus configuration:

🟥	ID-Code and process data length of the devices are different.
🟩	ID-Code and process data length of the devices are identical.
🟦	This device is not available in the configured bus configuration.
⬜	Device has not been compared.
*	The following device is not available in the connected bus configuration.
**	The following bus level

The user usually only activates the *Connected Bus* window for the read-in time of the connected INTERBUS system. The communication path of this window is taken on from the central settings and can be called via the *Selected Control System* list.

Legend

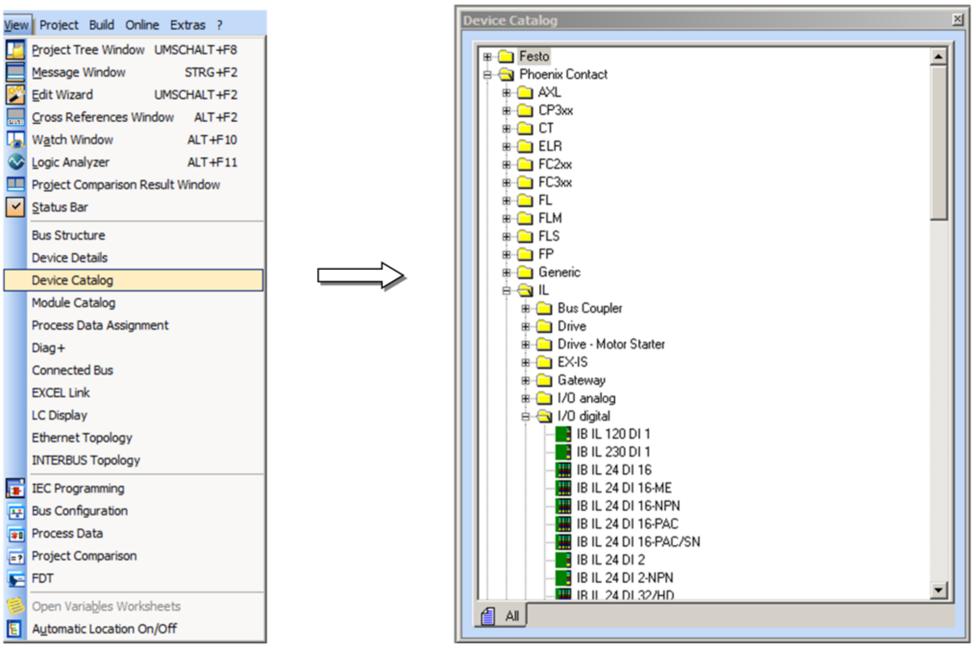
RED	ID code and process data length of the devices are different.
GREEN	ID code and process data length of the devices are identical.
BLUE	This device is not available in the configured bus configuration.
GRAY	Device has not been compared.

***** The following device is not available in the current bus configuration.

***** The following bus level is not available in the current bus configuration.

X / Y	ID code and process data length

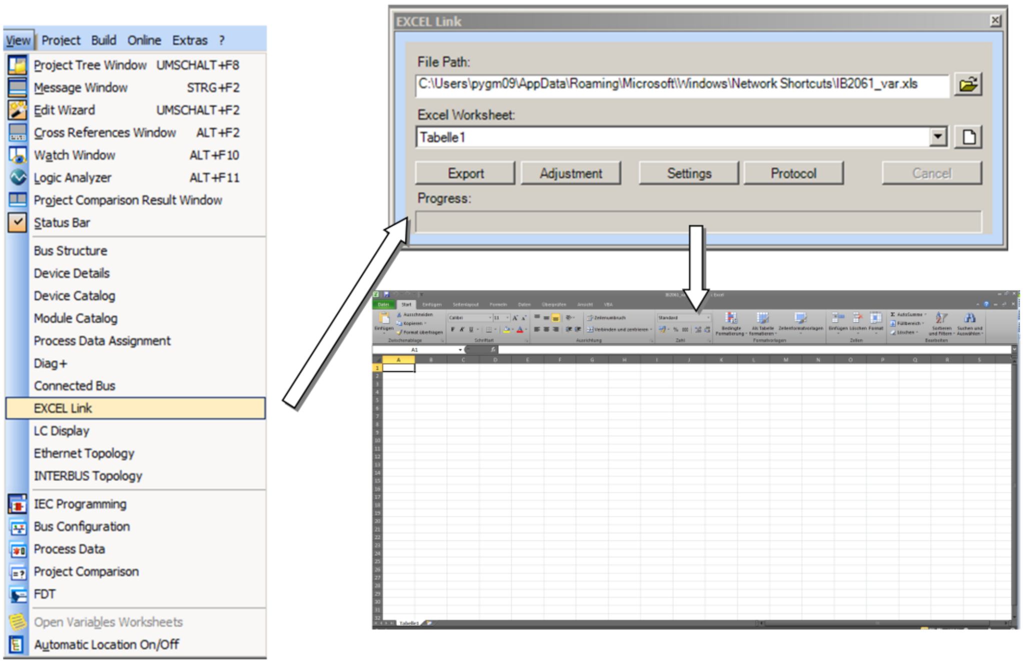
Device Catalog



The *Device Catalog* window is required for offline configuration of the bus systems and only has to be shown during this operating phase. It is also required for corrections of devices incorrectly selected during online configuration.

In the device catalog, all device description files are listed on the *All* tab. Via the context menu, the user can adjust the tree structure.

Excel Link

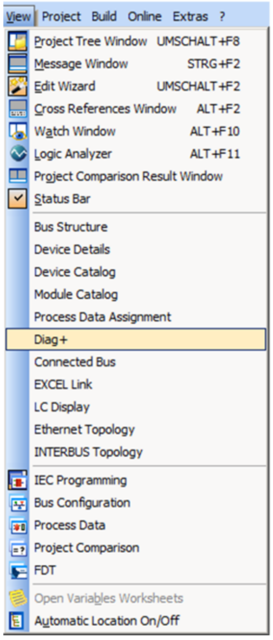


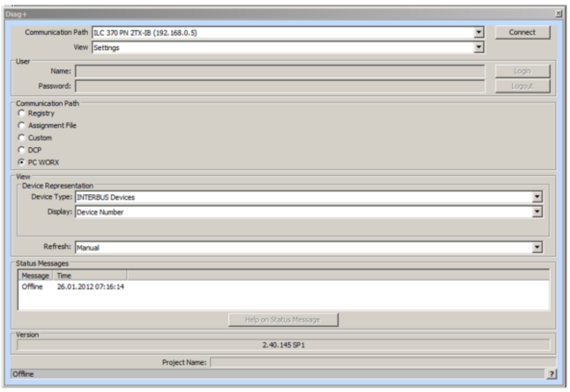
File Path: C:\Users\pygm09\AppData\Roaming\Microsoft\Windows\Network Shortcuts\IB2061_var.xls

Excel Worksheet: Tabelle1

As an interface, the Excel link integrated in PC WORX (compatible with MS Excel) enables the organized export and import of global variables linked to process data items.

Diag+ 2.0





Communication Path: ILC 370 PN 2TX-IB (192.168.0.5)
View: Settings

Name:
Password:

Communication Path
○ Registry
○ Assignment File
○ Custom
○ DCP
● PC WORX

View
Device Representation
Device Type: INTERBUS Devices
Display: Device Number

Refresh: Manual

Status Messages

Message	Time
Offline	26.01.2012 07:16:14

Version
2.40.145 SP1

Project Name:

Offline

The Diag+ 2.0 software integrated in PC WORX 6 offers extensive diagnostic function for both INTERBUS and PROFINET, which are the bus systems supported by the control systems. Like many other windows, Diag+ uses the centrally set up communication path.

Notes:

Hardware Architecture

Chapter 2

Notes:

Contents

This chapter offers information on the functions of the ProConOS operating system, the implementation on different hardware platforms and on the management of hardware resources with ProConOS.

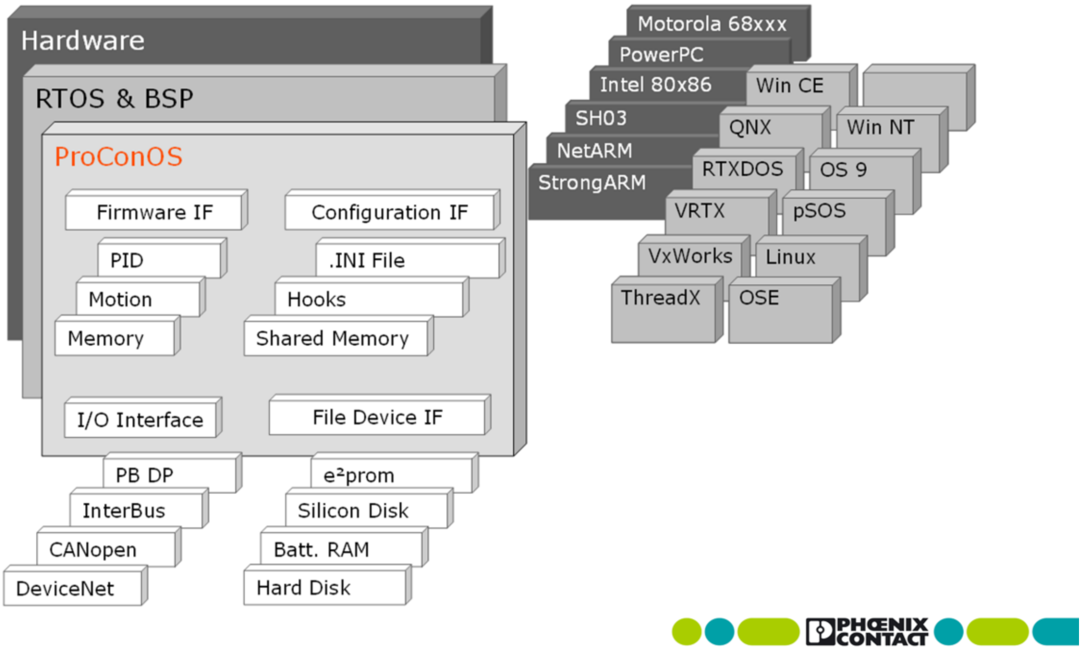
⚠ Attention!

ℹ Information

💡 Tip

Notes:

Overview

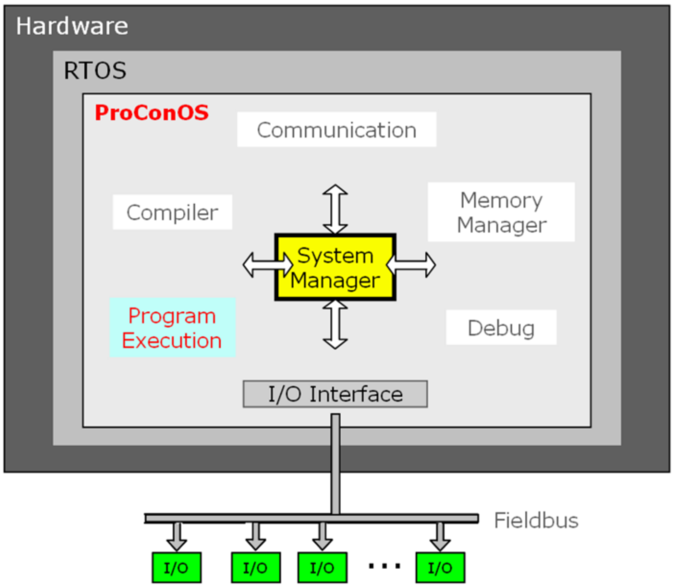


ProConOS (**Pro**grammable **Con**troller **O**perating **S**ystem) is required for the provision of PLC-typical system services on specific or standard hardware platforms. This includes loading and processing of externally created PLC programs as well as the provision of debug functions for programming, installation and maintenance of PLC-controlled machines and systems.

ProConOS is used as an operating system for most Phoenix Contact control systems. This offers the advantage that all control systems can be parameterized and programmed in the same development environment (PC WORX).

Notes:

Multitasking



ProConOS is based on standard multitasking operating systems that are controlled by means of task priorities. In ProConOS there are three different priority levels:

- Priority level for user tasks
- Priority level for ProConOS tasks
- Priority level for system manager tasks

This classification ensures that the processor computing time entirely available to the PLC is always distributed in favor of application-relevant tasks. Only if their timing allows it, the other system tasks receive their demanded computing time. This is the ProConOS-internal prerequisite for the temporal determinism of the application. The use of multitasking in the ProConOS system architecture thus aims at calculable time response, optimization of performance, i.e. minimizing response times, and specific reaction to runtime errors.

The user tasks include cyclic tasks, event tasks and DEFAULT tasks. Within a determined time interval, cyclic tasks are executed cyclically according to the priority defined by the user. Furthermore, a reaction to acyclically occurring events, e.g. hardware interrupt, can be reacted to in the event tasks.

The DEFAULT task is the task with the lowest priority and is executed when no other task is active. The DEFAULT task is a cyclic task.

During the adaptation of ProConOS to the different Phoenix Contact control platforms, the different interfaces were used to implement control-specific features.

Notes:

Communication PC Control System

Chapter 3

Notes:

Contents

This chapter informs the reader on the communication between the programming device and control system. The different communication paths that are available according to the control system type used are described. For networked-based communication, the transmission of IP addresses to control systems are presented, as is the use of BootP services.

⚠ Attention!

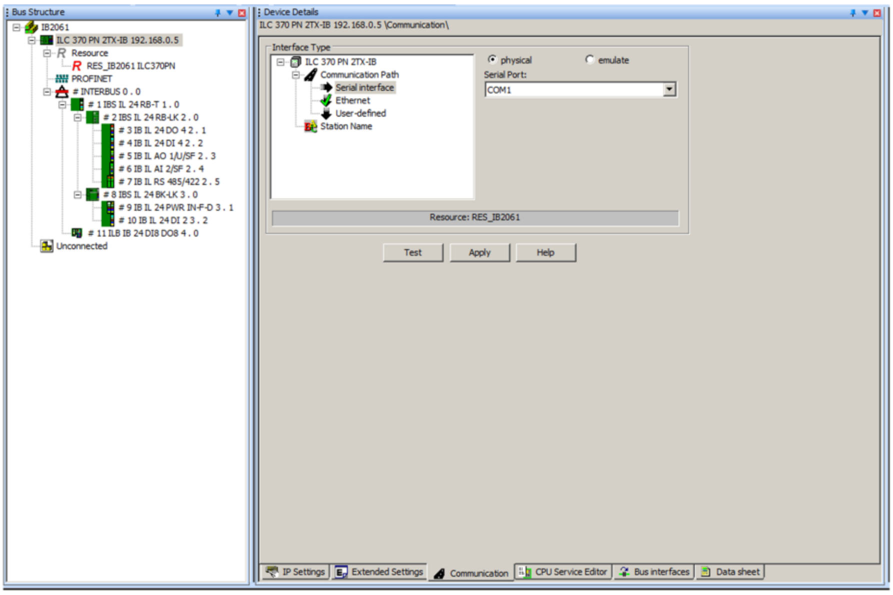
ℹ Information

💡 Tip

Notes:

Communication Paths

Serial Interface



```
IB2061
  ILC 370 PN 2TX-IB 192.168.0.5
    Resource
      RES_IB2061 ILC370PN
    PROFINET
    # INTERBUS 0 . 0
      # 1 IBS IL 24 RB-T 1 . 0
        # 2 IBS IL 24 RB-LK 2 . 0
          # 3 IB IL 24 DO 4 2 . 1
          # 4 IB IL 24 DI 4 2 . 2
          # 5 IB IL AO 1/U/SF 2 . 3
          # 6 IB IL AI 2/SF 2 . 4
          # 7 IB IL RS 485/422 2 . 5
        # 8 IBS IL 24 BK-LK 3 . 0
          # 9 IB IL 24 PWR IN-F-D 3 . 1
          # 10 IB IL 24 DI 2 3 . 2
      # 11 ILB IB 24 DI8 DO8 4 . 0
  Unconnected
```

ILC 370 PN 2TX-IB 192.168.0.5 \Communication\

Interface Type

```
ILC 370 PN 2TX-IB
  Communication Path
    Serial interface
    Ethernet
    User-defined
  Station Name
```

● physical ○ emulate

Serial Port: COM1

Resource: RES_IB2061

The communication path for the connection of the programming device to the control system is selected on the *Communication* tab when the element *CONTROL SYSTEM* is selected. Depending on the control system type, different media are available.

Every control system provides a serial interface for which a standard IBS connecting cable or the connecting cable with Mini-DIN connector can be used.

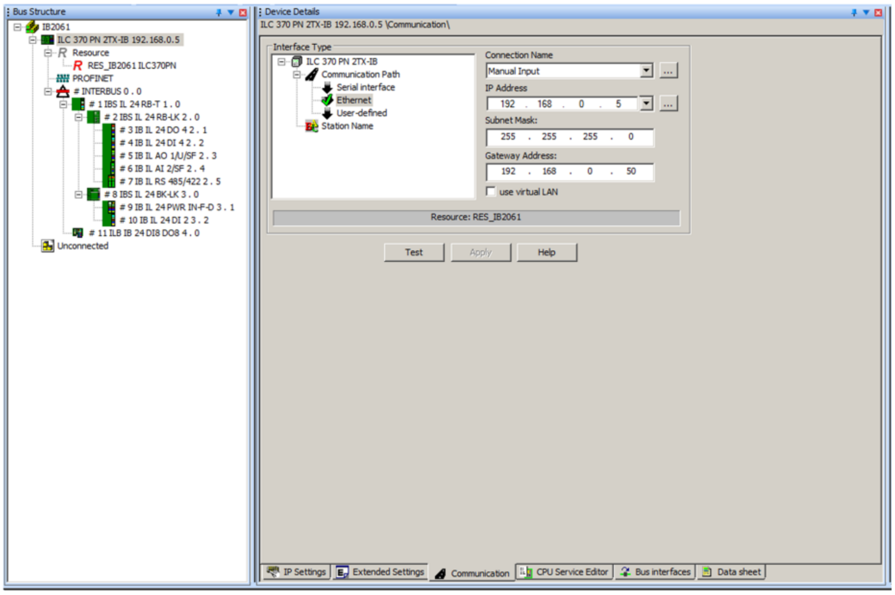
In case the programming device does not provide a serial interface and a USB adapter has to be used, it might be necessary to select the emulated interface.

Pressing the *Test* button activates a connection establishment which is displayed in the status line.

Via the *Apply* button you can specify the communication path for the current control system in the currently worked-on project. Using these settings, other windows and dialogs can easily access the control system all the time.

When using previously configured control systems, the message *Connection names do not match* may pop up because a connection name for the connection is saved to the control system. This is to prevent an accidental access to a control system. At that point, an acknowledgement is not necessary.

TCP/IP



Bus Structure

- IB2061
  - ILC 370 PN 2TX-IB 192.168.0.5
    - Resource
      - RES_IB2061 ILC370PN
    - PROFINET
    - # INTERBUS 0 . 0
      - # 1 IBS IL 24 RB-T 1 . 0
        - # 2 IBS IL 24 RB-LK 2 . 0
          - # 3 IB IL 24 DO 4 2 . 1
          - # 4 IB IL 24 DI 4 2 . 2
          - # 5 IB IL AO 1/U/SF 2 . 3
          - # 6 IB IL AI 2/SF 2 . 4
          - # 7 IB IL RS 485/422 2 . 5
        - # 8 IBS IL 24 BK-LK 3 . 0
          - # 9 IB IL 24 PWR IN-F-D 3 . 1
          - # 10 IB IL 24 DI 2 3 . 2
      - # 11 ILB IB 24 DI8 DO8 4 . 0
  - Unconnected

Device Details

ILC 370 PN 2TX-IB 192.168.0.5 \Communication\

Interface Type

- ILC 370 PN 2TX-IB
  - Communication Path
    - Serial interface
    - ✓ Ethernet
    - User-defined
  - Station Name

Connection Name: Manual Input

IP Address: 192 . 168 . 0 . 5

Subnet Mask: 255 . 255 . 255 . 0

Gateway Address: 192 . 168 . 0 . 50

☐ use virtual LAN

Resource: RES_IB2061

In addition to the serial interface, most control systems provide the possibility to communicate via TCP/IP.

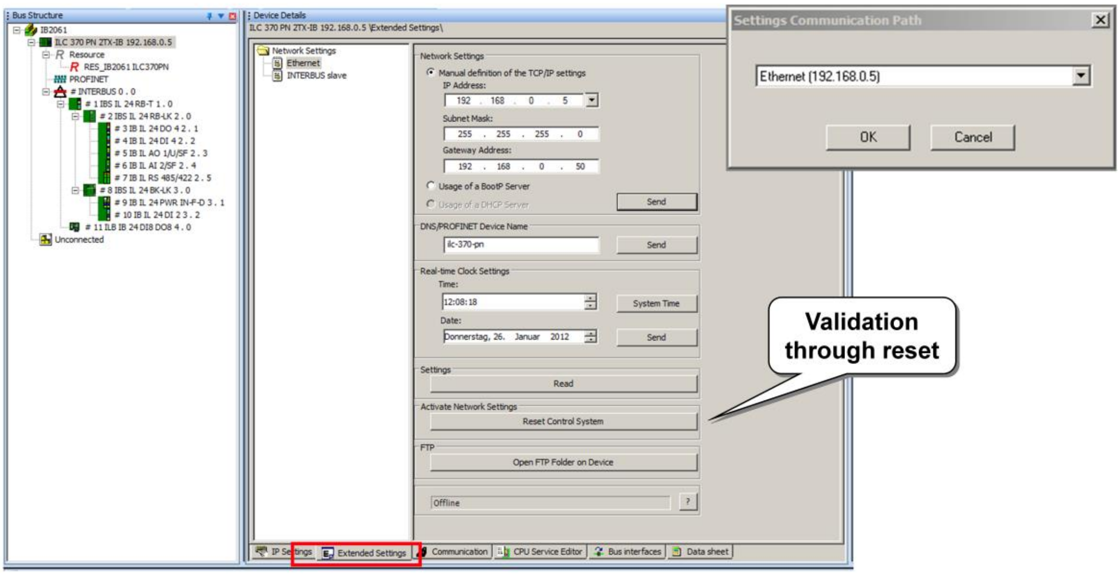
For this type of communication, the IP address must be saved to the control system.
Furthermore, this address must be communicated to PC WORX.

Notes:

Control System Network Settings

Notes:

Setting the IP Address • Real-Time Clock



Bus Structure

```
IB2061
  ILC 370 PN 2TX-IB 192.168.0.5
    Resource
      RES_IB2061 ILC370PN
    PROFINET
    # INTERBUS 0 . 0
      # 1 IBS IL 24 RB-T 1 . 0
        # 2 IBS IL 24 RB-LK 2 . 0
          # 3 IB IL 24 DO 4 2 . 1
          # 4 IB IL 24 DI 4 2 . 2
          # 5 IB IL AO 1/U/SF 2 . 3
          # 6 IB IL AI 2/SF 2 . 4
          # 7 IB IL RS 485/422 2 . 5
        # 8 IBS IL 24 BK-LK 3 . 0
          # 9 IB IL 24 PWR IN-F-D 3 . 1
          # 10 IB IL 24 DI 2 3 . 2
      # 11 ILB IB 24 DI8 DO8 4 . 0
  Unconnected
```

Device Details

ILC 370 PN 2TX-IB 192.168.0.5 \Extended Settings\

Network Settings
- Ethernet
- INTERBUS slave

Network Settings
- ● Manual definition of the TCP/IP settings
  - IP Address: 192 . 168 . 0 . 5
  - Subnet Mask: 255 . 255 . 255 . 0
  - Gateway Address: 192 . 168 . 0 . 50
- ○ Usage of a BootP Server
- ○ Usage of a DHCP Server

DNS/PROFINET Device Name: ilc-370-pn

Real-time Clock Settings
- Time: 12:08:18
- Date: Donnerstag, 26. Januar 2012

Status: Offline

Settings Communication Path: Ethernet (192.168.0.5)

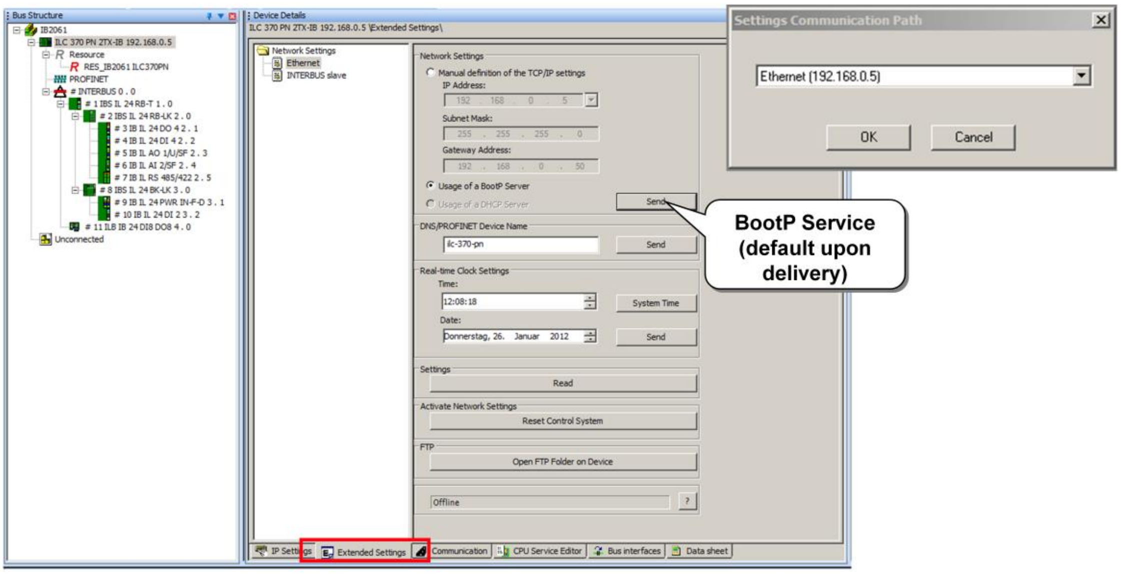
Validation through reset

With control systems with real-time clock, the IP address and the time are set on the *Extended Settings* tab after the element *Control System* has been selected. In the *Network Settings* frame, the network configuration of the control system is displayed (default upon delivery: *Usage of BootP server*) after the *Read* function has been selected. For this, you need to select a communication path ready for operation. Usually, this is the serial interface.

The communication is set up according to the project requirements. If necessary, adjust the real-time clock of the control system by calling and sending the *System Time*. The validation of the changes made is implemented by resetting the control system.

As the control systems have an FTP server, with a successfully configured network communication, the software for FTP access that is installed under the operating system can be started via the *Open FTP Folder on Device* button. Thus, the content of the FTP area of the CompactFlash card is displayed.

Assigning the IP Address via BootP



Bus Structure

IB2061
- ILC 370 PN 2TX-IB 192.168.0.5
  - Resource
    - RES_IB2061 ILC370PN
  - PROFINET
  - # INTERBUS 0 . 0
    - # 1 IBS IL 24 RB-T 1 . 0
      - # 2 IBS IL 24 RB-LK 2 . 0
        - # 3 IB IL 24 DO 4 2 . 1
        - # 4 IB IL 24 DI 4 2 . 2
        - # 5 IB IL AO 1/U/SF 2 . 3
        - # 6 IB IL AI 2/SF 2 . 4
        - # 7 IB IL RS 485/422 2 . 5
      - # 8 IBS IL 24 BK-LK 3 . 0
        - # 9 IB IL 24 PWR IN-F-D 3 . 1
        - # 10 IB IL 24 DI 2 3 . 2
    - # 11 ILB IB 24 DI8 DO8 4 . 0
- Unconnected

Device Details

ILC 370 PN 2TX-IB 192.168.0.5 \Extended Settings\

Network Settings
- Ethernet
- INTERBUS slave

Network Settings

○ Manual definition of the TCP/IP settings

IP Address: 192 . 168 . 0 . 5

Subnet Mask: 255 . 255 . 255 . 0

Gateway Address: 192 . 168 . 0 . 50

● Usage of a BootP Server

○ Usage of a DHCP Server

DNS/PROFINET Device Name: ilc-370-pn

Real-time Clock Settings

Time: 12:08:18

Date: Donnerstag, 26. Januar 2012

Offline

Settings Communication Path

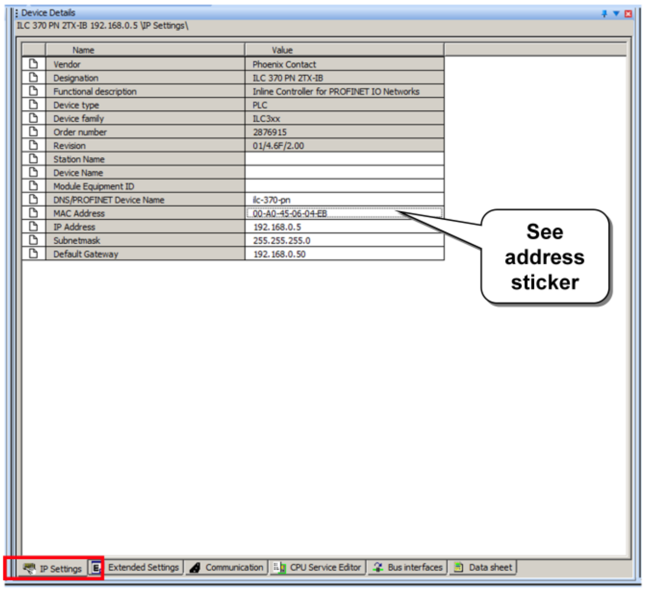
Ethernet (192.168.0.5)

BootP Service (default upon delivery)

If after startup of a BootP server (e.g. PC WORX), the control system is to receive its IP address, the option *Usage of BootP server* is to be selected in the extended settings. As with the transmission of a new static IP address, this configuration has to be sent and then validated by resetting the controller.

When using the BootP option, it is not sufficient to just change the CF card when a control system is replaced. The new MAC address has to be communicated to the BootP server.

MAC Address • Activating the BootP Server

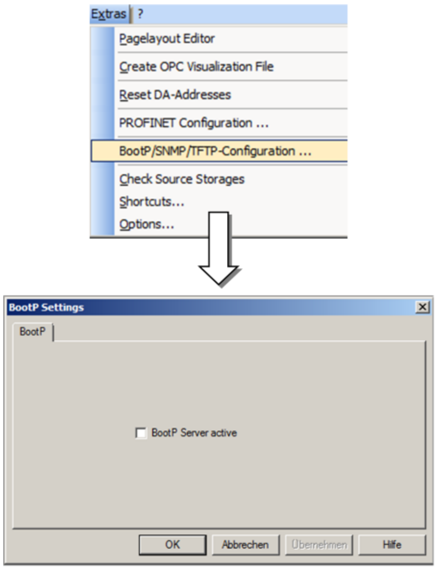


Device Details

ILC 370 PN 2TX-IB 192.168.0.5 \IP Settings\

Name	Value
Vendor	Phoenix Contact
Designation	ILC 370 PN 2TX-IB
Functional description	Inline Controller for PROFINET IO Networks
Device type	PLC
Device family	ILC3xx
Order number	2876915
Revision	01/4.6F/2.00
Station Name	
Device Name	
Module Equipment ID	
DNS/PROFINET Device Name	ilc-370-pn
MAC Address	00-A0-45-06-04-EB
IP Address	192.168.0.5
Subnetmask	255.255.255.0
Default Gateway	192.168.0.50

See address sticker



□ BootP Server active

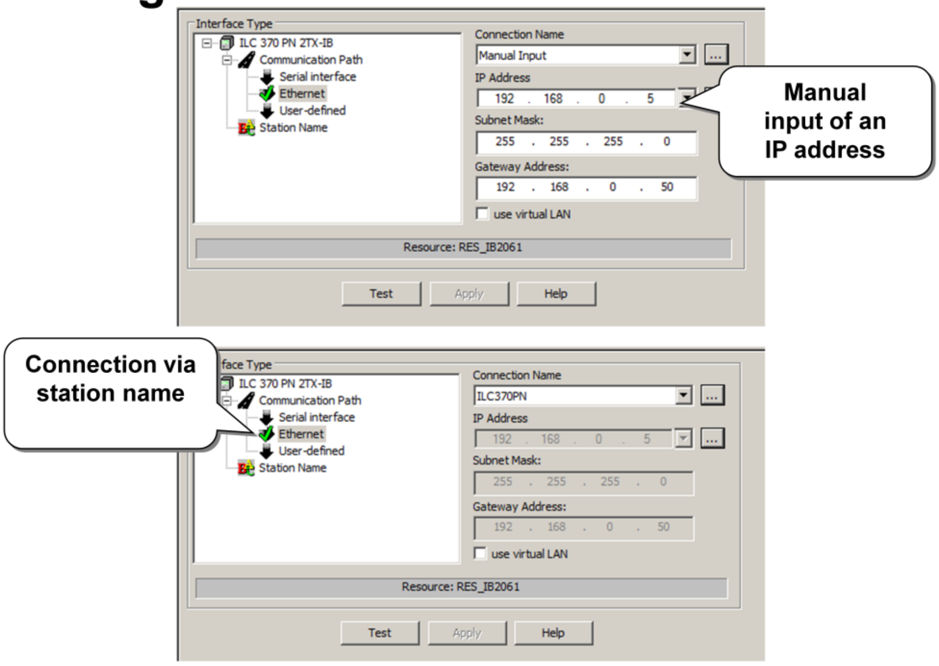
In order to use the BootP option for a control system by means of the BootP server integrated in PC WORX, it might be necessary to activate the BootP server via the menu *Tools* and → *BootP/SNMP/TFTP-Configuration*. For the assignment to the control system, the MAC address for the control system has to be entered in the IP Settings.

You can find the MAC address on the combined hexadecimal/barcode sticker on the front or left side of the housing.

Notes:

Network Communication

Selecting the Network Connection



Connection Name: Manual Input
IP Address: 192 . 168 . 0 . 5
Subnet Mask: 255 . 255 . 255 . 0
Gateway Address: 192 . 168 . 0 . 50
□ use virtual LAN
Resource: RES_IB2061

Manual input of an IP address

Connection Name: ILC370PN
IP Address: 192 . 168 . 0 . 5
Subnet Mask: 255 . 255 . 255 . 0
Gateway Address: 192 . 168 . 0 . 50
□ use virtual LAN
Resource: RES_IB2061

Connection via station name

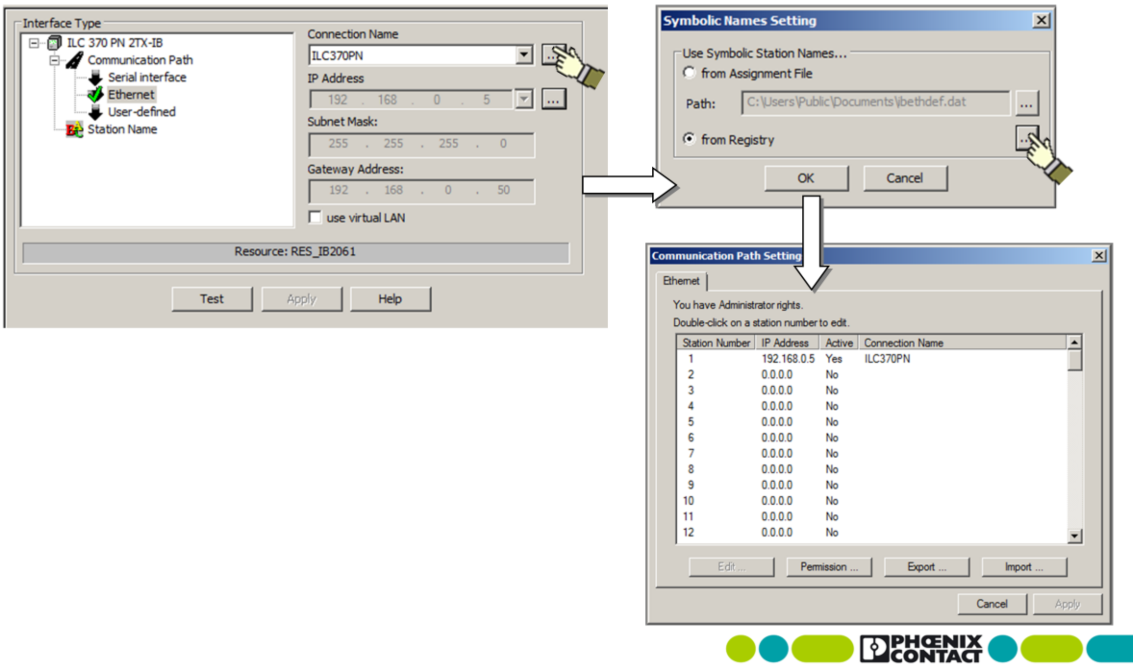
If the controller has been configured with a valid IP address, this address has to be made available to PC WORX.

Having selected the *TCP/IP (Ethernet/Local host)* communication path, there are two possibilities to do so:

1. Select the element *Manual Input* and enter the desired IP address in the enabled fields.

2. Select a station from the list of connection names. Station names are connected to IP addresses via an address file or an entry in the registry.

Editing the Station Names



Interface Type

- ILC 370 PN 2TX-IB
  - Communication Path
    - Serial interface
    - Ethernet
    - User-defined
  - Station Name

Connection Name: ILC370PN

IP Address: 192 . 168 . 0 . 5

Subnet Mask: 255 . 255 . 255 . 0

Gateway Address: 192 . 168 . 0 . 50

□ use virtual LAN

Resource: RES_IB2061

Symbolic Names Setting

Use Symbolic Station Names...

○ from Assignment File

Path: C:\Users\Public\Documents\bethdef.dat

● from Registry

Communication Path Setting

Ethernet

You have Administrator rights.
Double-click on a station number to edit.

Station Number	IP Address	Active	Connection Name
1	192.168.0.5	Yes	ILC370PN
2	0.0.0.0	No	
3	0.0.0.0	No	
4	0.0.0.0	No	
5	0.0.0.0	No	
6	0.0.0.0	No	
7	0.0.0.0	No	
8	0.0.0.0	No	
9	0.0.0.0	No	
10	0.0.0.0	No	
11	0.0.0.0	No	
12	0.0.0.0	No	

Via the *... button*, you can load an address file or change the station name entries.

The easiest way to create an address file is to change entries in the registry and then export the list. Using a text editing tool, the file (*.dat) created can then be edited as a tabulator-divided text file.

Entries in this address file or the registry are only displayed in the *Connection Name* field if they have been activated.

Notes:

INTERBUS Configuration

Chapter 4

Notes:

Contents

This chapter provides an overview of the configuration of an INTERBUS system connected to the control system.

In case that a system that is ready to operate is connected to the control system, the online configuration can be used. For adaptations and the configuration before startup of the hardware, the offline configuration is described.

This Chapter also describes the creation of user-defined device catalogs (especially interesting for mass production) and the import of device description files.

ℹ For both configuration methods, the device description files for the devices used in the project are required. A wide range is already provided on the PC due to the installation of PC WORX. Further device description files (XML format) can be registered later after copying the respective directories via the *Device Catalog* window.

For online configuration, a ready-to-operate communication between PC and control system (see Chapter *Communication PC - Control System*) is required as well.

⚠ Attention!

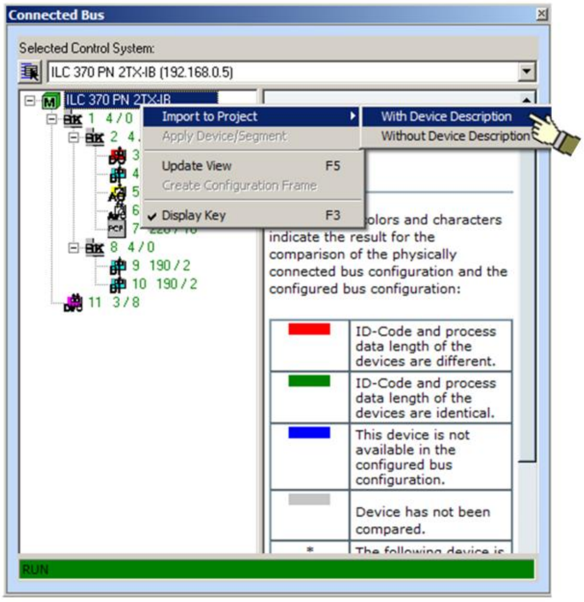
ℹ Information

💡 Tip

Notes:

Online Configuration

Read-In of the Connected Bus System



…olors and characters indicate the result for the comparison of the physically connected bus configuration and the configured bus configuration:

🟥	ID-Code and process data length of the devices are different.
🟩	ID-Code and process data length of the devices are identical.
🟦	This device is not available in the configured bus configuration.
⬜	Device has not been compared.
*	The following device is

For online configuration, the *View → Connected Bus* menu is displayed. This window is not embedded in the combination of the already opened windows, but remains undocked. In order to keep the currently used workspace (usually the *Bus Configuration* workspace) clearly arranged, this window should be switched offline and hidden again after the read-in.

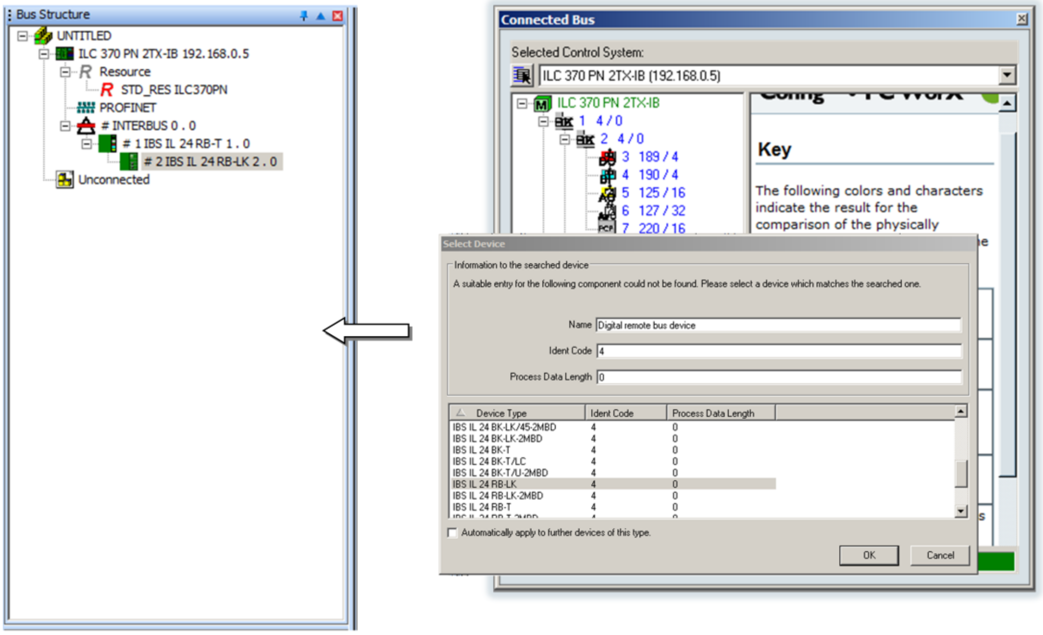
For reading in the connected INTERBUS system, you have to select the control system currently worked on from the *Selected Control System* list. PC WORX will then display the configuration frame of the INTERBUS saved to the control system or, in case a system in the *Ready* state is used, create a new configuration frame.

If the system is not ready to operate or faulty, then this window cannot display the configuration frame. For diagnostics, the *Diag+* window (operation see Chapter *Diag+ 2.0 in PC WORX 5*) can be used.

If the bus system is displayed, it can be copied from the context menu into the *Bus Configuration* window via the function *Import to Project → With Device Description*.

⚠ Please consider that when you select this function, an already configured bus system is discarded. If just individual devices from the bus system are to be applied, (e.g. as an addition if configuration has not been completed), then the function *apply Device/Segment* is to be used.

Inserting the Detected Devices



The following colors and characters indicate the result for the comparison of the physically

Select Device

Information to the searched device

A suitable entry for the following component could not be found. Please select a device which matches the searched one.

Name: Digital remote bus device

Ident Code: 4

Process Data Length: 0

Device Type	Ident Code	Process Data Length
IBS IL 24 BK-LK/45-2MBD	4	0
IBS IL 24 BK-LK-2MBD	4	0
IBS IL 24 BK-T	4	0
IBS IL 24 BK-T/LC	4	0
IBS IL 24 BK-T/U-2MBD	4	0
IBS IL 24 RB-LK	4	0
IBS IL 24 RB-LK-2MBD	4	0
IBS IL 24 RB-T	4	0

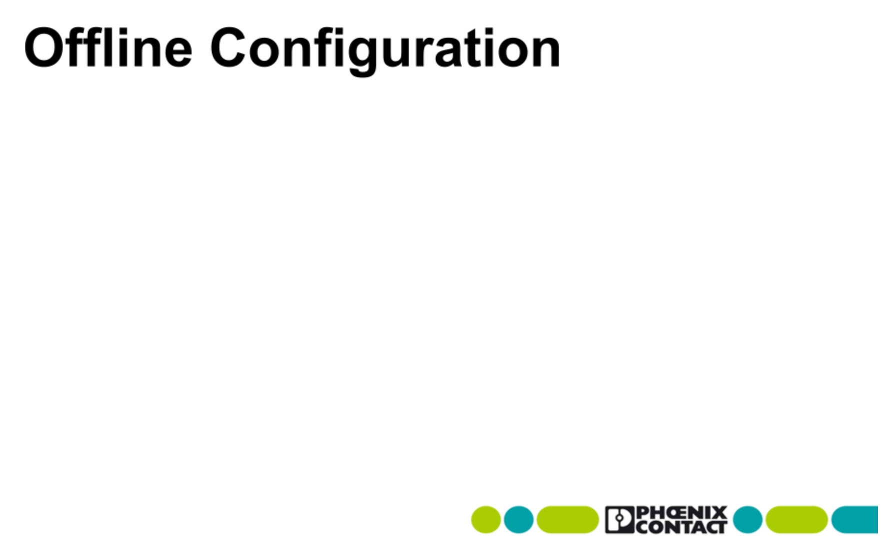
☐ Automatically apply to further devices of this type.

Depending on the device currently worked on, a list of the device description files registered in PC WORX is displayed in the *Select Device* dialog. Furthermore, the selection is also specified by means of the combination of identification code and process data length of the current device.

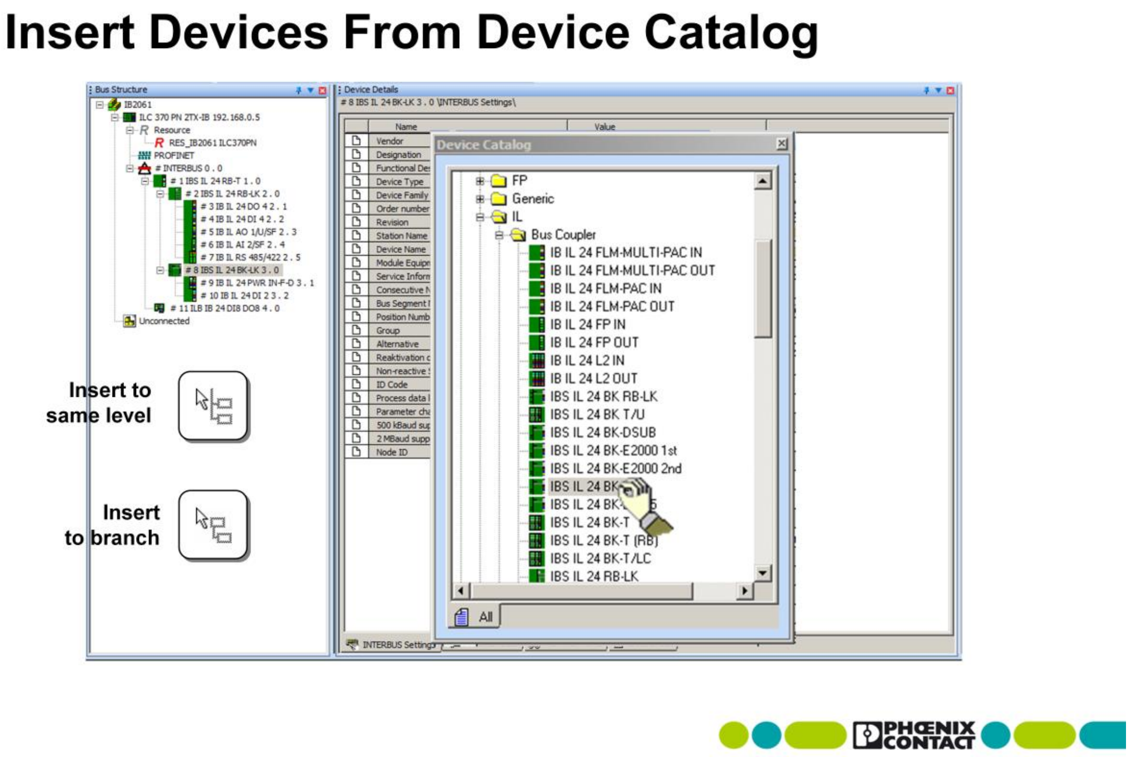
In case user-defined device catalogs have been created in PC WORX, before selecting a device you have to select the catalog, which the function shall access during its search.

If the selection was faulty, the selection process should not be aborted, but continued with. In most cases, a subsequent correction via the device catalog takes less time than the repeated read-in of the system.

Notes:



Offline Configuration



Insert Devices From Device Catalog

Bus Structure

- IB2061
  - ILC 370 PN 2TX-IB 192.168.0.5
    - Resource
      - RES_IB2061 ILC370PN
    - PROFINET
    - # INTERBUS 0 . 0
      - # 1 IBS IL 24 RB-T 1 . 0
        - # 2 IBS IL 24 RB-LK 2 . 0
          - # 3 IB IL 24 DO 4 2 . 1
          - # 4 IB IL 24 DI 4 2 . 2
          - # 5 IB IL AO 1/U/SF 2 . 3
          - # 6 IB IL AI 2/SF 2 . 4
          - # 7 IB IL RS 485/422 2 . 5
        - # 8 IBS IL 24 BK-LK 3 . 0
          - # 9 IB IL 24 PWR IN-F-D 3 . 1
          - # 10 IB IL 24 DI 2 3 . 2
      - # 11 ILB IB 24 DI8 DO8 4 . 0
  - Unconnected

Device Details

8 IBS IL 24 BK-LK 3 . 0 \INTERBUS Settings\

Device Catalog

- FP
- Generic
- IL
  - Bus Coupler
    - IB IL 24 FLM-MULTI-PAC IN
    - IB IL 24 FLM-MULTI-PAC OUT
    - IB IL 24 FLM-PAC IN
    - IB IL 24 FLM-PAC OUT
    - IB IL 24 FP IN
    - IB IL 24 FP OUT
    - IB IL 24 L2 IN
    - IB IL 24 L2 OUT
    - IBS IL 24 BK RB-LK
    - IBS IL 24 BK T/U
    - IBS IL 24 BK-DSUB
    - IBS IL 24 BK-E2000 1st
    - IBS IL 24 BK-E2000 2nd
    - IBS IL 24 BK
    - IBS IL 24 BK-
    - IBS IL 24 BK-T
    - IBS IL 24 BK-T (RB)
    - IBS IL 24 BK-T/LC
    - IBS IL 24 RB-LK

For offline configuration, the *Device Catalog* window from the views menu has to be activated. It is advisable to position this window between the *Bus configuration* and the *Device Details* window. Thus, the latter is partly forced out of the visible area, but, just as the *Connected Bus* window, the device catalog will only be shown until it is no longer needed. The catalog function regarding the bus configuration is limited to three actions:

1. Insert device to same level (remote bus device behind remote bus device or local bus device behind local bus device).

This can be done via the context menu (in the catalog *Copy device*, in the bus configuration *Insert to same level*) or using the mouse. For this, the device has to be selected in the catalog and (mouse-button pressed down) drawn to the device in the bus configuration, after which you would like to insert the new device. The mouse pointer displays below symbol.

Symbol when inserting to same level:

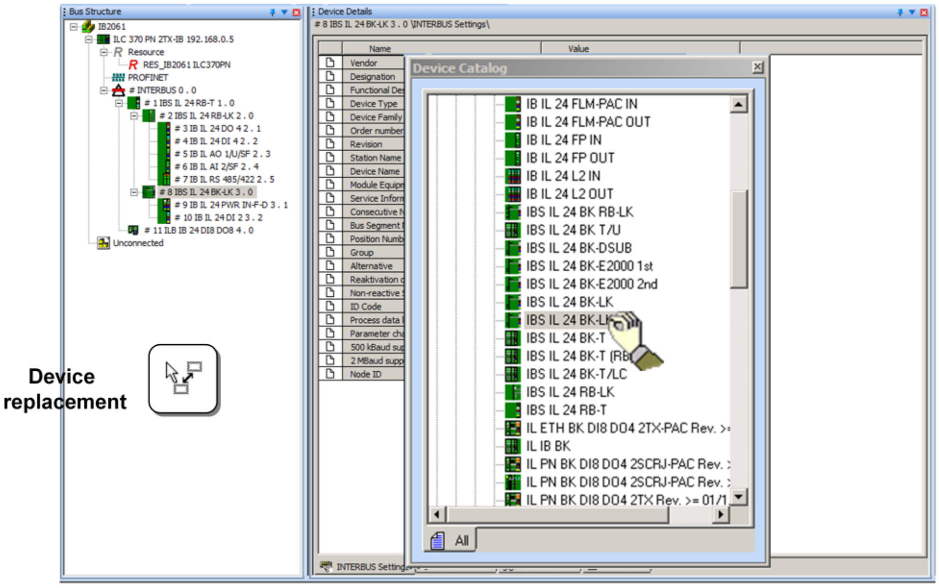
2. Insert device in lower level (*branch*) (remote bus device in remote bus branch or local bus device behind local bus terminal module).

This can also be done via the context menu (in the catalog *Copy Device*, in the bus configuration *Insert to lower level*) or using the mouse. Select the desired device in the catalog and draw it with the mouse button pressed to the bus configuration. In contrast to the insertion to the same level, on the device after which you would like to insert a new one, the mouse has to be moved to the right-hand side until below symbol is displayed.

Symbol when inserting to lower level:

When using the mouse and pressing the shift button, you can switch between the two possibilities *Insert to same level* and *Insert to lower level.*

Device Replacement Using the Device Catalog



3. Replace device (only possible if in principle the device offers the same interfaces). First of all, via the context menu, the replacement device has to be copied in the device catalog, then the context menu function *Replace* must be implemented.

Using mouse and keyboard, the desired device has to be selected in the catalog and, keeping the mouse button pressed down, has to be drawn to the device to be replaced in the bus configuration. For replacing instead of inserting, the same control button has to be pressed, so that below symbol shows up as a mouse pointer.

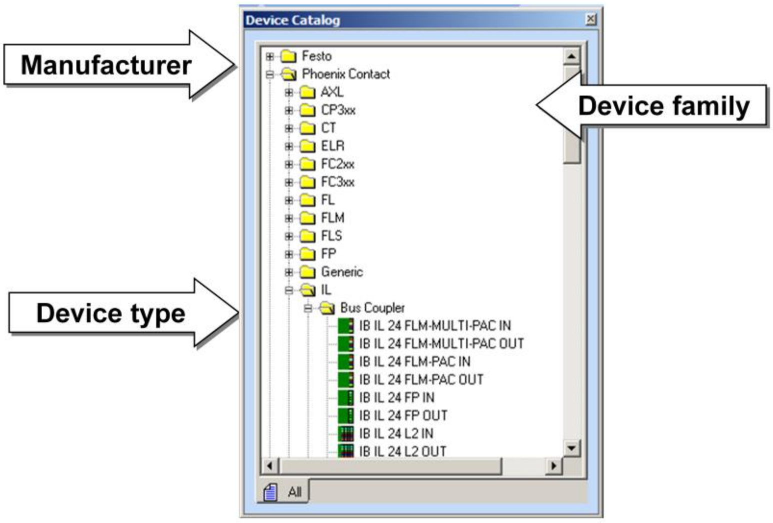
Symbol during replacement:

Notes:

Device Catalog and Device Files

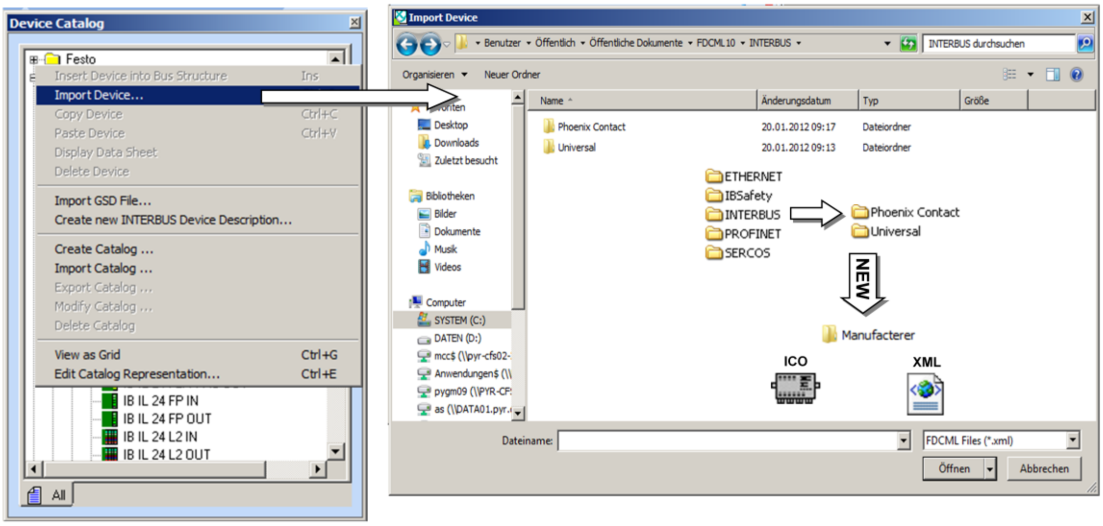
Notes:

Display



After the software installation, the catalog in its tree structure as a standard follows the system *Vendor – Device Family – Device Type*. This sorting has proven to be practical for the entire catalog, but can nevertheless be adjusted via the *Modify Catalog* function in the context menu. Doing without any structure might make sense especially for user-defined catalogs with a narrow range of devices.

Registration of Device Files (Import)



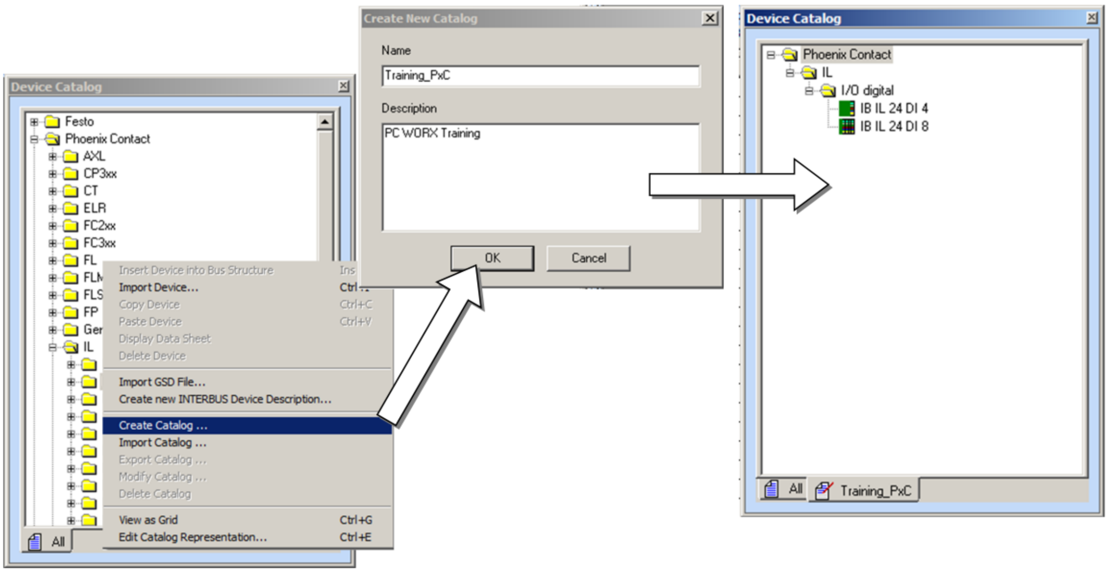
The registration (import) of device description files enables PC WORX to access files and thus devices that were made available subsequently.

In addition to an XML file, a full device description includes an icon file to which can also be referred to in the XML file. Without such a file, PC WORX would display a standard symbol.

Both files have to be stored in the same directory. It makes sense to create a folder with the name of the device supplier on the directory level shown above (for INTERBUS devices next to *Phoenix Contact* and *Universal* in order to keep an overview of the additionally registered devices. This folder does not affect the layout in the device catalog. For example, the supplier will not be read in from the folder name, to which the XML file is saved, but from the XML file itself.

⚠ The registration/import cannot be implemented from a mobile device (e.g. USB stick), as no local copies of the files are created.

User-Defined Device Catalog



Name: Training_PxC

Description: PC WORX Training

Creating a user-defined device catalog makes sense if a variety of projects is to be implemented using a limited number of devices. A new catalog can be created via context menu of the *Device Catalog* window. This catalog is first empty and unstructured.

The new catalog can be filled using the context menu to copy a device from the main catalog and paste it into the new catalog, as shown above.

The user-defined catalog enables faster access during offline and online configuration. For the latter, only the limited number of devices will be displayed when inserting devices.

Every user-defined device catalog can only contain the devices that are in the main catalog. There is no differentiated registration of device descriptions.

Notes:

PROFINET Configuration

Chapter 5

Notes:

Contents

This chapter describes the configuration of the PROFINET system connected to the Ethernet network.

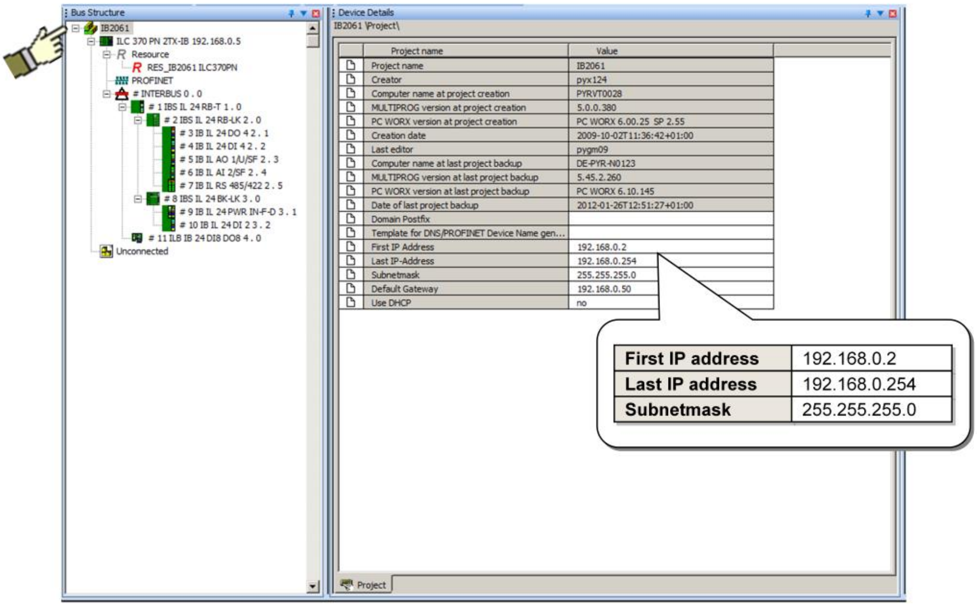
The online configuration describes the configuration for the case that the PROFINET system is installed. For adaptations and the configuration before startup of the hardware, the offline configuration is described.

⚠	Attention!
ℹ	Information
💡	Tip

Notes:

Basic Settings

Setting the Address Area



IB2061 \Project\

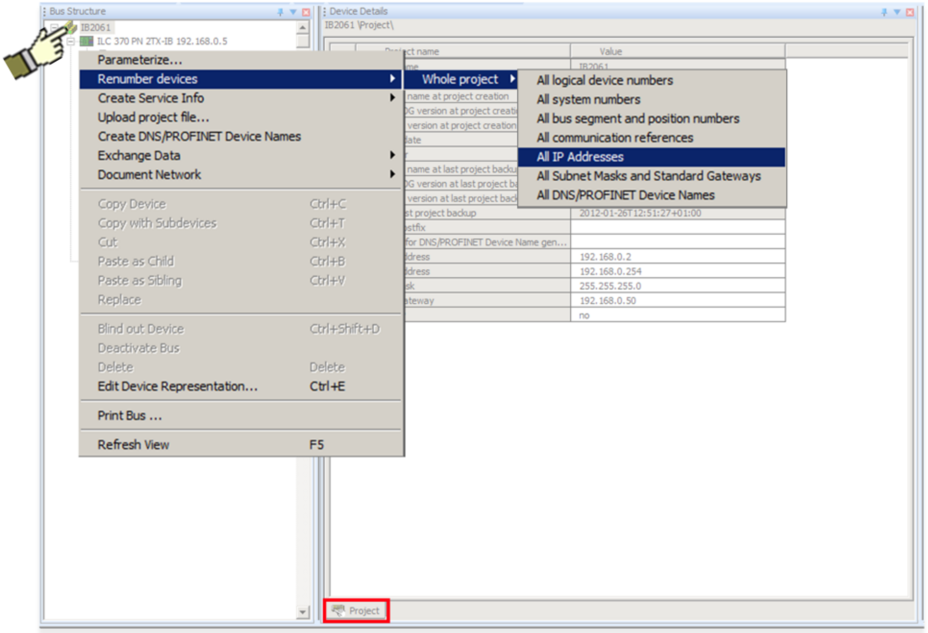
Project name	Value
Project name	IB2061
Creator	pyx124
Computer name at project creation	PYRVT0028
MULTIPROG version at project creation	5.0.0.380
PC WORX version at project creation	PC WORX 6.00.25 SP 2.55
Creation date	2009-10-02T11:36:42+01:00
Last editor	pygm09
Computer name at last project backup	DE-PYR-N0123
MULTIPROG version at last project backup	5.45.2.260
PC WORX version at last project backup	PC WORX 6.10.145
Date of last project backup	2012-01-26T12:51:27+01:00
Domain Postfix	
Template for DNS/PROFINET Device Name gen...	
First IP Address	192.168.0.2
Last IP-Address	192.168.0.254
Subnetmask	255.255.255.0
Default Gateway	192.168.0.50
Use DHCP	no

First IP address	192.168.0.2
Last IP address	192.168.0.254
Subnetmask	255.255.255.0

The address range of the PROFINET network in a PC WORX project is created via the project element in the *Bus Configuration* window. For every PROFINET device added, an IP address is automatically imported from this address range in the bus configuration.

For entering the IP address range, the start address may not be larger than the end address.

Importing Addresses

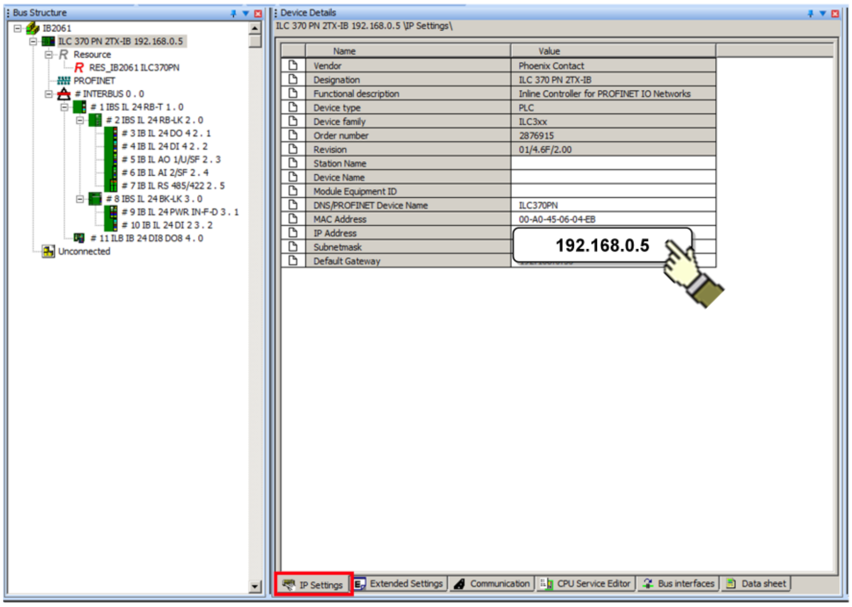


Via the context menu of the project you can centrally adapt the IP addresses in the bus configuration to the set address area.

If the subnetmask or the standard gateway is changed, the command *Renumber devices → Whole project → All Subnetmasks and Standard Gateways* has to be implemented.

Notes:

Direct IP Address Assignment



```
IB2061
  ILC 370 PN 2TX-IB 192.168.0.5
    Resource
      RES_IB2061 ILC370PN
    PROFINET
    # INTERBUS 0 . 0
      # 1 IBS IL 24 RB-T 1 . 0
        # 2 IBS IL 24 RB-LK 2 . 0
          # 3 IB IL 24 DO 4 2 . 1
          # 4 IB IL 24 DI 4 2 . 2
          # 5 IB IL AO 1/U/SF 2 . 3
          # 6 IB IL AI 2/SF 2 . 4
          # 7 IB IL RS 485/422 2 . 5
        # 8 IBS IL 24 BK-LK 3 . 0
          # 9 IB IL 24 PWR IN-F-D 3 . 1
          # 10 IB IL 24 DI 2 3 . 2
      # 11 ILB IB 24 DI8 DO8 4 . 0
  Unconnected
```

ILC 370 PN 2TX-IB 192.168.0.5 \IP Settings\

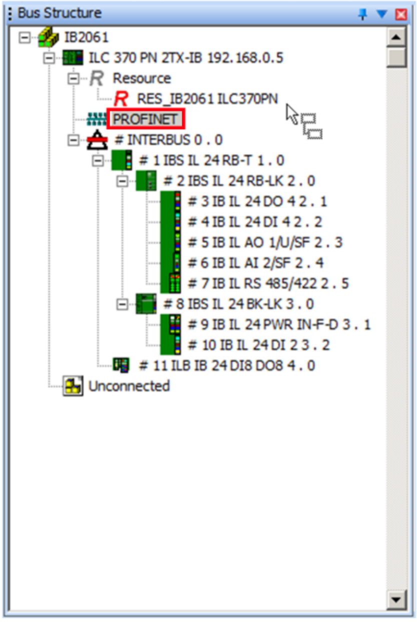
Name	Value
Vendor	Phoenix Contact
Designation	ILC 370 PN 2TX-IB
Functional description	Inline Controller for PROFINET IO Networks
Device type	PLC
Device family	ILC3xx
Order number	2876915
Revision	01/4.6F/2.00
Station Name	
Device Name	
Module Equipment ID	
DNS/PROFINET Device Name	ILC370PN
MAC Address	00-A0-45-06-04-EB
IP Address	
Subnetmask	
Default Gateway	

Via the *IP Settings* tab in the device details, an individual address assignment can be made for each device.

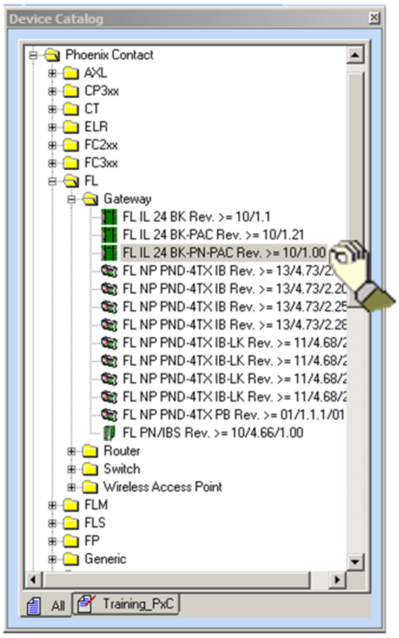
Notes:

Online/Offline Configuration

Offline Configuration



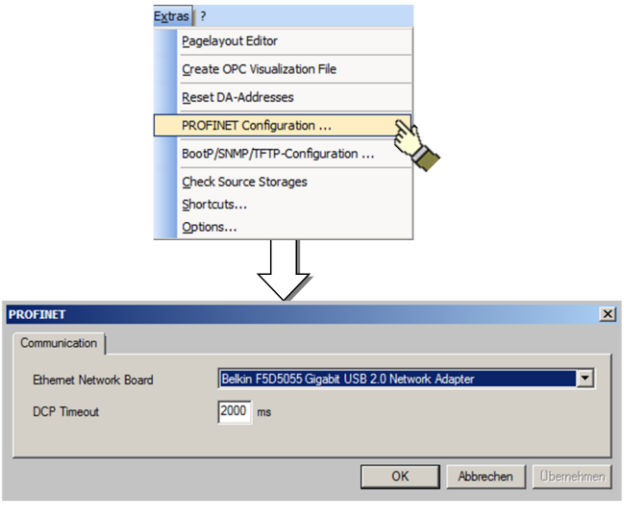
```
Bus Structure
IB2061
  ILC 370 PN 2TX-IB 192.168.0.5
    Resource
      RES_IB2061 ILC370PN
    PROFINET
    # INTERBUS 0 . 0
      # 1 IBS IL 24 RB-T 1 . 0
        # 2 IBS IL 24 RB-LK 2 . 0
          # 3 IB IL 24 DO 4 2 . 1
          # 4 IB IL 24 DI 4 2 . 2
          # 5 IB IL AO 1/U/SF 2 . 3
          # 6 IB IL AI 2/SF 2 . 4
          # 7 IB IL RS 485/422 2 . 5
        # 8 IBS IL 24 BK-LK 3 . 0
          # 9 IB IL 24 PWR IN-F-D 3 . 1
          # 10 IB IL 24 DI 2 3 . 2
      # 11 ILB IB 24 DI8 DO8 4 . 0
  Unconnected
```



```
Device Catalog
Phoenix Contact
  AXL
  CP3xx
  CT
  ELR
  FC2xx
  FC3xx
  FL
    Gateway
      FL IL 24 BK Rev. >= 10/1.1
      FL IL 24 BK-PAC Rev. >= 10/1.21
      FL IL 24 BK-PN-PAC Rev. >= 10/1.00
      FL NP PND-4TX IB Rev. >= 13/4.73/2.
      FL NP PND-4TX IB Rev. >= 13/4.73/2.20
      FL NP PND-4TX IB Rev. >= 13/4.73/2.25
      FL NP PND-4TX IB Rev. >= 13/4.73/2.28
      FL NP PND-4TX IB-LK Rev. >= 11/4.68/2
      FL NP PND-4TX IB-LK Rev. >= 11/4.68/2
      FL NP PND-4TX IB-LK Rev. >= 11/4.68/2
      FL NP PND-4TX IB-LK Rev. >= 11/4.68/2
      FL NP PND-4TX PB Rev. >= 01/1.1.1/01
      FL PN/IBS Rev. >= 10/4.66/1.00
    Router
    Switch
    Wireless Access Point
  FLM
  FLS
  FP
  Generic
All | Training_PxC
```

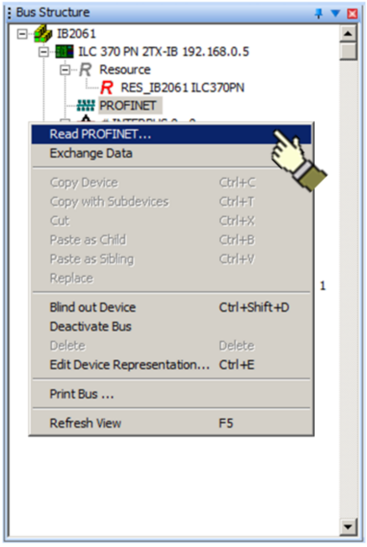
From the device catalog, PROFINET devices can be added to the network configuration. This can be done using Drag & Drop or by copying and pasting from the context menu.

Online Configuration



Ethernet Network Board: Belkin F5D5055 Gigabit USB 2.0 Network Adapter

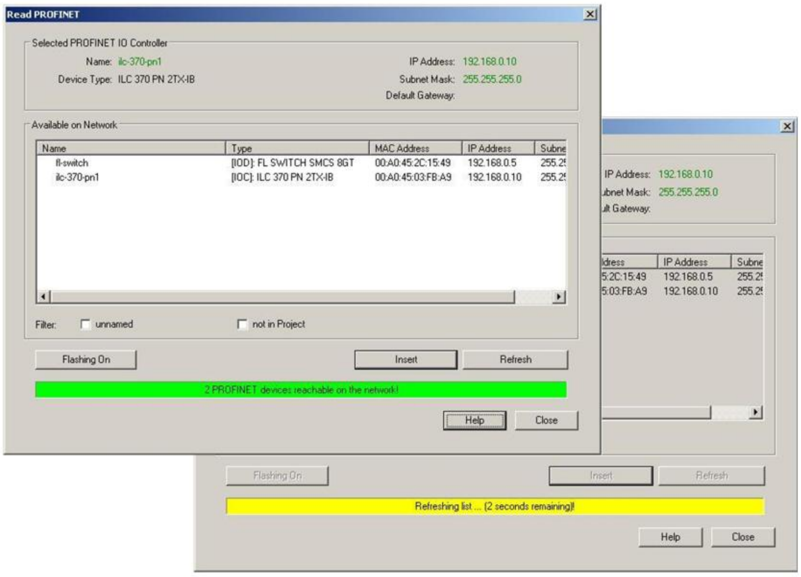
DCP Timeout: 2000 ms



The PROFINET network can also be configured online using PC WORX if the devices to be configured are installed and their device description files are registered in PC WORX.

If no PROFINET devices are found although the PROFINET system is connected, then maybe the wrong network card is installed in the PROFINET configuration.

Show Reachable Devices



Read PROFINET

Selected PROFINET IO Controller

Name: ilc-370-pn1

Device Type: ILC 370 PN 2TX-IB

IP Address: 192.168.0.10

Subnet Mask: 255.255.255.0

Default Gateway:

Available on Network

Name	Type	MAC Address	IP Address	Subne
fl-switch	[IOD]: FL SWITCH SMCS 8GT	00:A0:45:2C:15:49	192.168.0.5	255.2
ilc-370-pn1	[IOC]: ILC 370 PN 2TX-IB	00:A0:45:03:FB:A9	192.168.0.10	255.2

Filter: ☐ unnamed ☐ not in Project

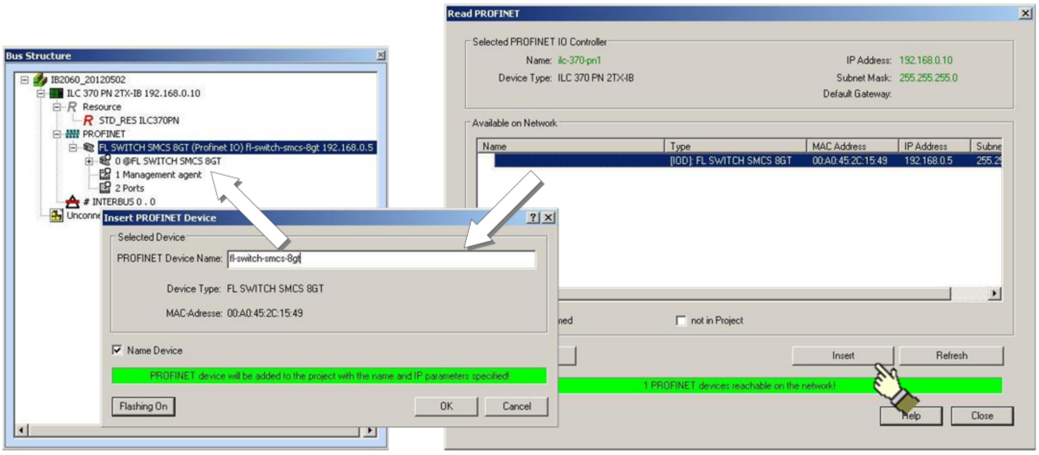
2 PROFINET devices reachable on the network!

Refreshing list ... (2 seconds remaining)!

Above list shows all PROFINET I/O devices available on the network. If some connected devices are not in the list, this might be due to the filter options below the window.

The selected I/O device is inserted to the PC WORX bus configuration via the *Insert* button.

Inserting Devices into the Project



Bus Structure

```
IB2060_20120502
  ILC 370 PN 2TX-IB 192.168.0.10
    Resource
      STD_RES ILC370PN
    PROFINET
      FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5
        0 @FL SWITCH SMCS 8GT
        1 Management agent
        2 Ports
    # INTERBUS 0 . 0
  Unconne
```

Insert PROFINET Device

Selected Device

PROFINET Device Name: fl-switch-smcs-8gt

Device Type: FL SWITCH SMCS 8GT

MAC-Adresse: 00:A0:45:2C:15:49

☑ Name Device

PROFINET device will be added to the project with the name and IP parameters specified!

Read PROFINET

Selected PROFINET IO Controller

Name: ilc-370-pn1

Device Type: ILC 370 PN 2TX-IB

IP Address: 192.168.0.10

Subnet Mask: 255.255.255.0

Default Gateway:

Available on Network

Name	Type	MAC Address	IP Address	Subne
	[IOD]: FL SWITCH SMCS 8GT	00:A0:45:2C:15:49	192.168.0.5	255.2

☐ not in Project

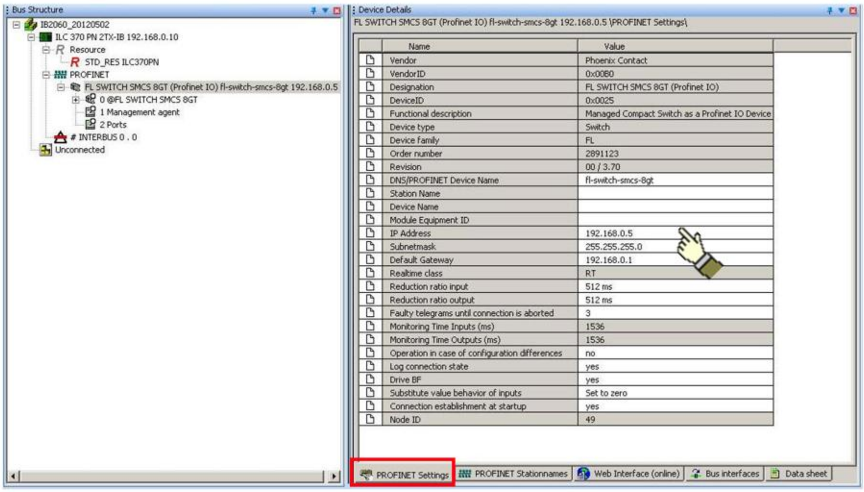
1 PROFINET devices reachable on the network!

In another dialog, a PROFINET device name can directly be assigned to the I/O device. First, the device name entered only refers to the assignment in the bus configuration. The actual naming, i.e. writing the device name to the I/O device itself, only happens by activating the *Name Device* control field.

Notes:

Device Settings

Device Settings



- IB2060_20120502
  - ILC 370 PN 2TX-IB 192.168.0.10
    - Resource
      - STD_RES ILC370PN
    - PROFINET
      - FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5
        - 0 @FL SWITCH SMCS 8GT
        - 1 Management agent
        - 2 Ports
    - # INTERBUS 0 . 0
  - Unconnected

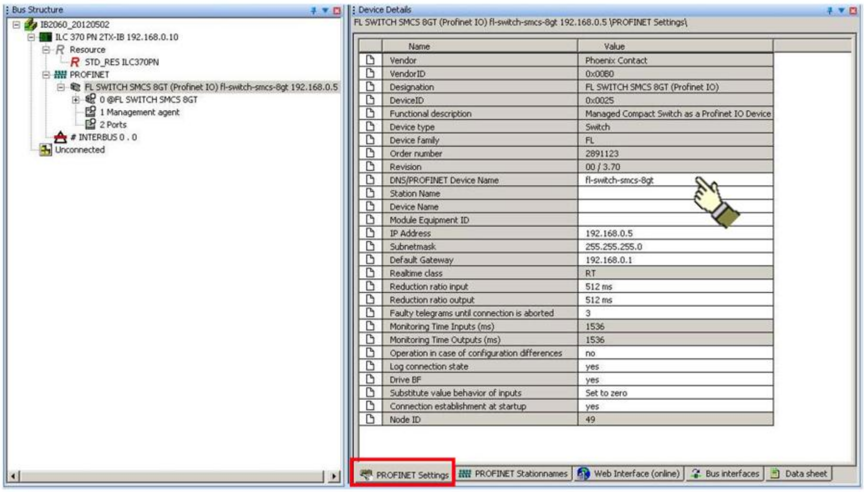
FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5 \PROFINET Settings\

Name	Value
Vendor	Phoenix Contact
VendorID	0x00B0
Designation	FL SWITCH SMCS 8GT (Profinet IO)
DeviceID	0x0025
Functional description	Managed Compact Switch as a Profinet IO Device
Device type	Switch
Device family	FL
Order number	2891123
Revision	00 / 3.70
DNS/PROFINET Device Name	fl-switch-smcs-8gt
Station Name	
Device Name	
Module Equipment ID	
IP Address	192.168.0.5
Subnetmask	255.255.255.0
Default Gateway	192.168.0.1
Realtime class	RT
Reduction ratio input	512 ms
Reduction ratio output	512 ms
Faulty telegrams until connection is aborted	3
Monitoring Time Inputs (ms)	1536
Monitoring Time Outputs (ms)	1536
Operation in case of configuration differences	no
Log connection state	yes
Drive BF	yes
Substitute value behavior of inputs	Set to zero
Connection establishment at startup	yes
Node ID	49

For every PROFINET I/O device, the update times for the inputs and outputs can be determined via the *PROFINET Settings* tab.

The update times for the individual I/O devices as well as for the inputs and outputs per device can be different.

Device Names



FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5 \PROFINET Settings\

Name	Value
Vendor	Phoenix Contact
VendorID	0x00B0
Designation	FL SWITCH SMCS 8GT (Profinet IO)
DeviceID	0x0025
Functional description	Managed Compact Switch as a Profinet IO Device
Device type	Switch
Device family	FL
Order number	2891123
Revision	00 / 3.70
DNS/PROFINET Device Name	fl-switch-smcs-8gt
Station Name	
Device Name	
Module Equipment ID	
IP Address	192.168.0.5
Subnetmask	255.255.255.0
Default Gateway	192.168.0.1
Realtime class	RT
Reduction ratio input	512 ms
Reduction ratio output	512 ms
Faulty telegrams until connection is aborted	3
Monitoring Time Inputs (ms)	1536
Monitoring Time Outputs (ms)	1536
Operation in case of configuration differences	no
Log connection state	yes
Drive BF	yes
Substitute value behavior of inputs	Set to zero
Connection establishment at startup	yes
Node ID	49

The device names of the individual I/O devices in the PC WORX project can be assigned via the *PROFINET Settings* tab.

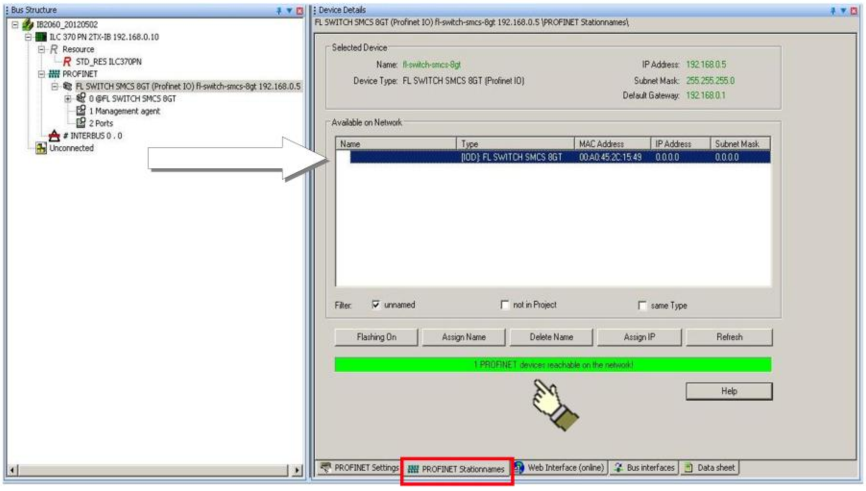
Allowed character set for assigning device names:

Letters:	*a-z*
Numerals:	*0-9*
hyphen:	-

The assignment of the device name refers exclusively to the assignment in the PC WORX project. The actual device naming has to be implemented separately.

The device name is of highest priority in PROFINET. It serves for identification of I/O devices in a PROFINET system and thus should be clear for use in the entire network.

PROFINET Devices in the Network



Bus Structure

- IB2060_20120502
  - ILC 370 PN 2TX-IB 192.168.0.10
    - Resource
      - STD_RES ILC370PN
    - PROFINET
      - FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5
        - 0 @FL SWITCH SMCS 8GT
        - 1 Management agent
        - 2 Ports
    - # INTERBUS 0 . 0
  - Unconnected

Device Details

FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5 \PROFINET Stationnames\

Selected Device

Name: fl-switch-smcs-8gt
Device Type: FL SWITCH SMCS 8GT (Profinet IO)
IP Address: 192.168.0.5
Subnet Mask: 255.255.255.0
Default Gateway: 192.168.0.1

Available on Network

Name	Type	MAC Address	IP Address	Subnet Mask
	[IOD]: FL SWITCH SMCS 8GT	00:A0:45:2C:15:49	0.0.0.0	0.0.0.0

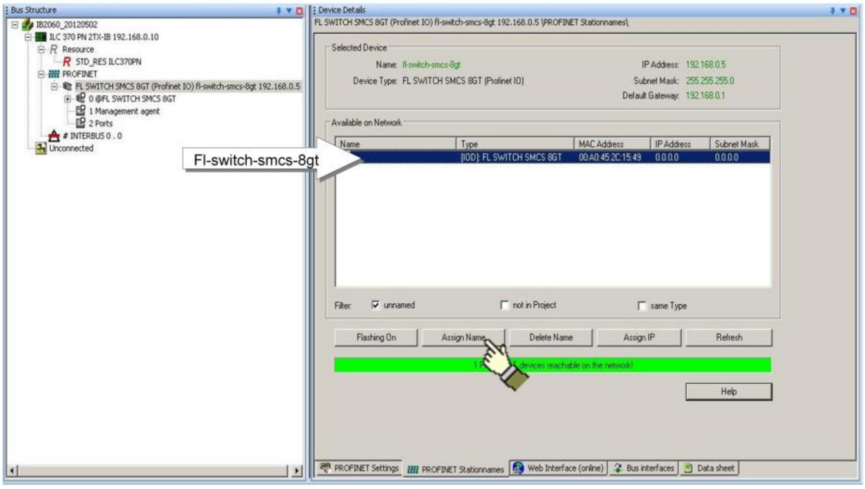
Filter: ☑ unnamed ☐ not in Project ☐ same Type

1 PROFINET devices reachable on the network!

Via the *PROFINET Stationnames* tab, all PROFINET I/O devices available on the network are read in and listed as shown above. Using this dialog you can write the PROFINET device names and the corresponding IP addresses to the respective I/O device. The actual device naming is implemented using the *Assign Name* button.

The *PROFINET Stationnames* tab is only available if a PROFINET I/O device was selected in the bus configuration.

PROFINET Device Naming



Bus Structure

- IB2060_20120502
  - ILC 370 PN 2TX-IB 192.168.0.10
    - Resource
      - STD_RES ILC370PN
    - PROFINET
      - FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5
        - 0 @FL SWITCH SMCS 8GT
        - 1 Management agent
        - 2 Ports
    - # INTERBUS 0 . 0
  - Unconnected

Fl-switch-smcs-8gt

Device Details

FL SWITCH SMCS 8GT (Profinet IO) fl-switch-smcs-8gt 192.168.0.5 \PROFINET Stationnames\

Selected Device

Name: fl-switch-smcs-8gt

Device Type: FL SWITCH SMCS 8GT (Profinet IO)

IP Address: 192.168.0.5

Subnet Mask: 255.255.255.0

Default Gateway: 192.168.0.1

Available on Network

Name	Type	MAC Address	IP Address	Subnet Mask
	[IOD]: FL SWITCH SMCS 8GT	00:A0:45:2C:15:49	0.0.0.0	0.0.0.0

Filter: ☑ unnamed ☐ not in Project ☐ same Type

1 P [illegible] T devices reachable on the network!

When assigning a name, the name of the I/O device selected in the bus configuration is assigned to the I/O device selected in the *Device Details* window. Please make sure that the selected devices are of the same type. The device name is permanently saved onto the respective I/O device.

The process described here is generally referred to as device naming. Unlike the volatile IP address, the device name remains saved on the PROFINET I/O device after a voltage reset.

Notes:

Process Data Variables

Chapter 6

Notes:

Contents

This chapter describes the creation of process data variables (inputs and outputs) based on the process data items of the bus systems configured in PC WORX.

A second step describes how global variables already available in the programming can be connected with process data items in order to enable access to the input and output signals of the bus system.

ℹ For both procedures, a bus system configured in PC WORX is required (see section *INTERBUS Configuration* or section *PROFINET Configuration*).

The connection between variables and process data items can be disconnected again via the context menu of the variables or the process data in the process data assignment view. Variables no longer needed, however, cannot be deleted in this view, but only in a corresponding global variable table.

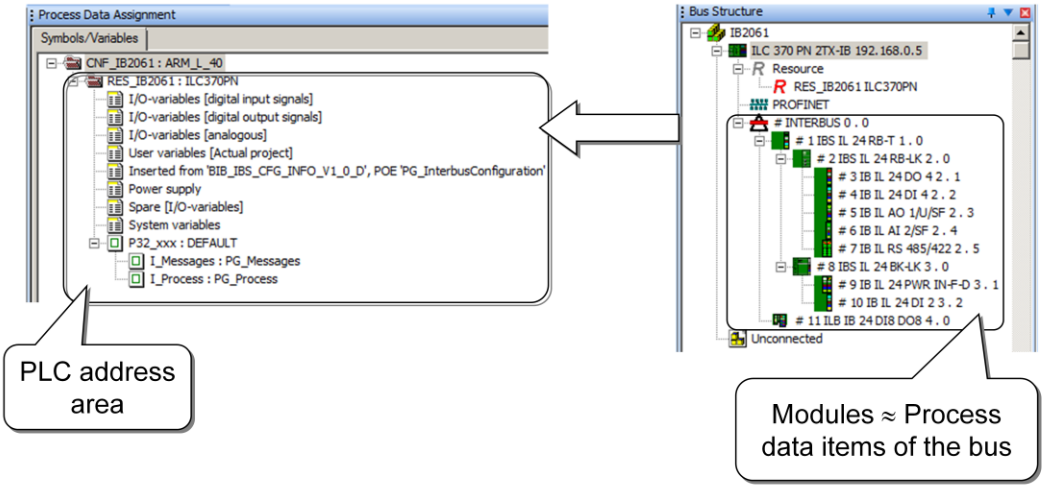
⚠ Attention!

ℹ Information

💡 Tip

Notes:

Process Data Assignment Workspace



```
CNF_IB2061 : ARM_L_40
  RES_IB2061 : ILC370PN
    I/O-variables [digital input signals]
    I/O-variables [digital output signals]
    I/O-variables [analogous]
    User variables [Actual project]
    Inserted from 'BIB_IBS_CFG_INFO_V1_0_D', POE 'PG_InterbusConfiguration'
    Power supply
    Spare [I/O-variables]
    System variables
    P32_xxx : DEFAULT
      I_Messages : PG_Messages
      I_Process : PG_Process
```

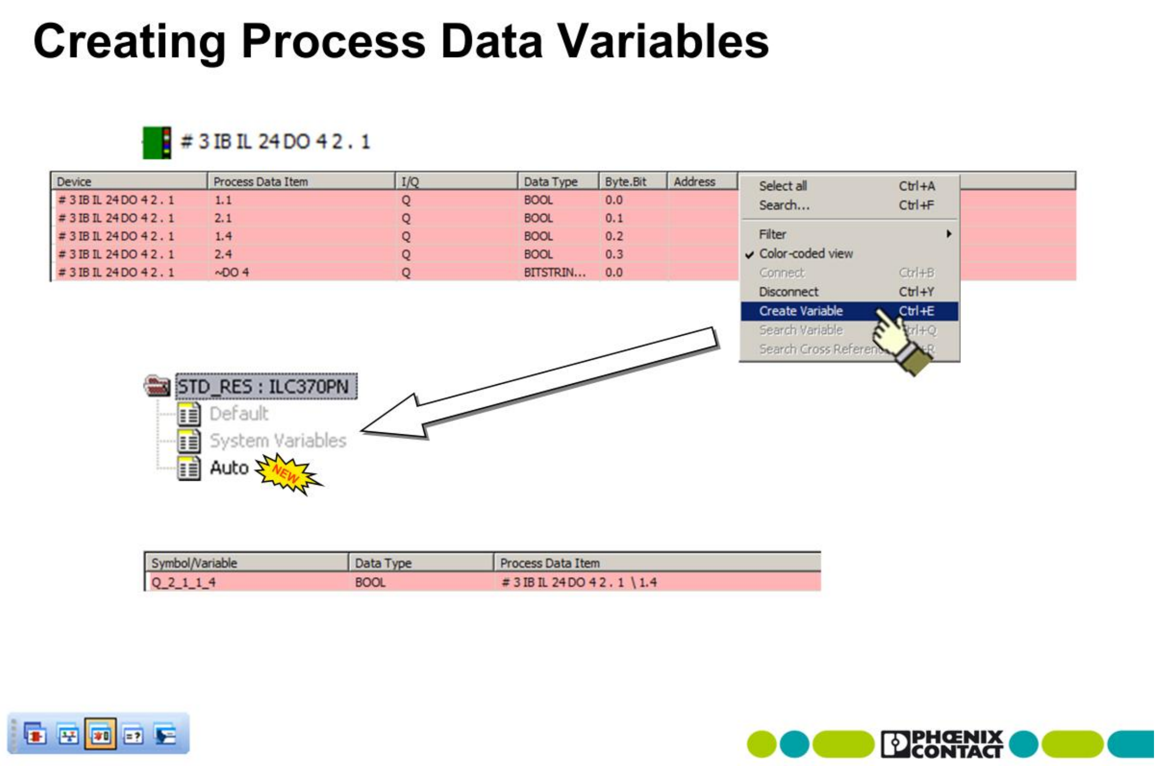
```
IB2061
  ILC 370 PN 2TX-IB 192.168.0.5
    Resource
      RES_IB2061 ILC370PN
    PROFINET
    # INTERBUS 0 . 0
      # 1 IBS IL 24 RB-T 1 . 0
        # 2 IBS IL 24 RB-LK 2 . 0
          # 3 IB IL 24 DO 4 2 . 1
          # 4 IB IL 24 DI 4 2 . 2
          # 5 IB IL AO 1/U/SF 2 . 3
          # 6 IB IL AI 2/SF 2 . 4
          # 7 IB IL RS 485/422 2 . 5
        # 8 IBS IL 24 BK-LK 3 . 0
          # 9 IB IL 24 PWR IN-F-D 3 . 1
          # 10 IB IL 24 DI 2 3 . 2
      # 11 ILB IB 24 DI8 DO8 4 . 0
  Unconnected
```

PLC address area

Modules ≈ Process data items of the bus

The process data items made available by the bus systems (no matter which system) through the XML files must be made available for the acquisition of input signals and the output of output signals of the PLC. This means that these objects must be accessed from the PLC address area. With PC WORX and other symbolic-only working systems, this happens through the assignment of variables to the process data items.

The PC WORX process data assignment workspace, which was already described together with the PC WORX interfaces, offers the right tools to create variables based on the process data items or connect existing variables with available process data items. Both methods are described in the following.



Creating Process Data Variables

3 IB IL 24 DO 4 2 . 1

Device	Process Data Item	I/Q	Data Type	Byte.Bit	Address
# 3 IB IL 24 DO 4 2 . 1	1.1	Q	BOOL	0.0	
# 3 IB IL 24 DO 4 2 . 1	2.1	Q	BOOL	0.1	
# 3 IB IL 24 DO 4 2 . 1	1.4	Q	BOOL	0.2	
# 3 IB IL 24 DO 4 2 . 1	2.4	Q	BOOL	0.3	
# 3 IB IL 24 DO 4 2 . 1	~DO 4	Q	BITSTRIN...	0.0	

Symbol/Variable	Data Type	Process Data Item
Q_2_1_1_4	BOOL	# 3 IB IL 24 DO 4 2 . 1 \ 1.4

For the creation of process data variables, select the element corresponding to the CPU of the PLC in the upper quadrant on the left-hand side.

Unless the user renamed the element after creating the project, in a project based on a standard project template, the element bears the designation *STD_RES (Standard Resource)* (CPU of the PLC).

Depending on the selected element in the upper quadrant on the right-hand side (bus system hardware), in the lower quadrant on the right-hand side for

a) system files (PROFINET, IBS), all process data items belonging to the bus systems are displayed;,

b) for bus couplers, all process data items of the bus modules assigned to the bus segment

c) and for individual modules, only the process data items assigned to the module are displayed.

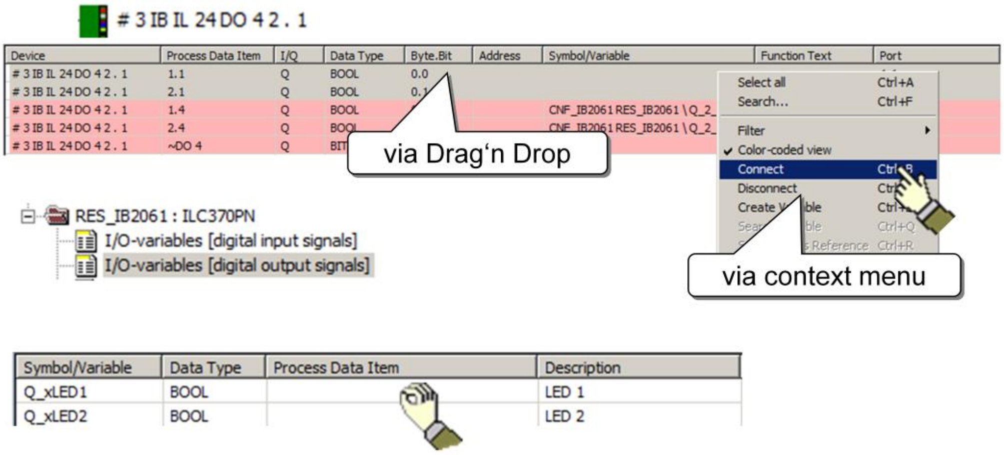
In the filter you can select whether standard and/or individual process data item, connected and/or unconnected process data items are displayed or not. The creation is done by selecting the *Create Variable* function from the context menu of a selected process data line. Please make sure that the process data items do not interfere with each other. This would later lead to an error message during compilation of the project. The variables created are always entered in the *Auto* variable group, which, if it has not been created so far, is automatically created by PC WORX.

The name of the automatically created variables is created according to the following pattern:

<data direction>_<Device ID IBS/PN>_<connection point> (see above example)

Characters not allowed in a variable name are replaced by underscores. The names of the variables can only be adapted in the variable tables, not in the process data assignment.

Process Data Items and Global Variables



Device	Process Data Item	I/Q	Data Type	Byte.Bit	Address	Symbol/Variable	Function Text	Port
# 3 IB IL 24 DO 4 2 . 1	1.1	Q	BOOL	0.0				
# 3 IB IL 24 DO 4 2 . 1	2.1	Q	BOOL	0.1				
# 3 IB IL 24 DO 4 2 . 1	1.4	Q	BOOL			CNF_IB2061 RES_IB2061 \ Q_2_		
# 3 IB IL 24 DO 4 2 . 1	2.4	Q	BOOL			CNF_IB2061 RES_IB2061 \ Q_2_		
# 3 IB IL 24 DO 4 2 . 1	~DO 4	Q	BIT					

via Drag‘n Drop

via context menu

Symbol/Variable	Data Type	Process Data Item	Description
Q_xLED1	BOOL		LED 1
Q_xLED2	BOOL		LED 2

For connecting process data items with already existing global variables, select the variable group, in which the variable to be connected is saved, in the upper quadrant on the left-hand side. Select the module and then the process data, to which the variable is to be connected to, from the upper quadrant on the right-hand side.

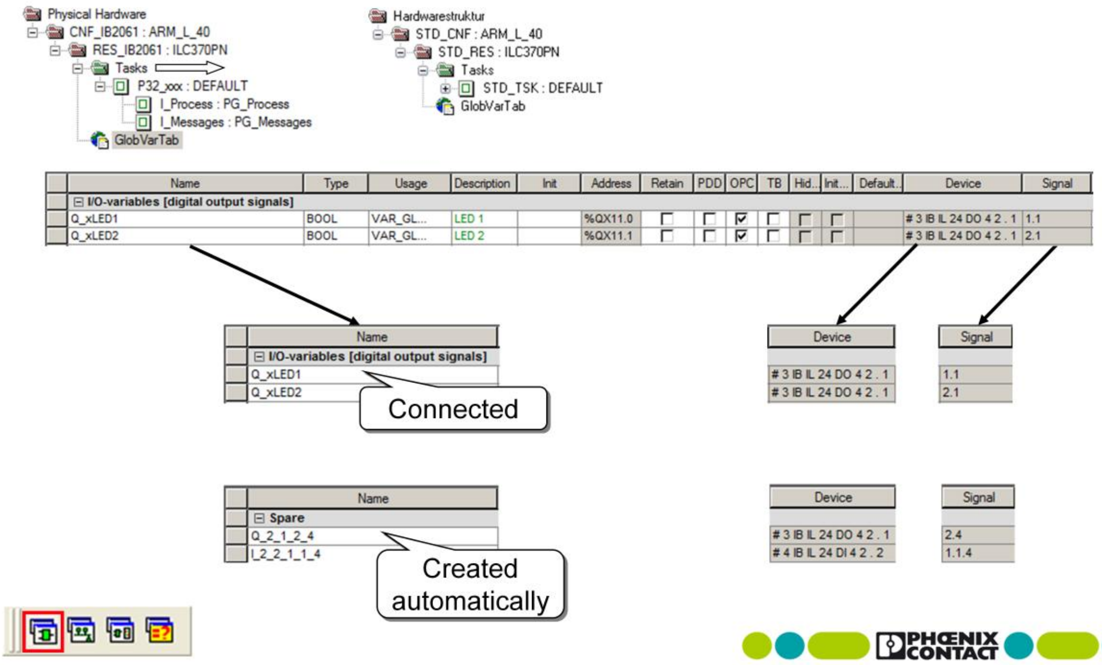
The *Connect* function will then be offered in the context menu of the variable as well as the process data.

As an alternative, the connection can be established using Drag’n Drop.

If more than one process data item of a module are to be connected to variables, a selection of multiple PD objects can be connected to the existing variables via Drag’n Drop. The correct order has to be adhered to.

Notes:

Global Variables with Process Data Connection



- Physical Hardware
  - CNF_IB2061 : ARM_L_40
    - RES_IB2061 : ILC370PN
      - Tasks
        - P32_xxx : DEFAULT
          - I_Process : PG_Process
          - I_Messages : PG_Messages
      - GlobVarTab

- Hardwarestruktur
  - STD_CNF : ARM_L_40
    - STD_RES : ILC370PN
      - Tasks
        - STD_TSK : DEFAULT
      - GlobVarTab

Name	Type	Usage	Description	Init	Address	Retain	PDD	OPC	TB	Hid...	Init...	Default...	Device	Signal
⊟ I/O-variables [digital output signals]														
Q_xLED1	BOOL	VAR_GL...	LED 1		%QX11.0	☐	☐	☑	☐	☐	☐		# 3 IB IL 24 DO 4 2 . 1	1.1
Q_xLED2	BOOL	VAR_GL...	LED 2		%QX11.1	☐	☐	☑	☐	☐	☐		# 3 IB IL 24 DO 4 2 . 1	2.1

Name	Device	Signal
⊟ I/O-variables [digital output signals]		
Q_xLED1	# 3 IB IL 24 DO 4 2 . 1	1.1
Q_xLED2	# 3 IB IL 24 DO 4 2 . 1	2.1

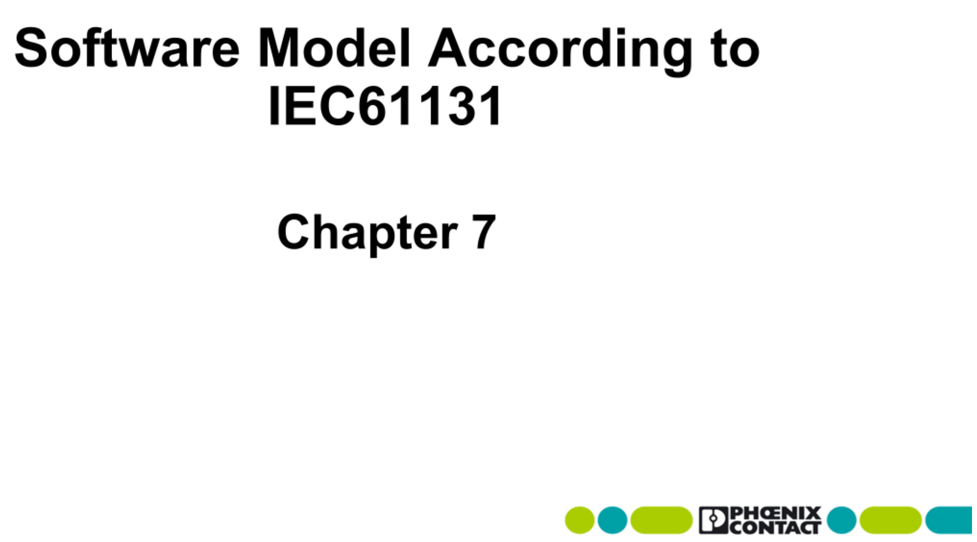
Connected

Name	Device	Signal
⊟ Spare		
Q_2_1_2_4	# 3 IB IL 24 DO 4 2 . 1	2.4
I_2_2_1_1_4	# 4 IB IL 24 DI 4 2 . 2	1.1.4

Created automatically

No matter which procedure you choose for making available process data variables, the result will be the same. Only the symbols for created variables need to be adapted depending on the application requirements.

Notes:



Software Model According to IEC61131

Chapter 7

Notes:

Contents

In this chapter you learn about how to categorize the integrated control systems according to IEC 61131. Four performance classes for control systems and the processors and types available regarding these are presented.

The documentation also describes how the performance of the CPU can be organized with the help of tasks according to the application requirements.

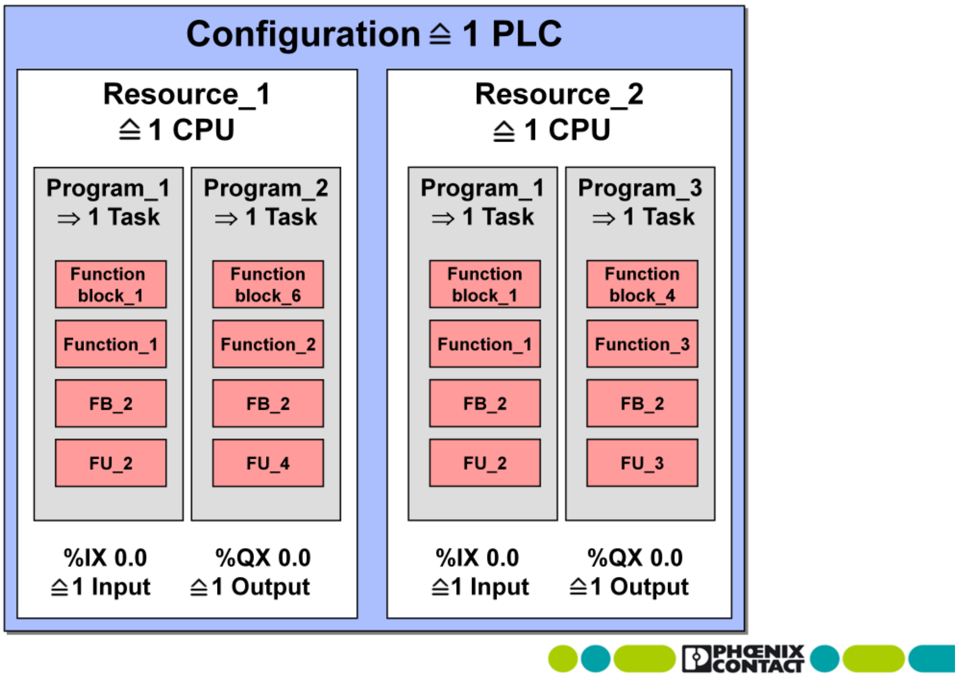
⚠ Attention!

ℹ Information

💡 Tip

Notes:

Terminology



Configuration ≙ 1 PLC

Resource_1
≙ 1 CPU

Program_1
⇒ 1 Task

Function block_1

Function_1

FB_2

FU_2

Program_2
⇒ 1 Task

Function block_6

Function_2

FB_2

FU_4

%IX 0.0
≙1 Input

%QX 0.0
≙1 Output

Resource_2
≙ 1 CPU

Program_1
⇒ 1 Task

Function block_1

Function_1

FB_2

FU_2

Program_3
⇒ 1 Task

Function block_4

Function_3

FB_2

FU_3

%IX 0.0
≙1 Input

%QX 0.0
≙1 Output

Configuration
Description (physics and logic) of a complete PLC system.

Resource
Offers support for all features that are required for the performance programs; interface between one program and the physical I/O channels of the PLC.

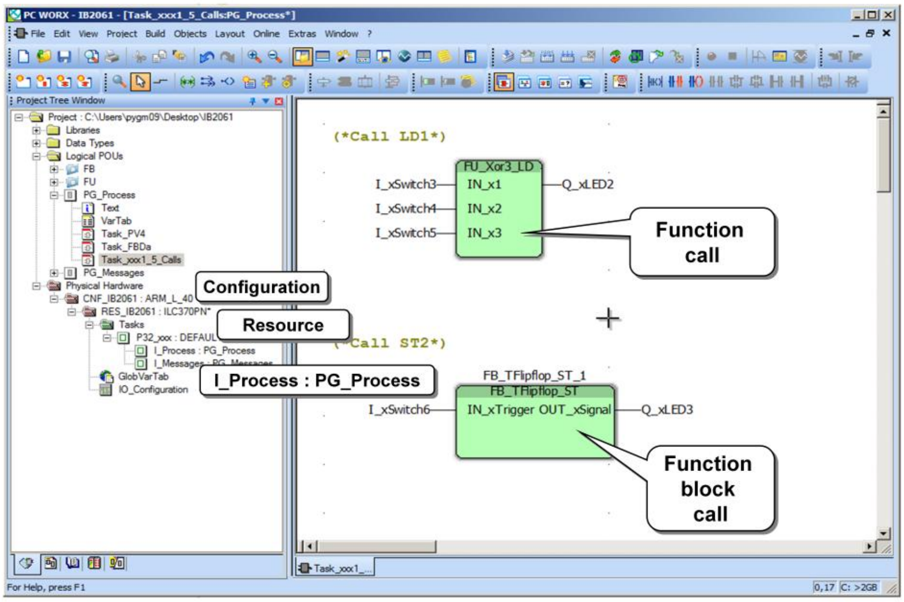
Tasks
Time to control the execution of different application parts.

Program (PG)
Program organization unit (POU), is assigned to a task

Function block (FB)
Subprogram (POU), contains static data

Function (FU)
Subprogram (POU), without comprehensive data

Implementation



The program *PG_Process* is assigned to the instance *I_Process*.

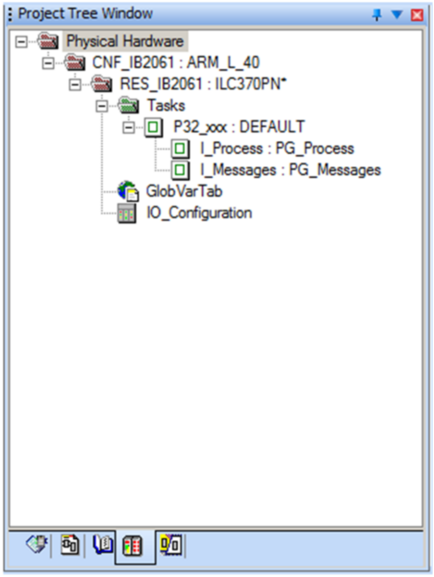
The left-hand side of the workspace displays the project description according to IEC 61131.

The upper part of the project tree contains the description of the logic part of the project: libraries, data types and logic POUs.

The lower part describes the physical structure: CPU type(s), task structure, access to process data.

The right-hand side of the workspace displays the details for individual elements.

Hardware Tree

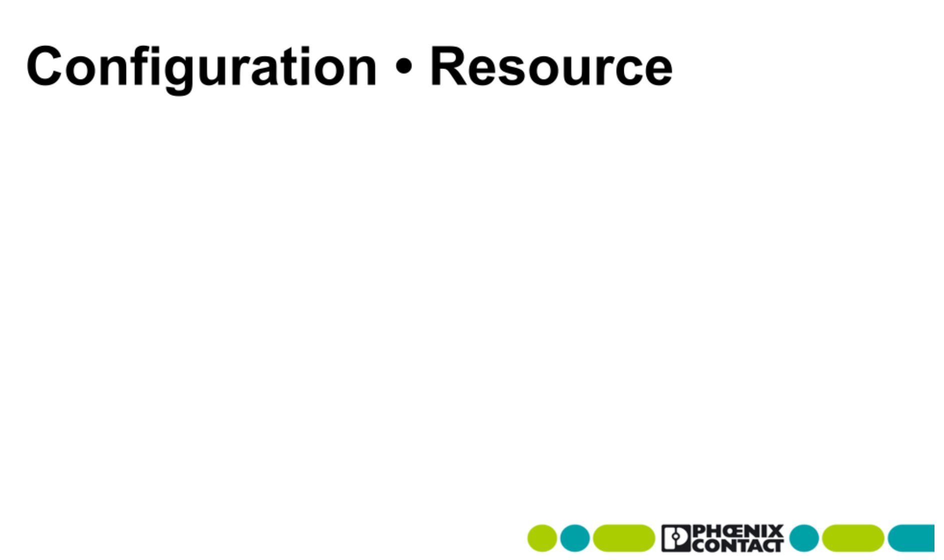


```
Physical Hardware
  CNF_IB2061 : ARM_L_40
    RES_IB2061 : ILC370PN*
      Tasks
        P32_xxx : DEFAULT
          I_Process : PG_Process
          I_Messages : PG_Messages
      GlobVarTab
      IO_Configuration
```

In PC WORX, the PLC hardware structure mainly consists of the PLC rack, the PLC processor and the management of the processor capacity.

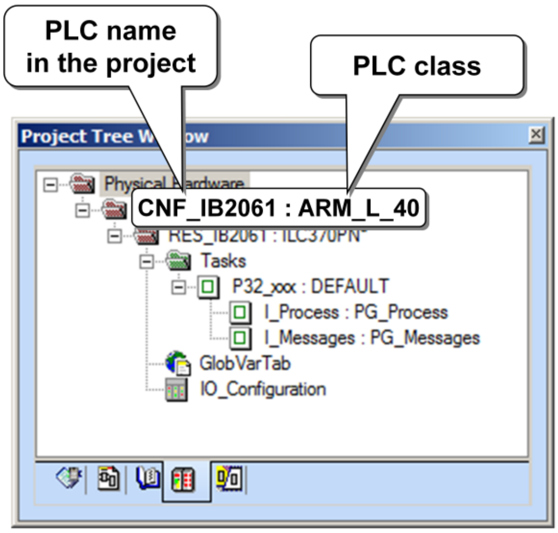
The part of the project tree, to which the hardware structure is saved, can be viewed on a separate tab.

Notes:



Configuration • Resource

Configuration: PLC Performance Class/Rack

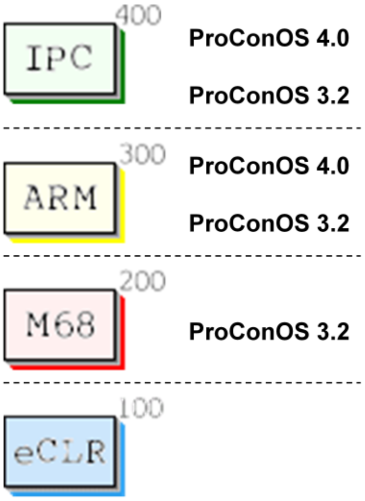


PLC name in the project

PLC class

Project Tree Window

- Physical Hardware
  - CNF_IB2061 : ARM_L_40
    - RES_IB2061 : ILC370PN
      - Tasks
        - P32_xxx : DEFAULT
          - I_Process : PG_Process
          - I_Messages : PG_Messages
      - GlobVarTab
      - IO_Configuration



According to IEC 61131, the higher-level element of the hardware structure for a PLC is called *Configuration*. For classic control systems, this corresponds to the PLC rack and serves for classification for the performance class.

After installation, four [1] different performance classes are available in PC WORX. These classes already differ regarding the processors used. Furthermore, for the two high performance classes, different versions of runtime systems are available.

Class 400 counts on INTEL processors; in class 300, ARM processors are used and for class 200 control systems, Motorola processors are used. The latest class 100 control systems use eCLR.

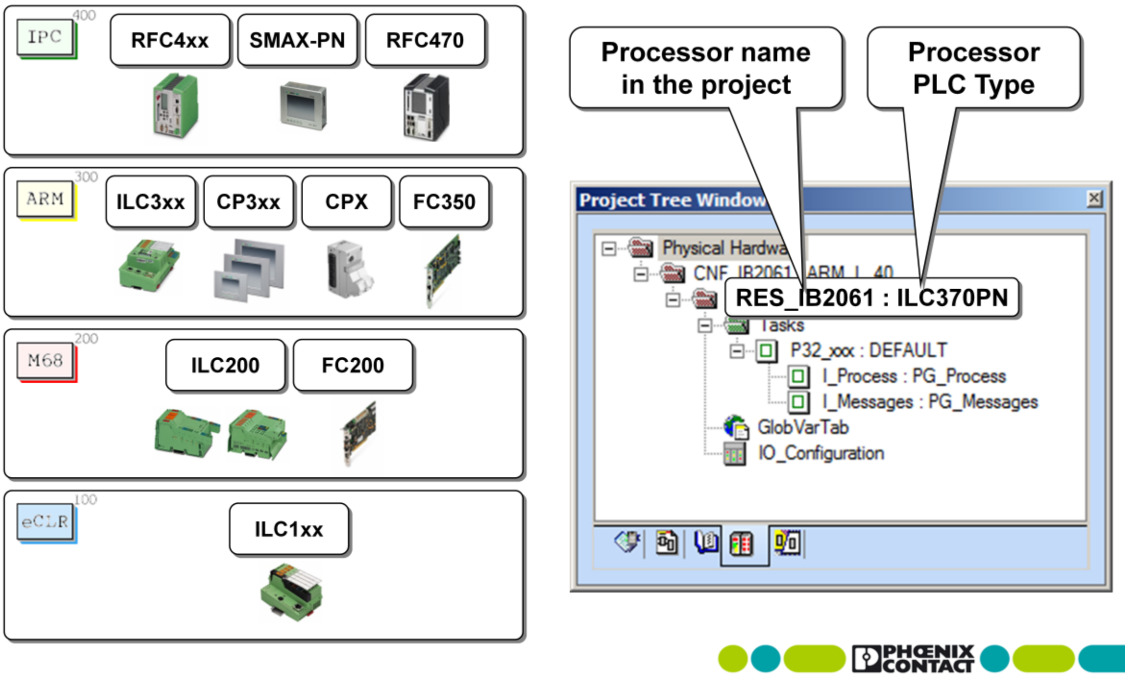
The difference between the two operating system versions 3.2 and 4.0 is basically the degree of editing options during operation of the PLC.

Due to the possibility to work with several control systems, in addition to the performance class of the control system, a name has to be assigned. The project templates in PC WORX as a standard use the name *STD_CNF* (standard configuration).

⚠ The term *Configuration* used here is not at all linked to the INTERBUS configuration.

[1] As of PC WORX 6.00.25

Resource: PLC Processor



Processor name in the project

Processor PLC Type

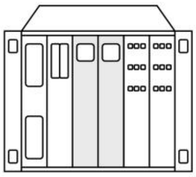
Project Tree Window

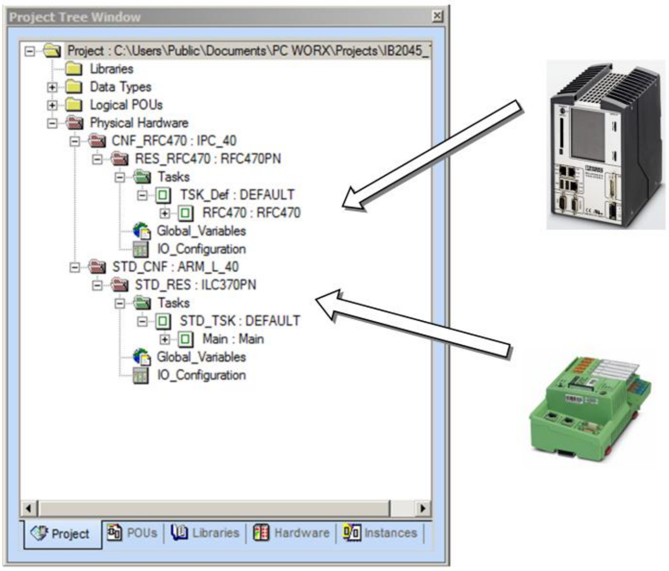
- Physical Hardwa
  - CNF_IB2061 : ARM_L_40
    - RES_IB2061 : ILC370PN
      - Tasks
        - P32_xxx : DEFAULT
          - I_Process : PG_Process
          - I_Messages : PG_Messages
      - GlobVarTab
      - IO_Configuration

The processors to be used and therefore the matching control systems depend on the performance class. Above figure shows which control systems are connected to which performance class. Some control systems are available with the two ProConOS versions 3.2 and 4.0.

As with the performance class, in addition to entering the processor type, a name for the element *Resource* has to be assigned. The project templates in PC WORX as a standard use the name STD_RES (standard resource).

Distributed Resources



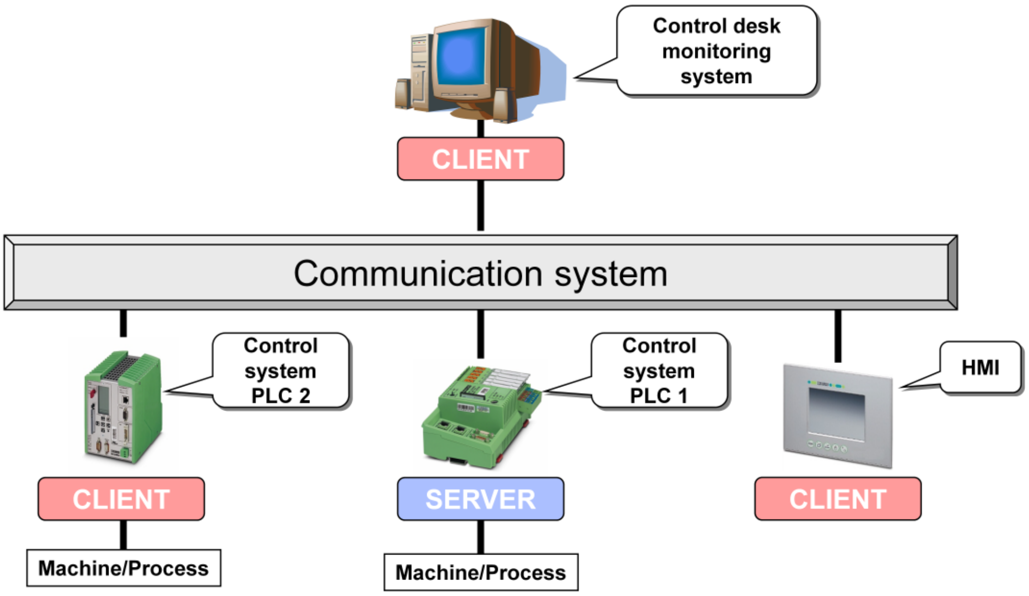


```
Project : C:\Users\Public\Documents\PC WORX\Projects\IB2045_
  Libraries
  Data Types
  Logical POUs
  Physical Hardware
    CNF_RFC470 : IPC_40
      RES_RFC470 : RFC470PN
        Tasks
          TSK_Def : DEFAULT
            RFC470 : RFC470
        Global_Variables
        IO_Configuration
    STD_CNF : ARM_L_40
      STD_RES : ILC370PN
        Tasks
          STD_TSK : DEFAULT
            Main : Main
        Global_Variables
        IO_Configuration
```

Regarding the hardware in the bus configuration and the PLC, PC WORX supports several systems. However, in the following, this documentation will only refer to one control system per project.

The programs created in the upper part of the project tree can be assigned to different, distributed configurations and thus, control systems. The control systems use the same algorithms, but process them individually.

Communication model IEC61131-5



Control desk monitoring system

CLIENT

Communication system

Control system PLC 2

CLIENT

Machine/Process

Control system PLC 1

SERVER

Machine/Process

HMI

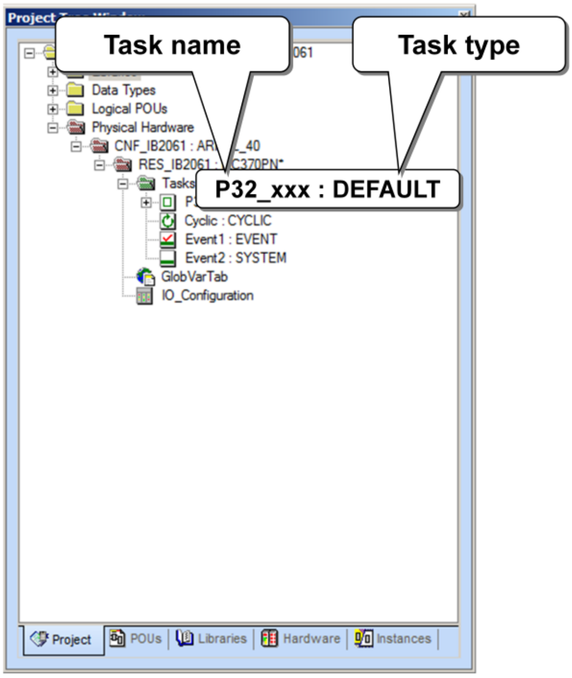
CLIENT

If more than one control system is configured, the data exchange can be implemented via standardized blocks. In part 5 of IEC 61131, the communication model describes the data exchange between several devices of the automation process. It corresponds to a usual client/server architecture and will be dealt with in the PC WORX 6 – Basics Of Communication course.

Notes:

Tasks

Task: Processor Management



Cyclic tasks

□ **DEFAULT task**

↻ **User-defined, cyclic tasks**

Event-controlled tasks

▭ **Tasks for system-internal events (PLC related)**

☑ **Tasks for system-external events (PLC related)**

Thanks to the task management in PC WORX, the processor capacity can be used precisely and tuned to the application.

The tasks are divided into two classes: cyclic and event-controlled tasks.

The process in a PLC usually features cyclic execution. The DEFAULT task as a cyclic task therefore can be used for most applications. It works with the lowest priority and as a minimum cycle time offers tmin=2*system tick. For IPC and ARM processors, the system tick is set to 1ms, for M68 processors to 5 ms. The DEFAULT task can only be created once and because of its low priority it is interrupted by every other task.
In contrast do the DEFAULT task, user-defined tasks (CYCLIC tasks) offer the possibility to freely select priority and cycle time within the scope of technical marginal conditions. The priority can be set between *0* (highest) and *31* (lowest).

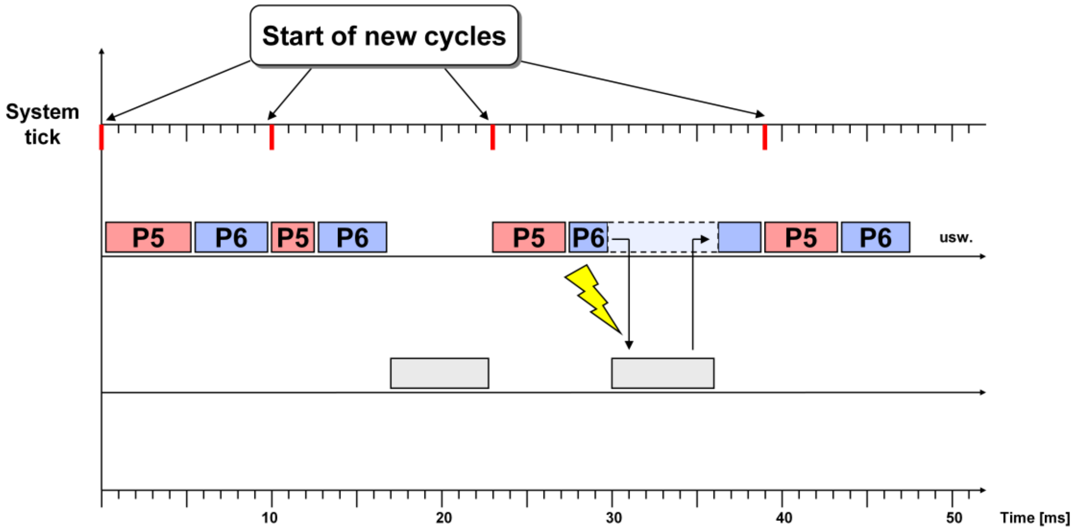
The second task class covers the SYSTEM and EVENT tasks. The SYSTEM tasks enable a reaction to PLC-internal events (e.g. cold restart, warm start, division by *0* etc.). This includes regular and irregular operating states of the control system.
It depends on the control system used which external events can be acquired with the EVENT tasks. One example are the direct inputs, where state changes are defined as an event with the ILCs. One of the events is the finishing of a cycle when the INTERBUS is connected.

The events for EVENT and SYSTEM tasks can be set in the PC WORX dialogs via plain-text selection.

No programming can be carried out in tasks. According to their configuration, tasks trigger program instances (see section *Hardware Architecture*).

You can ensure a clearly arranged hardware structure by assigning names that correspond to features. Process data preprocessing tasks always have to have the name PDP.

PLC Timing: DEFAULT-Task



The timing of the DEFAULT task corresponds to the time response of most classic control systems: Directly after execution of a cycle, the next one is started. There are no fixed intervals. Due to the programming, cycle times may fluctuate significantly. The interruption of the default task by a server task (debug or communication) also results in fluctuations in the cycle time.

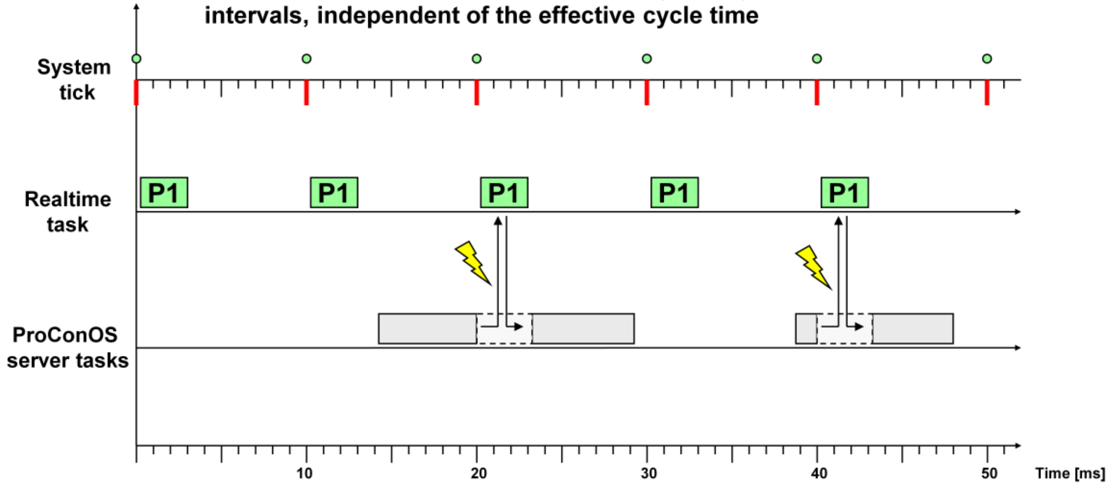
DEFAULT task: After each cycle (code P5 and code P6), the next one starts immediately.

Watchdog error: The code cannot be executed within t=500ms (default setting – can be changed.

Debugging affects the time response of the program execution.

PLC Timing: Realtime Task

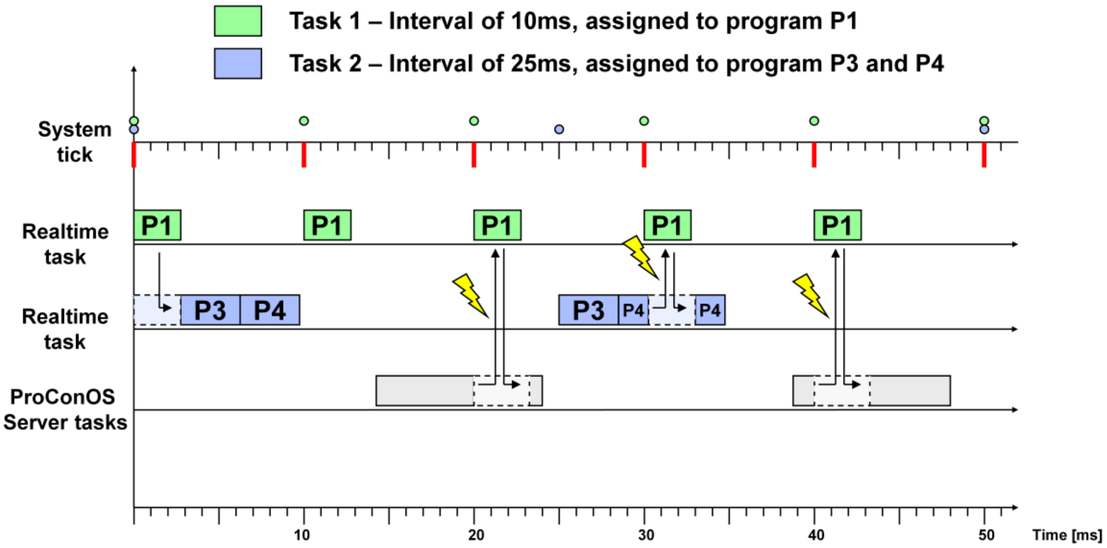
Deterministic time response: Processing of a cycle within fixed intervals, independent of the effective cycle time



The realtime tasks features a temporal determinism: The code (P1) assigned to a task is executed exactly once in fixed intervals. The response times can therefore be calculated. Server tasks only become active if the realtime tasks do not need computing time.

The time monitoring (watchdog) ensures that even if the processing of an assigned code cannot be finished within an interval because, e.g. the code contains too many processor commands, a calculable reaction still is implemented. Either the control system switches to the operating state STOP or a user-defined reaction is executed.

PLC Timing: More than one Realtime Task



🟩 Task 1 – Interval of 10ms, assigned to program P1

🟦 Task 2 – Interval of 25ms, assigned to program P3 and P4

Notes:

Program Organization Units

Chapter 8

Notes:

Contents

This chapter informs you on the software blocks of your project, which, according to IEC 61131, are called program organization units.

The features of the three available types, how to call them in you project and how to implement the data exchange between individual blocks are described.

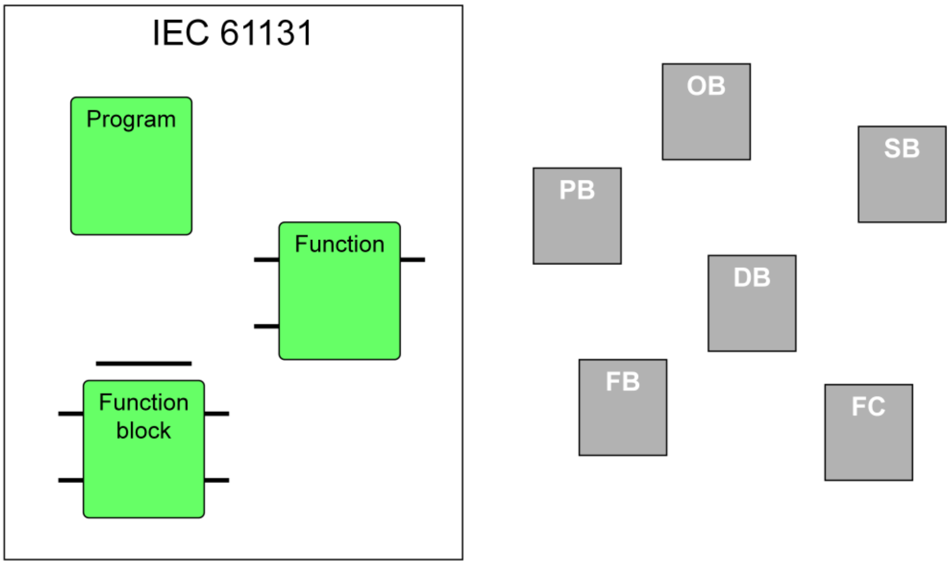
⚠ Attention!

ℹ Information

💡 Tip

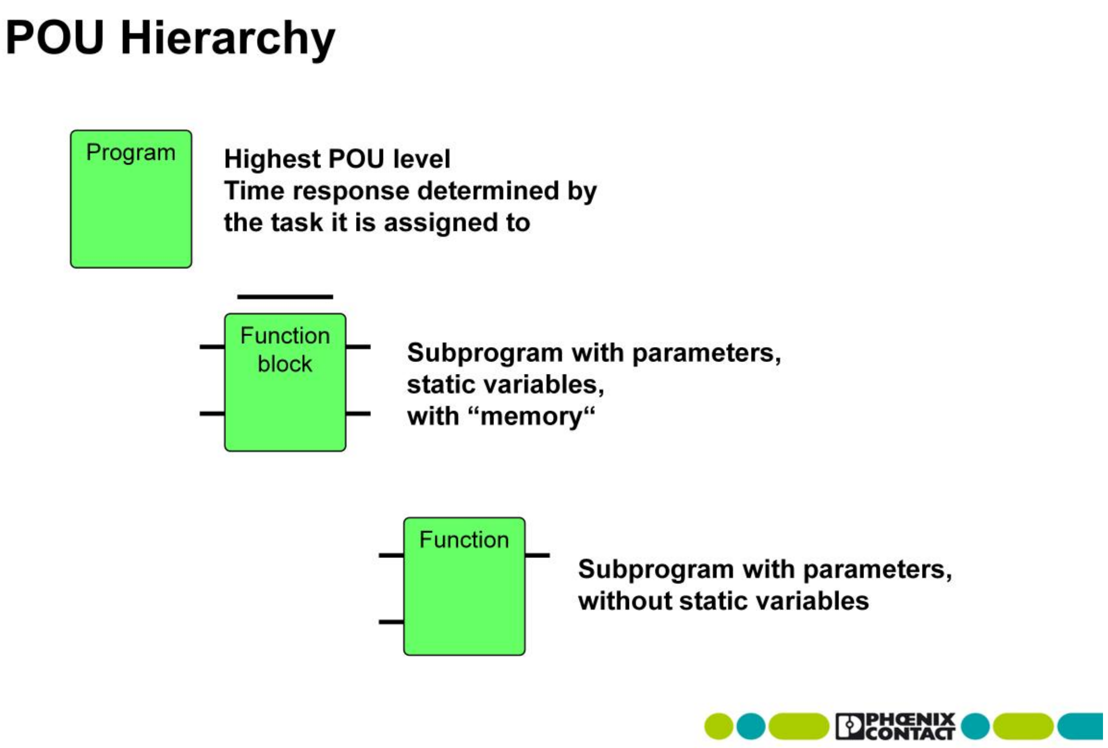
Notes:

POU Types 1



According to IEC 61131, three different POU types are defined: the program, the function block and the function. In general, programming could be implemented as an unstructured lining up of commands. However, the use of these three types allows for a much higher degree of clarity in the project, and what's more, it offers advantages regarding efficient programming in the editing phase.

In the following, the features and areas of application for the individual POU types are described as well as a first overview on how data can be exchanged between configured POUs.



POU Hierarchy

Program

Highest POU level
Time response determined by
the task it is assigned to

Function block

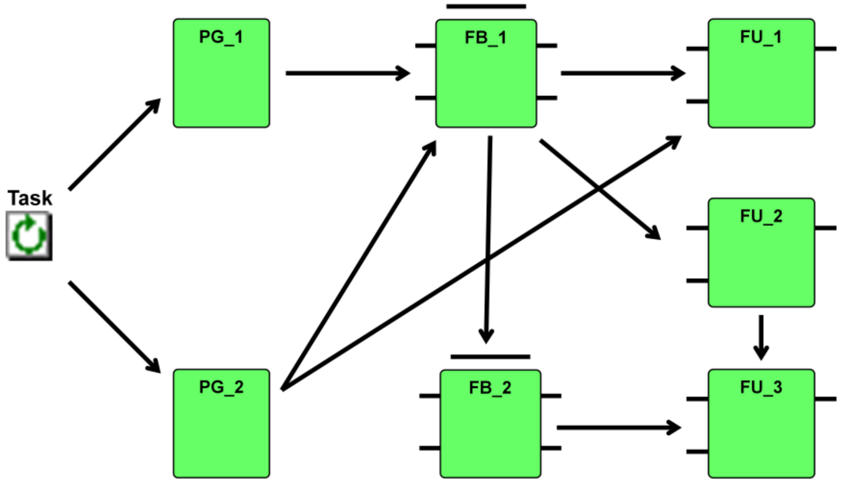
Subprogram with parameters,
static variables,
with “memory“

Function

Subprogram with parameters,
without static variables

The overview shows the hierarchy of the three POU types and their basic properties.

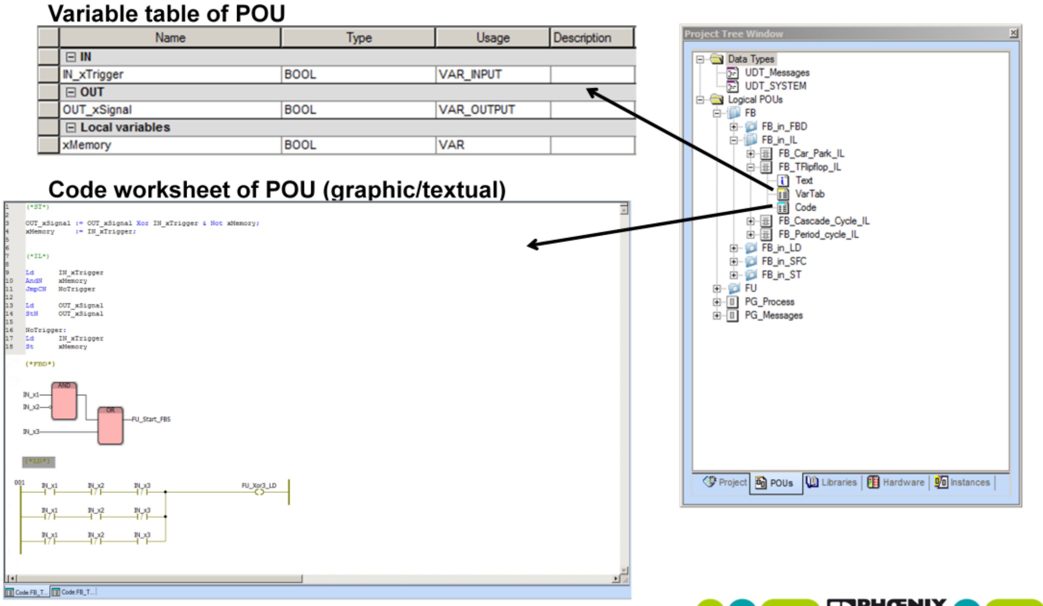
POU Call Structure



The call structure is organized hierarchically.
The degree of complexity of the POU calls depends on the memory stack.
For security reasons, recursive calls are not allowed.

Notes:

POU Components



Variable table of POU

Name	Type	Usage	Description
⊟ IN			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ Local variables			
xMemory	BOOL	VAR	

Code worksheet of POU (graphic/textual)

```
(*ST*)

OUT_xSignal := OUT_xSignal Xor IN_xTrigger & Not xMemory;
xMemory     := IN_xTrigger;


(*IL*)

Ld       IN_xTrigger
AndN     xMemory
JmpCN    NoTrigger

Ld       OUT_xSignal
StN      OUT_xSignal

NoTrigger:
Ld       IN_xTrigger
St       xMemory
```

Each POU (no matter which type) can be provided with one ore more comment worksheets in which the programmer can save a description of the implemented functions or information on the version.

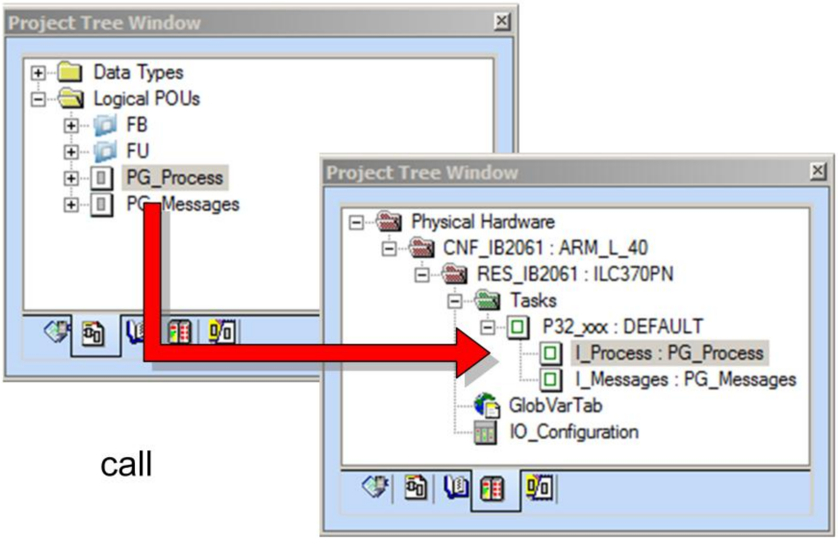
Exactly one local variable table is available for saving the variables required for function of the POU.

For carrying out the desired algorithm, one or more graphical or textural worksheets are available to the programmer.

Notes:

The Program

Use of Programs



The program is the POU of the highest level. In a project, at least one POU of this type is required.

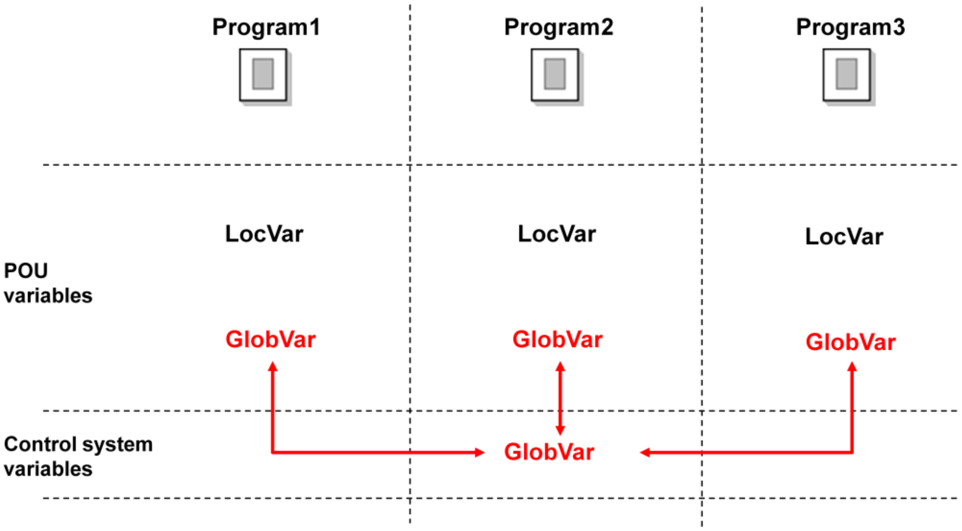
Programs as well as the other POUs are saved to the Logical POUs folder and at that point have no effect on the automation project. Only when a program in form of a program instance is used in a task, the code saved to the program is carried out on the PLC.

A program instance is the memory area of a CPU, in which a code available as a program is carried out depending on a task definition.

To release a CPU program for execution by assigning a program instance element to a task is called program instantiation (compare to function block instantiation).

PC WORX supports multiple instantiation of programs. The opportunities deriving from this are explained in detail in the *PC WORX IEC 61131 Communication* course.

POU Type *Program*: Data Exchange



One of the main tasks of the programs is linking the hardware signals to the programming.

Global variables are the only way to establish this link. They are also used to exchange global information regarding the control system.

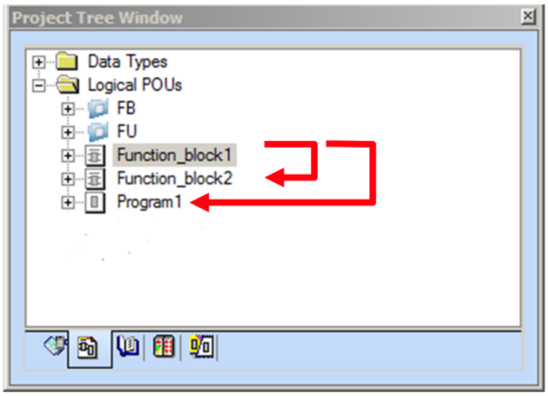
For programs, global variables are the only possibility to exchange data with other programming elements.

As with the other POU types, flags are also declared as local variables as long as their values are not of importance to the entire project.

Notes:

The Function Block

POE Type *Function Block*



Call

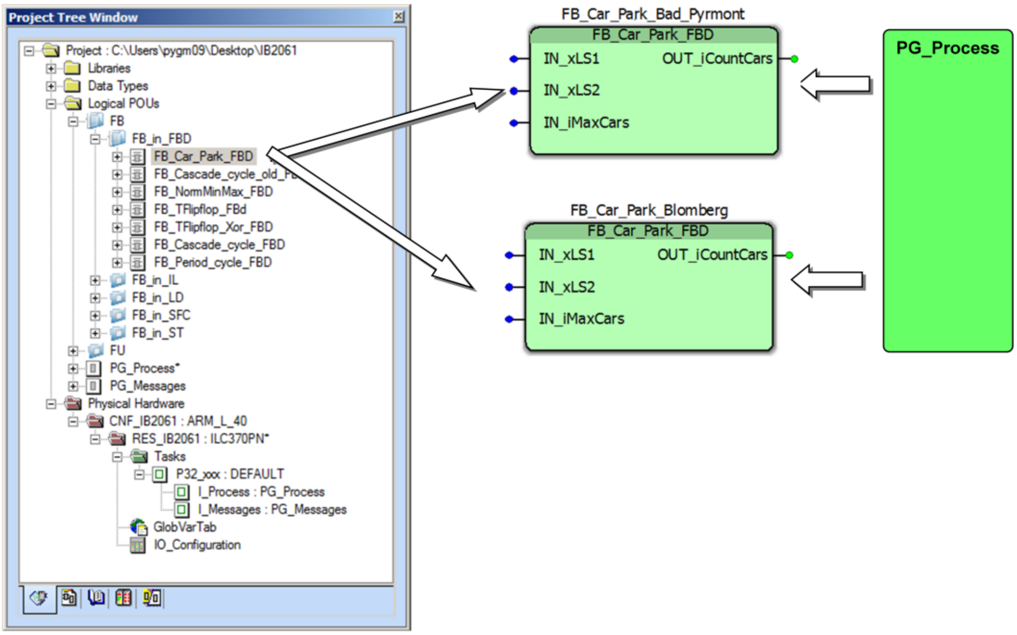
As parameterized elements, function blocks can be called in programs and other function blocks.

Creating function blocks (and functions) basically aims at two things. Firstly, once a program code has been created, it can simply be used again and again. Secondly, the parameterized POUs also offer the advantage of encapsulation. This means that a complicated programming is created, e.g., in a text-based language. The call including the connection of only few input and output values can than be carried out in a graphical language, which offers higher clarity.

As shown in the screenshot, function blocks can be called in other function blocks as well as in programs. A recursive call is not possible.

As function blocks can hold values over several program cycles, for each call of a function block, individual memory must be made available on the control system. This memory is called function block instance (cf. program instance) and can be compared to a complex variable. The function block instance is listed in the variable list of the POU in which the function block is called.

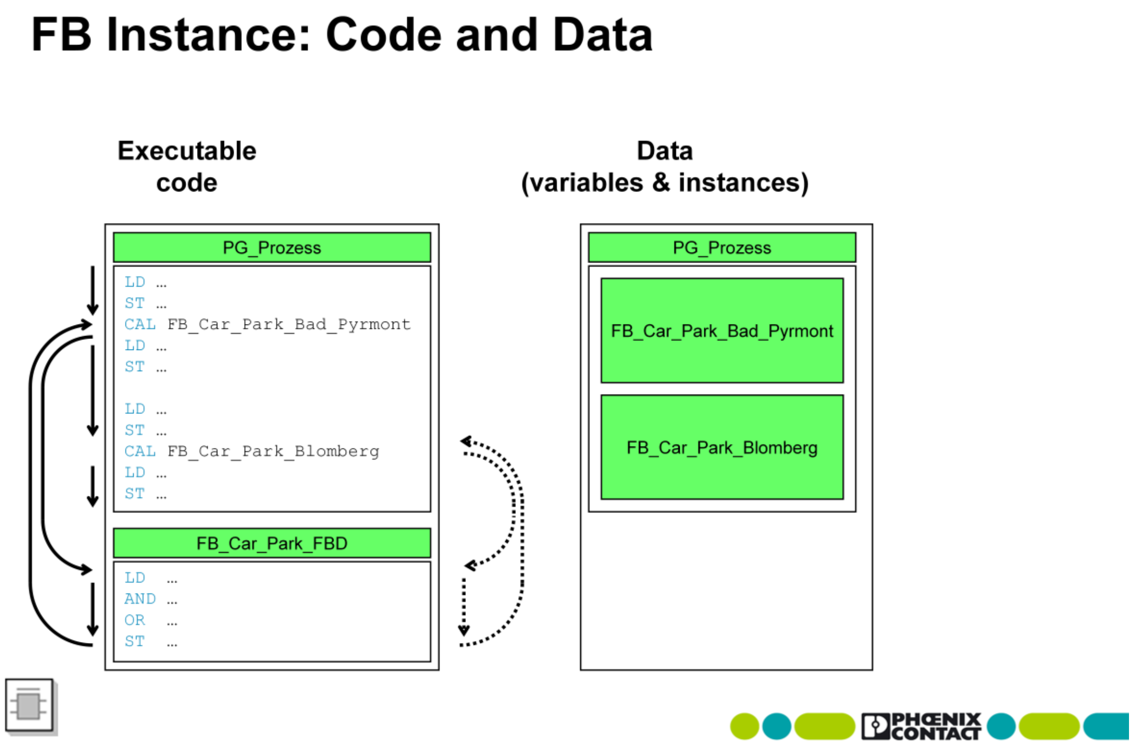
Instantiation of Function Blocks



Project Tree Window

```
Project : C:\Users\pygm09\Desktop\IB2061
  Libraries
  Data Types
  Logical POUs
    FB
      FB_in_FBD
        FB_Car_Park_FBD
        FB_Cascade_cycle_old_F
        FB_NormMinMax_FBD
        FB_TFlipflop_FBd
        FB_TFlipflop_Xor_FBD
        FB_Cascade_cycle_FBD
        FB_Period_cycle_FBD
      FB_in_IL
      FB_in_LD
      FB_in_SFC
      FB_in_ST
    FU
    PG_Process*
    PG_Messages
  Physical Hardware
    CNF_IB2061 : ARM_L_40
      RES_IB2061 : ILC370PN*
        Tasks
          P32_xxx : DEFAULT
            I_Process : PG_Process
            I_Messages : PG_Messages
        GlobVarTab
        IO_Configuration
```

The same algorithm type (FB_Car_Park_FBD) is called twice: The program PG_Process uses the Car_Park function block twice for different tasks. For both tasks, individual memory is made available (with the names FB_Car_Park_Bad_Pyrmont and FB_Car_Park_Blomberg).



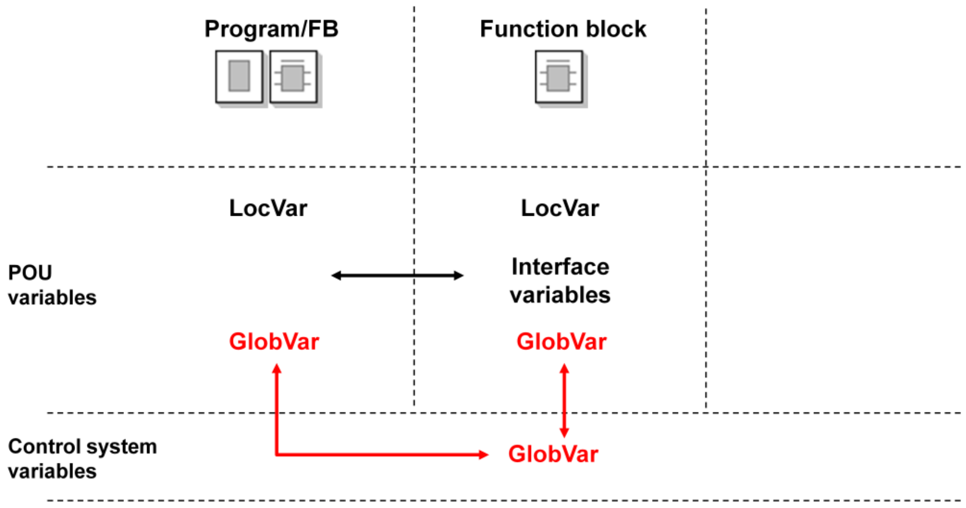
FB Instance: Code and Data

```
LD …
ST …
CAL FB_Car_Park_Bad_Pyrmont
LD …
ST …

LD …
ST …
CAL FB_Car_Park_Blomberg
LD …
ST …
```

```
LD  …
AND …
OR  …
ST  …
```

This overhead shows the two executions of the FB structure on different, discrete memories.

POU Type *Function Block*: Data Exchange



Program/FB

Function block

LocVar

LocVar

POU variables

Interface variables

GlobVar

GlobVar

Control system variables

GlobVar

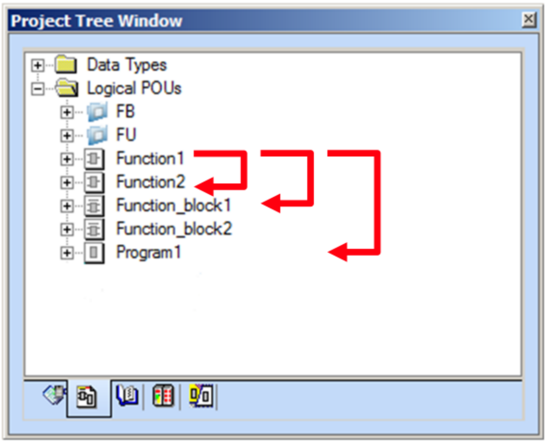
Regarding data exchange, function blocks offer the highest flexibility. According to the classic approach, data exchange is only carried out via input and output parameters (interface variables).

For specific applications, however, it might make sense to exchange data with the function block through the "backdoor", i.e. via global variables. The disadvantage for the programmer when using this method is that there is no direct transparency regarding the processed data during the call.

Notes:

The Function

Use of Functions



Call

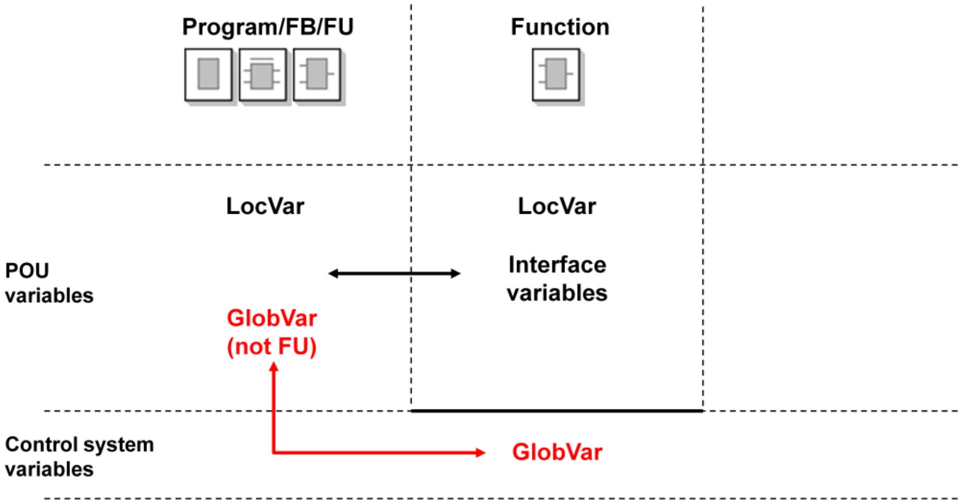
The functions are the second type of parameterized POUs. As shown in the graphic, they can be used in other functions, function blocks and programs. Just like with the function blocks, a recursive call is not possible.

Unlike the function blocks, functions are limited in so far that they cannot save data over several PLC cycles.

If a function block has to do exactly this, a function cannot be carried out. Functions are also disqualified if a block is to hold more than one output parameter as functions can only return a single result.

The advantage of functions compared to function blocks lies in the fact that a function does not have to be instantiated. For multiple use, this means that the same memory is used again and again for execution.

POU Type Function: Data Exchange



As regards to data exchange, functions are clearly defined. They have to provide at least one input parameter and return exactly one result. Internally, the return value of the function must be saved onto a parameter that is already available due to the creation of the function in the project tree (parameter name = function name).

Functions cannot access global variables.

Notes:

Data Management

Chapter 9

Notes:

Contents

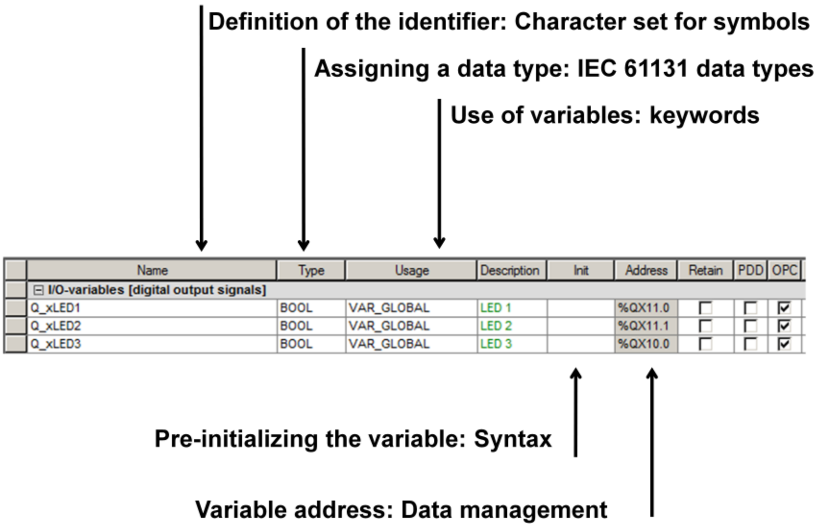
This chapter informs you on the parameters that determine a variable and the variable applications that can be used in the programming organization units according to IEC 61131. Furthermore, the connection between local, locally used global and global reference variables is described as well as the applications available in functions and function blocks.

For management and organization, variable groups and the variable dialog are available.

⚠	Attention!
ℹ	Information
💡	Tip

Notes:

Principle of Variable Declaration



Definition of the identifier: Character set for symbols

Assigning a data type: IEC 61131 data types

Use of variables: keywords

	Name	Type	Usage	Description	Init	Address	Retain	PDD	OPC
	⊟ I/O-variables [digital output signals]								
	Q_xLED1	BOOL	VAR_GLOBAL	LED 1		%QX11.0	□	□	☑
	Q_xLED2	BOOL	VAR_GLOBAL	LED 2		%QX11.1	□	□	☑
	Q_xLED3	BOOL	VAR_GLOBAL	LED 3		%QX10.0	□	□	☑

Pre-initializing the variable: Syntax

Variable address: Data management

According to IEC 61131, all upper and lower case letters (no Umlaut or ß), all numbers and the underscore as a separator or leading symbol can be used as a character set for variable names[1].

One important variable parameter is the data type. After installation of PC WORX, a variety of the data types defined by IEC 61131 are available and after declaration by the programmer, user-defined data types are available as well.

The use of a variable indicates the validity it has concerning the PLC and which function it might take on for data exchange of its POU with other POUs and the PLC.

The description of a variable is optional and can contain all characters.

The physical address on the control system presents the exchange point of the variable with the peripherals. For this, the IEC 61131 defines a fixed syntax:

Abstract syntax:
%<Data direction><Capacity prefix><ByteOffset[.BitPosition]>
Example: %QX4.0 (output bit 4.0)

A variable can be pre-initialized with a value (standard 0)
(for syntax refer to constant declaration)

[1] PC WORX offers the opportunity to enable the use of DIN designations. When this possibility is used, the programming of the POUs written in structured text must be unique.

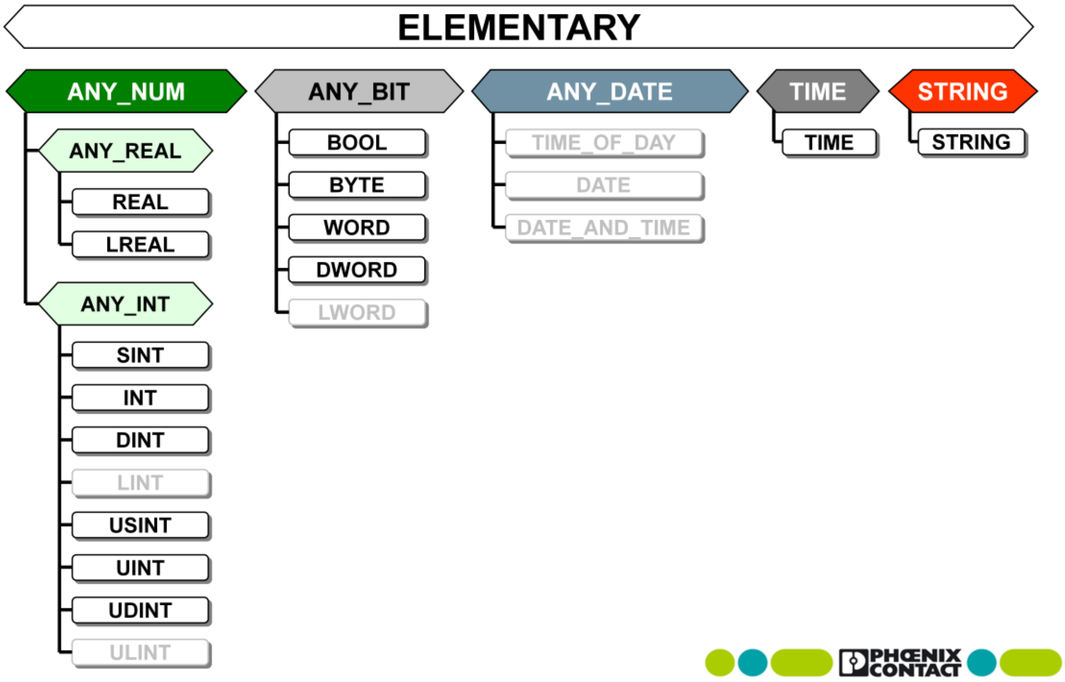
Menu *Extras* → *Options* → sheet *Code* → *Enable DIN-identifiers*

Notes:

Elementary Data Types in IEC 61131

Notes:

Hierarchy of Data Types



ELEMENTARY

- ANY_NUM
  - ANY_REAL
    - REAL
    - LREAL
  - ANY_INT
    - SINT
    - INT
    - DINT
    - LINT
    - USINT
    - UINT
    - UDINT
    - ULINT
- ANY_BIT
  - BOOL
  - BYTE
  - WORD
  - DWORD
  - LWORD
- ANY_DATE
  - TIME_OF_DAY
  - DATE
  - DATE_AND_TIME
- TIME
  - TIME
- STRING
  - STRING

A major part of the data types defined by IEC 61131 is available as a standard in PC WORX. These data types are divided into groups. Certain functions and function blocks (with overloaded input and output parameters) do not necessarily have to be equipped with the variable of a certain data type but with a data type from a group (e.g. function blocks of the boolean logic AND, OR, XOR, and NOT with ANY_BIT).

Numeric Data Types

Keyword	Data type	Size [Bits]	Value range
SINT	8 bit integer (with sign bit)	8	-128..127
INT	Integer (with sign bit)	16	-32768..32767
DINT	Double integer (with sign bit)	32	-2.147.483.648..2.147.483.647
LINT	64 bit integer (with sign bit)	64	-9.223.372.036.854.775.808.. 9.223.372.036.854.775.807
USINT	8 bit Integer (w/o sign bit)	8	0..255
UINT	Integer (w/o sign bit)	16	0..65.535
UDINT	Double integer (w/o sign bit)	32	0..4.294.967.295
ULINT	64 bit integer (w/o sign bit)	64	0..18.446.744.073.709.551.615
REAL	32-bit floating-point value	32	+/- 1.5*10^-45 .. +/- 3.4*10^38
LREAL	64-bit floating-point value	64	

Bit-Based Data Types

Keyword	Data type	Size [Bits]	Value range
BOOL	Bool	1	0..1_{HEX} (also false / true)
BYTE	Bit string 8	8	0..FF_{HEX}
WORD	Bit string 16	16	0..$FFFF_{HEX}$
DWORD	Bit string 32	32	0..FFFF $FFFF_{HEX}$
LWORD	Bit string 64	64	0..FFFF FFFF FFFF $FFFF_{HEX}$

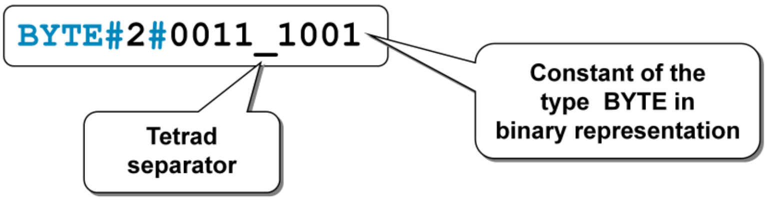
Extended Data Types

Keyword	Data type	Size [Bits]	Value range
TIME	Duration	32	0..4.294.967.295ms (cf. UDINT)
DATE	Date		
TOD	Time of day		
DT	Date and time		
STRING	String (standard)	80[bytes]	

Constants - Literals

Syntax of Constants

```
<Data type>#<Basis>#<Constant value><Unit>
```

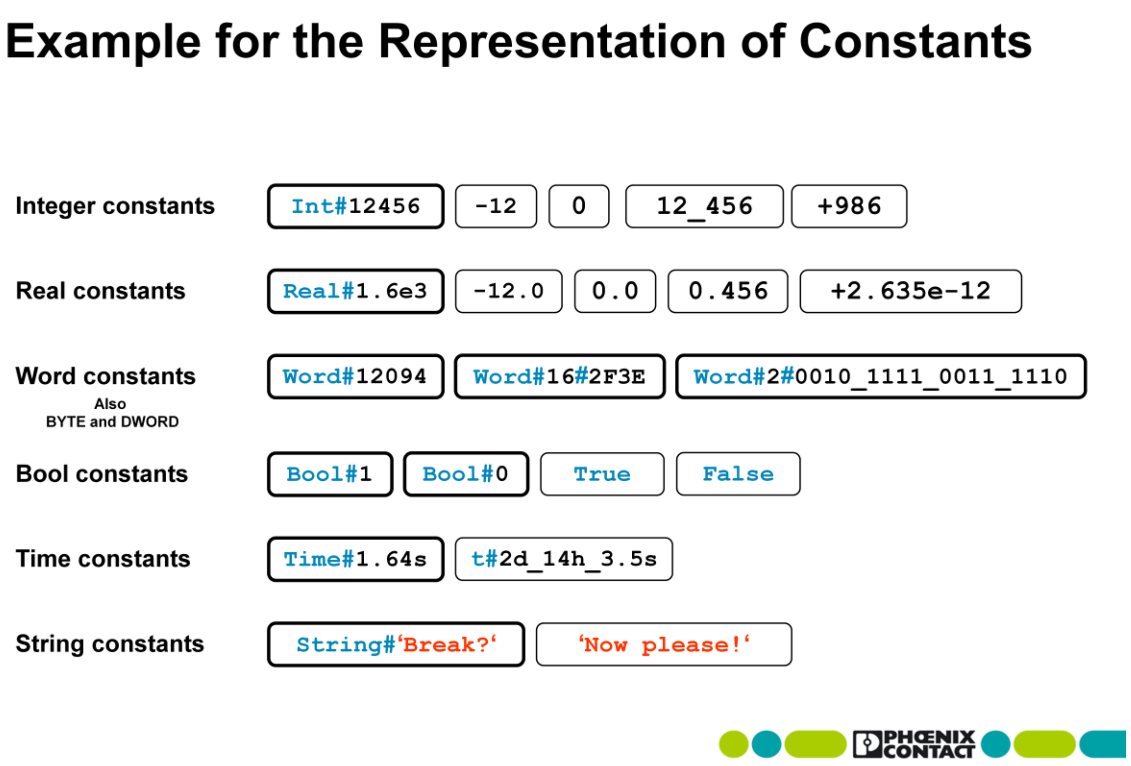


```
BYTE#2#0011_1001
```

The formally correct syntax for declaring constants (official designation of constants according to IEC 61131 is "literals") is as follows:

- Data type
- Basis (if other than *10*
  2 for a binary
  8 for an octal
  16 for a hexadecimally coded constant)
- Constant value
- Unit (only data type TIME), e.g. *s*, *ms*

As shown in the example, underscores can improve readability. They do not affect the constant value.



Example for the Representation of Constants

Integer constants	Int#12456	-12	0	12_456	+986
Real constants	Real#1.6e3	-12.0	0.0	0.456	+2.635e-12
Word constants Also BYTE and DWORD	Word#12094	Word#16#2F3E	Word#2#0010_1111_0011_1110		
Bool constants	Bool#1	Bool#0	True	False	
Time constants	Time#1.64s	t#2d_14h_3.5s			
String constants	String#‘Break?‘	‘Now please!‘			

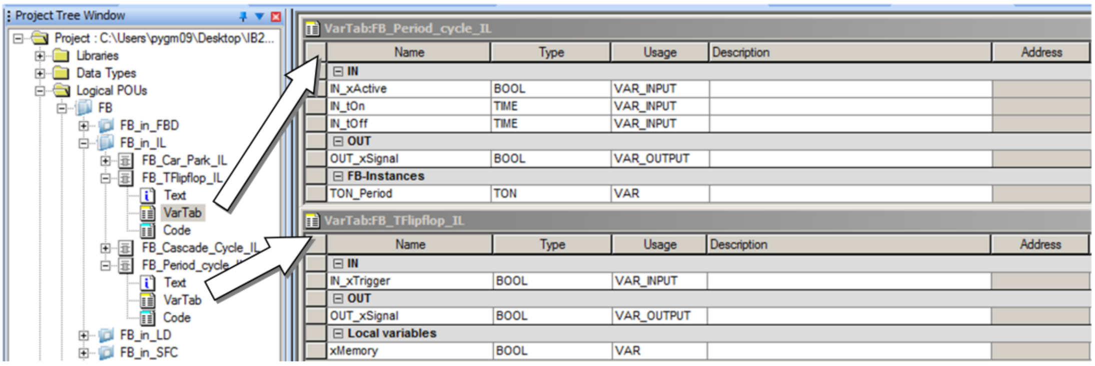
This page shows possible formats for constants. Please remember that for some data types it is allowed to use an abbreviated syntax that is often found in programming. The abbreviated form, e.g. like for the integer data type, is not allowed for the other members of the integer group.

Only in a special range of programming you will find the boolean *1* and *0* instead of *True* and *False*.

Notes:

Use of Variables

Local Data



VarTab:FB_Period_cycle_IL

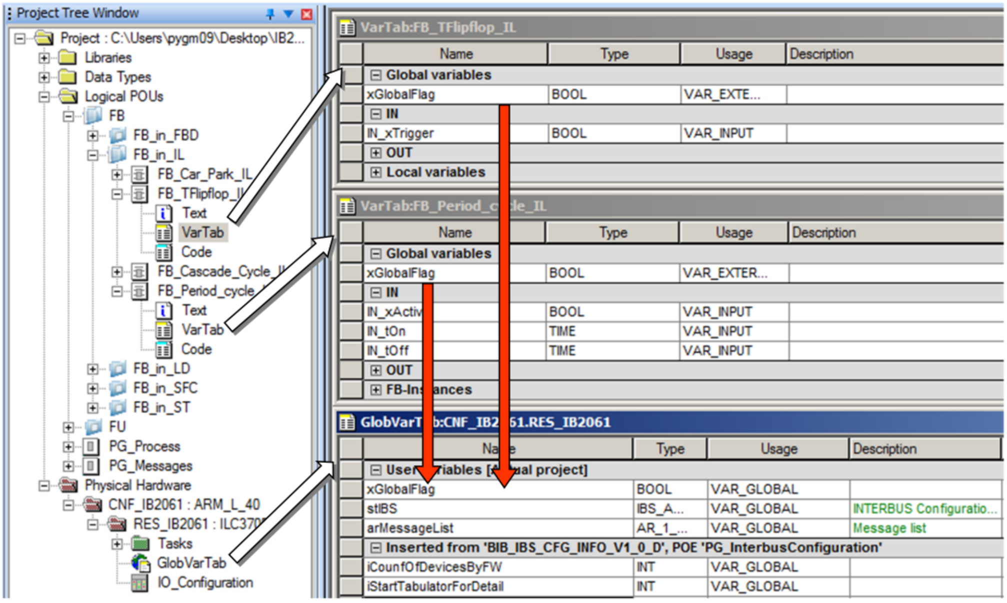
Name	Type	Usage	Description	Address
⊟ **IN**				
IN_xActive	BOOL	VAR_INPUT		
IN_tOn	TIME	VAR_INPUT		
IN_tOff	TIME	VAR_INPUT		
⊟ **OUT**				
OUT_xSignal	BOOL	VAR_OUTPUT		
⊟ **FB-Instances**				
TON_Period	TON	VAR		

VarTab:FB_TFlipflop_IL

Name	Type	Usage	Description	Address
⊟ **IN**				
IN_xTrigger	BOOL	VAR_INPUT		
⊟ **OUT**				
OUT_xSignal	BOOL	VAR_OUTPUT		
⊟ **Local variables**				
xMemory	BOOL	VAR		

The data encapsulation in IEC 61131 follows the example of organization in high-level languages. Using this method, two variables of the same name are handled in two different POUs and therefore completely discretely.

Global Data



VarTab:FB_TFlipflop_IL

Name	Type	Usage	Description
⊟ Global variables			
xGlobalFlag	BOOL	VAR_EXTE...	
⊟ IN			
IN_xTrigger	BOOL	VAR_INPUT	
⊞ OUT			
⊞ Local variables			

VarTab:FB_Period_c le_IL

Name	Type	Usage	Description
⊟ Global variables			
xGlobalFlag	BOOL	VAR_EXTER...	
⊟ IN			
IN_xActiv	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊞ OUT			
⊞ FB-Ins ances			

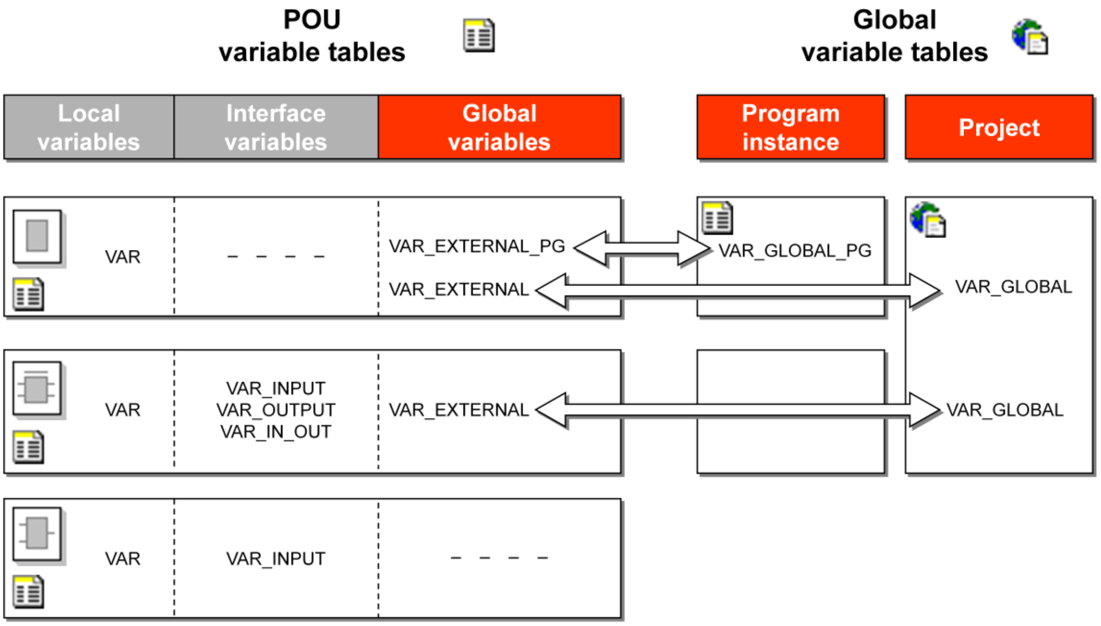
GlobVarT b:CNF_IB2 61.RES_IB2061

Na e	Type	Usage	Description
⊟ User riables [ual project]			
xGlobalFlag	BOOL	VAR_GLOBAL	
stIBS	IBS_A...	VAR_GLOBAL	INTERBUS Configuratio...
arMessageList	AR_1_...	VAR_GLOBAL	Message list
⊟ Inserted from 'BIB_IBS_CFG_INFO_V1_0_D', POE 'PG_InterbusConfiguration'			
iCounfOfDevicesByFW	INT	VAR_GLOBAL	
iStartTabulatorForDetail	INT	VAR_GLOBAL	

Global data are centrally held on the global variable table. Data storage is only carried out there. Local use of such variables refers to this global reference.

Notes:

Global Variables • Local Variables



POU variable tables			Global variable tables	
Local variables	Interface variables	Global variables	Program instance	Project
VAR	– – – –	VAR_EXTERNAL_PG VAR_EXTERNAL	VAR_GLOBAL_PG	VAR_GLOBAL
VAR	VAR_INPUT VAR_OUTPUT VAR_IN_OUT	VAR_EXTERNAL		VAR_GLOBAL
VAR	VAR_INPUT	– – – –		

Until this point, the documentation has been limited to the differentiation between the local and global variables in the POU area.

As an addition it has to be said that the global reference variable has the designation VAR_GLOBAL while the variable used in programming and entered in the POU has the designation VAR_EXTERNAL.

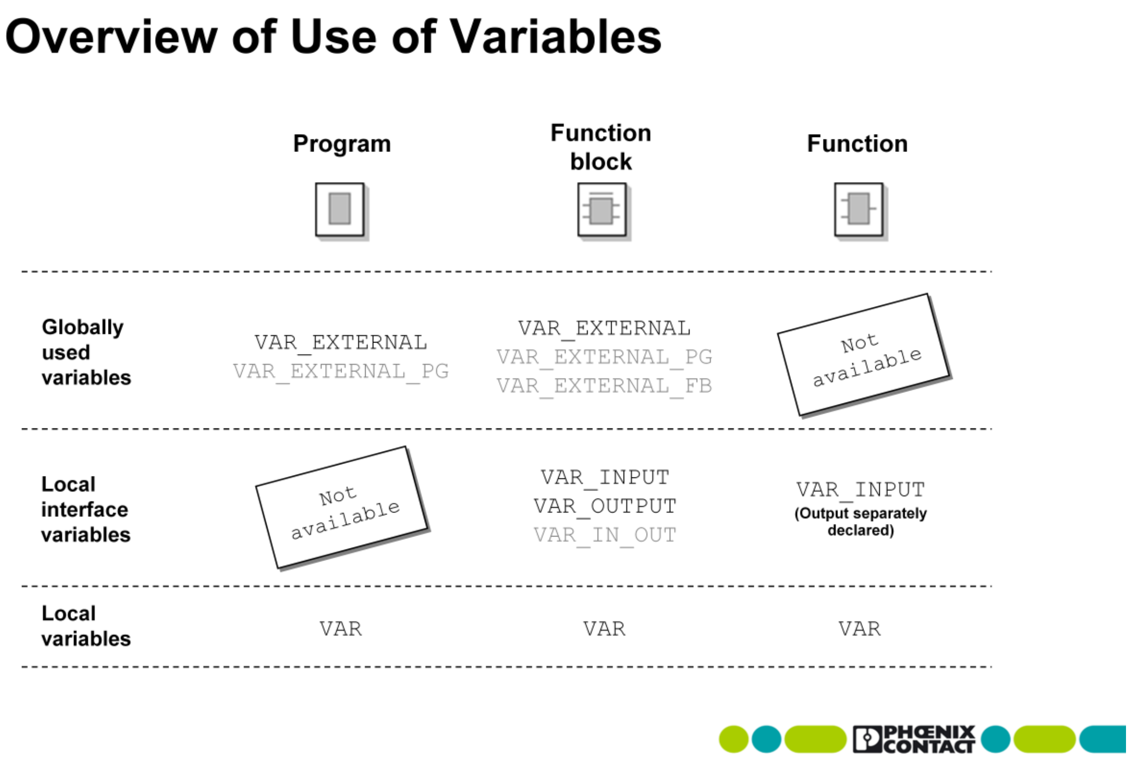
Thus, changing a created variable at a later point only makes sense if the changes are implemented in the central data storage of the global variable table (PC WORX supports the synchronization of the variable entered in the POU after changes have been made). Regarding organization, the variable table is assigned to the resource and therefore is part of the hardware structure.

In a local variable table, local and global variables that are to be linked to one another in the program in the POU are entered. Therefore *Local variable table* is not the designation for a local variable table, but for the local table belonging to a POU for local and locally used global variables.

Further information

In the context of global variables, the variables of the use VAR_EXTERNAL_PG and VAR_GLOBAL_PG present a special case. These variables have all the features global variables have. They can therefore be connected to process points of the bus systems. However, for every program instance that is created for a program with such variables, an individual address area is reserved. Thus, a differentiated coupling with the hardware is achieved.

Detailed information and examples regarding the issue of *Multiple instantiation of programs* are dealt with in the *PC WORX Basics of Communication* course.



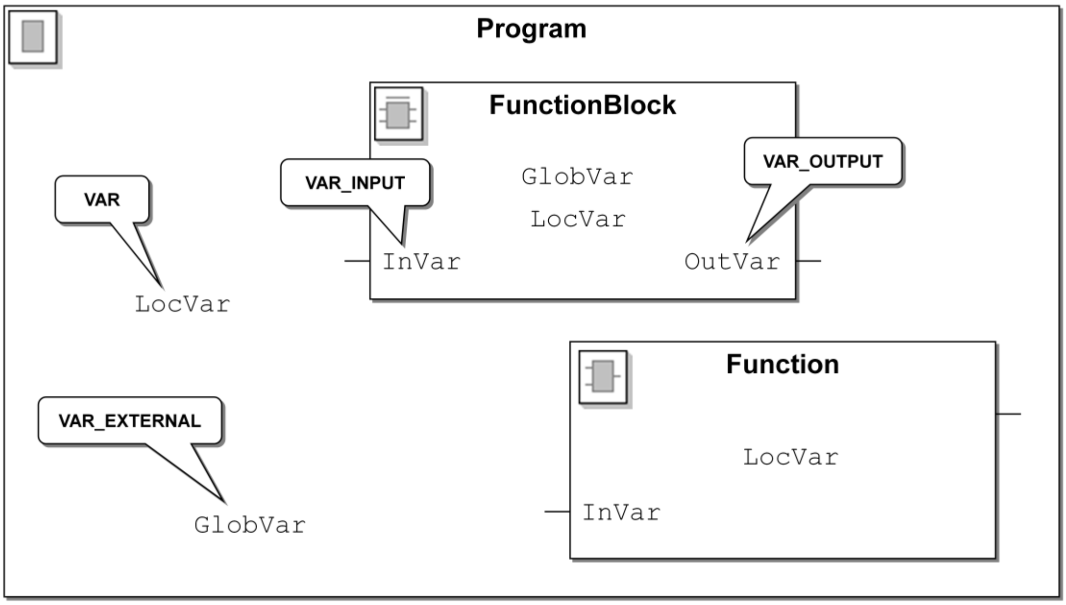
Overview of Use of Variables

	Program	Function block	Function
Globally used variables	VAR_EXTERNAL VAR_EXTERNAL_PG	VAR_EXTERNAL VAR_EXTERNAL_PG VAR_EXTERNAL_FB	Not available
Local interface variables	Not available	VAR_INPUT VAR_OUTPUT VAR_IN_OUT	VAR_INPUT (Output separately declared)
Local variables	VAR	VAR	VAR

In addition to the already described applications, this graphic shows the possibilities of use in functions and function blocks.

The use *VAR_IN_OUT*, which is displayed in gray for function blocks, is especially interesting for the transmission of very large amounts of data (field and structure variables) as with this use it is not necessary to copy data; only a pointer operation is carried out.

Graphical Overview of Use of Variables



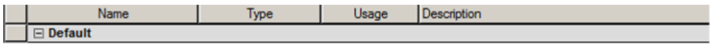
This graphic shows the most common uses.

Notes:

Managing variables

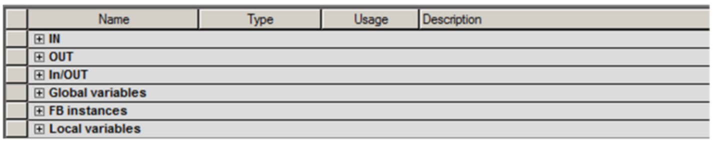
Variable Groups

Standard group



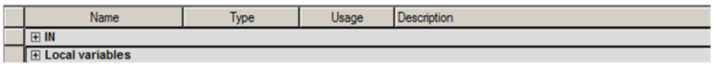
Name	Type	Usage	Description
⊟ Default			

Groups for function blocks



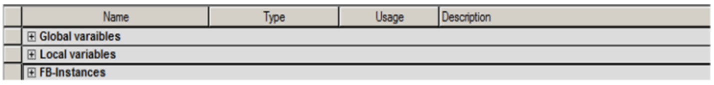
Name	Type	Usage	Description
⊞ IN			
⊞ OUT			
⊞ In/OUT			
⊞ Global variables			
⊞ FB instances			
⊞ Local variables			

Groups for functions



Name	Type	Usage	Description
⊞ IN			
⊞ Local variables			

Groups for programs



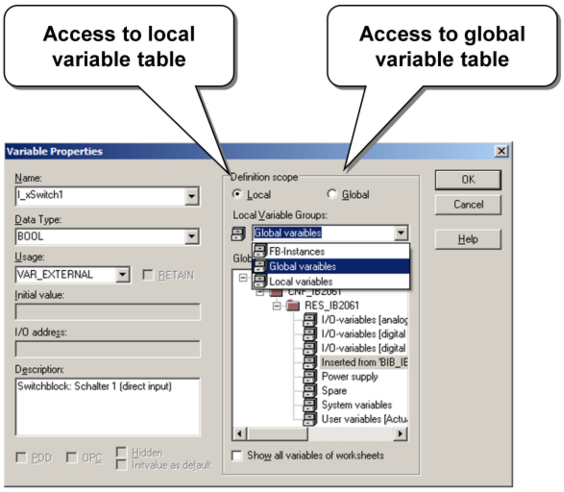
Name	Type	Usage	Description
⊞ Global varaibles			
⊞ Local variables			
⊞ FB-Instances			

In order to achieve a more clearly arranged variable structure within the variable tables, it is recommended to use variable groups. They do not affect programming, but simply serve as a means for better organization and more efficiency when working with variables.

The *Default* group is available as a standard when a worksheet is created. Depending on the POU type, the groups shown in the screenshot have proven to be a good basis. In more complex projects, the use of further, function-related groups will make sense.

This type of grouping is not only suitable for local variable tables but also for global variable tables. The *System Variables* group, which contains the system variables, can also be renamed.

Variable Dialog



Access to local variable table

Access to global variable table

Variable Properties

Name: I_xSwitch1

Data Type: BOOL

Usage: VAR_EXTERNAL

Initial value:

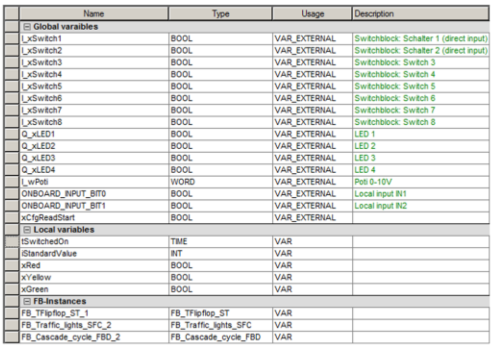
I/O address:

Description: Switchblock: Schalter 1 (direct input)

Definition scope: ● Local ○ Global

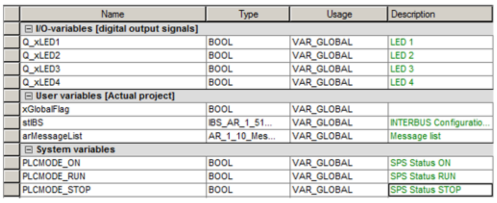
Local Variable Groups: Global varaibles

Local variable table



Name	Type	Usage	Description
⊟ Global varaibles			
I_xSwitch1	BOOL	VAR_EXTERNAL	Switchblock: Schalter 1 (direct input)
I_xSwitch2	BOOL	VAR_EXTERNAL	Switchblock: Schalter 2 (direct input)
I_xSwitch3	BOOL	VAR_EXTERNAL	Switchblock: Switch 3
I_xSwitch4	BOOL	VAR_EXTERNAL	Switchblock: Switch 4
I_xSwitch5	BOOL	VAR_EXTERNAL	Switchblock: Switch 5
I_xSwitch6	BOOL	VAR_EXTERNAL	Switchblock: Switch 6
I_xSwitch7	BOOL	VAR_EXTERNAL	Switchblock: Switch 7
I_xSwitch8	BOOL	VAR_EXTERNAL	Switchblock: Switch 8
Q_xLED1	BOOL	VAR_EXTERNAL	LED 1
Q_xLED2	BOOL	VAR_EXTERNAL	LED 2
Q_xLED3	BOOL	VAR_EXTERNAL	LED 3
Q_xLED4	BOOL	VAR_EXTERNAL	LED 4
I_wPoti	WORD	VAR_EXTERNAL	Poti 0-10V
ONBOARD_INPUT_BIT0	BOOL	VAR_EXTERNAL	Local input IN1
ONBOARD_INPUT_BIT1	BOOL	VAR_EXTERNAL	Local input IN2
xCfgReadStart	BOOL	VAR_EXTERNAL	
⊟ Local variables			
tSwitchedOn	TIME	VAR	
iStandardValue	INT	VAR	
xRed	BOOL	VAR	
xYellow	BOOL	VAR	
xGreen	BOOL	VAR	
⊟ FB-Instances			
FB_TFlipflop_ST_1	FB_TFlipflop_ST	VAR	
FB_Traffic_lights_SFC_2	FB_Traffic_lights_SFC	VAR	
FB_Cascade_cycle_FBD_2	FB_Cascade_cycle_FBD	VAR	

Global variable table



Name	Type	Usage	Description
⊟ I/O-variables [digital output signals]			
Q_xLED1	BOOL	VAR_GLOBAL	LED 1
Q_xLED2	BOOL	VAR_GLOBAL	LED 2
Q_xLED3	BOOL	VAR_GLOBAL	LED 3
Q_xLED4	BOOL	VAR_GLOBAL	LED 4
⊟ User variables [Actual project]			
xGlobalFlag	BOOL	VAR_GLOBAL	
stIBS	IBS_AR_1_51...	VAR_GLOBAL	INTERBUS Configuratio...
arMessageList	AR_1_10_Mes...	VAR_GLOBAL	Message list
⊟ System variables			
PLCMODE_ON	BOOL	VAR_GLOBAL	SPS Status ON
PLCMODE_RUN	BOOL	VAR_GLOBAL	SPS Status RUN
PLCMODE_STOP	BOOL	VAR_GLOBAL	SPS Status STOP

The variable dialog allows the access to variables in the corresponding local variable tables and to the global variable tables. The dialog also enables viewing and changing existing variables and declaring new ones.

If global variables are to be viewed and declared, the global variable group has to be selected in addition to the local variable group.

Notes:

Programming languages according to IEC61131-3

Chapter 10

Notes:

Contents

This chapter offers a quick overview of the five languages according to IEC 61131 and their specific advantages.

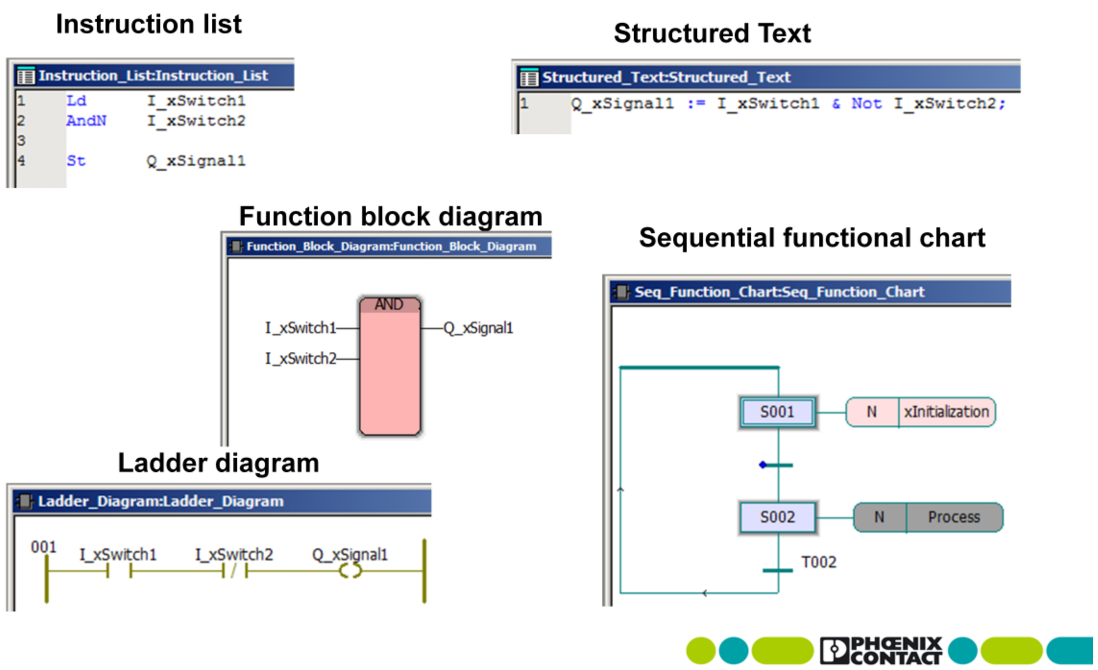
⚠ Attention!

ℹ Information

💡 Tip

Notes:

The IEC61131-3 Languages



Instruction list

Instruction_List:Instruction_List

```
Ld      I_xSwitch1
AndN    I_xSwitch2

St      Q_xSignal1
```

Structured Text

Structured_Text:Structured_Text

```
Q_xSignal1 := I_xSwitch1 & Not I_xSwitch2;
```

Function block diagram

Sequential functional chart

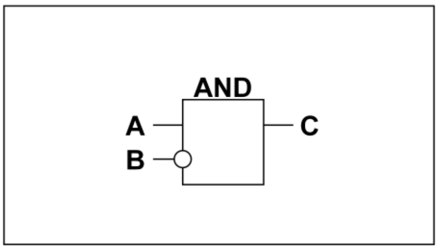
Ladder diagram

Function_Block_Diagram:Function_Block_Diagram
AND
I_xSwitch1
I_xSwitch2
Q_xSignal1
Seq_Function_Chart:Seq_Function_Chart
S001
N
xInitialization
S002
N
Process
T002
Ladder_Diagram:Ladder_Diagram
001
I_xSwitch1
I_xSwitch2
Q_xSignal1

As a standard, PC WORX offers the five languages described in IEC 61131. The additional Machine Sequential Function Chart Language and FFLD (Fix Format Ladder Diagram) are not available.

Function Block Diagram (FBD)

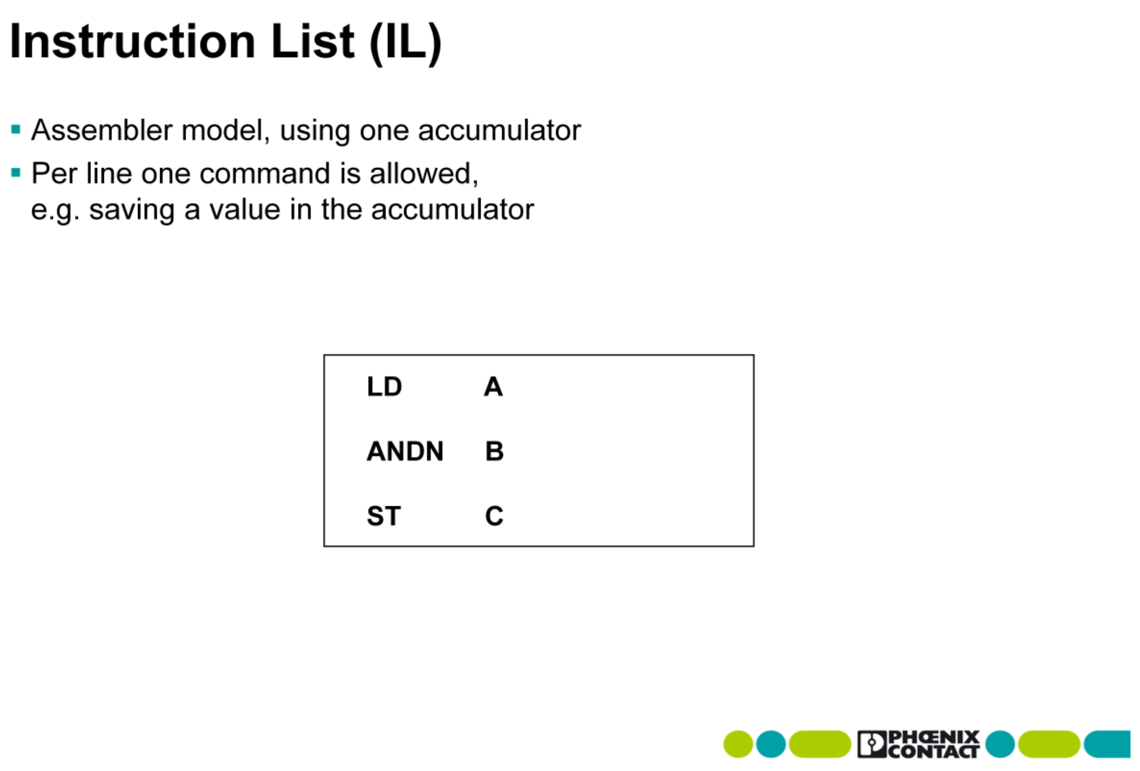
- Graphical language, widely used in Europe
- Programming elements in form of function blocks
- Function blocks can be “wired“ similarly as in a circuit diagram
- Used in a variety of applications responsible for the flow of information between control system components.



Ladder Diagram (LD)

- Standardized, limited set of programming symbols for relay control systems
- Based on the North-American programming style, similar to the US standard for drawing circuit diagrams

```
     A         B                    C
--|    |----|  /  |--------------(     )--
```



Instruction List (IL)

- Assembler model, using one accumulator
- Per line one command is allowed,
  e.g. saving a value in the accumulator

```
LD      A

ANDN    B

ST      C
```

Structured Text (ST)

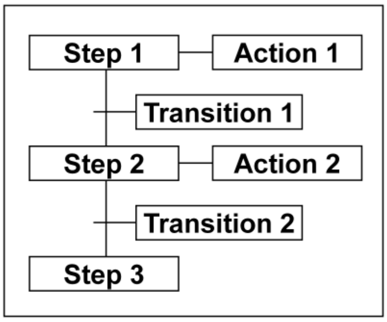
- High-level language, structured through subprograms
- Syntax similar to PASCAL
- Complex and nested instructions

```
C := A AND NOT B;
```

Notes:

Sequential Function Chart (SFC)

- Powerful, graphical programming language for describing the sequence behavior of control programs
- Used for structuring control programs
- Clearly arranged programming language, allows for quick diagnostics
- Basic elements: Steps with action blocks and transitions
- Supports alternative and parallel sequences



Step 1

Action 1

Transition 1

Step 2

Action 2

Transition 2

Step 3

Notes:

Programming in PC WORX

Chapter 11

Notes:

Contents

This chapter gives basic information on the project structure in PC WORX. This includes inserting and handling programming organization units as well as the presentation of the IEC 61131 standard functions and function blocks.

⚠ Attention!

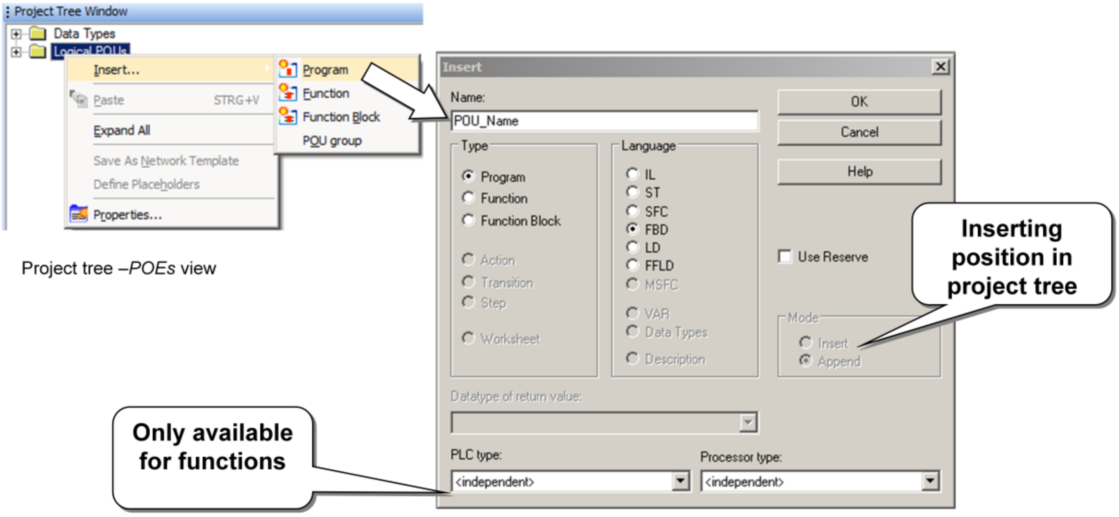
ℹ Information

💡 Tip

Notes:

Program Organization Units

Inserting POU



Name: POU_Name

Type: ● Program ○ Function ○ Function Block ○ Action ○ Transition ○ Step ○ Worksheet

Language: ○ IL ○ ST ○ SFC ● FBD ○ LD ○ FFLD ○ MSFC ○ VAR ○ Data Types ○ Description

□ Use Reserve

Mode: ○ Insert ● Append

PLC type: <independent>

Processor type: <independent>

Project tree –*POEs* view

Inserting position in project tree

Only available for functions

If the *Main* program, which is automatically inserted via the project template, is not to be used, the first step for creating a programming is inserting a POU of the type *Program*. When an element *Logical POUs* or an existing POU is selected, then inserting a POU can be done via the context menu or the menu bar.

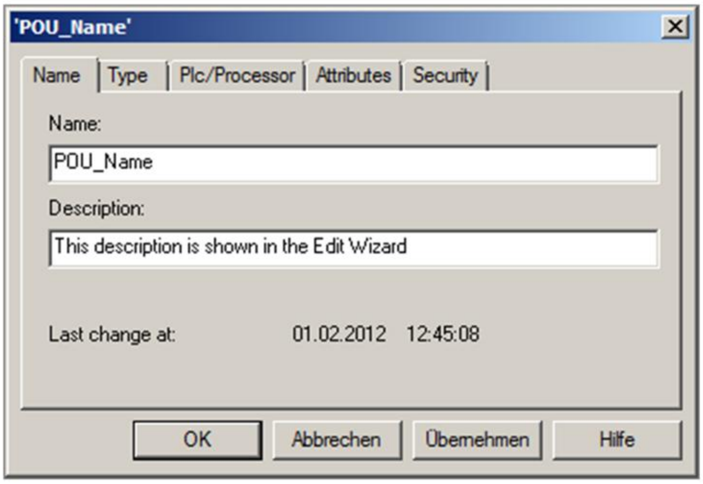
Depending on the element to be inserted (frame *Type*), several options are available. For example, not all languages are selectable for every POU type (functions do not support sequential function charts, programs do not support the Machine Sequential Function Chart Language).

The option *Datatype of return value* is only active for functions. As already described in section *Variable structure according to IEC 61131*, functions only have one output parameter. The data type of this parameter is determined via this selection, its name via the POU name.

The selection <*independent*> for PLC and processor type only has to be adapted if PLC- or processor-specific functions and function blocks are to be used. If this option is used, this POU can only be operated on a PLC or processor of the determined type.

The position of a POU in the project tree does not affect the execution order. The call of a program in the form of a program instance through a task, or the call of functions and function blocks in code worksheets determines the program sequence.

POU Properties


'POU_Name'

Name: POU_Name

Description: This description is shown in the Edit Wizard

Last change at: 01.02.2012 12:45:08

With the exception of the language, the POU properties can be subsequently adapted in the POU properties dialog.

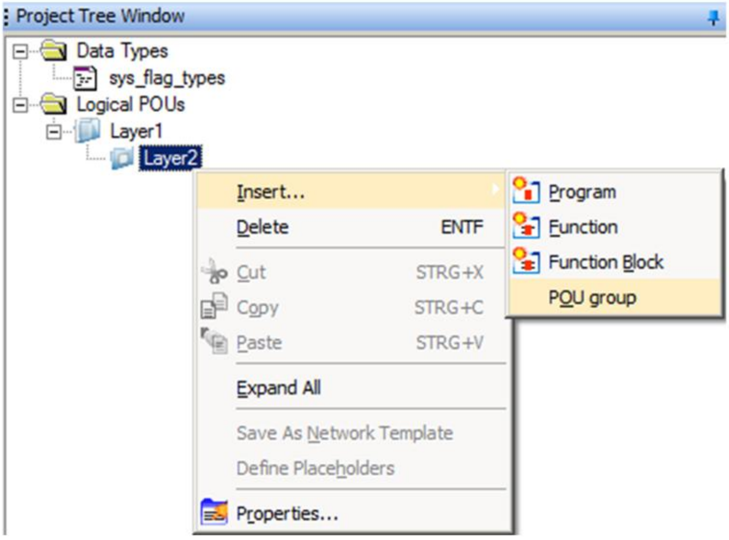
A change of type is possible, but, depending on the frequency of use of the POU, this means more adaptation effort for the project. The reserve refers to the memory reserved for the POU during the download for subsequent changes. As a standard, the resource (hardware structure) is set in a way so that memory is available for every POU and an individual reserve is not necessary.

About the *PLC/Processor* tab.

The *Read only* attribute is used for the multi-user function, but can also be used as a protection against involuntary changes.

The security settings for a POU allow for the expertise protection for programming. The protection is only activated after entering a password via the menu *File* → *Enter password*.

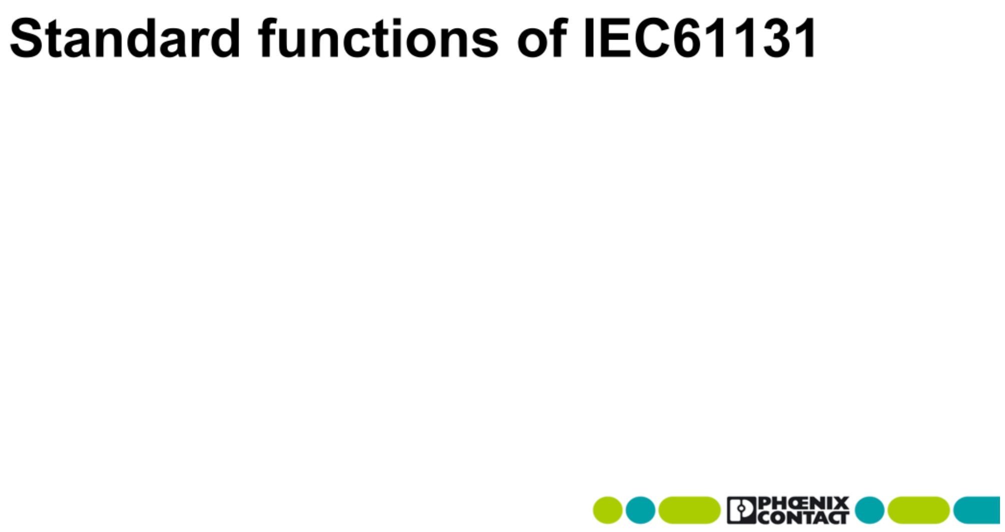
POU Groups



In order to keep the program organization units that you created yourself clearly arranged in your project, it is recommended to use POU groups once a certain project size has been reached.

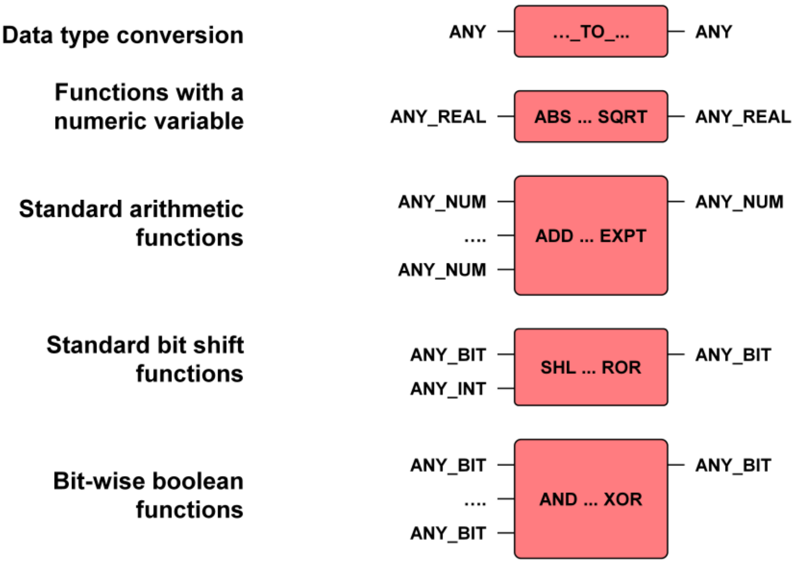
The POU groups do not affect the programming assignments. They only serve as a means for better organization and can be created in as complex an order as you like.

POU groups are always listed as the highest elements in the *Logical POU* folder and appear in alphabetical order on one level.



Standard functions of IEC61131

Standard Functions 1



Data type conversion: ANY — …_TO_… — ANY

Functions with a numeric variable: ANY_REAL — ABS ... SQRT — ANY_REAL

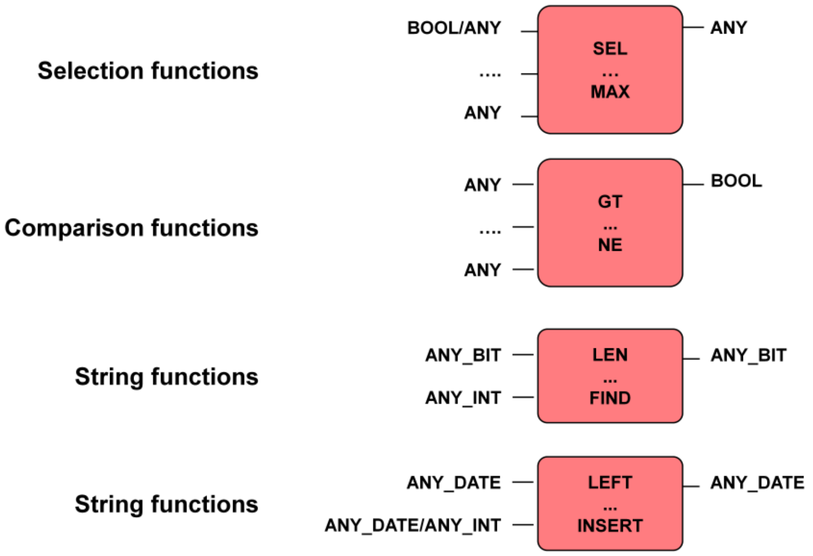
Standard arithmetic functions: ANY_NUM, …., ANY_NUM — ADD ... EXPT — ANY_NUM

Standard bit shift functions: ANY_BIT, ANY_INT — SHL ... ROR — ANY_BIT

Bit-wise boolean functions: ANY_BIT, …., ANY_BIT — AND ... XOR — ANY_BIT

After the installation of PC WORX, a variety of functions is available. These can be viewed via the edit wizard. They are defined by IEC 61131. Due to the variety of data types, especially the group of type converting functions is very large.

Standard Functions 2



Selection functions

BOOL/ANY

....

ANY

SEL
...
MAX

ANY

Comparison functions

ANY

....

ANY

GT
...
NE

BOOL

String functions

ANY_BIT

ANY_INT

LEN
...
FIND

ANY_BIT

String functions

ANY_DATE

ANY_DATE/ANY_INT

LEFT
...
INSERT

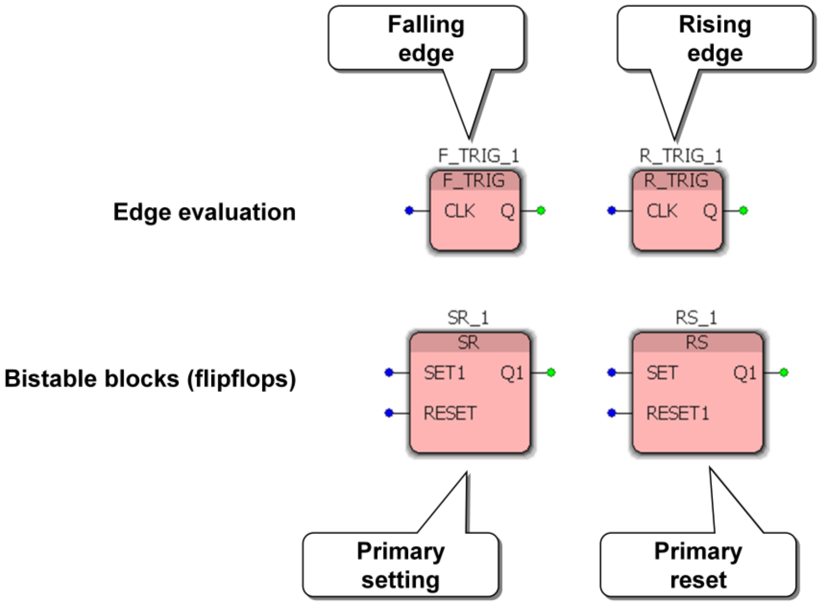
ANY_DATE

Details regarding the connection and function of the blocks can be obtained from the individual HTML helps for each block in PC WORX.

Notes:

Standard Function Blocks of IEC61131

Standard Function Blocks 1



Falling edge

Rising edge

Edge evaluation

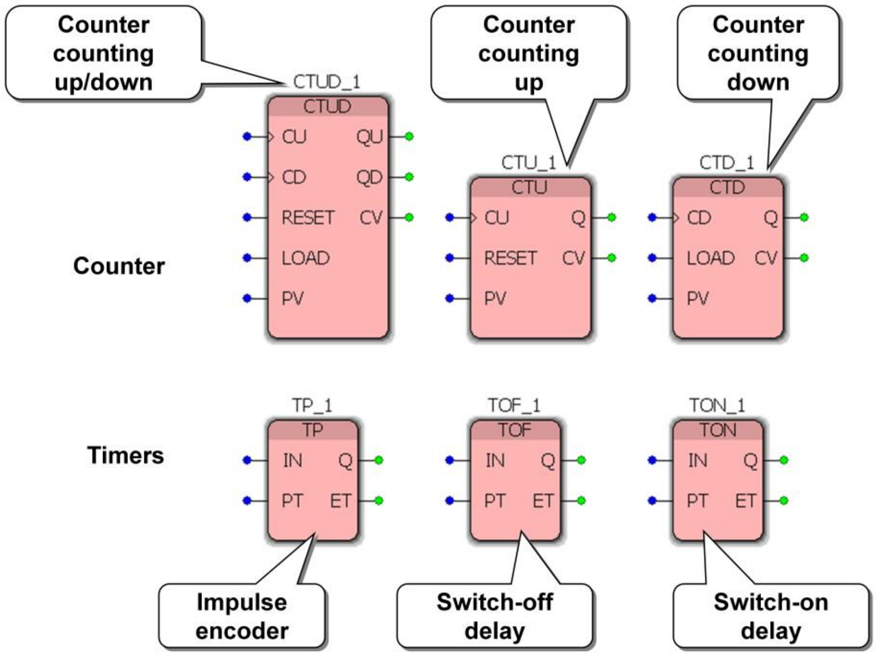
Bistable blocks (flipflops)

Primary setting

Primary reset

IEC 61131 defines function blocks that can be divided into four groups as shown above. As regards to the functions, there are deviations from blocks of classic PLC programming systems. One example for this is the fact that the RS flipflop is dominant in the reset and the SR flipflop dominant in the setting.

Standard Function Blocks 2


Counter counting up/down

Counter counting up

Counter counting down

Counter

Timers

Impulse encoder

Switch-off delay

Switch-on delay

Further information on wiring and function of function blocks and functions can be obtained via the PC WORX HTML help.

Notes:

FBD – Function Block Diagram

Chapter 12

Notes:

Contents

This chapter describes the programming in function block diagram. This includes inserting functions and function blocks as well as accessing variables of different validities.

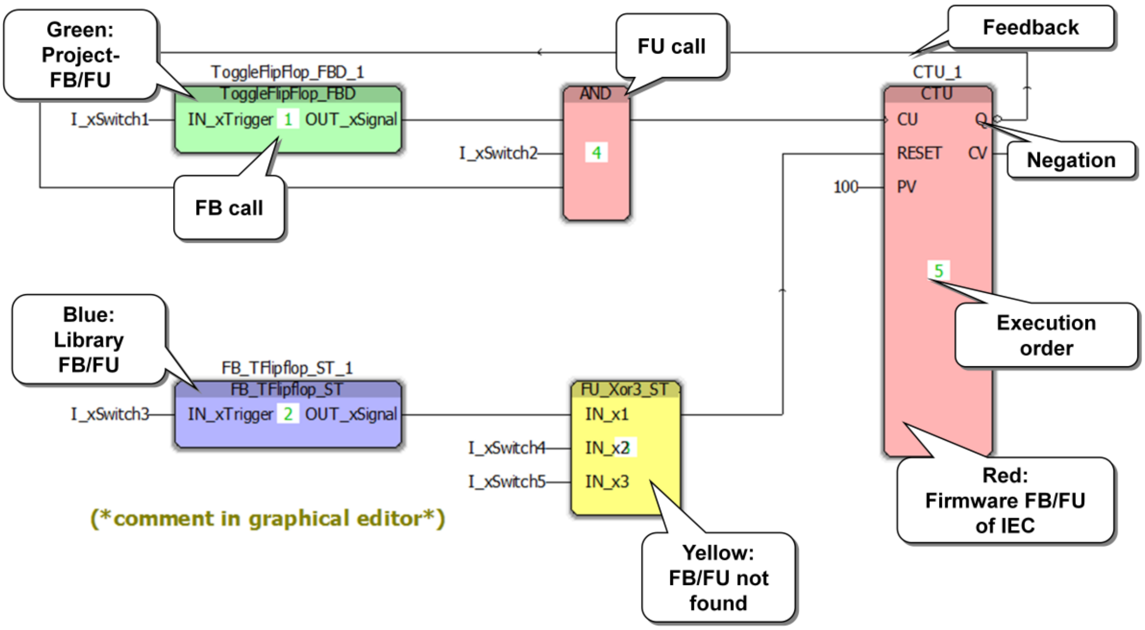
⚠ Attention!

ℹ Information

💡 Tip

Notes:

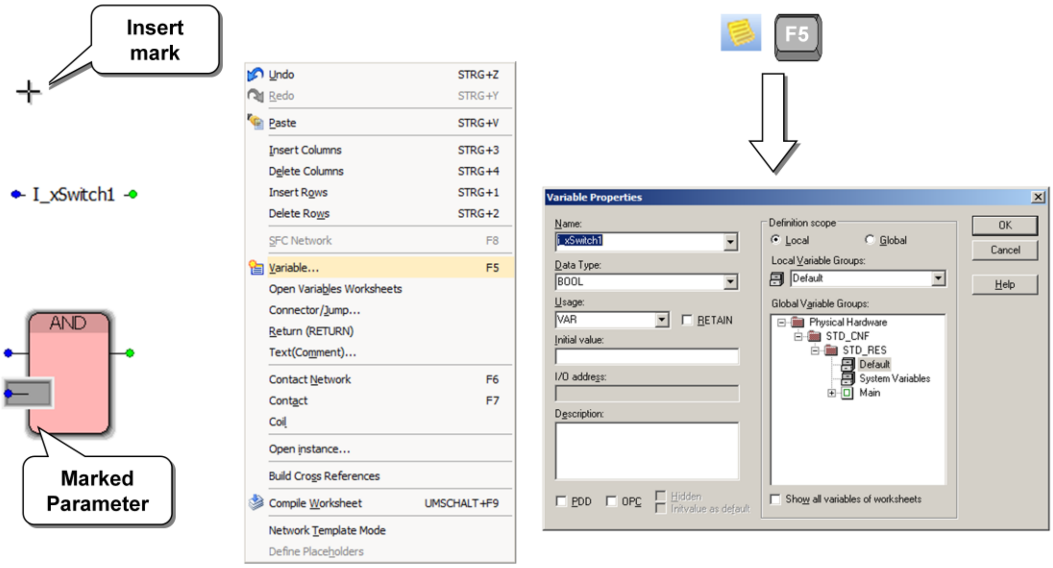
Elements of Function Block Diagram



This overhead gives an overview of the language elements available in function block diagram. Basically, these are blocks and variables.

Notes:

Accessing Variables



Name: I_xSwitch1

Data Type: BOOL

Usage: VAR ☐ RETAIN

Initial value:

I/O address:

Description:

☐ PDD ☐ OPC ☐ Hidden ☐ Initvalue as default

Definition scope: ● Local ○ Global

Local Variable Groups: Default

Global Variable Groups:

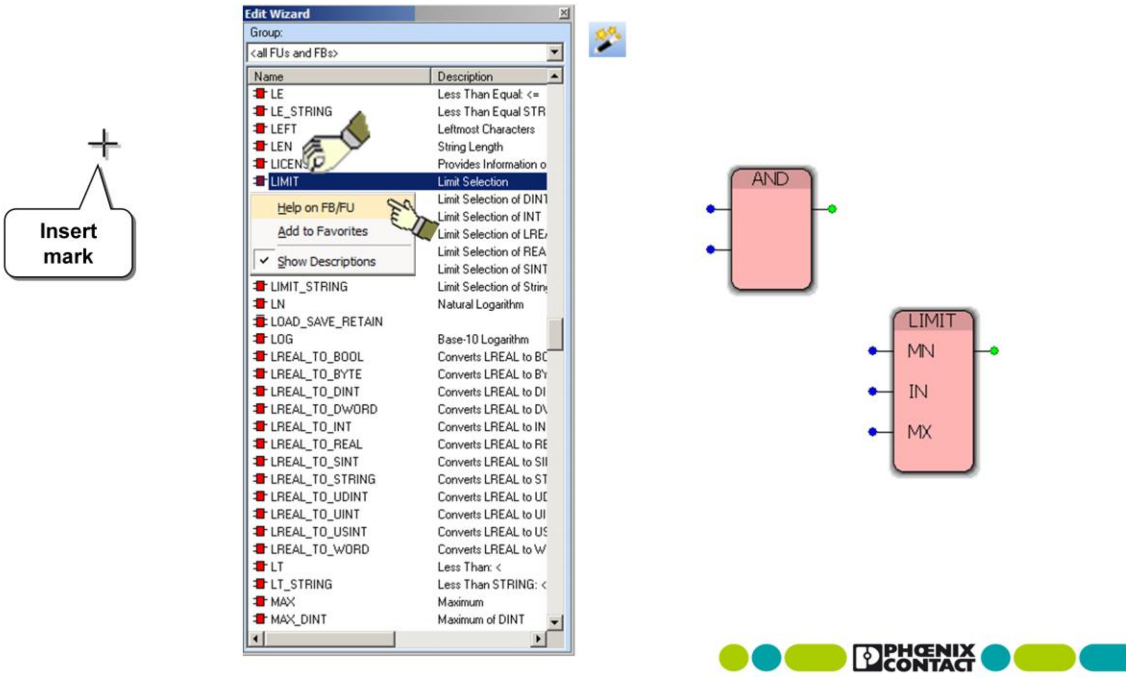
☐ Show all variables of worksheets

In FBD, inserting a variable is carried out via the variable dialog. This dialog offers access to local and global variables.

In the free FBD editor, first the insert mark has to be set or a connection point of a function or function block has to be marked before the dialog can be shown.

The dialog can be activated via the context menu, the button shown above or via the F5 function button (default setting).

Accessing Functions



Edit Wizard

Group:

<all FUs and FBs>

Name	Description
LE	Less Than Equal: <=
LE_STRING	Less Than Equal STR
LEFT	Leftmost Characters
LEN	String Length
LICENS	Provides Information o
LIMIT	Limit Selection
	Limit Selection of DINT
	Limit Selection of INT
	Limit Selection of LRE/
	Limit Selection of REA
	Limit Selection of SINT
LIMIT_STRING	Limit Selection of Strin
LN	Natural Logarithm
LOAD_SAVE_RETAIN	
LOG	Base-10 Logarithm
LREAL_TO_BOOL	Converts LREAL to BC
LREAL_TO_BYTE	Converts LREAL to BY
LREAL_TO_DINT	Converts LREAL to DI
LREAL_TO_DWORD	Converts LREAL to D\
LREAL_TO_INT	Converts LREAL to IN
LREAL_TO_REAL	Converts LREAL to RE
LREAL_TO_SINT	Converts LREAL to SII
LREAL_TO_STRING	Converts LREAL to ST
LREAL_TO_UDINT	Converts LREAL to UE
LREAL_TO_UINT	Converts LREAL to UI
LREAL_TO_USINT	Converts LREAL to US
LREAL_TO_WORD	Converts LREAL to W
LT	Less Than: <
LT_STRING	Less Than STRING: <
MAX	Maximum
MAX_DINT	Maximum of DINT

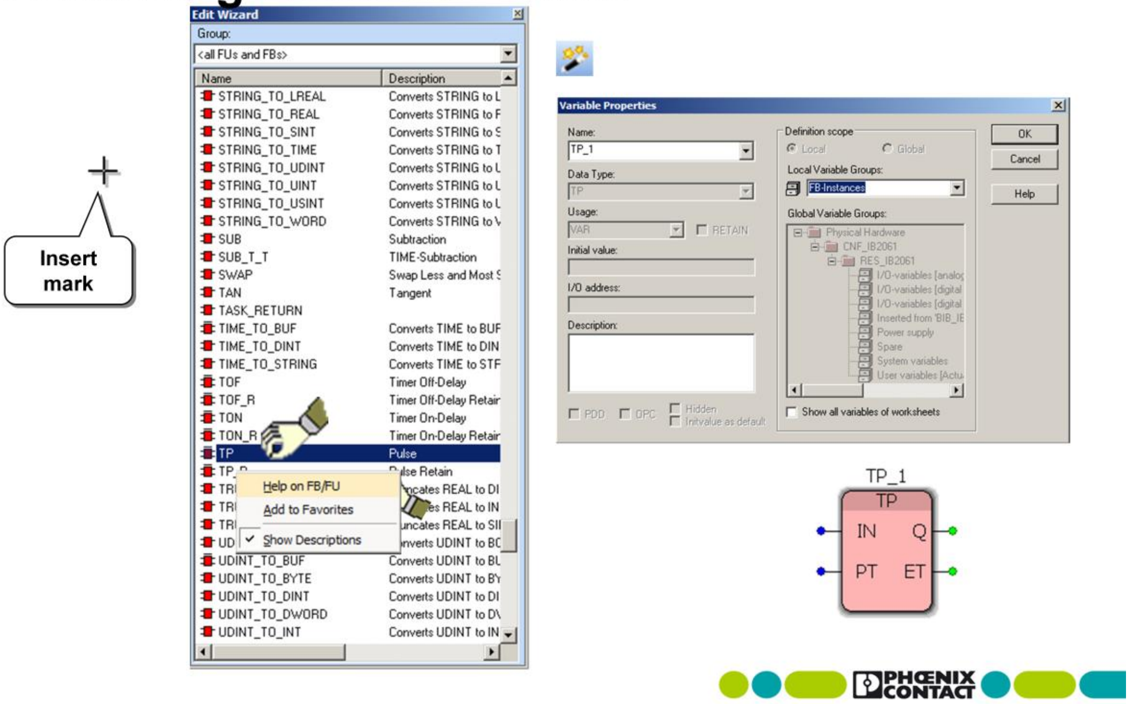
In order to access a function via the Edit Wizard, first of all the insert mark must be set on the graphic worksheet. To insert a mark, make a double click.

As an alternative, you can use Drag'n Drop.

For each standard block, an HTML help is available, which can be called via the context menu.

Via the button or the standard key combination Shift+F2, the edit wizard can be shown/hidden.

Accessing Function Blocks



Edit Wizard

Group:

<all FUs and FBs>

Name	Description
STRING_TO_LREAL	Converts STRING to L
STRING_TO_REAL	Converts STRING to F
STRING_TO_SINT	Converts STRING to S
STRING_TO_TIME	Converts STRING to T
STRING_TO_UDINT	Converts STRING to U
STRING_TO_UINT	Converts STRING to U
STRING_TO_USINT	Converts STRING to U
STRING_TO_WORD	Converts STRING to V
SUB	Subtraction
SUB_T_T	TIME-Subtraction
SWAP	Swap Less and Most S
TAN	Tangent
TASK_RETURN	
TIME_TO_BUF	Converts TIME to BUF
TIME_TO_DINT	Converts TIME to DIN
TIME_TO_STRING	Converts TIME to STF
TOF	Timer Off-Delay
TOF_R	Timer Off-Delay Retair
TON	Timer On-Delay
TON_R	Timer On-Delay Retair
TP	Pulse
TP_R	Pulse Retain
TRI	ncates REAL to DI
TRI	es REAL to IN
TRI	uncates REAL to SII
UD	nverts UDINT to BC
UDINT_TO_BUF	Converts UDINT to BU
UDINT_TO_BYTE	Converts UDINT to BY
UDINT_TO_DINT	Converts UDINT to DI
UDINT_TO_DWORD	Converts UDINT to D\
UDINT_TO_INT	Converts UDINT to IN

Variable Properties

Name: TP_1

Data Type: TP

Usage: VAR ☐ RETAIN

Initial value:

I/O address:

Description:

☐ PDD ☐ OPC ☐ Hidden ☐ Initvalue as default

Definition scope: ● Local ○ Global

Local Variable Groups: FB-Instances

Global Variable Groups:

- Physical Hardware
  - CNF_IB2061
    - RES_IB2061
      - I/O-variables [analog
      - I/O-variables [digital
      - I/O-variables [digital
      - Inserted from 'BIB_IE
      - Power supply
      - Spare
      - System variables
      - User variables [Actu

☐ Show all variables of worksheets

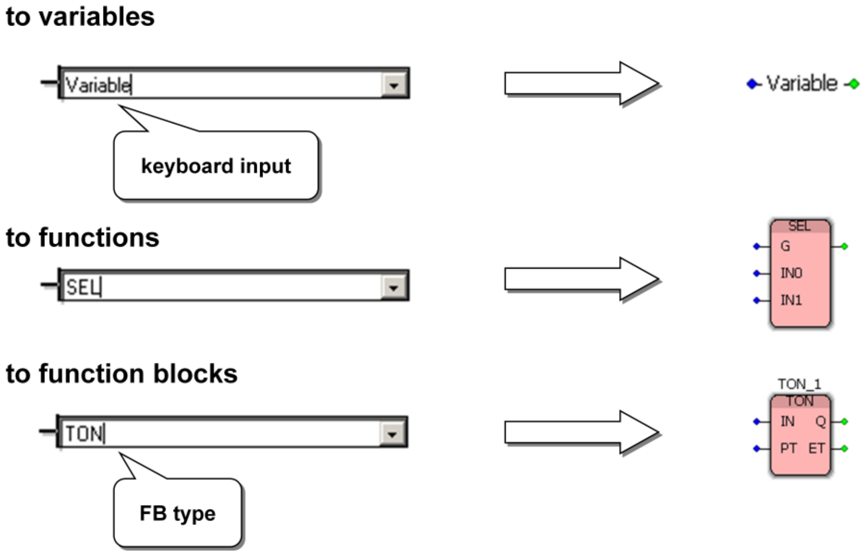
Inserting function blocks is done in the same way as inserting functions. However, before inserting a function block, its instance has to be declared. As an instance name, PC WORX proposes the function block name plus an increment linked by an underscore. This name can be selected individually, i.e. function-related, by the user, though.

For each standard block, an HTML help is available, which can be called via the context menu.

Above example shows the inserting dialog of a project, in which a local variable group for FB instances has already been created. Otherwise, only the *Default* group would be available.

Notes:

Simplified Access via the Keyboard



to variables

keyboard input

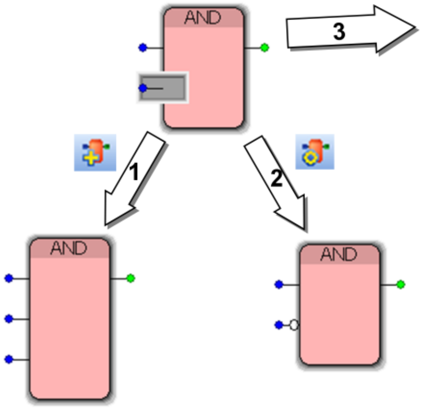
to functions

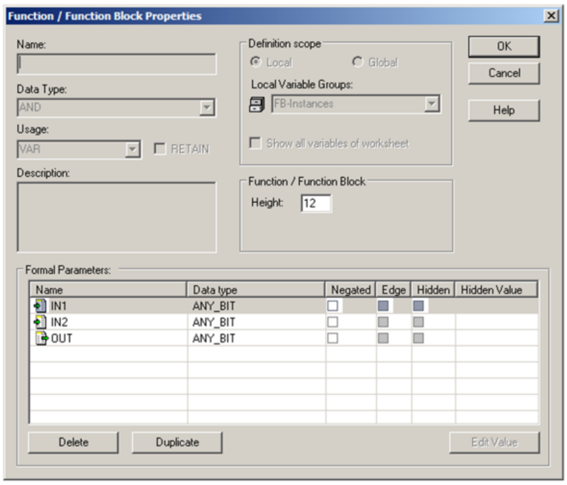
to function blocks

FB type

The graphic editor in PC WORX allows you to access programming elements directly via keyboard input, instead of access via dialogs. To this end, a formal parameter must be highlighted or an insert mark must be set into the workspace so that you can type the name of element to be inserted using the keyboard. In case of function blocks, this must be the FB type, not the name desired for the instance.

Editing the Formal Parameters





Function / Function Block Properties

Name:

Data Type: AND

Usage: VAR ☐ RETAIN

Description:

Definition scope

● Local ○ Global

Local Variable Groups: FB-Instances

☐ Show all variables of worksheet

Function / Function Block

Height: 12

OK

Cancel

Help

Formal Parameters:

Name	Data type	Negated	Edge	Hidden	Hidden Value
IN1	ANY_BIT	☐	☐	☐	
IN2	ANY_BIT	☐	☐	☐	
OUT	ANY_BIT	☐	☐	☐	

Delete

Duplicate

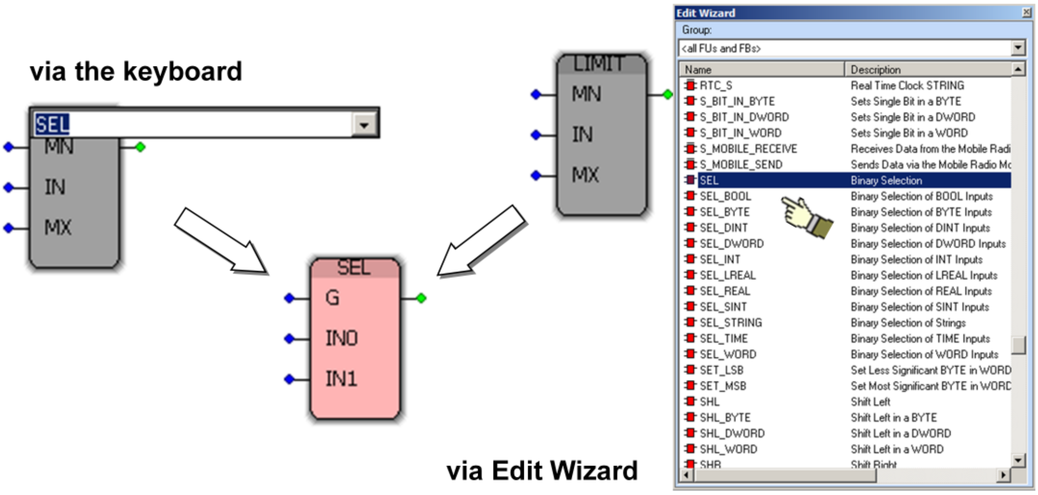
Edit Value

This page shows the different options of editing the formal parameters of blocks (operators, functions and function blocks).

These include (1) the adding of input parameters (which is generally supported by operators only) and (2) the inverting of in- and output parameters of bit-based parameters (not supported for all blocks). On selecting a parameter, the buttons presented on the slide are released.

The (3) Properties dialog of standard blocks can be accessed via double-click, if blocks are user-defined, access is given via the Properties option in the context menu.

Exchanging a Block



<all FUs and FBs>

Name	Description
RTC_S	Real Time Clock STRING
S_BIT_IN_BYTE	Sets Single Bit in a BYTE
S_BIT_IN_DWORD	Sets Single Bit in a DWORD
S_BIT_IN_WORD	Sets Single Bit in a WORD
S_MOBILE_RECEIVE	Receives Data from the Mobile Radi
S_MOBILE_SEND	Sends Data via the Mobile Radio Mo
SEL	Binary Selection
SEL_BOOL	Binary Selection of BOOL Inputs
SEL_BYTE	Binary Selection of BYTE Inputs
SEL_DINT	Binary Selection of DINT Inputs
SEL_DWORD	Binary Selection of DWORD Inputs
SEL_INT	Binary Selection of INT Inputs
SEL_LREAL	Binary Selection of LREAL Inputs
SEL_REAL	Binary Selection of REAL Inputs
SEL_SINT	Binary Selection of SINT Inputs
SEL_STRING	Binary Selection of Strings
SEL_TIME	Binary Selection of TIME Inputs
SEL_WORD	Binary Selection of WORD Inputs
SET_LSB	Set Less Significant BYTE in WORD
SET_MSB	Set Most Significant BYTE in WORD
SHL	Shift Left
SHL_BYTE	Shift Left in a BYTE
SHL_DWORD	Shift Left in a DWORD
SHL_WORD	Shift Left in a WORD
SHR	Shift Right

A function or a function block can be exchanged via the Edit Wizard using Drag‘n Drop or, if the block is highlighted, by double-clicking on the block. As an alternative, the simplified access to blocks is given via the keyboard input.

Notes:

Function Blocks and Functions Programmed by the User

Chapter 13

Notes:

Contents

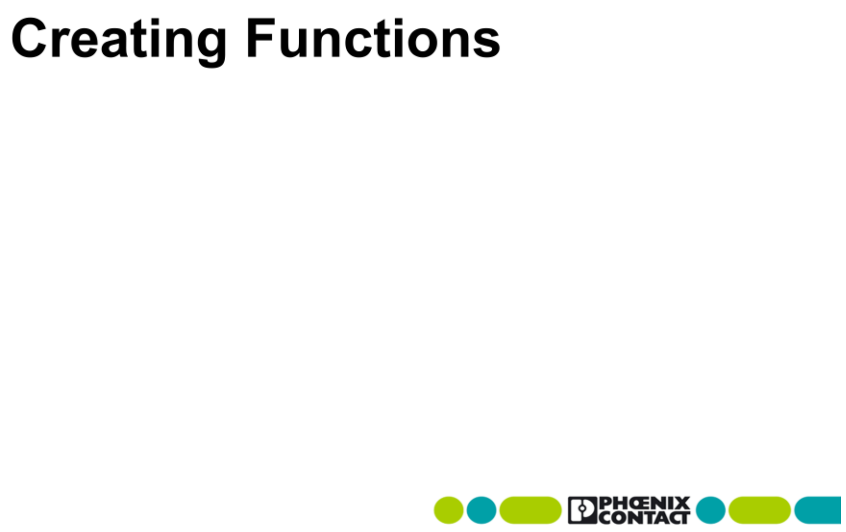
This chapter describes how you can create function and function blocks, what has to be considered and how these blocks can be used in the current project and made available to other projects.

⚠ Attention!

ℹ Information

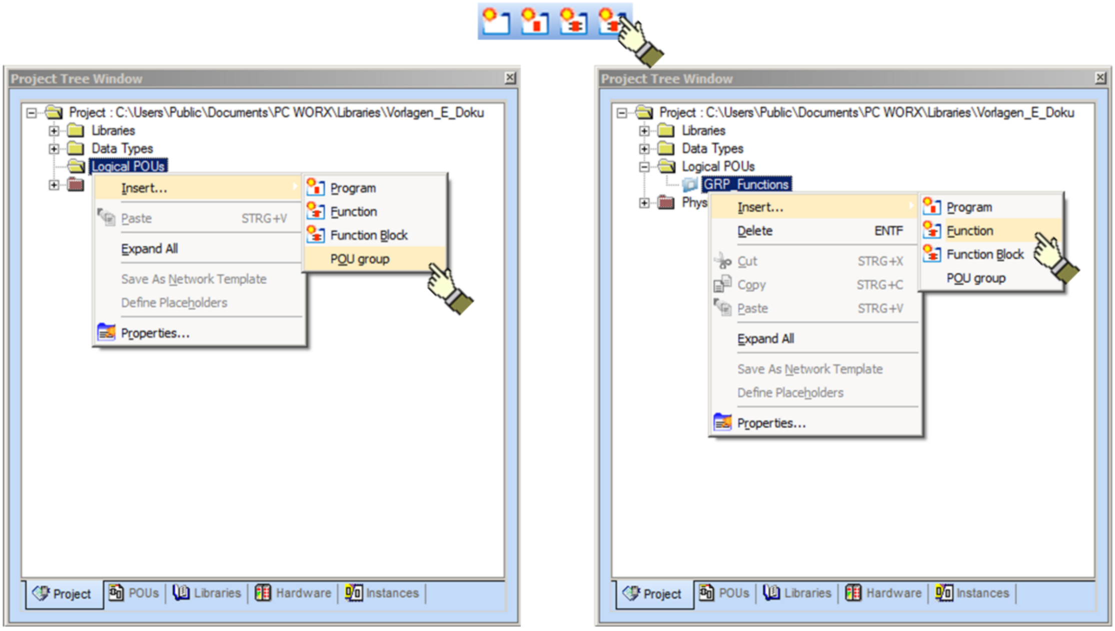
💡 Tip

Notes:



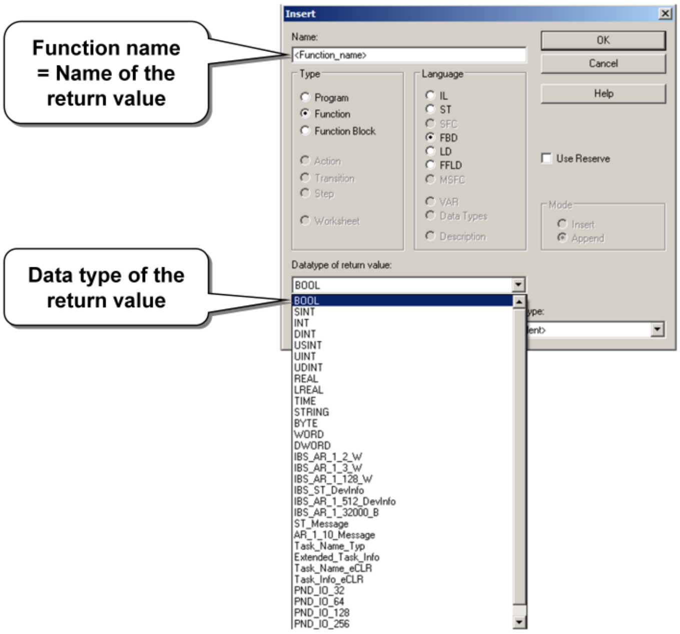
Creating Functions

Inserting into the Project Tree



Inserting a function into the project tree is possible when the *Logical POUs* folder or the POU group is marked. Depending on the project size, it might make sense to organize blocks you created in POU groups. These blocks can also be added to the groups later.

Setting Function Properties



Function name = Name of the return value

Data type of the return value

Insert

Name: <Function_name>

Type: ○ Program ● Function ○ Function Block ○ Action ○ Transition ○ Step ○ Worksheet

Language: ○ IL ○ ST ○ SFC ● FBD ○ LD ○ FFLD ○ MSFC ○ VAR ○ Data Types ○ Description

□ Use Reserve

Mode: ○ Insert ● Append

Datatype of return value: BOOL

Only few parameters have to be set in order to insert a function.

Function name

Later, the function name of the block will be listed for selection in the edit wizard. Furthermore, this name will be used as a return value name for internal programming of the block. The name has to correspond to the conventions for variable names.

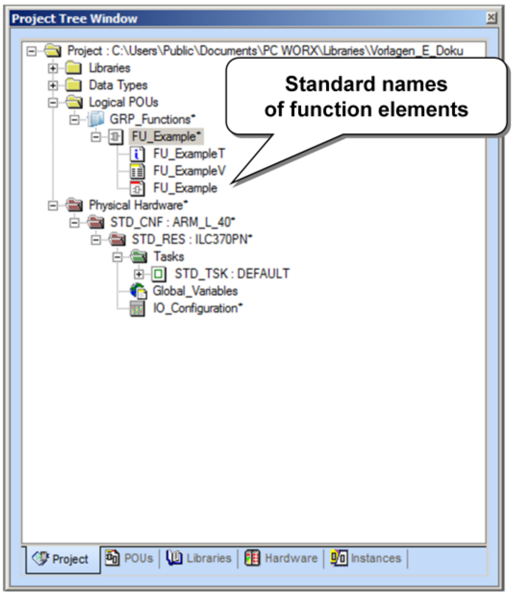
Return value of the function

The type of the single output value of the function is determined via the data type of the return value.

PLC and processor type

The properties only have to be determined if corresponding, specific blocks for the selected type are to be accessed. This is unlikely as most specific blocks are function blocks and thus cannot be called.

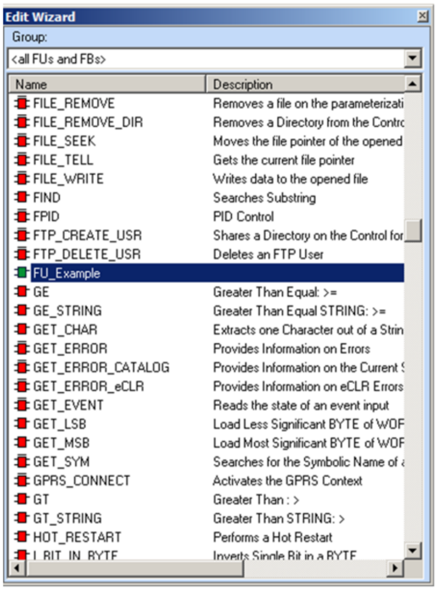
Function in the Project Tree



Project Tree Window

```
Project : C:\Users\Public\Documents\PC WORX\Libraries\Vorlagen_E_Doku
  Libraries
  Data Types
  Logical POUs
    GRP_Functions*
      FU_Example*
        FU_ExampleT
        FU_ExampleV
        FU_Example
  Physical Hardware*
    STD_CNF : ARM_L_40*
      STD_RES : ILC370PN*
        Tasks
          STD_TSK : DEFAULT
        Global_Variables
        IO_Configuration*
```

Standard names of function elements



Edit Wizard

Group: <all FUs and FBs>

Name	Description
FILE_REMOVE	Removes a file on the parameterizati
FILE_REMOVE_DIR	Removes a Directory from the Contro
FILE_SEEK	Moves the file pointer of the opened
FILE_TELL	Gets the current file pointer
FILE_WRITE	Writes data to the opened file
FIND	Searches Substring
FPID	PID Control
FTP_CREATE_USR	Shares a Directory on the Control for
FTP_DELETE_USR	Deletes an FTP User
FU_Example	
GE	Greater Than Equal: >=
GE_STRING	Greater Than Equal STRING: >=
GET_CHAR	Extracts one Character out of a Strin
GET_ERROR	Provides Information on Errors
GET_ERROR_CATALOG	Provides Information on the Current S
GET_ERROR_eCLR	Provides Information on eCLR Errors
GET_EVENT	Reads the state of an event input
GET_LSB	Load Less Significant BYTE of WOR
GET_MSB	Load Most Significant BYTE of WOR
GET_SYM	Searches for the Symbolic Name of a
GPRS_CONNECT	Activates the GPRS Context
GT	Greater Than : >
GT_STRING	Greater Than STRING: >
HOT_RESTART	Performs a Hot Restart
I_BIT_IN_BYTE	Inverts Single Bit in a BYTE

After the function was inserted into the project tree, it is listed with the chosen name and the names for the three basic elements that are based on the first name.

<Function name>T for the comment sheet

<Function name>V for the variable table

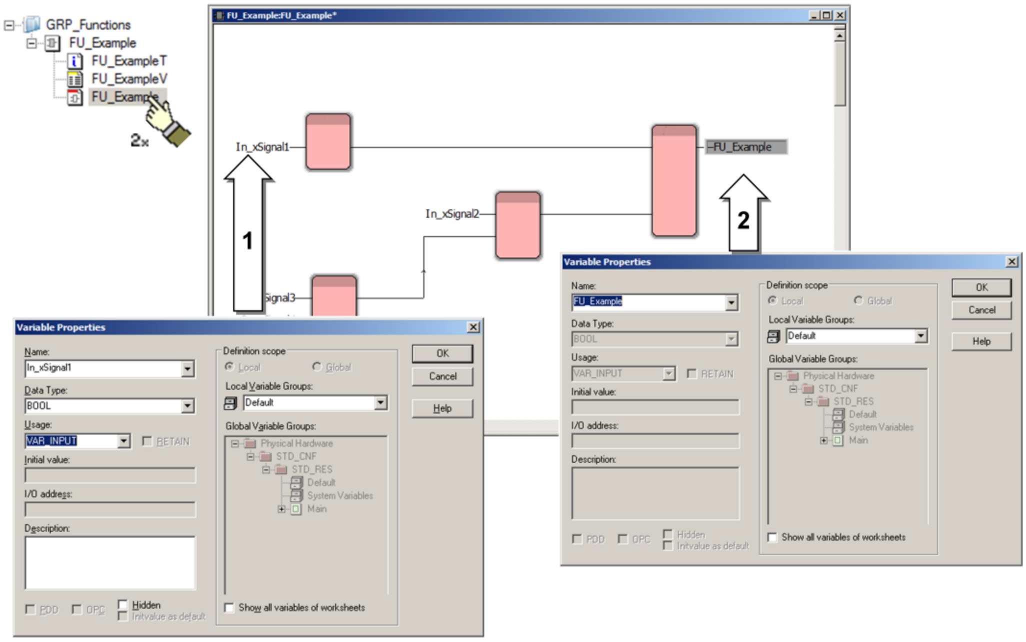
<Function name> for the first code worksheet

In addition to this, in the edit wizard the function shows up in the *<all FUs and FBs>* group and in the group with the name of the current project.

Although they are functions, for functions created by the user there is not entry in the *<Functions>* group.

At this point, the function cannot yet be called. The asterisks after the variable table and the code worksheet indicate that these elements have not yet been compiled.

Editing the Function



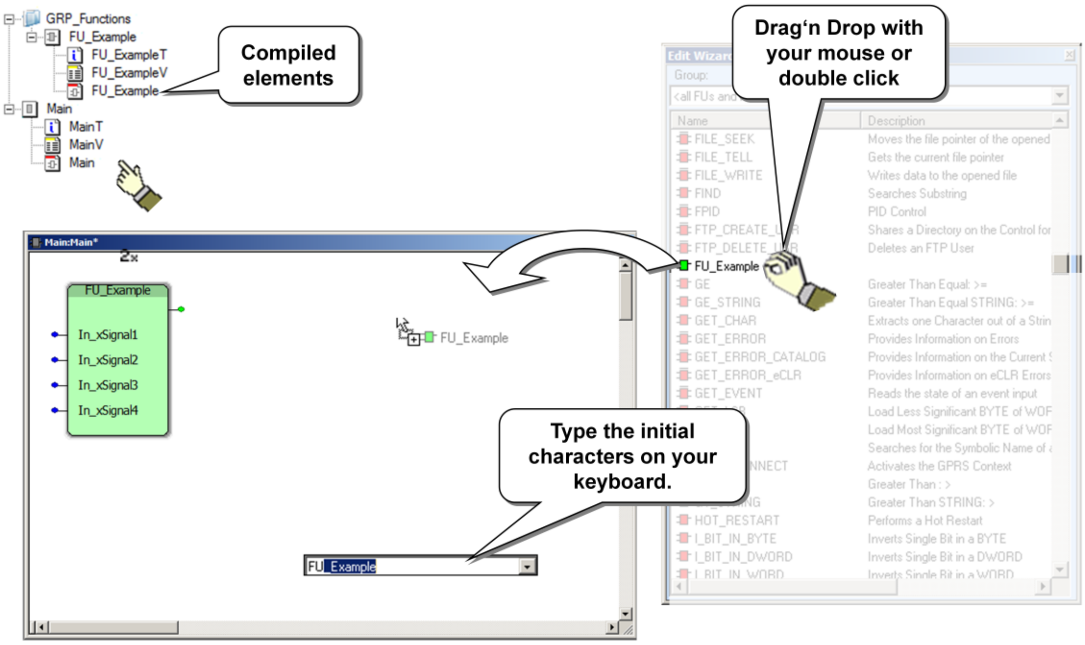
Variable Properties (1):
Name: In_xSignal1
Data Type: BOOL
Usage: VAR_INPUT □ RETAIN
Initial value:
I/O address:
Description:
Local Variable Groups: Default
□ PDD □ OPC □ Hidden □ Initvalue as default
□ Show all variables of worksheets

Variable Properties (2):
Name: FU_Example
Data Type: BOOL
Usage: VAR_INPUT □ RETAIN
Initial value:
I/O address:
Description:
Local Variable Groups: Default
□ PDD □ OPC □ Hidden □ Initvalue as default
□ Show all variables of worksheets

The function itself is edited in its worksheet(s) according to the standard procedure for programming. As shown above, at least (1) one input parameter has to be declared. Furthermore, the (2) return value (name = function name) has to be used to make the value computed by means of the function algorithm available to the project.

Notes:

Use of the Function



Compiled elements

Drag‘n Drop with your mouse or double click

Type the initial characters on your keyboard.

Name	Description
FILE_SEEK	Moves the file pointer of the opened
FILE_TELL	Gets the current file pointer
FILE_WRITE	Writes data to the opened file
FIND	Searches Substring
FPID	PID Control
FTP_CREATE_U R	Shares a Directory on the Control for
FTP_DELETE_U R	Deletes an FTP User
FU_Example	
GE	Greater Than Equal: >=
GE_STRING	Greater Than Equal STRING: >=
GET_CHAR	Extracts one Character out of a Strin
GET_ERROR	Provides Information on Errors
GET_ERROR_CATALOG	Provides Information on the Current S
GET_ERROR_eCLR	Provides Information on eCLR Errors
GET_EVENT	Reads the state of an event input
	Load Less Significant BYTE of WOF
	Load Most Significant BYTE of WOF
	Searches for the Symbolic Name of a
NNECT	Activates the GPRS Context
	Greater Than : >
NG	Greater Than STRING: >
HOT_RESTART	Performs a Hot Restart
I_BIT_IN_BYTE	Inverts Single Bit in a BYTE
I_BIT_IN_DWORD	Inverts Single Bit in a DWORD
I_BIT_IN_WORD	Inverts Single Bit in a WORD

The variable table is compiled by closing it.

The code worksheets can be compiled by closing them or by explicitly using the corresponding button or key combination (as a standard Shift+F9).

After successful compilation of the variable table, a function can be called for use in another POU.

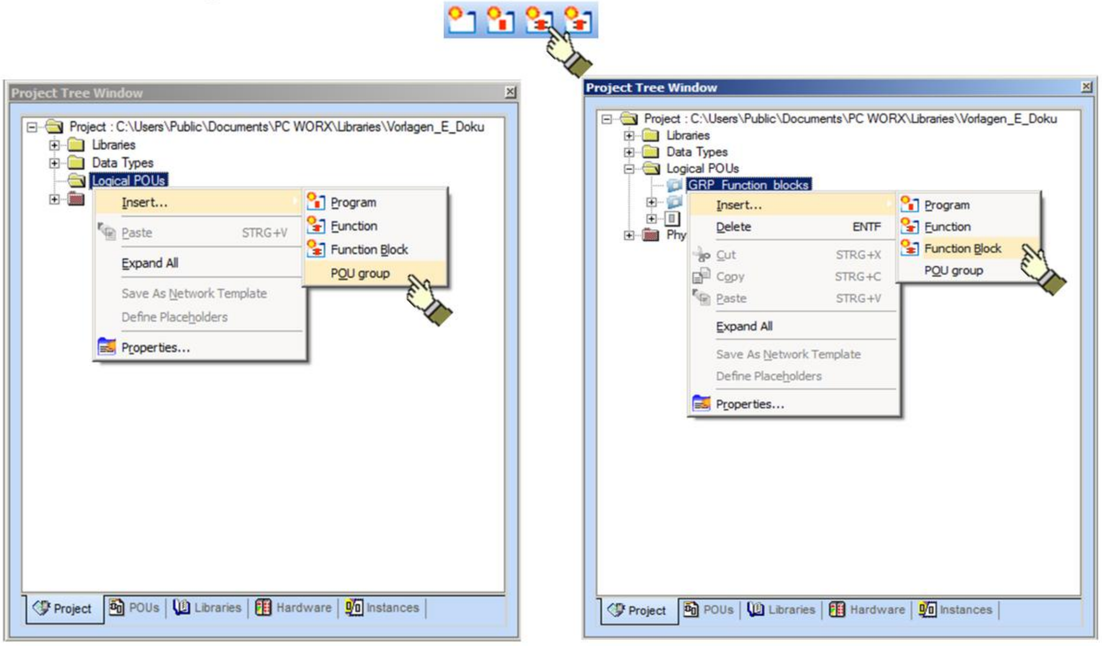
When inserting, you can choose between Drag‘n Drop and, if the insert mark is set, double click. Additionally, it is possible to type on the keyboard. When the insert mark is set, the available variables and parametrizable block types are listed in a combobox after a valid character has been inserted, The inserted string is completed by PC WORX automatically and the desired function can be inserted by pressing the Return key.

If the asterisks remain behind the variable table after the function has been closed, then it could not be compiled correctly. The reason for this can be the lack of input variables or a non-translated library.

Notes:

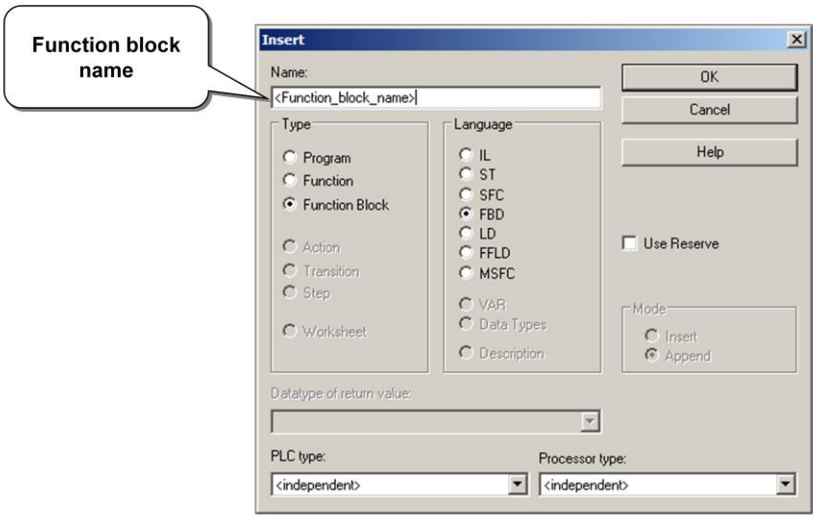
Creating Function Blocks

Inserting into the Project Tree



Inserting a function block into the project tree is possible when the *Logical POUs* folder or the POU group is marked. Depending on the project size, it might make sense to organize blocks you created in POU groups. These blocks can also be added to the groups later.

Set Function Block Properties



Insert

Name: <Function_block_name>

Type
○ Program
○ Function
● Function Block
○ Action
○ Transition
○ Step
○ Worksheet

Language
○ IL
○ ST
○ SFC
● FBD
○ LD
○ FFLD
○ MSFC
○ VAR
○ Data Types
○ Description

Datatype of return value:

PLC type: <independent>

Processor type: <independent>

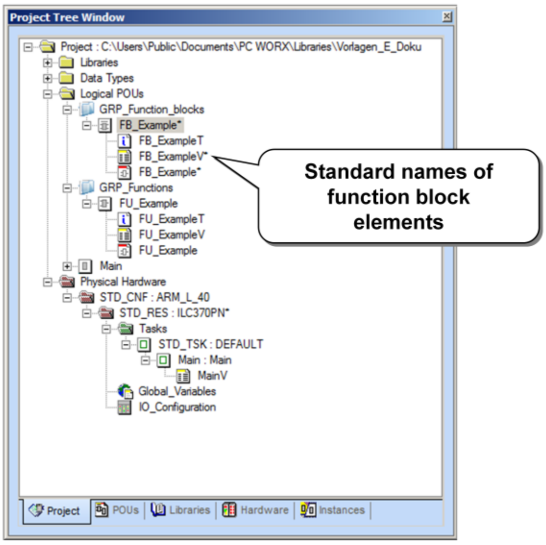
□ Use Reserve

Mode
○ Insert
● Append

For the insertion of function blocks, a larger variety of languages is available than for the insertion of functions. The data type of the return values does not have to be specified.

Later, the function block name of the block will be listed for selection in the edit wizard. The name is not used internally for programming the FB. The name has to correspond to the conventions for variable names.

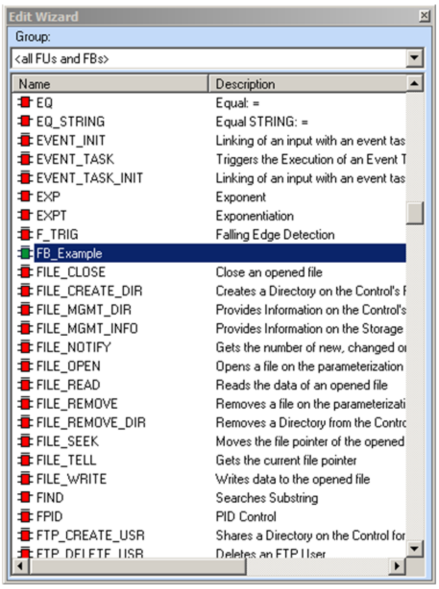
Function Block in the Project Tree



Project Tree Window

```
Project : C:\Users\Public\Documents\PC WORX\Libraries\Vorlagen_E_Doku
  Libraries
  Data Types
  Logical POUs
    GRP_Function_blocks
      FB_Example*
        FB_ExampleT
        FB_ExampleV*
        FB_Example*
    GRP_Functions
      FU_Example
        FU_ExampleT
        FU_ExampleV
        FU_Example
    Main
  Physical Hardware
    STD_CNF : ARM_L_40
      STD_RES : ILC370PN*
        Tasks
          STD_TSK : DEFAULT
            Main : Main
              MainV
        Global_Variables
        IO_Configuration
```

Standard names of function block elements



Edit Wizard

Group: <all FUs and FBs>

Name	Description
EQ	Equal: =
EQ_STRING	Equal STRING: =
EVENT_INIT	Linking of an input with an event tas
EVENT_TASK	Triggers the Execution of an Event T
EVENT_TASK_INIT	Linking of an input with an event tas
EXP	Exponent
EXPT	Exponentiation
F_TRIG	Falling Edge Detection
FB_Example	
FILE_CLOSE	Close an opened file
FILE_CREATE_DIR	Creates a Directory on the Control's F
FILE_MGMT_DIR	Provides Information on the Control's
FILE_MGMT_INFO	Provides Information on the Storage
FILE_NOTIFY	Gets the number of new, changed or
FILE_OPEN	Opens a file on the parameterization
FILE_READ	Reads the data of an opened file
FILE_REMOVE	Removes a file on the parameterizati
FILE_REMOVE_DIR	Removes a Directory from the Contro
FILE_SEEK	Moves the file pointer of the opened
FILE_TELL	Gets the current file pointer
FILE_WRITE	Writes data to the opened file
FIND	Searches Substring
FPID	PID Control
FTP_CREATE_USR	Shares a Directory on the Control for
FTP_DELETE_USR	Deletes an FTP User

After the function block was inserted into the project tree, it is listed with the chosen name and the names for the three basic elements that are based on the first name.

<Function block name>T for the comment sheet

<Function block name>V for the variable table

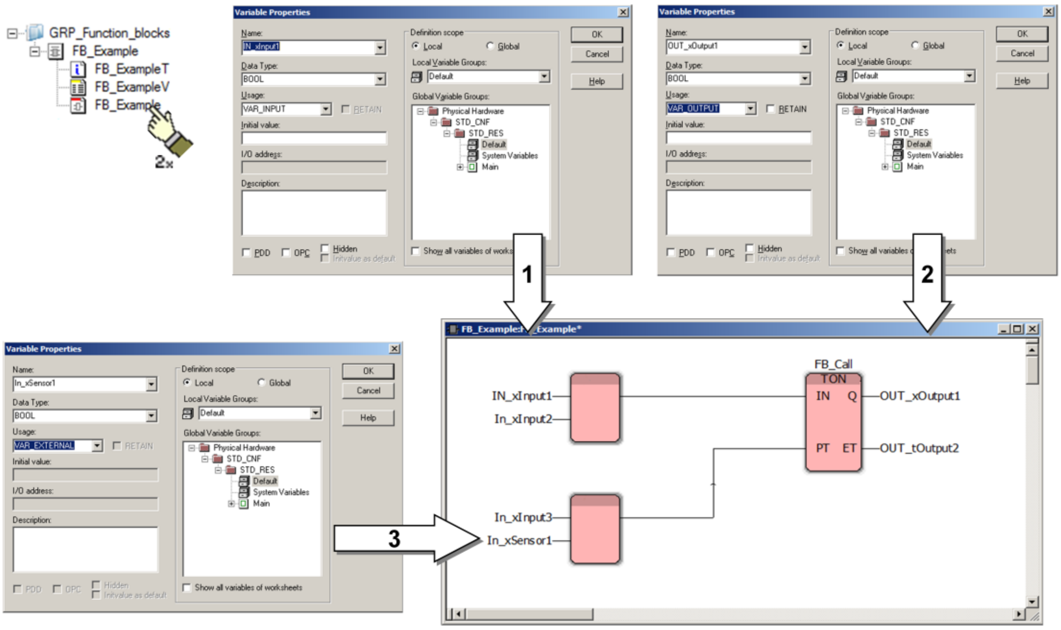
<Function block name> for the first code worksheet

In addition to this, in the edit wizard the function block shows up in the *<all FUs and FBs>* group and in the group with the name of the current project.

Although they are function blocks, for function blocks created by the user there is no entry in the *<Functions block>* group.

At this point, the function block cannot yet be called. The asterisks in the variable table and code worksheet indicate that these elements have not yet been compiled.

Editing the Function Block



Variable Properties

Name: IN_xInput1
Data Type: BOOL
Usage: VAR_INPUT ☐ RETAIN
Initial value:
I/O address:
Description:
☐ PDD ☐ OPC ☐ Hidden ☐ Initvalue as default
Definition scope: ● Local ○ Global
Local Variable Groups: Default
Global Variable Groups: Physical Hardware / STD_CNF / STD_RES / Default, System Variables, Main
☐ Show all variables of worksheets

Variable Properties

Name: OUT_xOutput1
Data Type: BOOL
Usage: VAR_OUTPUT ☐ RETAIN
Initial value:
I/O address:
Description:
☐ PDD ☐ OPC ☐ Hidden ☐ Initvalue as default
Definition scope: ● Local ○ Global
Local Variable Groups: Default
Global Variable Groups: Physical Hardware / STD_CNF / STD_RES / Default, System Variables, Main
☐ Show all variables of worksheets

Variable Properties

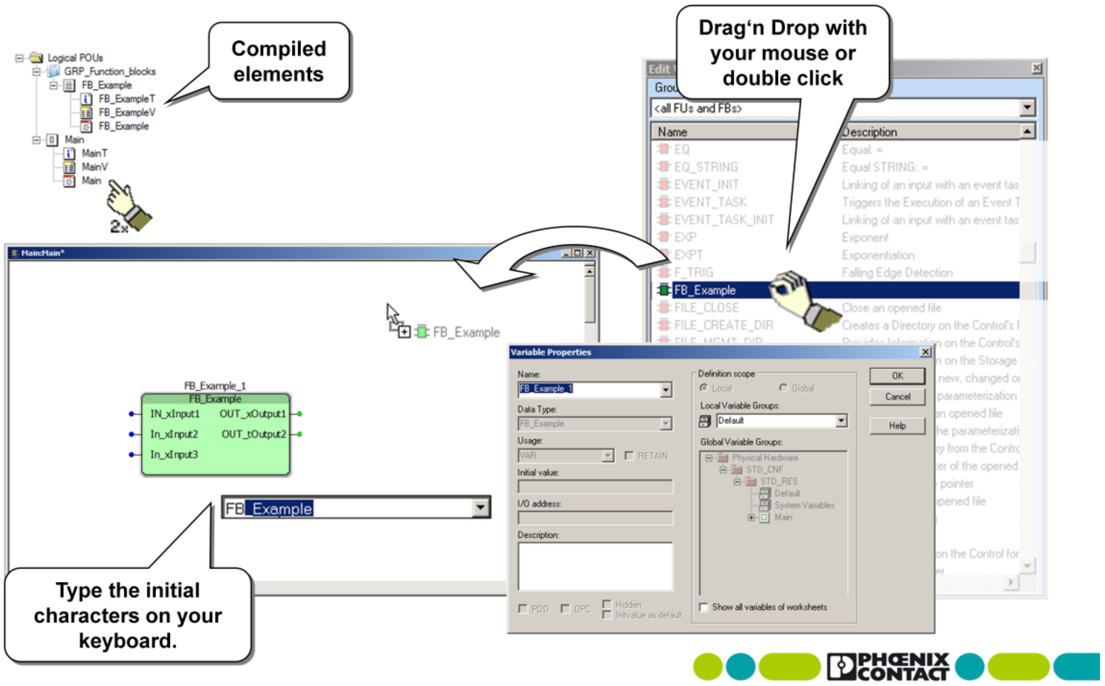
Name: In_xSensor1
Data Type: BOOL
Usage: VAR_EXTERNAL ☐ RETAIN
Initial value:
I/O address:
Description:
☐ PDD ☐ OPC ☐ Hidden ☐ Initvalue as default
Definition scope: ● Local ○ Global
Local Variable Groups: Default
Global Variable Groups: Physical Hardware / STD_CNF / STD_RES / Default, System Variables, Main
☐ Show all variables of worksheets

The function block itself is edited in its worksheet(s) according to the standard procedure for the programming. Unlike with functions, there is a significantly higher degree of freedom regarding the design. Input (1) and output (2) parameters can be declared in any number and function blocks also allow for the access to global variables (3).

Remember that the function block name should not be used as a variable name during internal programming of the FB.

Notes:

Use of the Function block



Compiled elements

Drag'n Drop with your mouse or double click

Type the initial characters on your keyboard.

Variable Properties

Name: FB_Example_1

Data Type: FB_Example

Usage: VAR

Definition scope: ● Local ○ Global

Local Variable Groups: Default

The variable table is compiled by closing it.

The code worksheets can be compiled by closing them or by explicitly using the corresponding button or key combination (as a standard Alt+F9).

After successful compilation of the variable table, a function block can be called for use in another POU.

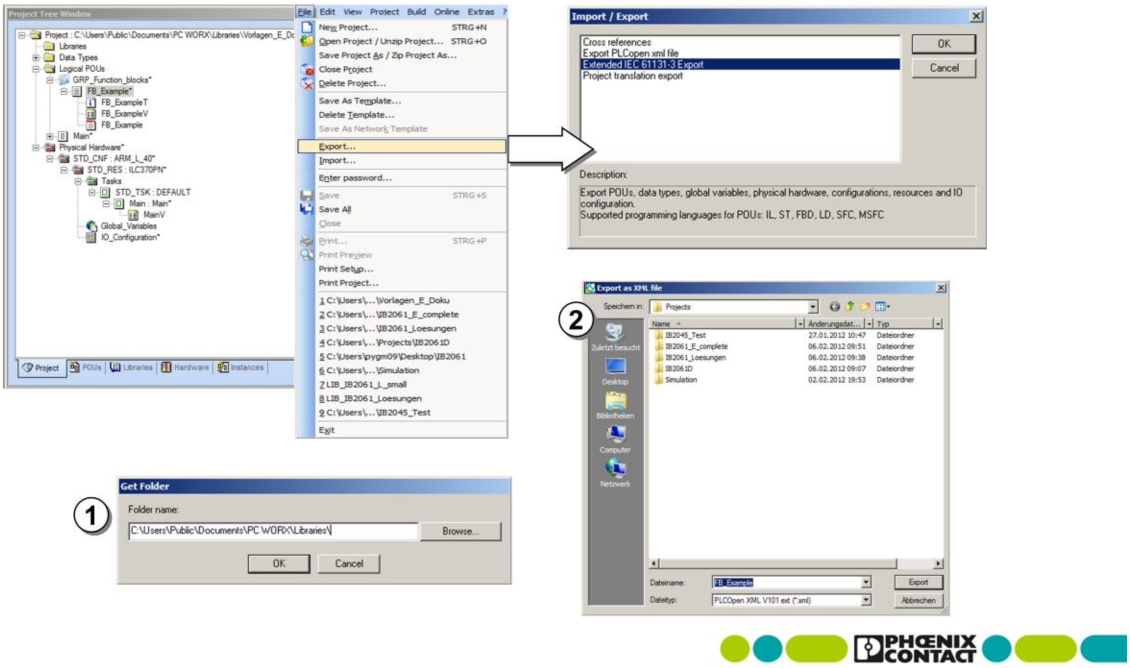
When inserting, you can choose between Drag'n Drop and, if the insert mark is set, double click. Additionally, it is possible to type on the keyboard. When the insert mark is set, the available variables and parametrizable block types are listed in a combobox after a valid character has been inserted, The inserted string is completed by PC WORX automatically and the desired function block type can be inserted by pressing the Return key. After that, the type must be instantiated using the variable dialog.

If the asterisks remain behind the variable table after the function has been closed, then it could not be compiled correctly. The reason for this can be the lack of input variables or a non-translated library.

Notes:

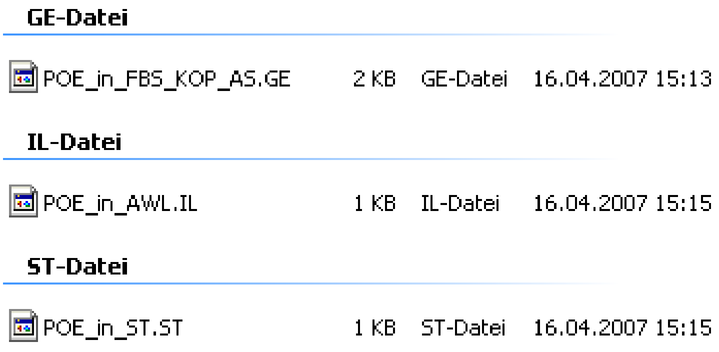
Exporting and Importing Function Blocks

Exporting a Block


Project Tree Window
Project : C:\Users\Public\Documents\PC WORX\Libraries\Vorlagen_E_Do
Libraries
Data Types
Logical POUs
GRP_Function_blocks*
FB_Example*
FB_ExampleT
FB_ExampleV
FB_Example
Main*
Physical Hardware*
STD_CNF : ARM_L_40*
STD_RES : ILC370PN*
Tasks
STD_TSK : DEFAULT
Main : Main*
MainV
Global_Variables
IO_Configuration*
Project
POUs
Libraries
Hardware
Instances
File
Edit
View
Project
Build
Online
Extras
?
New Project... STRG+N
Open Project / Unzip Project... STRG+O
Save Project As / Zip Project As...
Close Project
Delete Project...
Save As Template...
Delete Template...
Save As Network Template
Export...
Import...
Enter password...
Save STRG+S
Save All
Close
Print... STRG+P
Print Preview
Print Setup...
Print Project...
1 C:\Users\...\Vorlagen_E_Doku
2 C:\Users\...\IB2061_E_complete
3 C:\Users\...\IB2061_Loesungen
4 C:\Users\...\Projects\IB2061D
5 C:\Users\pygm09\Desktop\IB2061
6 C:\Users\...\Simulation
7 LIB_IB2061_L_small
8 LIB_IB2061_Loesungen
9 C:\Users\...\IB2045_Test
Exit
Import / Export
Cross references
Export PLCopen xml file
Extended IEC 61131-3 Export
Project translation export
OK
Cancel
Description:
Export POUs, data types, global variables, physical hardware, configurations, resources and IO configuration.
Supported programming languages for POUs: IL, ST, FBD, LD, SFC, MSFC
2
Export as XML file
Speichern in: Projects
Name
Änderungsdat...
Typ
IB2045_Test 27.01.2012 10:47 Dateiordner
IB2061_E_complete 06.02.2012 09:51 Dateiordner
IB2061_Loesungen 06.02.2012 09:38 Dateiordner
IB2061D 06.02.2012 09:07 Dateiordner
Simulation 02.02.2012 19:53 Dateiordner
Zuletzt besucht
Desktop
Bibliotheken
Computer
Netzwerk
Dateiname: FB_Example
Dateityp: PLCOpen XML V101 ext (*.xml)
Export
Abbrechen
1
Get Folder
Folder name: C:\Users\Public\Documents\PC WORX\Libraries\
Browse...
OK
Cancel

A block has to be marked in the project tree before it can be exported, saved as a file or made available to other projects. In the *File* menu, select *Export*. You can then select between saving the POU as (1) IEC 61131-3 file or as (2) PLC Open XML file.

IEC61131-3 files



GE-Datei

POE_in_FBS_KOP_AS.GE	2 KB	GE-Datei	16.04.2007 15:13

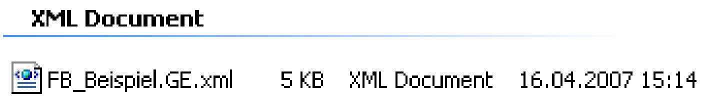
IL-Datei

POE_in_AWL.IL	1 KB	IL-Datei	16.04.2007 15:15

ST-Datei

POE_in_ST.ST	1 KB	ST-Datei	16.04.2007 15:15

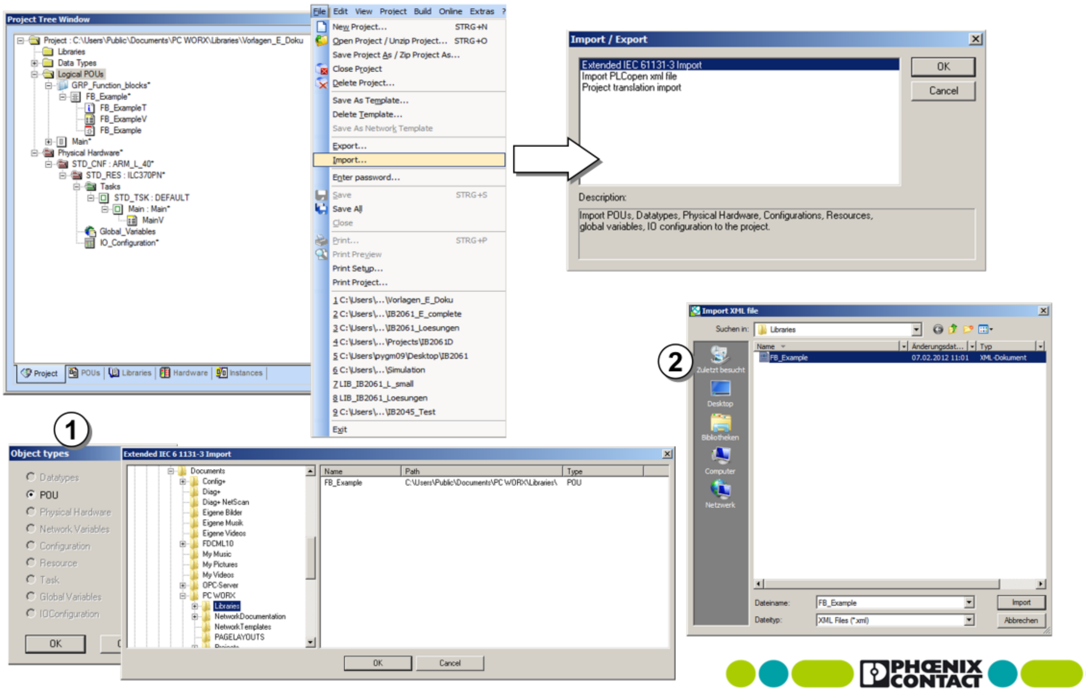
PLC Open XML file (graphic editor)



XML Document

FB_Beispiel.GE.xml	5 KB	XML Document	16.04.2007 15:14

Importing a Block



Description:
Import POUs, Datatypes, Physical Hardware, Configurations, Resources, global variables, IO configuration to the project.

Name	Path	Type
FB_Example	C:\Users\Public\Documents\PC WORX\Libraries\	POU

Name	Änderungsdat...	Typ
FB_Example	07.02.2012 11:01	XML-Dokument

Dateiname: FB_Example
Dateityp: XML Files (*.xml)

In order to import a block from (1) an IEC 61131-3 file or a (2) PLC Open XML file, the *Logical POUs* folder or one of its elements have to be selected. Selecting *Import* in the *File* menu opens the dialog for selecting the type to be imported.

Notes:

IL Instruction List

Chapter 14

Notes:

Contents

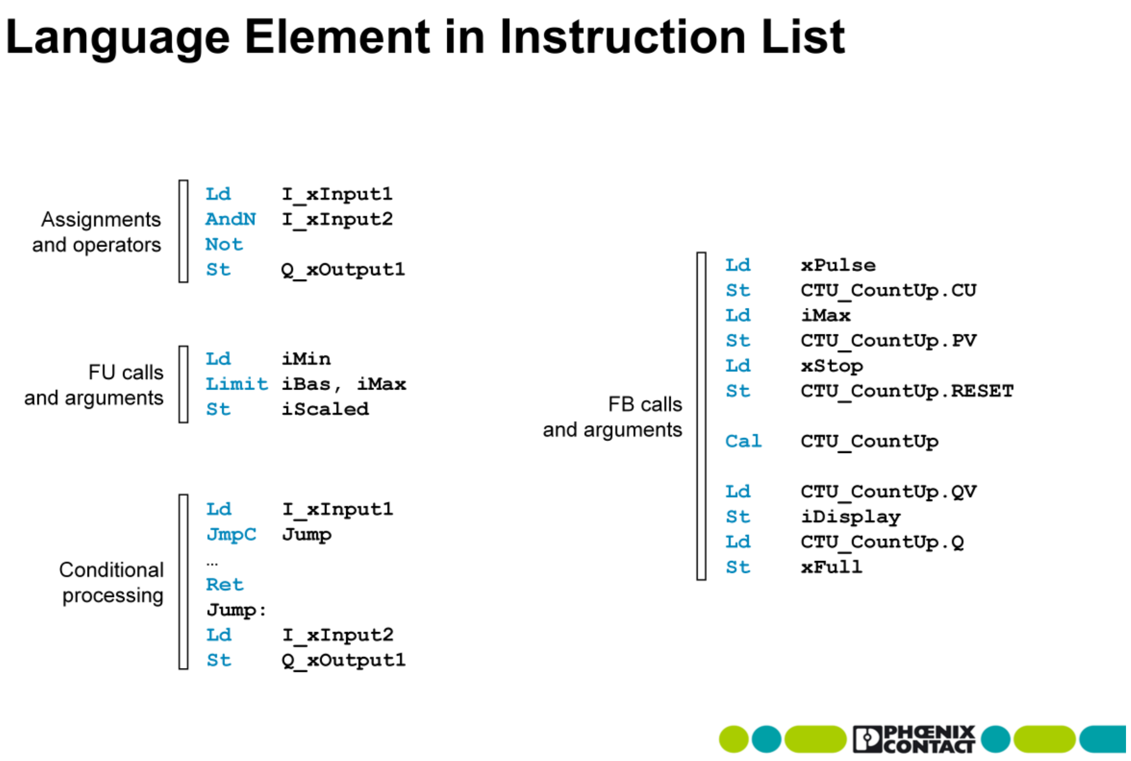
This chapter shows how to program in instruction list in PC WORX. In addition to simple instructions like operators and functions it is described how function blocks can be called and how conditionally executed programming can be implemented via jump and return.

⚠ Attention!

ℹ Information

💡 Tip

Notes:



Language Element in Instruction List

Assignments and operators

```
Ld    I_xInput1
AndN  I_xInput2
Not
St    Q_xOutput1
```

FU calls and arguments

```
Ld    iMin
Limit iBas, iMax
St    iScaled
```

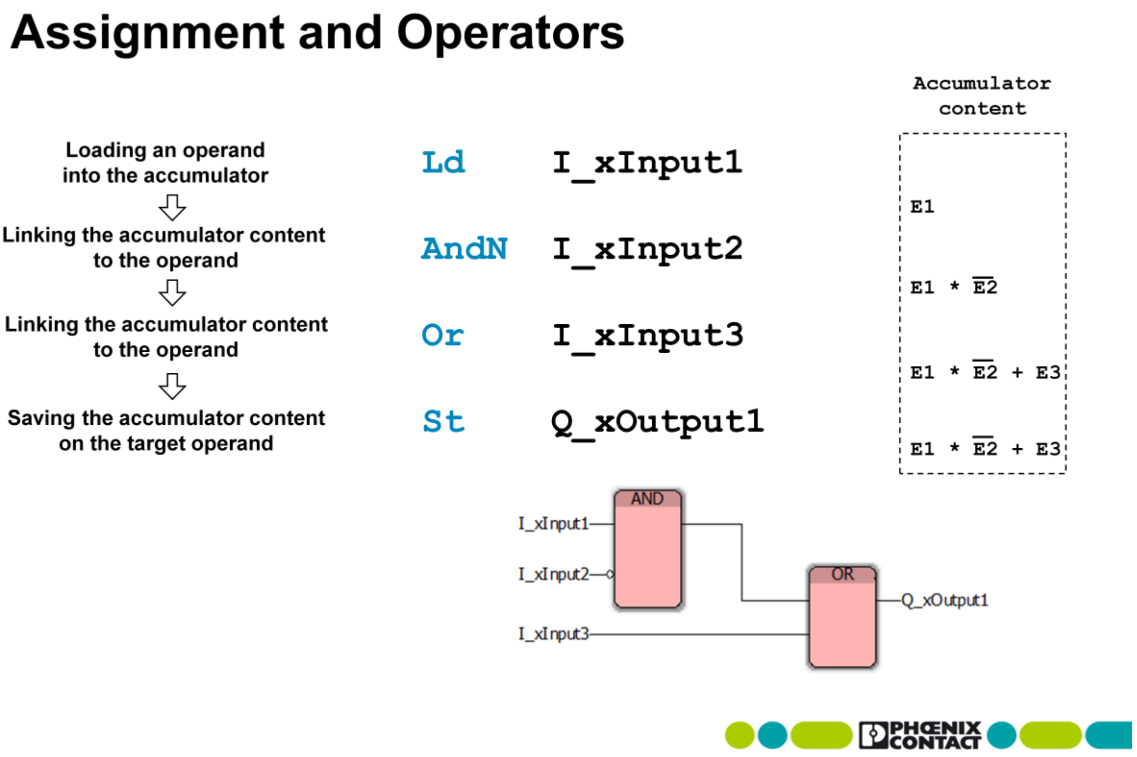
Conditional processing

```
Ld    I_xInput1
JmpC  Jump
…
Ret
Jump:
Ld    I_xInput2
St    Q_xOutput1
```

FB calls and arguments

```
Ld     xPulse
St     CTU_CountUp.CU
Ld     iMax
St     CTU_CountUp.PV
Ld     xStop
St     CTU_CountUp.RESET

Cal    CTU_CountUp

Ld     CTU_CountUp.QV
St     iDisplay
Ld     CTU_CountUp.Q
St     xFull
```

The access to the language elements available in IL is mostly supported by the edit wizard. In addition to the *Functions* and *Function block* groups, the group *Operators* is available in instruction list.

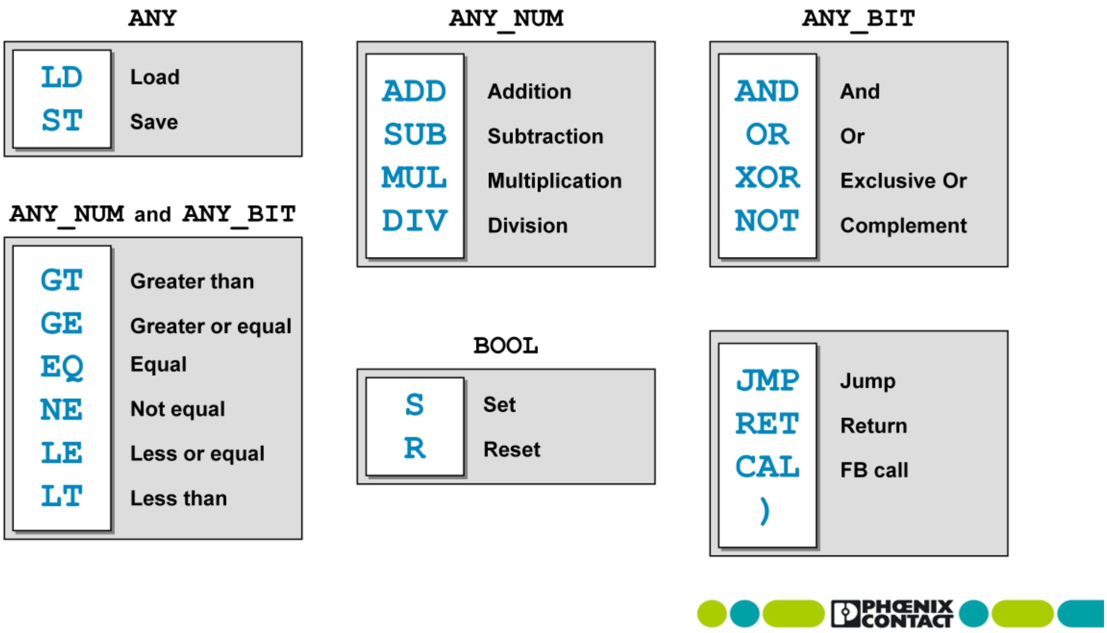


Assignment and Operators

			Accumulator content
Loading an operand into the accumulator ⇩	Ld	I_xInput1	E1
Linking the accumulator content to the operand ⇩	AndN	I_xInput2	E1 * $\overline{E2}$
Linking the accumulator content to the operand ⇩	Or	I_xInput3	E1 * $\overline{E2}$ + E3
Saving the accumulator content on the target operand	St	Q_xOutput1	E1 * $\overline{E2}$ + E3

Instruction list in PC WORX uses an accumulator and a uniform pair of load and save command for all data types.

Operators in Instruction List



ANY

LD	Load
ST	Save

ANY_NUM and ANY_BIT

GT	Greater than
GE	Greater or equal
EQ	Equal
NE	Not equal
LE	Less or equal
LT	Less than

ANY_NUM

ADD	Addition
SUB	Subtraction
MUL	Multiplication
DIV	Division

BOOL

S	Set
R	Reset

ANY_BIT

AND	And
OR	Or
XOR	Exclusive Or
NOT	Complement

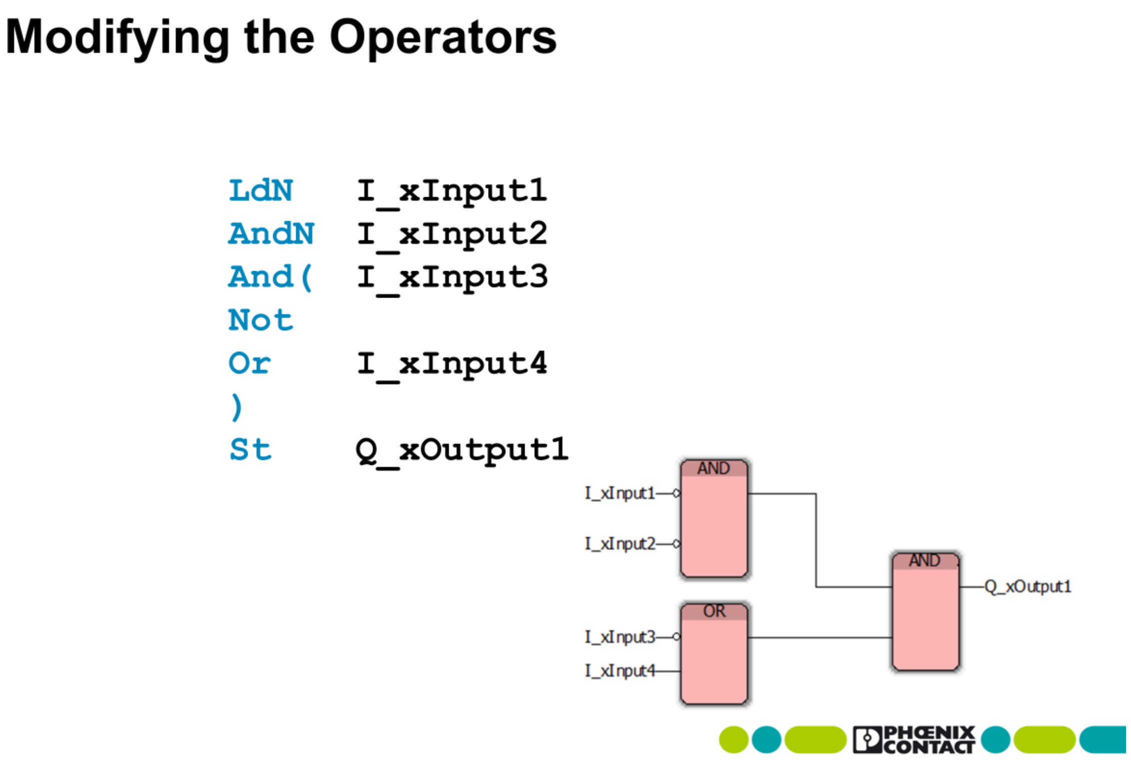
JMP	Jump
RET	Return
CAL	FB call
)	

This diagram gives an overview of the variables of data types and data type groups operators can be used with. The formatting regarding an instruction in instruction list is the same for all.

Exceptions are the complement *NOT*, the function block end *RETURN* and the closing bracket, which can be used without an operand in the same line.

As is the standard in classic PLC programming, the operators *S* and *R* once write, depending on a positive link, a boolean 1 or 0 to the connected operands.

Notes:



Modifying the Operators

```
LdN     I_xInput1
AndN    I_xInput2
And(    I_xInput3
Not
Or      I_xInput4
)
St      Q_xOutput1
```

By means of the modifier *N*, the operators for accessing the memory, *LD* and *ST*, can implement an inverted access to bit-based operands. The same holds true for operators for boolean operands.

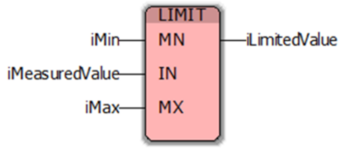
The opening bracket as a modifier results in a prioritized processing of the code in the brackets.

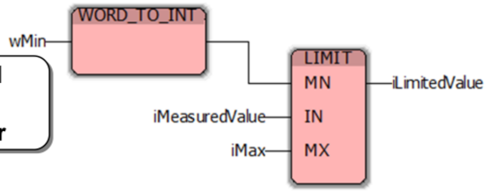
The closing bracket as an operator stands alone in one instruction line and triggers the processing of the prioritized code.

Function Calls

1st operand explicitly loaded

```
Ld      iMin
Limit   iMeasuredValue, iMax
St      iLimitedValue
```





1st operand from the accumulator

```
Ld      wMin
Word_To_Int
Limit iMeasuredValue, iMax
St      iLimitedValue
```

The call of functions in instruction list only differs slightly from the syntax of standard operations. The first parameter can be loaded explicitly via *LD* or be taken from the accumulator.

A difference only becomes clear if functions with more than one input are used. The absolute parameters following the second parameter are not written into separate lines but behind the function name, in correct order and separated through commas.

Editing in PC WORX

Inserting via the edit wizard

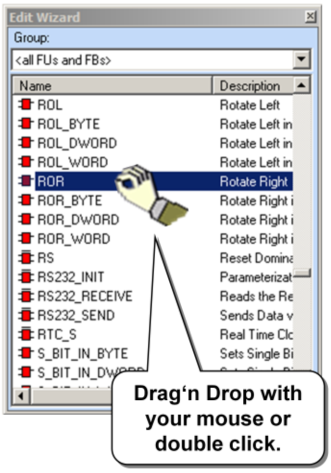
Comments as dummies

Parameter name and data type

```
LD     (* IN as ANY_BIT *)
ROR    (* N as ANY_INT *)
ST     (* Result as ANY_BIT *)
```

After editing by user

```
LD     wRawValue
ROR    INT#8
ST     wRotated
```



Edit Wizard

Group:

<all FUs and FBs>

Name	Description
ROL	Rotate Left
ROL_BYTE	Rotate Left in
ROL_DWORD	Rotate Left in
ROL_WORD	Rotate Left in
ROR	Rotate Right
ROR_BYTE	Rotate Right i
ROR_DWORD	Rotate Right i
ROR_WORD	Rotate Right i
RS	Reset Domina
RS232_INIT	Parameterizat
RS232_RECEIVE	Reads the Re
RS232_SEND	Sends Data v
RTC_S	Real Time Clc
S_BIT_IN_BYTE	Sets Single Bi
S_BIT_IN_DW	

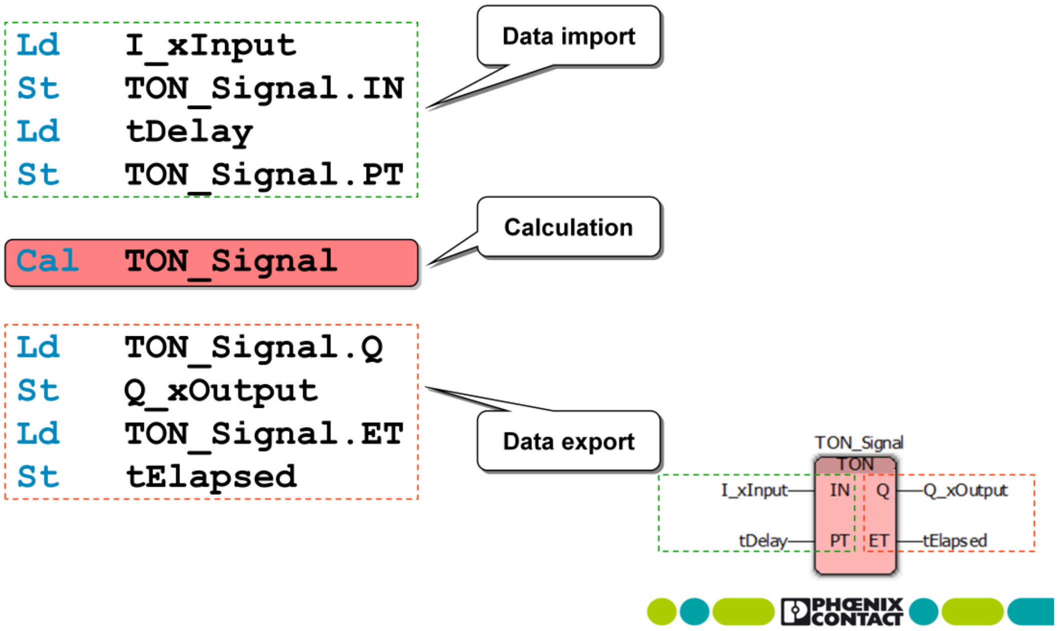
Drag'n Drop with your mouse or double click.

Editing a function in PC WORX can be carried out by entering the function syntax by means of the keyboard. If the parameters, the function spelling or the order of arguments are unknown, a syntax template can be inserted in the programming via the edit wizard.

The target variable and the arguments to be transmitted are inserted as comments by the edit wizard and must be replaced by variables.

Especially the nesting of functions cannot just be carried out using the edit wizard. It has proven to be helpful to view the syntax in blank lines and then do the actual programming.

Function Block Call



```
Ld    I_xInput
St    TON_Signal.IN
Ld    tDelay
St    TON_Signal.PT

Cal   TON_Signal

Ld    TON_Signal.Q
St    Q_xOutput
Ld    TON_Signal.ET
St    tElapsed
```

Data import

Calculation

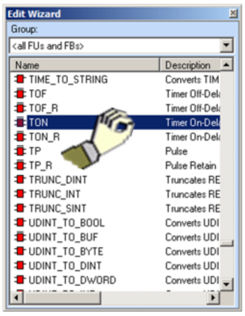
Data export

The function block call in instruction list is carried out, as in all other languages, in three phases:

1. Provision of values for the input parameters (data import).
2. Execution of block functionality, if necessary, by using saved data (calculation).
3. Saving the calculated values via the output parameters on variables created (data export).

Editing in PC WORX

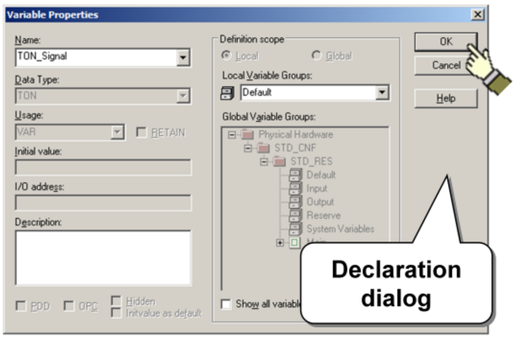
Inserting via the edit wizard



Edit Wizard

Group: <all FUs and FBs>

Name	Description
TIME_TO_STRING	Converts TIM
TOF	Timer Off-Del
TOF_R	Timer Off-Del
TON	Timer On-Del
TON_R	Timer On-Del
TP	Pulse
TP_R	Pulse Retain
TRUNC_DINT	Truncates RE
TRUNC_INT	Truncates RE
TRUNC_SINT	Truncates RE
UDINT_TO_BOOL	Converts UDI
UDINT_TO_BUF	Converts UDI
UDINT_TO_BYTE	Converts UDI
UDINT_TO_DINT	Converts UDI
UDINT_TO_DWORD	Converts UDI



Variable Properties

Name: TON_Signal

Data Type: TON

Usage: VAR □ RETAIN

Initial value:

I/O address:

Description:

□ PDD □ OPC □ Hidden □ Initvalue as default

Definition scope: ● Local ○ Global

Local Variable Groups: Default

Global Variable Groups: Physical Hardware, STD_CNF, STD_RES, Default, Input, Output, Reserve, System Variables

□ Show all variabl

Declaration dialog

```
LD   (* BOOL *)
ST   TON_Signal.IN
LD   (* TIME *)
ST   TON_Signal.PT
CAL  TON_Signal
LD   TON_Signal.Q
ST   (* BOOL *)
LD   TON_Signal.ET
ST   (* TIME *)
```

After editing

by the user

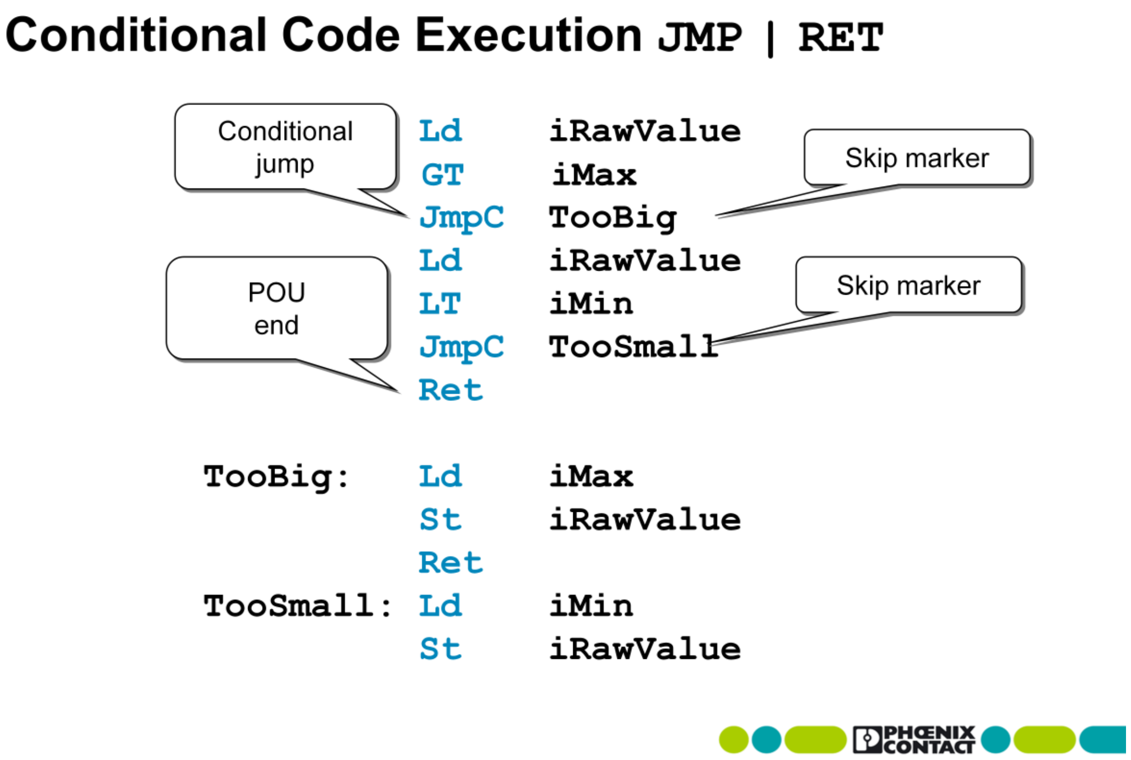
```
LD      xInput
ST      TON_Signal.IN
LD      TIME#250ms
ST      TON_Signal.PT

CAL     TON_Signal

LD      TON_Signal.Q
ST      xOutput
LD      TON_Signal.ET
ST      tElapsed
```

Just as for the functions, editing a function block in PC WORX can be carried out by entering the FB syntax via the keyboard. You have to keep in mind, though, that function blocks, unlike functions, have to be instantiated. This can either be done manually via the variable table or via the variable declaration dialog. The FB type is determined as data type.

If the edit wizard is used to add an FB to the programming, then the declaration of the instance is executed as in the graphical languages via the declaration dialog that opens. As for functions, the syntax parts of the call to be replaced with variables are inserted as comments.

The call of nested FBs cannot be implemented via the edit wizard. The wizard can only show the syntax of an individual call.

Notes:



Conditional Code Execution JMP | RET

```
          Ld     iRawValue
          GT     iMax
          JmpC   TooBig
          Ld     iRawValue
          LT     iMin
          JmpC   TooSmall
          Ret

TooBig:   Ld     iMax
          St     iRawValue
          Ret
TooSmall: Ld     iMin
          St     iRawValue
```

Conditional jump

Skip marker

POU end

Skip marker

The jump operator *JMP* and the function block end operator *RET* allow for the execution order deviating from the standard IL order and thus for the conditional execution of codes. For the *JMP*, the jump address must be determined. The skip marker is not declared. As a target it can be entered alone in one line or, as shown above, be placed in front of the instruction.

The modifier *C* allows for the conditional execution of the operators *JMP, RET* and *CAL*. The modifier *N* leads to an inversion of the execution condition.

Notes:

LD - Ladder Diagram

Chapter 15

Notes:

Contents

This chapter describes the language elements and their application required for programming in ladder diagram. These include contacts and spoils with their options as well as power rails.

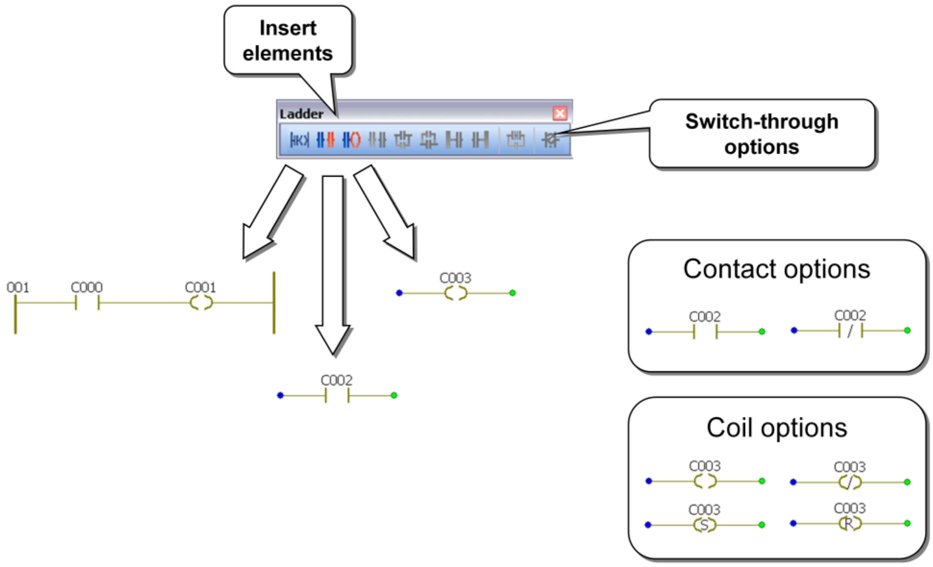
ℹ For programming in ladder diagram, which requires the use of non-boolean data types, variables, functions and function blocks can be used as in FBD. For information on editing, please refer to section *FBD – Function Block Diagram*.

⚠	Attention!
ℹ	Information
💡	Tip

Notes:

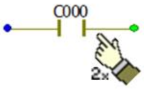
Basic Elements

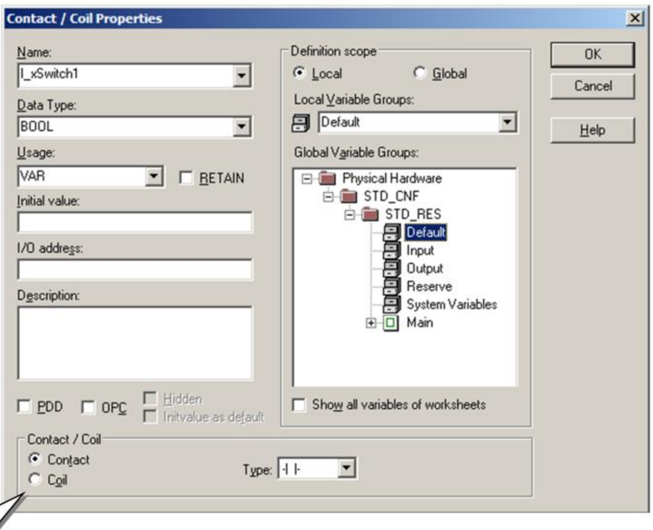
Editing in Ladder Diagram



The ladder diagram basic elements can be inserted and edited via the LD menu bar. Existing networks can also be extended and completed using these buttons. Depending on the element selected in the worksheet, the corresponding buttons are enabled.

Contact/Coil Dialog





Contact / Coil Properties

Name: I_xSwitch1

Data Type: BOOL

Usage: VAR □ RETAIN

Initial value:

I/O address:

Description:

□ PDD □ OPC □ Hidden □ Initvalue as default

Definition scope: ● Local ○ Global

Local Variable Groups: Default

Global Variable Groups:

- Physical Hardware
  - STD_CNF
    - STD_RES
      - Default
      - Input
      - Output
      - Reserve
      - System Variables
      - Main

□ Show all variables of worksheets

Contact / Coil: ● Contact ○ Coil Type: -| |-

Additional options compared to variable dialog

The dialog for setting the contact/coil properties only differs slightly from the standard variable dialog. Additionally, control elements for setting the LD element are available.

Furthermore, only those data types that allow access to a boolean parameter can be selected.

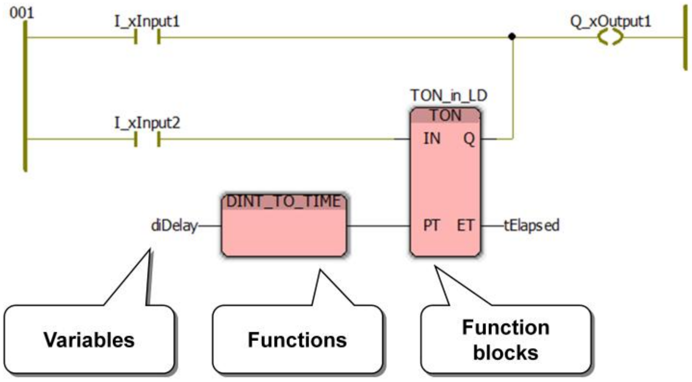
In addition to the *Bool* data type, this includes the other bit-based data types in case of an implicit bit access, e.g. **`I_wFeedback.x13`**. and thus the access to the 13th bit of the *Feedback* input process data word. Elements of a user-defined data type can also be used (see section *User-Defined Data Types*).

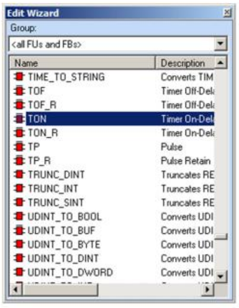
Notes:

Extended Editing

Notes:

Functions and Function Blocks in Ladder Diagram




Edit Wizard

Group:

<all FUs and FBs>

Name	Description
TIME_TO_STRING	Converts TIM
TOF	Timer Off-Del
TOF_R	Timer Off-Del
TON	Timer On-Del
TON_R	Timer On-Del
TP	Pulse
TP_R	Pulse Retain
TRUNC_DINT	Truncates RE
TRUNC_INT	Truncates RE
TRUNC_SINT	Truncates RE
UDINT_TO_BOOL	Converts UDI
UDINT_TO_BUF	Converts UDI
UDINT_TO_BYTE	Converts UDI
UDINT_TO_DINT	Converts UDI
UDINT_TO_DWORD	Converts UDI

As the same editor is used, the elements of function block diagram (variables, functions and function blocks) can be used without problems in ladder diagram. For details on editing and handling function block diagram, please refer to section *FBD – Function Block Diagram*.

Too much blending of the two languages within one network might lead to results that cannot be interpreted clearly by the compiler and thus to error messages.

Notes:

SFC - Sequential Function Chart

Chapter 16

Notes:

Contents

This chapter informs you on graphic step chain programming in PC WORX using the IEC 61131 language *sequential function chart*. First of all, the basic structure is described and afterwards the basic elements of programming (step, action and transition) as well as the different ways to program them.

ℹ Programming the details for transitions and actions is done in another IEC 61131 language. Therefore, the user needs to have knowledge of at least one further language for editing these elements of the sequential function chart.

⚠ Attention!

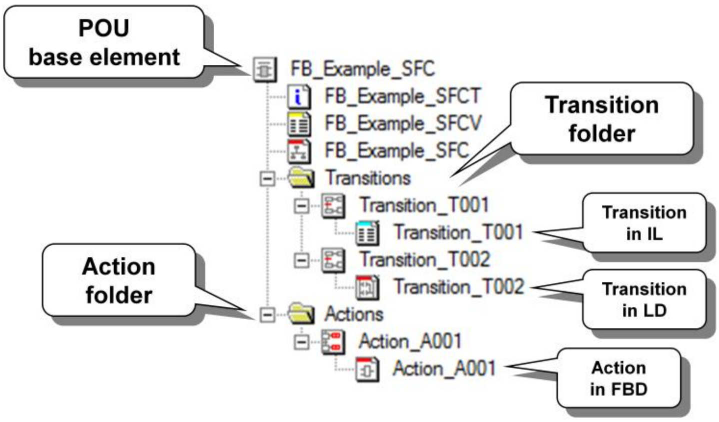
ℹ Information

💡 Tip

Notes:

Basic Structure of the Sequential Function Chart

Sequential Function Chart POU in the Project Tree

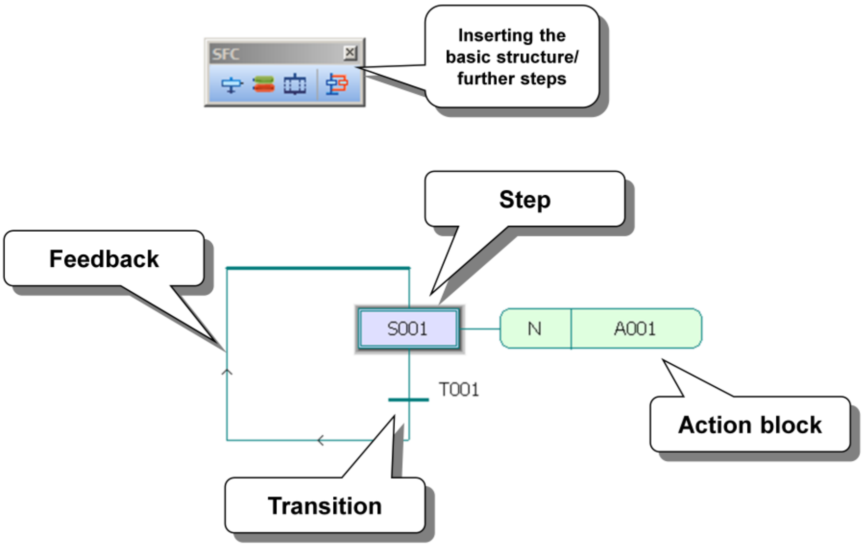


Above example shows a function block in sequential function chart in the project tree.

The step chain is no further POU type. The sequential function chart allows for programming POUs of the types *function blocks* and *program*.

Unlike the POU structure in the other IEC 61131 languages, the sequential function chart includes additional elements. In addition to text sheet, variable table and code worksheet, in which in SFC the actual step chain is programmed, there are two folders that contain the existing transitions and folders.

Basic Structure



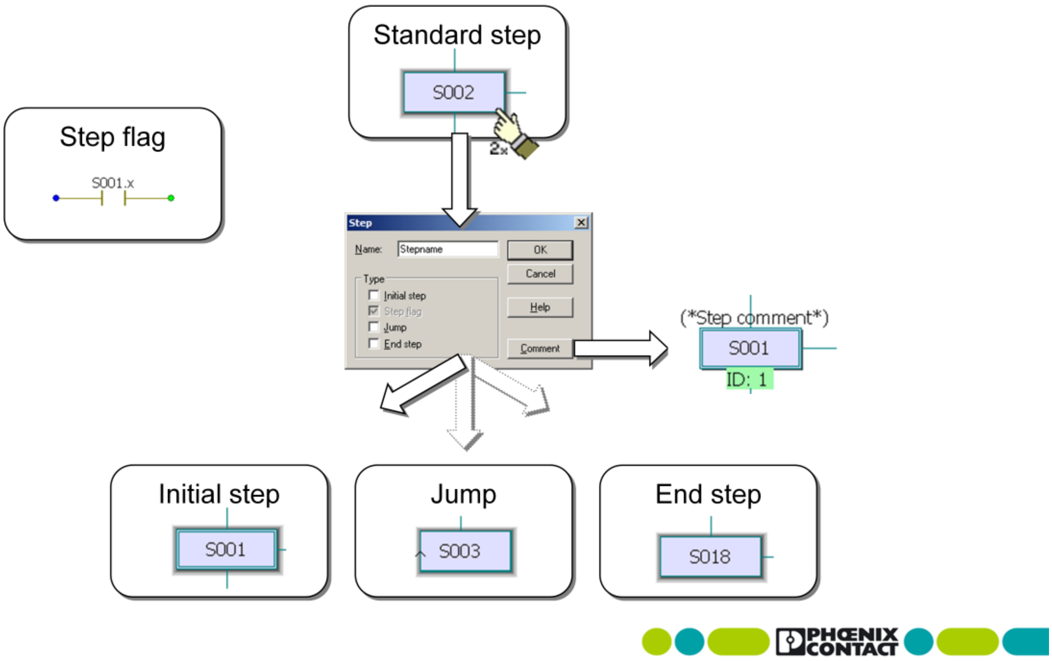
The basic structure, which can be inserted into a step chain worksheet via the button shown above after setting the insert mark, consists of the two basic elements Step (here: S001) and Transition (here: T001) with an action block assigned to the step (here: A001). The names of all elements can be adapted.

A step chain is also built from a sequence of step and transition. However, using actions is not required (Step without action waiting step).

Notes:

Basic Elements

The Step



The first step is part of the basic structure and inserted automatically. Afterwards, the system automatically only inserts standard steps. Via the step dialog a comment can be entered for each step.

However, only the step flag `<Stepname>.x` is automatically declared for a step in a step chain. This parameter represents the status, i.e. the activity of the step and can be read and, for special requirements, also be written (not allowed according to IEC 61131). It is valid within a POU.

Via the step dialog, the step can also be set to meet one of the versions required for specific requirements.

Initial step

Each step chain (in case of more than one worksheet) must at least have one defined initial step. If a start is implemented from a parallel branch, this is required for every parallel branch. Action blocks can be assigned to initial steps.

Jump

Unlike the other step types, a jump does not represent a process situation. It has to be regarded as a direct jump to the target step indicated above the jump name. For the jump the exception holds true that it has to have the same name as another step.

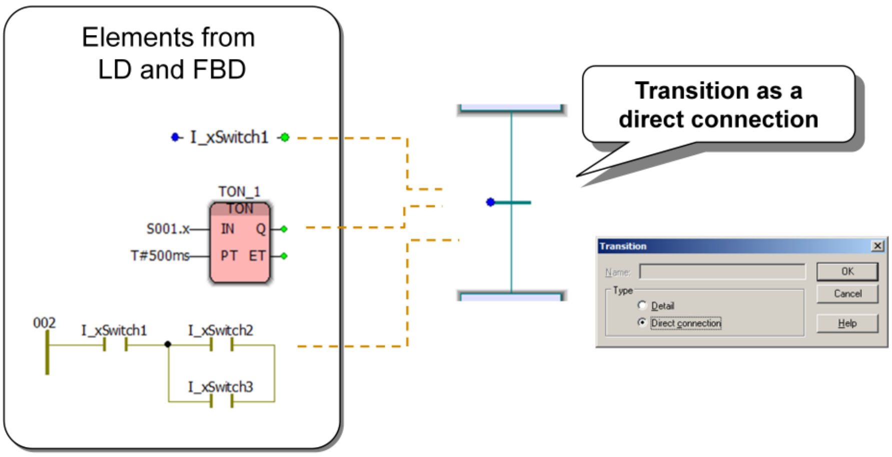
End step

For the program sequence, the jump represents a dead end. If it is reached once, a new execution of the step chain can only be achieved by manipulating the step chain by setting/resetting step flags.

A step chain ending with an end step is used for initialization. Action blocks can be assigned to end steps.

⚠ For the jump as well as for the end step, the connection of the step chain structure has to be interrupted. This change cannot be reversed later on.

The Transition



Elements from LD and FBD

Transition as a direct connection

Transition

Name:

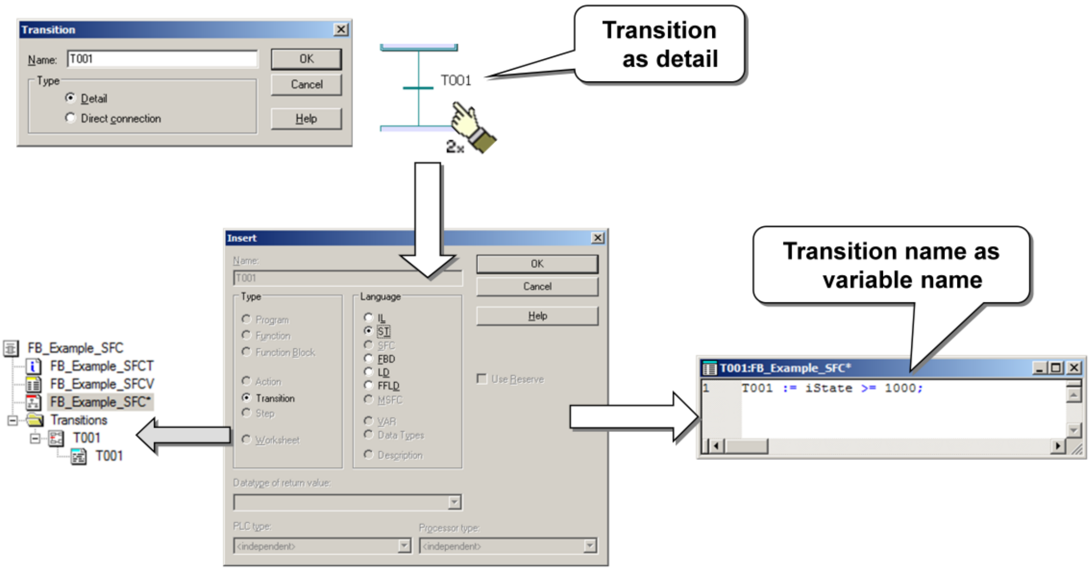
Type
○ Detail
● Direct connection

Direct connection

For this kind of programming on the step chain interface, the transition is switched to *Direct connection* via the properties dialog. The transition is then no longer named, but receives a terminal point as known with functions and function blocks. A boolean parameter is to be assigned to this terminal point to implement the switching requirement.

For this, as shown above, the ladder diagram and function block diagram elements can be used (variables, contacts, functions and function blocks).

The Transition Detail



Transition as detail

Transition name as variable name

Transition

Name: T001

Type: ● Detail ○ Direct connection

Insert

Name: T001

Type: ○ Program ○ Function ○ Function Block ○ Action ● Transition ○ Step ○ Worksheet

Language: ○ IL ● ST ○ SFC ○ FBD ○ LD ○ FFLD ○ MSFC ○ VAR ○ Data Types ○ Description

□ Use Reserve

Datatype of return value:

PLC type: <independent>

Processor type: <independent>

T001:FB_Example_SFC*

```
T001 := iState >= 1000;
```

Within a step chain, transitions implement the transition conditions of one process situation into the following. There are two ways to program these conditions.

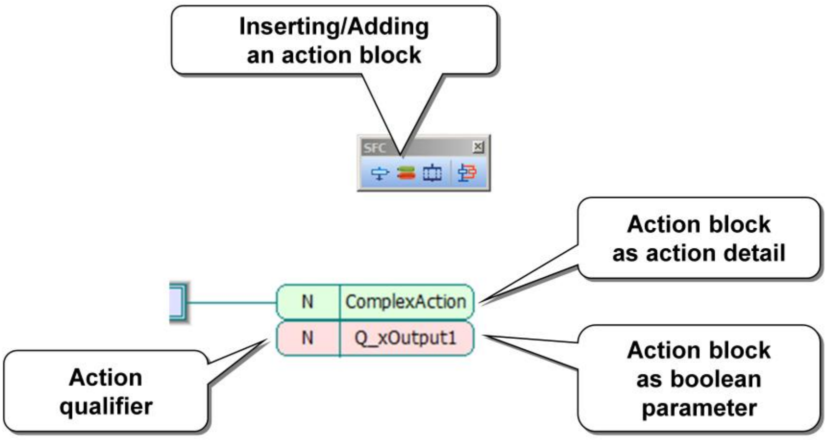
Transition detail

The transition detail represents one ore more worksheets that can be programmed in one of the IEC 61131 languages, with the exception of SFC. After adapting the transition name, you can open the insert dialog for the new element by double-clicking on the transition name. When the language has been selected, the transition is inserted into the POU transition folder. Concurrently, the first worksheet is opened. Programming in the transition has to create the switching requirement, which – similar to the return value for functions – has to be saved onto a variable with the name of the transition. Unlike the function names, this parameter is not already saved in the variable dialog, but has to be entered manually. The system nevertheless identifies the correctly entered name and then blocks the control elements that are usually enabled for a variable declaration.

For later adjustments, the transition detail can be opened from the step chain or via the corresponding element in the project tree.

The detail can be copied in the project tree for similar programming of transition details. All that has to be done after changing the name, is adapting the transition name and programming the deviations in the detail.

The Action Block


Inserting/Adding an action block
SFC
Action block as action detail
N
ComplexAction
N
Q_xOutput1
Action qualifier
Action block as boolean parameter

Action blocks influence the process situation just as steps represent process situations. They can be executed as action detail (standard color: light green). This means they can be used as programming in separate worksheets or for controlling boolean parameters (standard color: pale pink). The use can be set in the properties dialog of the action block.

In addition the status of the step, the action qualifier is important for executing the action. Besides the *N*, the character mostly used (*N* = nonstored, action active as long as step is active), all other characters according to IEC 61131-3 can be used:

Action qualifier

S **set** This action is continued until it is explicitly reset (even if the corresponding step becomes inactive).

R **reset** Terminates the execution of action previously started with character *S*.

L **time limited** After activating the corresponding step, the action is executed for the set time or until step becomes inactive.

D **time delayed** Delay starts when step becomes active. After a set time has passed, the action is executed until step is deactivated. If the step becomes inactive before the set time has passed, the action is not executed.

P **pulse** Starts when step becomes active/inactive and is executed once.

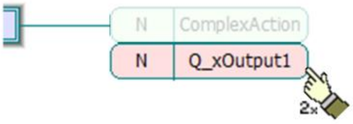
SD **stored+ time delayed** The action is executed until reset, but only starts after time delay has passed. Even if the step is active for less a time than the time delay.

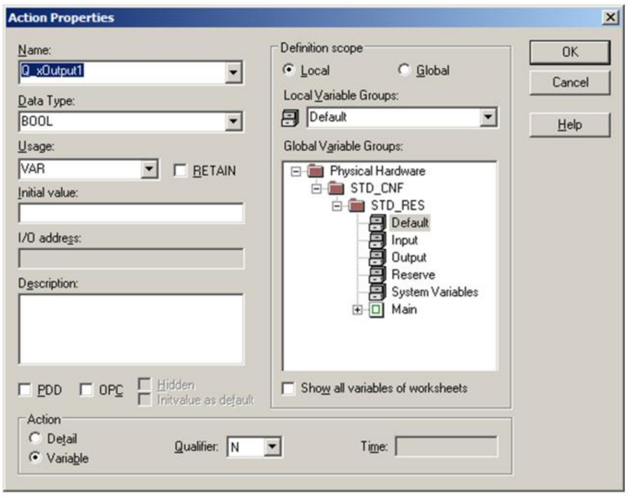
DS **time delayed+ stored** The action is executed until reset, but only starts after time delay has passed. Unlike with the previous character, this does not happen if the step becomes inactive before time delay has passed

SL **stored+ time limited** The action is executed for the set time, even if the step is active for a shorter time. The action ca be reset with the character *R*.

The time has to be set according to the standard of a time constant.

Action Variable





Action Properties

Name: Q_xOutput1

Data Type: BOOL

Usage: VAR □ RETAIN

Initial value:

I/O address:

Description:

□ PDD □ OPC □ Hidden □ Initvalue as default

Action
○ Detail
● Variable

Qualifier: N

Time:

Definition scope
● Local ○ Global

Local Variable Groups: Default

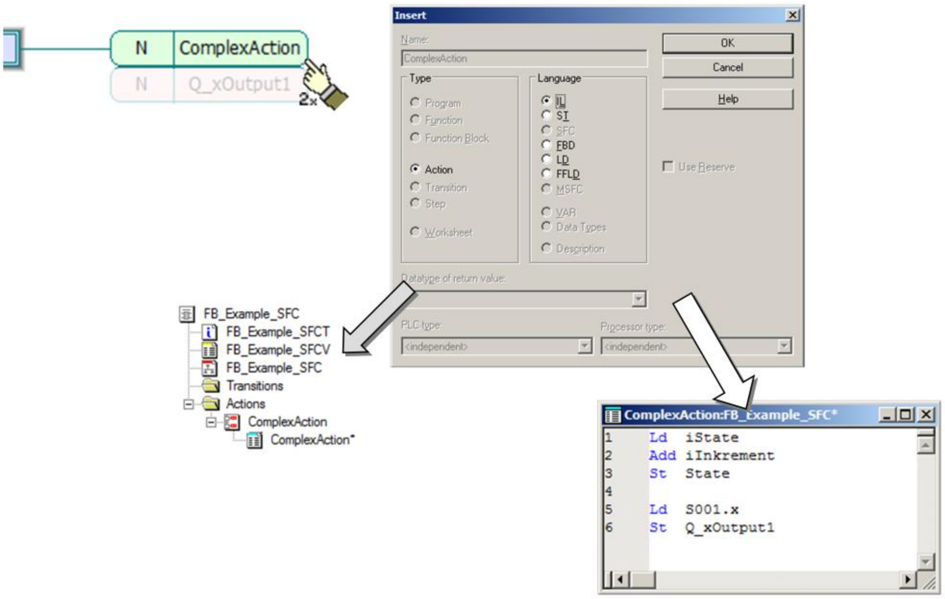
Global Variable Groups:
- Physical Hardware
  - STD_CNF
    - STD_RES
      - Default
      - Input
      - Output
      - Reserve
      - System Variables
      - Main

□ Show all variables of worksheets

As soon as an action is added to a step as a variable or the action is set to variable in the properties menu of the action block, a variable has to be declared or called following the known procedure.

For this, boolean variables or – as in ladder diagram – boolean parameters can be used by implicitly accessing individual bits as in word, byte or dword variables, or boolean parameters of structure and field variables.

Action Detail



Insert

Name: ComplexAction

Type: ○ Program ○ Function ○ Function Block ● Action ○ Transition ○ Step ○ Worksheet

Language: ● IL ○ ST ○ SFC ○ FBD ○ LD ○ FFLD ○ MSFC ○ VAR ○ Data Types ○ Description

□ Use Reserve

Datatype of return value:

PLC type: <independent>

Processor type: <independent>

ComplexAction:FB_Example_SFC*

```
Ld   iState
Add  iInkrement
St   State

Ld   S001.x
St   Q_xOutput1
```

If an action block is used as action detail, the name to be used for the complex action has to be edited via the context menu in the properties dialog. In this context, complex action refers to everything that exceeds controlling an individual boolean variable.

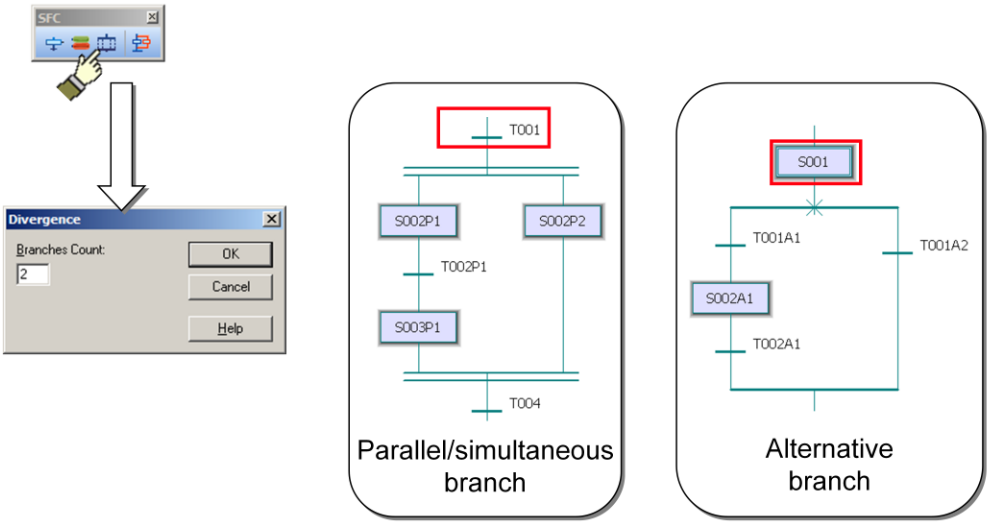
Once the name is assigned, the detail can be created by a double click. In the dialog opening now, only the language for programming has to be selected. Except for the sequential function charts, all IEC 61131 languages are available. With the opening of the first action detail worksheet, the corresponding element in the *Actions* folder of the POU is created. Now, the worksheets can be selected in the project tree or directly from the step chain using the action block.

The detail can be copied in the project tree for similar programming of action details. All that has to be done after changing the name, is adapting the action block name and programming the deviations in the detail.

For above detail the step flag *S001.x* was used to activate a boolean variable for the period of activity of the step. It would also have been possible to use an additional action block with the corresponding boolean variable and action qualifier *N*.

Notes:

Branches



Parallel/simultaneous branch

Alternative branch

The IEC 61131-3 sequential function chart allows for the creation of simultaneous and alternative branches.

Simultaneous branch

For inserting a simultaneous branch, the step after which the branch is to start must be selected The number of branches that can be inserted depends on the size of the worksheet. The branches are executed independently. The execution is only grouped if the end steps are active in all branches and the switching requirement of the grouping transition is met.

Integrating jumps and end steps in simultaneous branches is rejected by PC WORX as non-permissible programming.

Alternative branch

For inserting an alternative branch, the step after which the branch is to start has to be selected The number of branches that can be inserted depends on the size of the worksheet. The branch whose initial transition condition is met first is then executed. If more than one condition is met at the same time, execution is done according to the execution order in graphic worksheets, i.e. in prioritization from left to right.

It is possible to integrate jumps and end steps in alternative branches.

The insertion of parallel and alternative branches can also be done subsequently. For an alternative branch, both transitions must be highlighted above which the branches have to be inserted; for a parallel branch, both steps must be highlighted. The branch is inserted using the button in the SFC toolbar.

Notes:

User-Defined Data Types

Chapter 17

Notes:

Contents

A high number of IEC 61131-defined data types are made available by PC WORX for declaring variables. Moreover, the IEC provides for the declaration of user-defined data types for compliant programming systems. These data types (UDTs) are a summary of standard data types.

This section informs you about the different classes of these data types, how they are declared and how variables based on them can be used in programming.

💡 The declaration of user-defined data types and thus the use of complex variables is efficient, especially in combination with the high-level language-specific program structures of Structured Text. The programming examples are described for all languages, though.

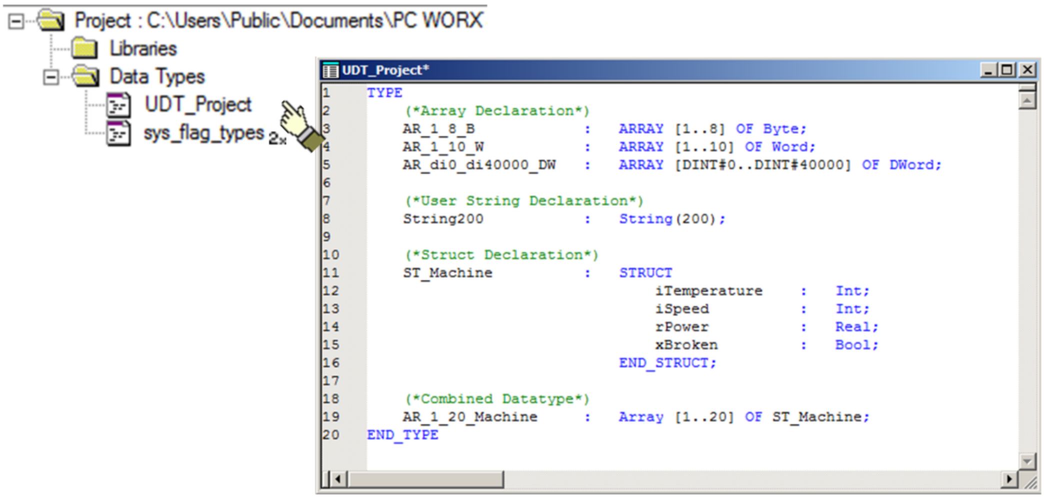
⚠ Attention!

ℹ Information

💡 Tip

Notes:

Worksheets for User-Defined Data Types

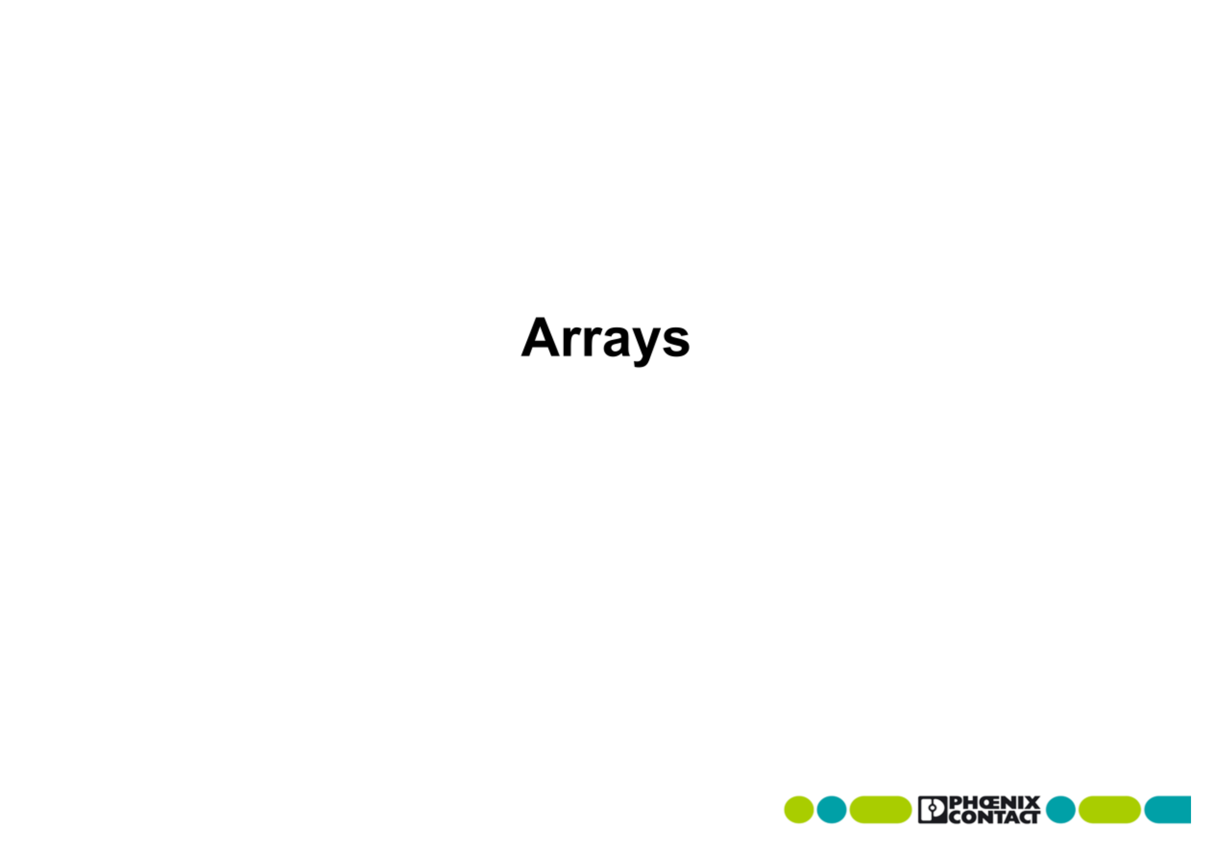


```
TYPE
    (*Array Declaration*)
    AR_1_8_B            :   ARRAY [1..8] OF Byte;
    AR_1_10_W           :   ARRAY [1..10] OF Word;
    AR_di0_di40000_DW   :   ARRAY [DINT#0..DINT#40000] OF DWord;

    (*User String Declaration*)
    String200           :   String(200);

    (*Struct Declaration*)
    ST_Machine          :   STRUCT
                                iTemperature    :   Int;
                                iSpeed          :   Int;
                                rPower          :   Real;
                                xBroken         :   Bool;
                            END_STRUCT;

    (*Combined Datatype*)
    AR_1_20_Machine     :   Array [1..20] OF ST_Machine;
END_TYPE
```

The declaration of user-defined data types is carried out on worksheets of the *Data Types* folder. The worksheet is included in each project template in PC WORX and is required for the system data types of the control systems. The worksheet itself can be renamed (*SYSTEM*), the data type declarations contained in it must not be amended, though.

Basically, all declarations for one project can be done in one worksheet. However, to keep everything clearly arranged, it is recommended to create function- related, individual worksheets.

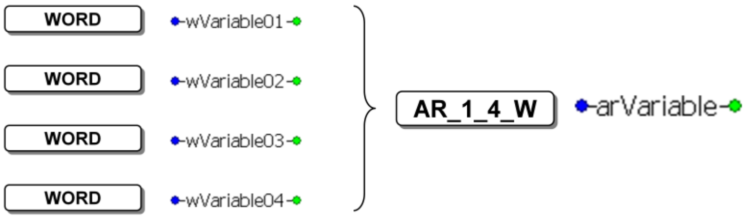
Notes:



Arrays

Notes:

Principle



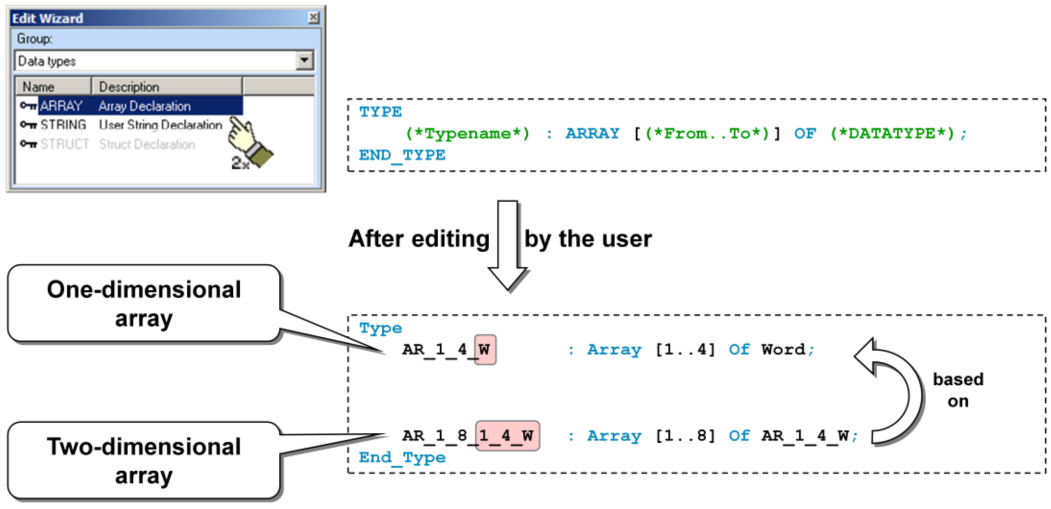
Name example

<AR_LowerBound>_<UpperBound>_<BasicType>

Data type declarations of the *arrays* class summarize elements in one basic data type. The example shows the summary of four variables of the data type Word in one array variable based on a new data type.

The name of the new data type can be selected freely. However, just as with all other elements in programming, it is recommended to use a convention for the names of user-defined data types. One example is shown above.

Declaration



```
TYPE
    (*Typename*) : ARRAY [(*From..To*)] OF (*DATATYPE*);
END_TYPE
```

After editing by the user

One-dimensional array

Two-dimensional array

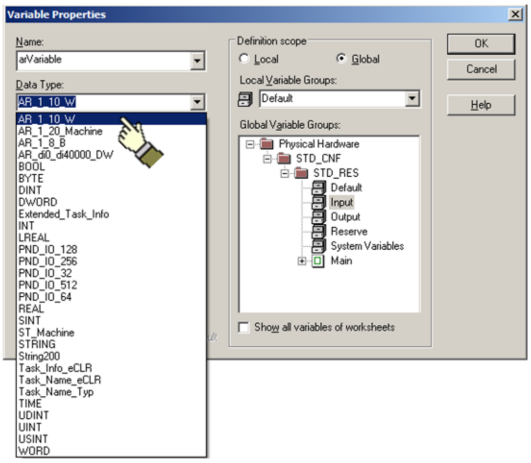
```
Type
    AR_1_4_W        : Array [1..4] Of Word;
    AR_1_8_1_4_W    : Array [1..8] Of AR_1_4_W;
End_Type
```

For declaring arrays, a certain format has to be used, which can be called as a help via the edit wizard adapted to the data type declarations. The desired name for the data type, the lower and upper bound of the array (in square brackets and as positive integer value) and the basic data type have to be entered.

Declaring multi-dimensional arrays is usually possible following the procedure described on the previous page. The example, however, shows that the declaration and thus the later use of variables of these data types might lead to difficulties regarding clarity.

For data types based on one another, the declaration order within the worksheet has to be adhered to (error message in PC WORX).

Use in Programming



Variable Properties

Name: arVariable

Data Type: AR_1_10_W

Definition scope: ○ Local ● Global

Local Variable Groups: Default

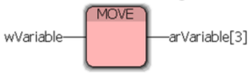
Global Variable Groups:

□ Show all variables of worksheets

Assigning two array variables of the same dimension

```
arVariable1 := arVariable2;
```

Assigning a single value to an element of an array variable using a constant



Assigning a single value to an element of an array variable using an index variable

```
Ld wVariable
St arVariable[iIndex]
```

After closing or compiling the data type worksheet (Alt+F9), the newly declared data types are available in the data type selection list. Two variables of the same data type, and this also holds true for arrays, can be assigned to each other.

Accessing individual elements can either be done after declaring the array variable in the POU and using an integer constant or using an integer variable which operates as an index for the array.

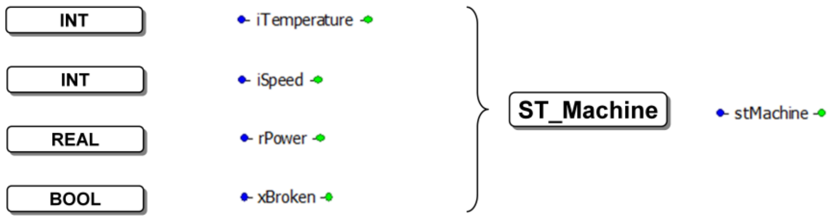
If variables are used for accessing single elements, it has to be assured through programming that array bounds are not violated.

Notes:

Structures

Notes:

Principle



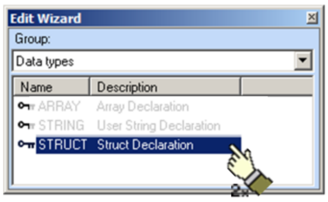
Name example

`ST_<Function>`

Data type declarations of the *Structures* class can summarize elements of one ore more basic data types. The example shows the summary of four variables of different data types in one structure variable based on a new data type. As with arrays, elements of the same type can be summarized if the access to the individual elements shall not be given by an index but by individual designations.

The name of the new data type can be selected freely. However, just as with all other elements in programming, it is recommended to use a convention for the names of user-defined data types. One example is shown above.

Declaration



```
TYPE
        (*Typename*)    :
        STRUCT
                (*Element 1 Name*)   :   (*DATATYPE*);
                (*Element 2 Name*)   :   (*DATATYPE*);
                (*Element 3 Name*)   :   (*DATATYPE*);
                (*       .           :          .    *);
                (*       .           :          .    *);
                (*Element n Name*)   :   (*DATATYPE*);
        END_STRUCT;
END_TYPE
```

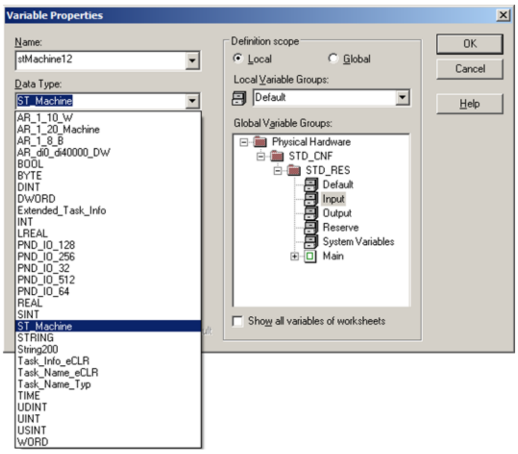
after editing ⇩ by the user

```
Type
    ST_Machine  :   Struct
                        iTemperature      : Int;
                        iSpeed            : Int;
                        rPower            : Real;
                        xBroken           : Bool;
                    End_Struct;
End_Type
```

For declaring structures, a certain format has to be used, which can be called as a help via the edit wizard adapted to the data type declarations. The desired data type name is to be entered and the named elements shown above have to be listed. The declaration of multi-dimensional structures is supported.

For data types based on one another, the declaration order within the worksheet has to be adhered to (error message in PC WORX).

Use in Programming



Variable Properties

Name: stMachine12

Data Type: ST_Machine

Definition scope: ● Local ○ Global

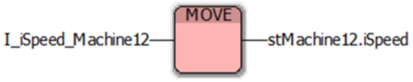
Local Variable Groups: Default

□ Show all variables of worksheets

Assigning two structure variables of the same dimension

```
stMachine12 := stMachine13;
```

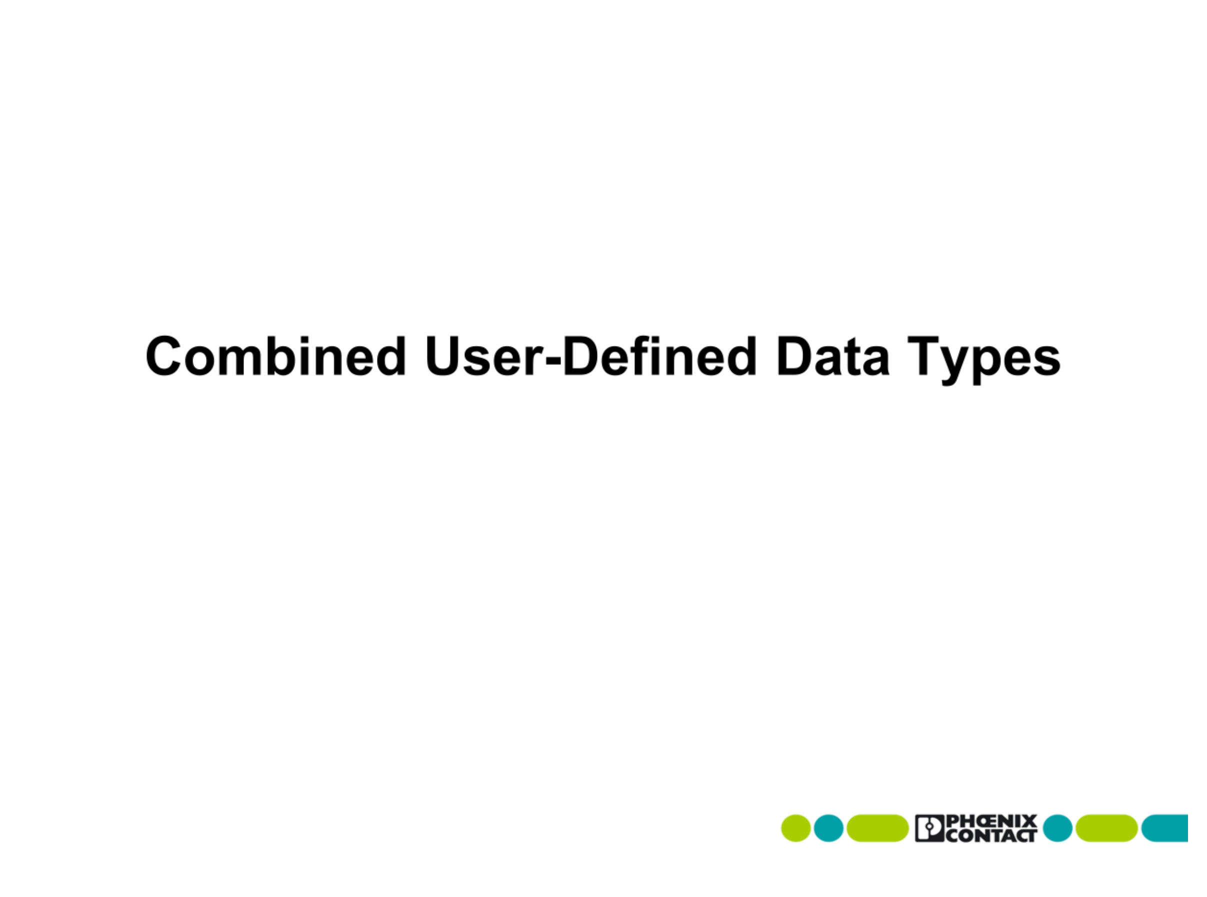
Assigning a single value to an element of a structure variable using named parameters



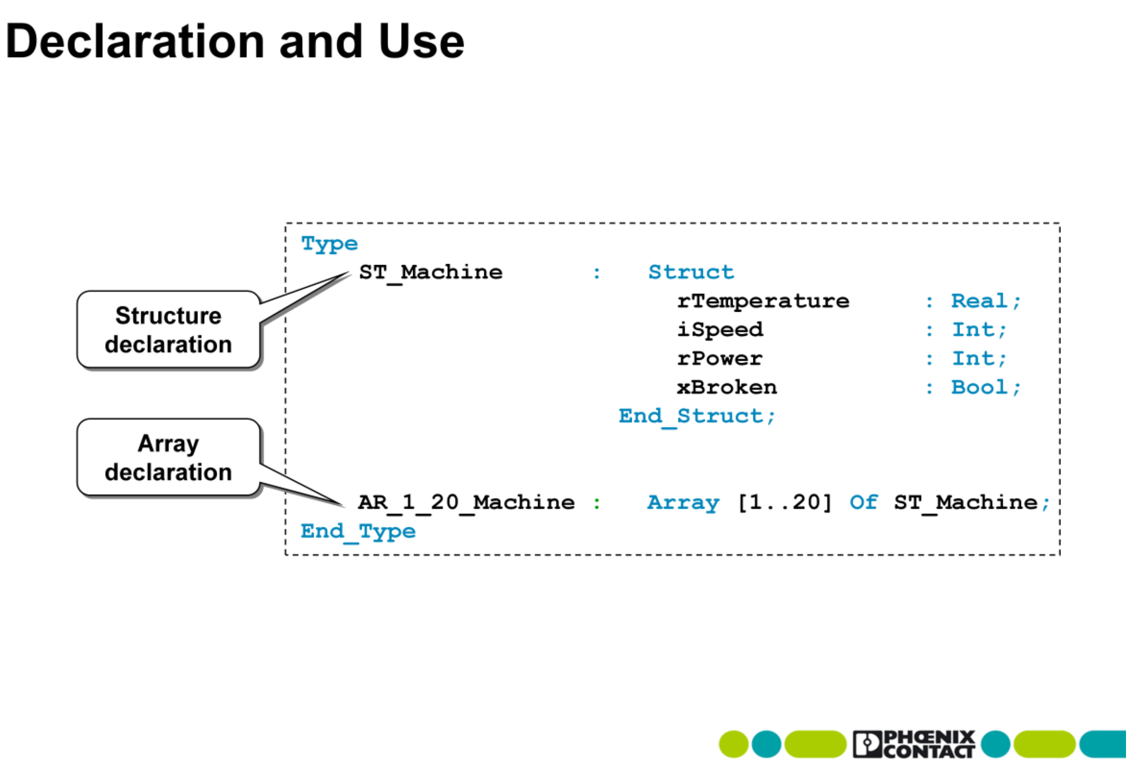
After closing or compiling the data type worksheet (Alt+F9), the newly declared data types are available in the data type selection list. Two variables of the same data type, and this also holds true for structures, can be assigned to each other.

After declaration of the structure variable in the POU, single elements can be accessed via the named parameter.

Notes:



Combined User-Defined Data Types



Declaration and Use

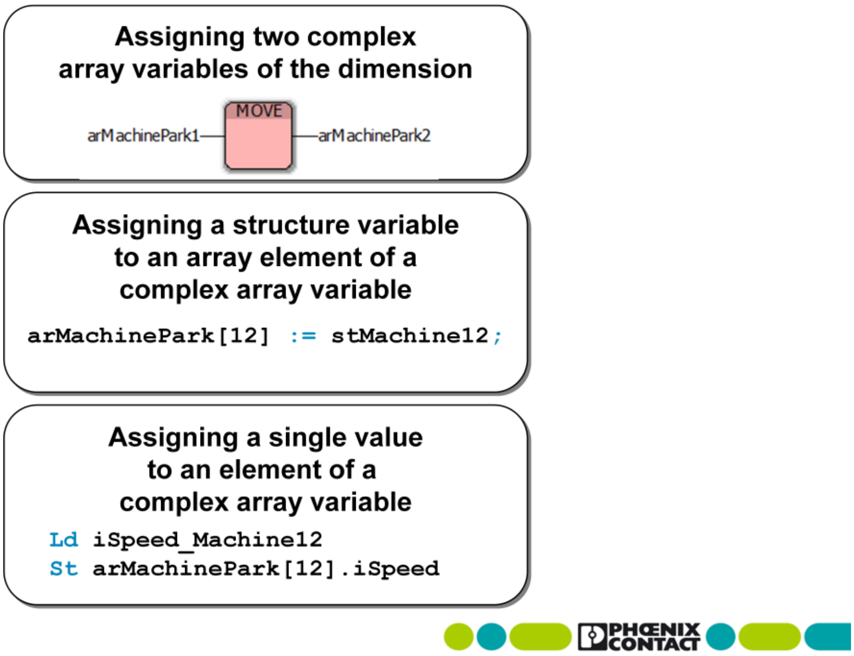
Structure declaration

Array declaration

```
Type
    ST_Machine      :   Struct
                          rTemperature      : Real;
                          iSpeed            : Int;
                          rPower            : Int;
                          xBroken           : Bool;
                        End_Struct;

    AR_1_20_Machine :   Array [1..20] Of ST_Machine;
End_Type
```

In many projects, the data type declarations of arrays and structures are combined. The example shows the declaration of a data type that comprises variables that can acquire the data of a pump. The declaration of an array is based on this declaration. The array combines 20 elements, indicated 1 to 20, of the pump structure.

Use in Programming



Assigning two complex array variables of the dimension

arMachinePark1
MOVE
arMachinePark2

Assigning a structure variable to an array element of a complex array variable

```
arMachinePark[12] := stMachine12;
```

Assigning a single value to an element of a complex array variable

```
Ld iSpeed_Machine12
St arMachinePark[12].iSpeed
```

The access to an element of a variable based on the pump array follows the motto "from large to small". This means that first of all it has to be specified, which pump within the array is to be accessed (INT variable or INT constant in square brackets) and then the parameter of the specified pump (via named parameter) has to be specified.

Notes:

ST Structured Text

Chapter 18

Notes:

Contents

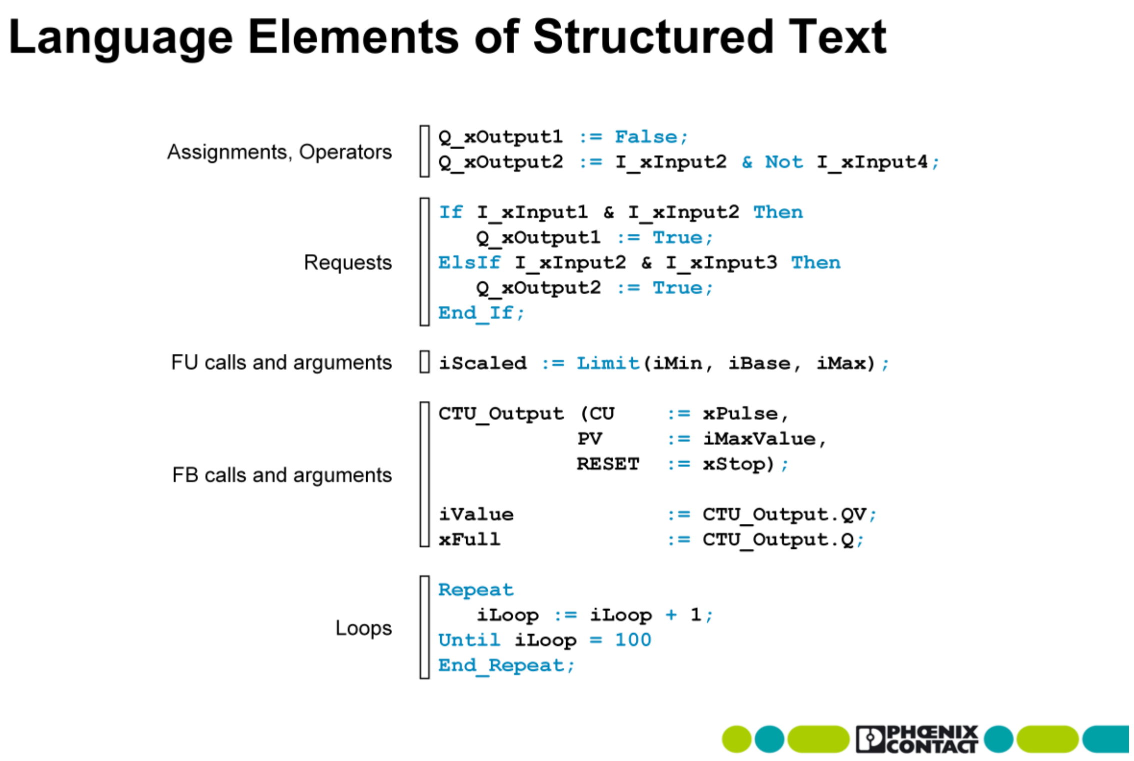
This chapter informs you about the programming possibilities in Structured Text. These include elements like operators, functions and function blocks, which are also available in the other languages, but also high-level language-specific elements for requests and loop programming.

⚠ Attention!

ℹ Information

💡 Tip

Notes:



Language Elements of Structured Text

Assignments, Operators

```
Q_xOutput1 := False;
Q_xOutput2 := I_xInput2 & Not I_xInput4;
```

Requests

```
If I_xInput1 & I_xInput2 Then
   Q_xOutput1 := True;
ElsIf I_xInput2 & I_xInput3 Then
   Q_xOutput2 := True;
End_If;
```

FU calls and arguments

```
iScaled := Limit(iMin, iBase, iMax);
```

FB calls and arguments

```
CTU_Output (CU    := xPulse,
            PV    := iMaxValue,
           RESET  := xStop);

iValue            := CTU_Output.QV;
xFull             := CTU_Output.Q;
```

Loops

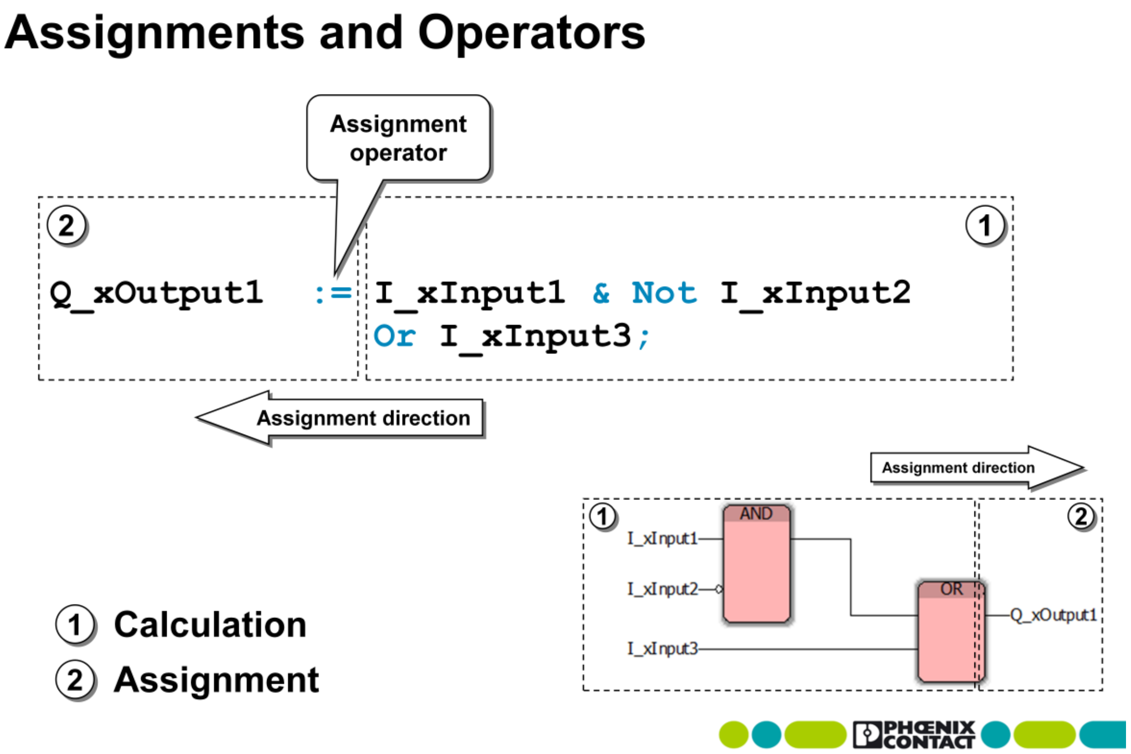
```
Repeat
   iLoop := iLoop + 1;
Until iLoop = 100
End_Repeat;
```

The access to the language elements available in Structured Text is mostly supported by the edit wizard. In addition to the *Functions* and *Function block* groups, the group *Keywords* is available in Structured Text.

Notes:

Basic Programming

Operators, Functions and Function Blocks



Assignments and Operators

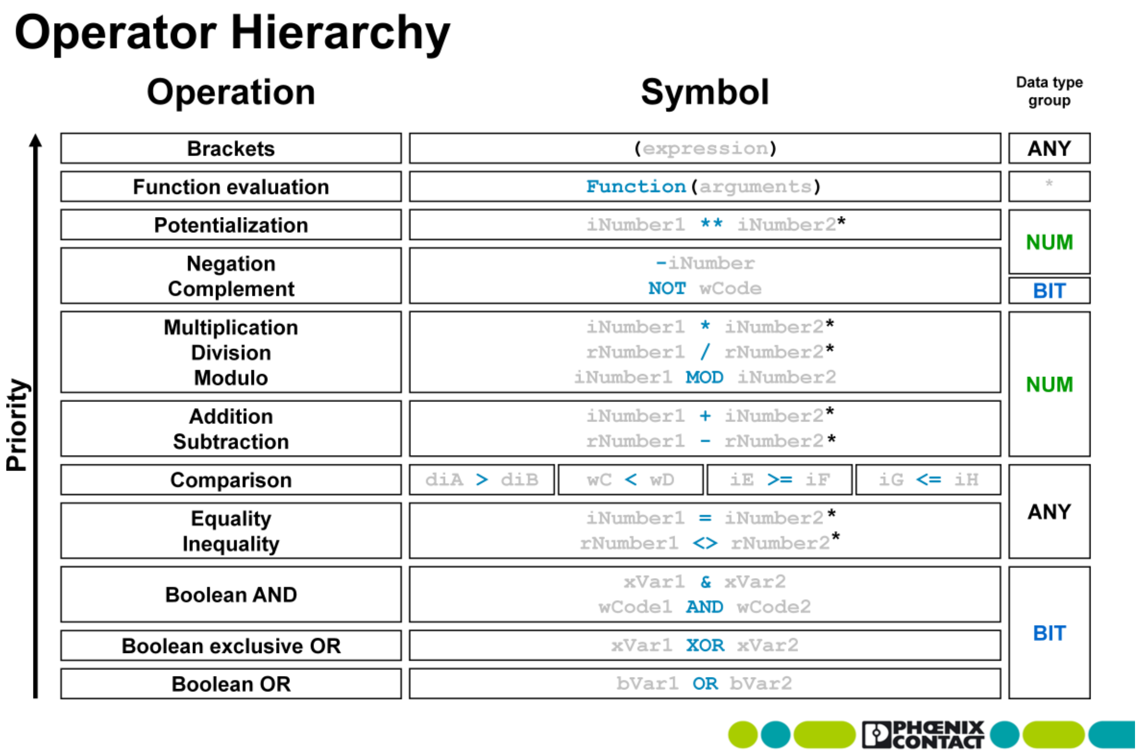
```
Q_xOutput1   := I_xInput1 & Not I_xInput2
                Or I_xInput3;
```

1 Calculation

2 Assignment

The major difference between graphical and textual languages is that the assignment of expressions (constants, variables and calculations) is carried out from left to right in graphical languages and from right to left in textual languages. Regarding the command set, the operators and language elements, there are no differences between these two types of programming languages.

The key elements that are the basis for the assignments are always the assignment operator := and the semicolon for ending an instruction. As shown in the example above, instructions can run over several lines to increase programming clarity.



Operator Hierarchy

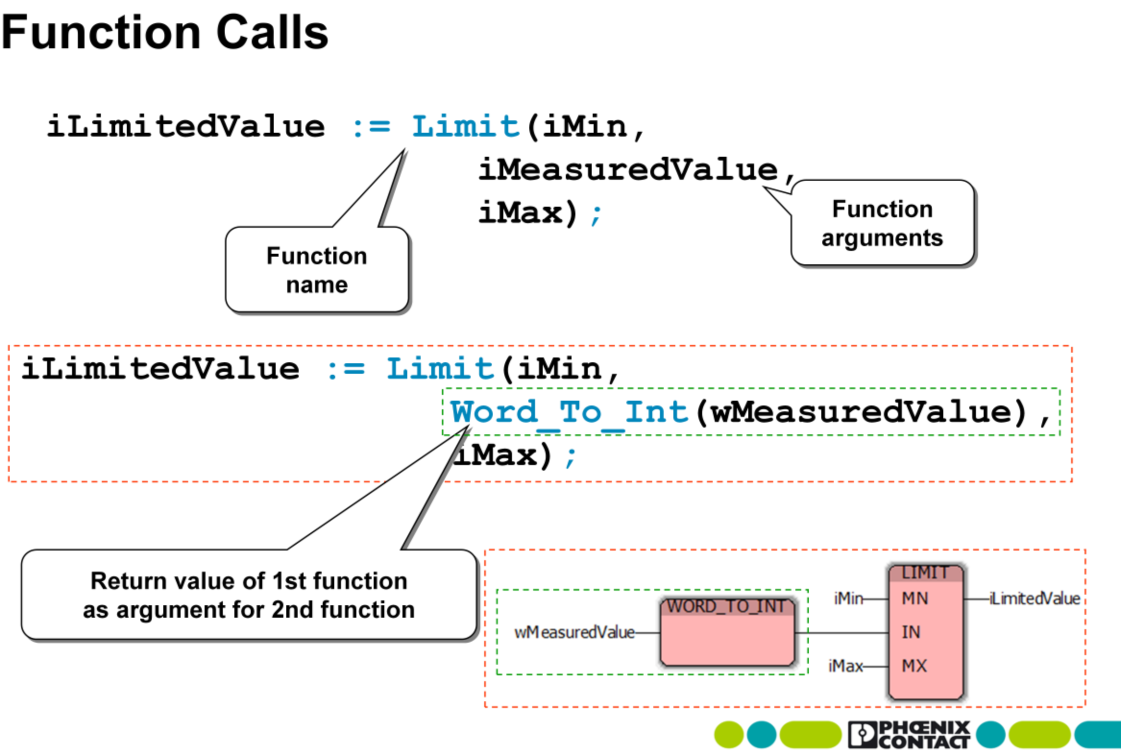
Priority

Operation	Symbol	Data type group
Brackets	(expression)	ANY
Function evaluation	Function(arguments)	*
Potentialization	iNumber1 ** iNumber2*	NUM
Negation	-iNumber	
Complement	NOT wCode	BIT
Multiplication Division Modulo	iNumber1 * iNumber2* rNumber1 / rNumber2* iNumber1 MOD iNumber2	NUM
Addition Subtraction	iNumber1 + iNumber2* rNumber1 - rNumber2*	
Comparison	diA > diB wC < wD iE >= iF iG <= iH	ANY
Equality Inequality	iNumber1 = iNumber2* rNumber1 <> rNumber2*	
Boolean AND	xVar1 & xVar2 wCode1 AND wCode2	BIT
Boolean exclusive OR	xVar1 XOR xVar2	
Boolean OR	bVar1 OR bVar2	

The operators available in Structured Text are subject to a certain order in their execution. This can be of significant importance if more than one operator is used in one expression. Using brackets, the user can always change these priorities. However, brackets can also be used to achieve higher clarity in programming and for relations.

The *Data type group* column in above list tells you for which data type or data type group the operators can be used.

* If the option *Enable DIN identifiers* is activated via *Tools* → *Options*→ *Code*, spaces have to be entered for a clear interpretation of the source code.



Function Calls

```
iLimitedValue := Limit(iMin,
                       iMeasuredValue,
                       iMax);
```

Function name

Function arguments

```
iLimitedValue := Limit(iMin,
                       Word_To_Int(wMeasuredValue),
                       iMax);
```

Return value of 1st function as argument for 2nd function

The use of functions in Structured Text is subject to the same rules that apply to the other languages. All input parameters have to be assigned to in the right order of the arguments.

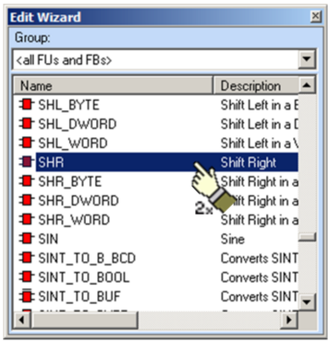
After the function name, the arguments to be transmitted follow in brackets, separated by commas. It is recommended to follow above pattern – using one line per argument – if lining up in one line is possible.

In the most simple application, the return value of the function is assigned to a target variable via the assignment operator.

The second example shows the nesting of functions. The result of the *Word_To_Int* in this example is used as a second argument for the *Limit* function.

Editing in PC WORX

Inserting via the edit wizard



Edit Wizard

Group:

<all FUs and FBs>

Name	Description
SHL_BYTE	Shift Left in a E
SHL_DWORD	Shift Left in a [
SHL_WORD	Shift Left in a \
SHR	Shift Right
SHR_BYTE	Shift Right in a
SHR_DWORD	ift Right in a
SHR_WORD	Shift Right in a
SIN	Sine
SINT_TO_B_BCD	Converts SINT
SINT_TO_BOOL	Converts SINT
SINT_TO_BUF	Converts SINT

Comments as dummies

Parameter name and data type

```
(* Result as ANY_BIT *) := SHR((* IN as ANY_BIT *),(* N as ANY_INT *));
```

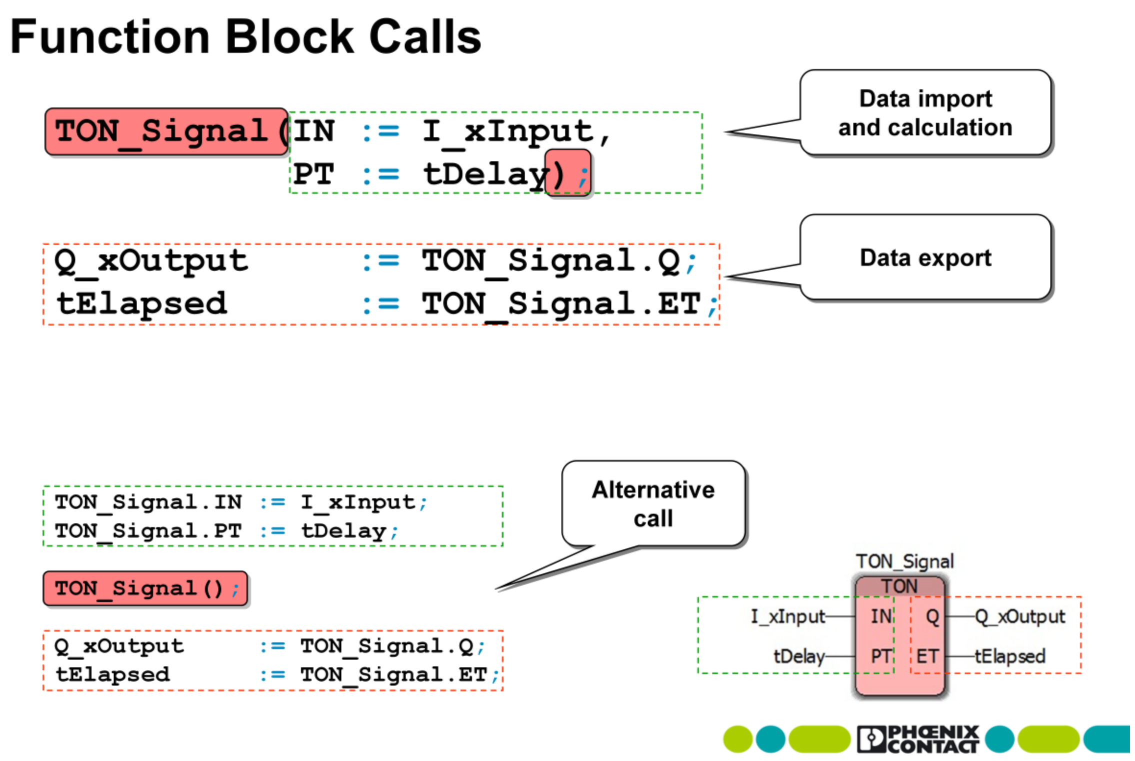
After editing by the user

```
wshifted := SHR(wMeasuredValue, INT#8);
```

Editing a function in PC WORX can be carried out by entering the function syntax by means of the keyboard. If the parameters, the function spelling or the order of arguments is unknown, a syntax template can be inserted in the programming via the edit wizard.

The target variables and the arguments to be transmitted are inserted as comments by the edit wizard and must be replaced by variables.

Especially the nesting of functions cannot just be carried out using the edit wizard. It has proven to be practical to have the syntax displayed in empty lines and then copy it to the actual programming.



Function Block Calls

```
TON_Signal(IN := I_xInput,
           PT := tDelay);

Q_xOutput       := TON_Signal.Q;
tElapsed        := TON_Signal.ET;
```

Data import and calculation

Data export

```
TON_Signal.IN  := I_xInput;
TON_Signal.PT  := tDelay;

TON_Signal();

Q_xOutput      := TON_Signal.Q;
tElapsed       := TON_Signal.ET;
```

Alternative call

The function block call in Structured Text is carried out, as in all other languages, in three phases:

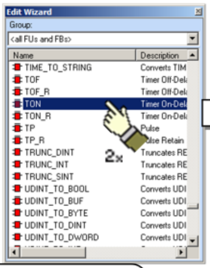
1. Provision of values for the input parameters (data import).
2. Execution of block functionality, if necessary, by using saved data (calculation).
3. Saving the calculated values via the output parameters on variables created (data export).

With the first call version, the actual parameters within a bracket are connected to the formal parameters of the function block instance. Therefore, the name of the instance does not have to be located in front of them as it is the case with the alternative call.

In any case, the brackets during call of the block have to be entered, even if the data import is done separately (see alternative call).

Editing in PC WORX

Inserting via the edit wizard

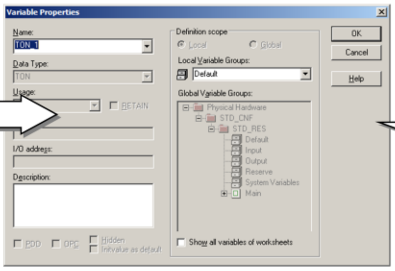


Edit Wizard

Group: <all FUs and FBs>

Name	Description
TIME_TO_STRING	Converts TIM
TOF	Timer Off-Del
TOF_R	Timer Off-Del
TON	Timer On-Del
TON_R	Timer On-Del
TP	Pulse
TP_R	Pulse Retain
TRUNC_DINT	Truncates RE
TRUNC_INT	Truncates RE
TRUNC_SINT	Truncates RE
UDINT_TO_BOOL	Converts UDI
UDINT_TO_BUF	Converts UDI
UDINT_TO_BYTE	Converts UDI
UDINT_TO_DINT	Converts UDI
UDINT_TO_DWORD	Converts UDI

2x



Variable Properties

Name: TON_1
Data Type: TON
Usage:
□ RETAIN
I/O address:
Description:
□ PDD □ OPC □ Hidden □ Initvalue as default

Definition scope
● Local ○ Global
Local Variable Groups: Default
Global Variable Groups:
Physical Hardware
STD_CNF
STD_RES
Default
Input
Output
Reserve
System Variables
Main
□ Show all variables of worksheets

Declaration dialog

Data type only

Comments as dummies

```
TON_1(IN:=(* BOOL *), PT:=(* TIME *));
(* BOOL *):=TON_1.Q;
(* TIME *):=TON_1.ET;
```

After editing by the user

```
TON_1(IN     := xActive,
      PT     := TIME#250ms);

xDelay      := TON_1.Q;
tElapsed    := TON_1.ET;
```

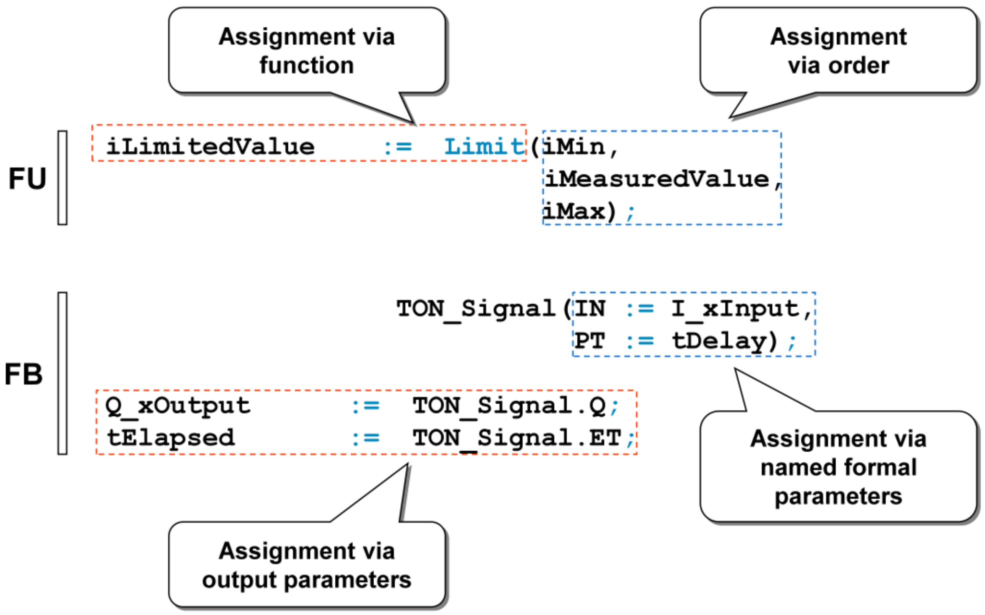
Just as for the functions, editing a function block in PC WORX can be carried out by entering the FB syntax via the keyboard. You have to keep in mind, though, that function blocks, unlike functions, have to be instantiated. This can either be done manually via the variable table or via the variable declaration dialog. The FB type is determined as data type.

If the edit wizard is used to add an FB to the programming, then the declaration of the instance is implemented as in the graphic languages via the declaration dialog that opens. As for functions, the syntax parts of the call to be replaced with variables are inserted as comments.

The call of nested FBs cannot be implemented via the edit wizard. Using the edit wizard, though, the syntax of a single call can be inserted and the FB instance declared.

Notes:

FU Call and FB Call in Comparison



FU

```
iLimitedValue    :=  Limit(iMin,
                           iMeasuredValue,
                           iMax);
```

FB

```
                      TON_Signal(IN  := I_xInput,
                                 PT  := tDelay);
Q_xOutput         :=  TON_Signal.Q;
tElapsed          :=  TON_Signal.ET;
```

Assignment via function

Assignment via order

Assignment via named formal parameters

Assignment via output parameters

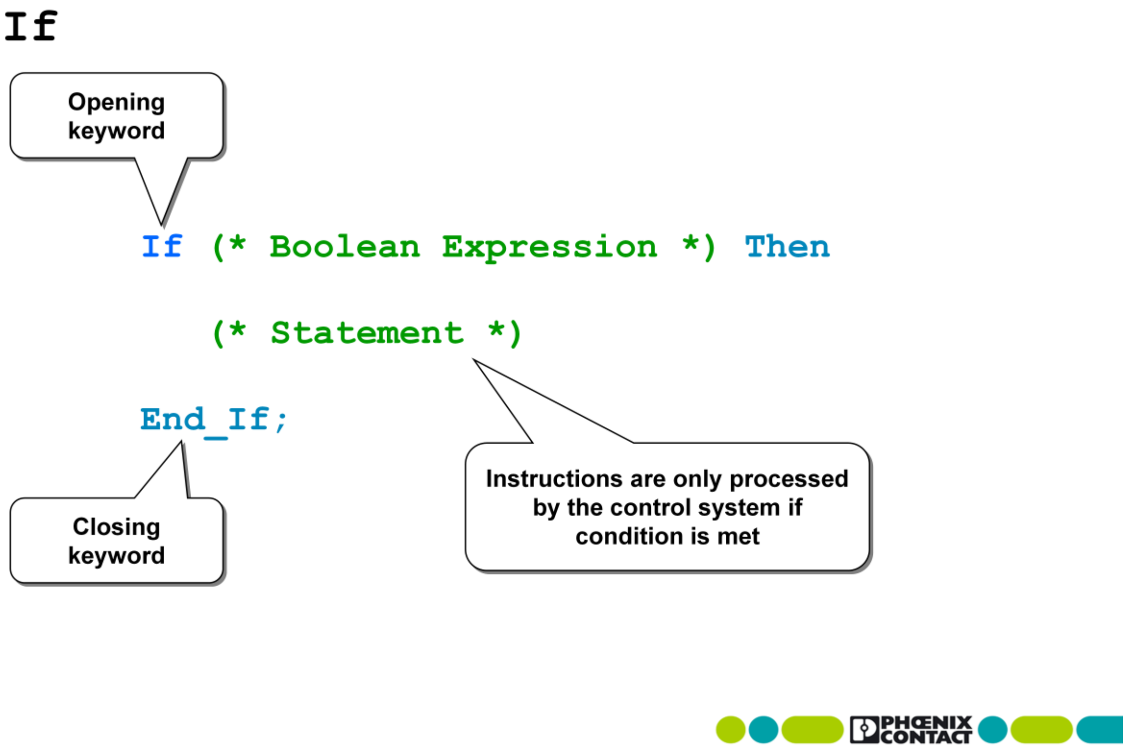
Above diagram is a comparison of the call structure for function blocks and functions. The function is very restrictive as regards to its connection possibilities, the arguments are assigned via the order. The various possibilities of the function blocks are based on the access to named input and output parameters.

Notes:

High-Level Language Elements 1

The Requests

Notes:



If

```
If (* Boolean Expression *) Then

   (* Statement *)

End_If;
```

Opening keyword

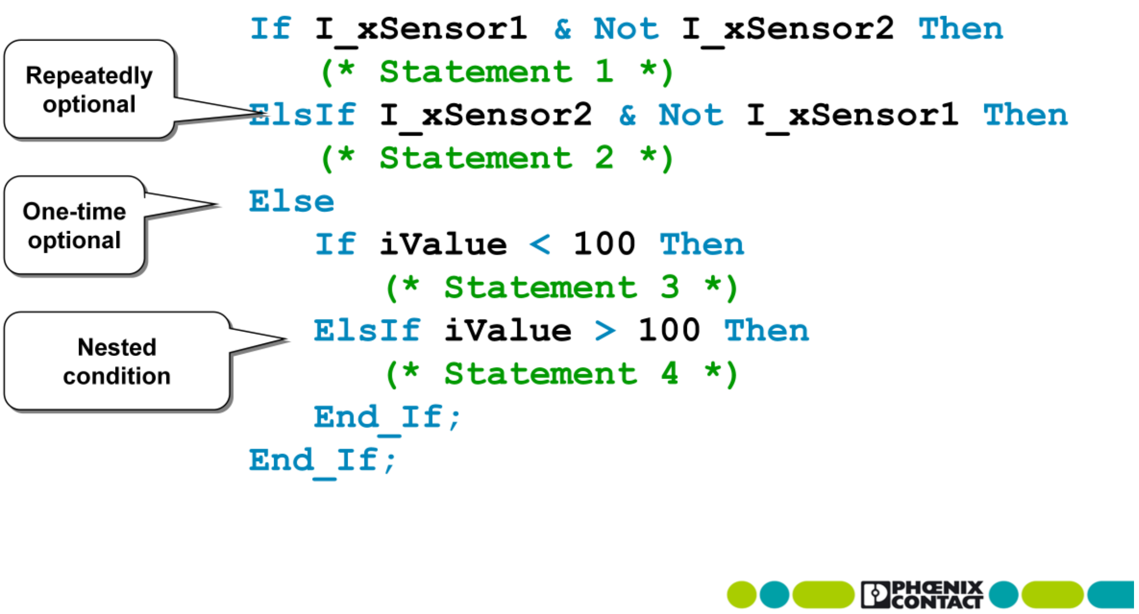
Closing keyword

Instructions are only processed by the control system if condition is met

The structure built on the keyword *If* makes it possible for the programmer to have code executed depending on a boolean condition.

This structure is an important supplement to the remaining elements known from the other languages, e.g. instructions, functions and function blocks. However, it should not be used to replace them.

If Extensions



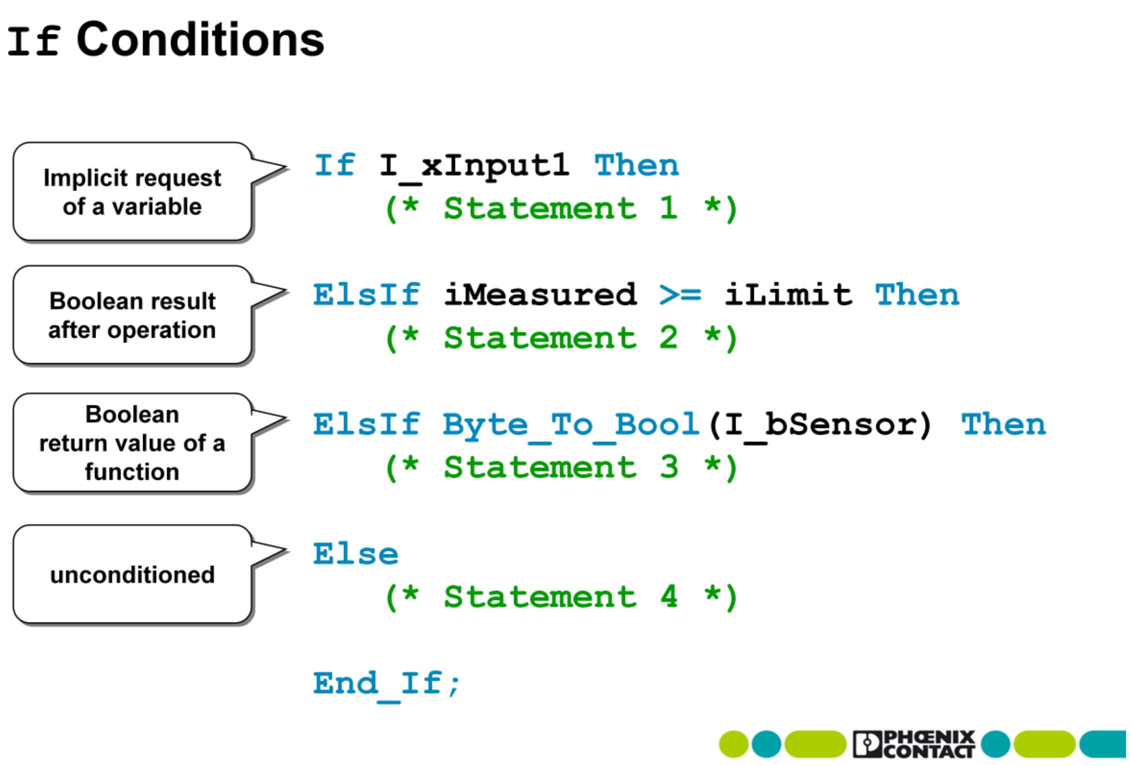
```
If I_xSensor1 & Not I_xSensor2 Then
   (* Statement 1 *)
ElsIf I_xSensor2 & Not I_xSensor1 Then
   (* Statement 2 *)
Else
   If iValue < 100 Then
      (* Statement 3 *)
   ElsIf iValue > 100 Then
      (* Statement 4 *)
   End_If;
End_If;
```

The basic structure can be extended using two keywords:

ElsIf allows an alternative, conditioned request if the condition of the opening *If* is not met. It can be used as often as you like. However, within an *If* structure, only the instruction depending on the first condition met is executed.

Else, unlike *ElsIf*, can only be used once. If within an *If* structure none of the previous requests were completed via *If* and *Elsif*, the instructions depending on *Else* are executed.

Above diagram also shows an example for a nested *If* structure. The second *If* block is only executed if *Else* is activated and then again follows the rules of an independent *If* structure.



`If` Conditions

Implicit request of a variable

Boolean result after operation

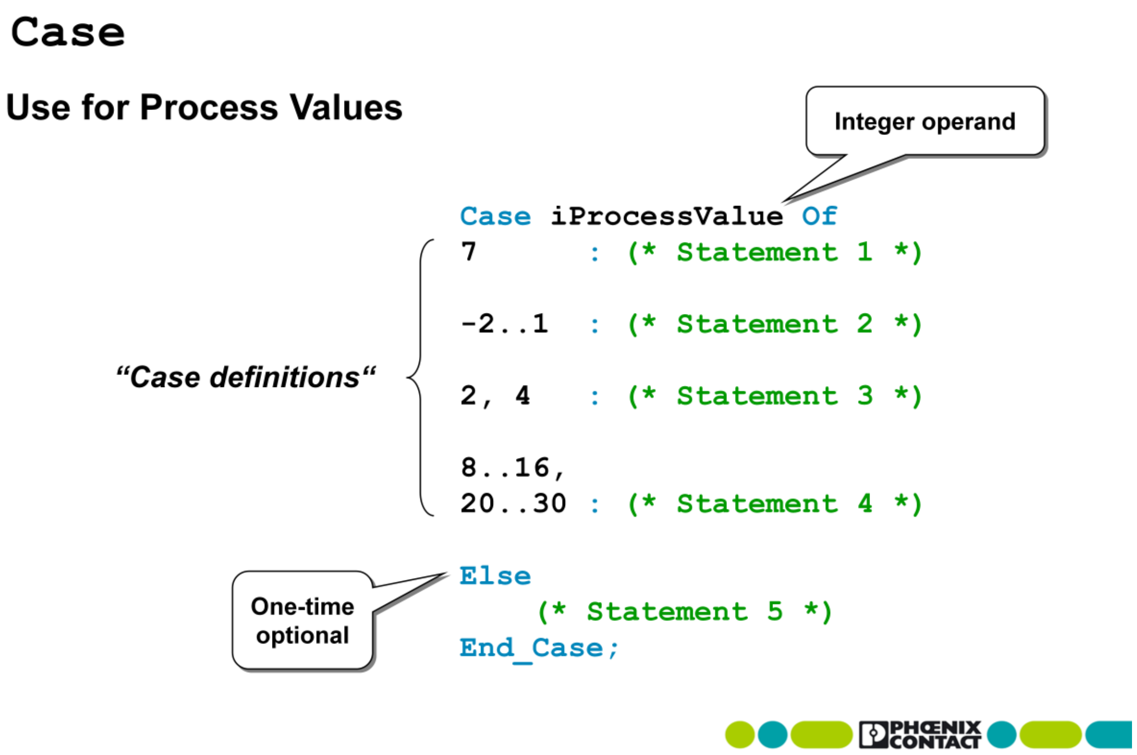
Boolean return value of a function

unconditioned

```
If I_xInput1 Then
   (* Statement 1 *)

ElsIf iMeasured >= iLimit Then
   (* Statement 2 *)

ElsIf Byte_To_Bool(I_bSensor) Then
   (* Statement 3 *)

Else
   (* Statement 4 *)

End_If;
```

Various programmings can be used as conditions for the elements and extensions referred to earlier:

- Boolean variables (with *NOT* for inverted requests)
- Boolean results of operations (nested also possible)
- Boolean return values of functions
- *Else* without condition (use within an *If* structure only possible once)



Case

Use for Process Values

Integer operand

“Case definitions“

```
Case iProcessValue Of
7        : (* Statement 1 *)

-2..1    : (* Statement 2 *)

2, 4     : (* Statement 3 *)

8..16,
20..30 : (* Statement 4 *)

Else
     (* Statement 5 *)
End_Case;
```

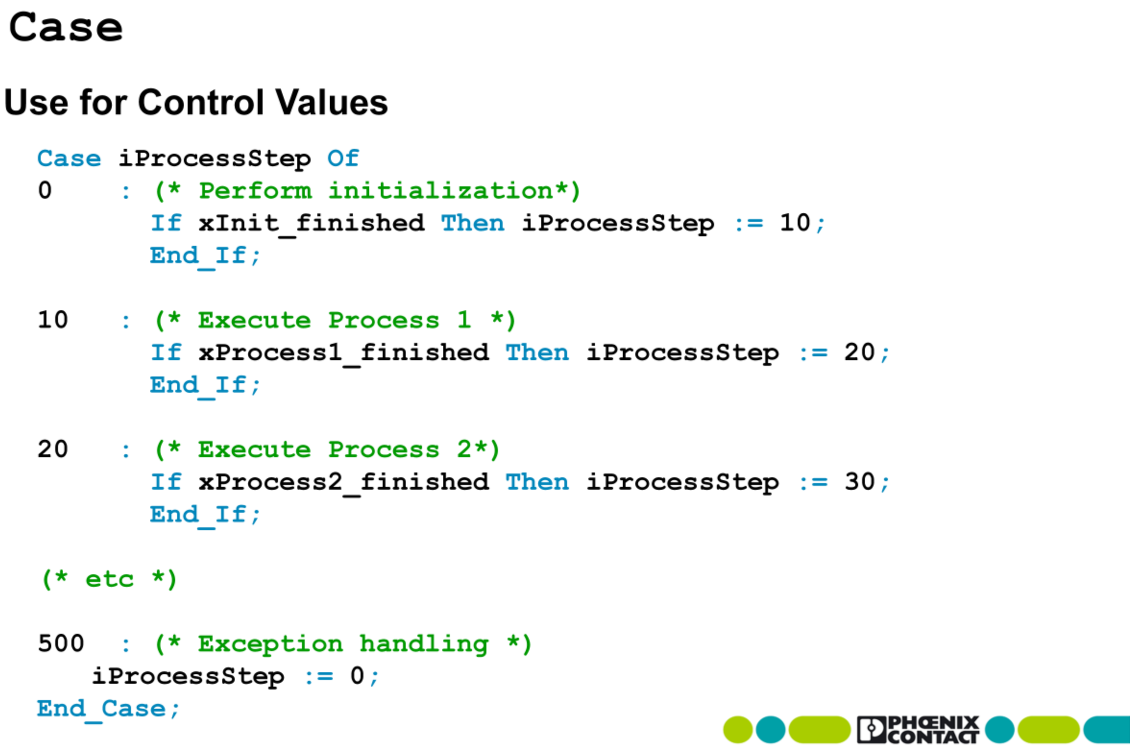
One-time optional

The request structure built on the keyword *Case* requests the value of a variable of the *Integer* data type. The example shows that individual values, values separated by commas, value ranges and the combination of the last two can be used as case definitions.

As with the *If* structure, the instruction block whose condition (value) corresponds to the value of the variable requested is executed. Unlike with the *If* structure, in which conditions do not have to be mutually exclusive (if *If* … *ElsIf* structures are used), an overlap of case definitions in *Case* structures is read out as an error.

The instruction block depending on the optional *Else* is executed if the variable values do not match the requested values.

All values have to be selected from the value range of the INT data type (-32.768 to 32.767).



Case

Use for Control Values

```
Case iProcessStep Of
0    : (* Perform initialization*)
       If xInit_finished Then iProcessStep := 10;
       End_If;

10   : (* Execute Process 1 *)
       If xProcess1_finished Then iProcessStep := 20;
       End_If;

20   : (* Execute Process 2*)
       If xProcess2_finished Then iProcessStep := 30;
       End_If;

(* etc *)

500  : (* Exception handling *)
   iProcessStep := 0;
End_Case;
```

If the *Case* structure is used to build a textual step chain programming, the requested variable is used for controlling the sequence. Its status shows the current process step.

In above example, the variable *iProcessstep* is initialized upon PLC start with *0* and the instruction block this value is assigned to is executed until the process reports the end of the initialization sequence via the boolean variable *xInit_completed*. As a result, the variable *iProcessstep* is set to *10* and in the next (!) PLC cycle, the action block the new value is assigned to is executed.

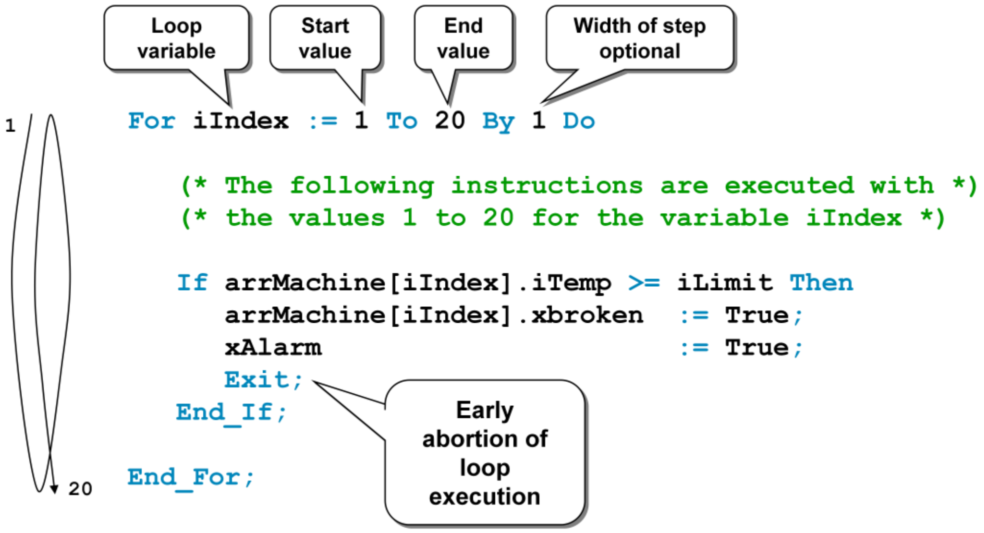
One advantage of this kind of programming is the possibility of always being able to set the step variable to a value via the end condition, which can be used to activate an exceptional handling.

Notes:

High-Level Language Elements 2

The Loops

For



```
For iIndex := 1 To 20 By 1 Do

   (* The following instructions are executed with *)
   (* the values 1 to 20 for the variable iIndex *)

   If arrMachine[iIndex].iTemp >= iLimit Then
      arrMachine[iIndex].xbroken  := True;
      xAlarm                      := True;
      Exit;
   End_If;

End_For;
```

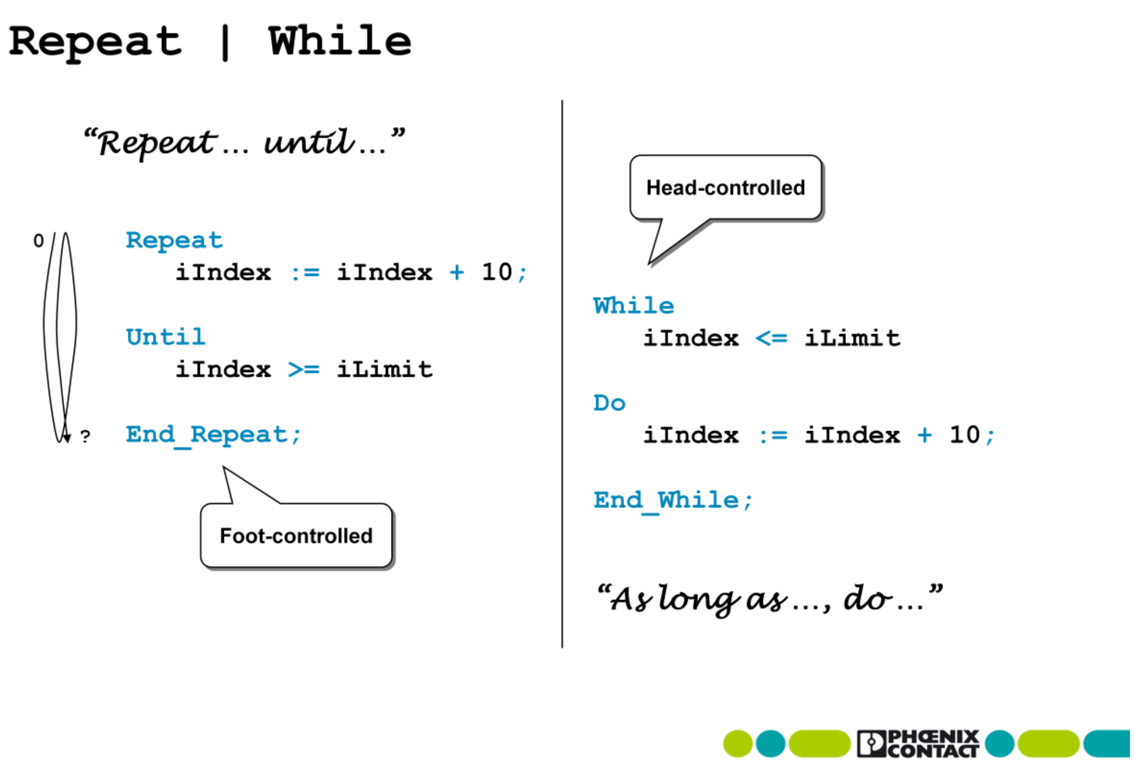
The *For* loop implements the repeated execution of instructions in a PLC cycle. All values from the start value through to the end value are acknowledged for the loop variable (here `iIndex`) under incrementation with the indicated step width.

All values (loop variable, start value, end value, step width) can be freely selected from the INT data type group. Only positive values can be selected for the step width.

If the step width is not indicated, the standard value INT#1 is set. Then, the other values have to be of the INT data type.

Once the loop has been run, the loop variable receives the end value plus step width (in above example: INT#21).

Early abortion of the loop run can be achieved via the *EXIT* instruction.



Repeat | While

“Repeat … until …”

```
Repeat
    iIndex := iIndex + 10;

Until
    iIndex >= iLimit

End_Repeat;
```

0
?
Foot-controlled
Head-controlled

```
While
    iIndex <= iLimit

Do
    iIndex := iIndex + 10;

End_While;
```

“As long as …, do …”

The *Repeat* and *While* loops, unlike the *For* loop, are no predetermined loops. In dependence on a boolean value it is decided if the depending programming is executed again. Here, the request time makes the difference between the *foot-controlled Repeat* loop and the *head-controlled While* loop. In the *While* loop, first of all the request is carried out while in the *Repeat* loop the dependent instructions are executed once before the loop is aborted due to non-compliance to the condition.

In the process and programming it has to be ensured that the loops are not executed undefined as the PLC would then not be able to continue execution. In addition to this, a watchdog error would be triggered due to the runtime monitoring.

Notes:

Project Management

Appendix A1

Notes:

Contents

This chapter offers an overview of the project files in PC WORX, the memory concept of the controllers as well as the different code forms and the project handling using different control systems.

⚠ This documentation and its description relate to the control using ProConOS as an operating system. The controls of the 100 class have dialogs and storage concepts that deviate from this presentation. Please read about the deviations in the related user manuals.

⚠	Attention!
ℹ	Information
💡	Tip

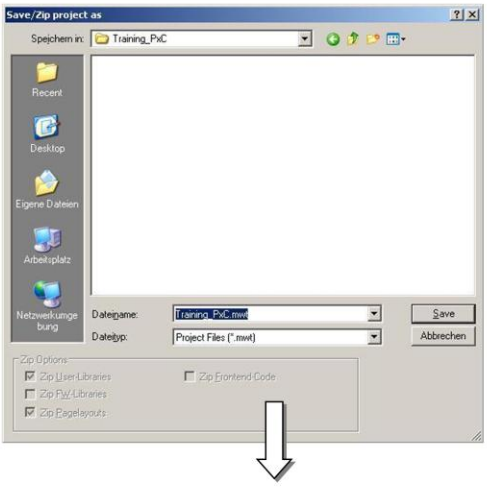
Notes:

Project Files

Notes:

Saving/Archiving

Save as …



Save/Zip project as

Speichern in: Training_PxC

Dateiname: Training_PxC.mwt

Dateityp: Project Files (*.mwt)

Zip Options

☑ Zip User-Libraries

□ Zip FW-Libraries

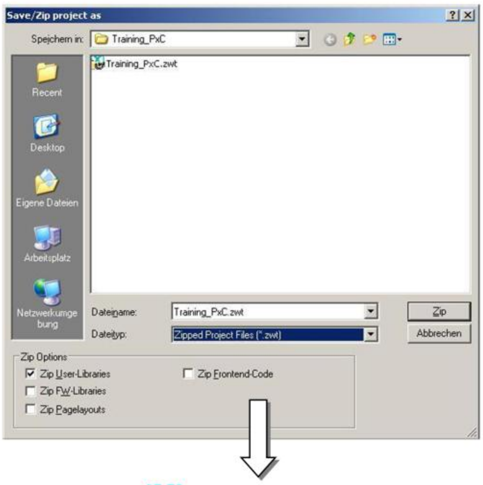
☑ Zip Pagelayouts

□ Zip Frontend-Code

Project name

Project name.mwt

Archiving



Save/Zip project as

Speichern in: Training_PxC

Training_PxC.zwt

Dateiname: Training_PxC.zwt

Dateityp: Zipped Project Files (*.zwt)

Zip Options

☑ Zip User-Libraries

□ Zip FW-Libraries

□ Zip Pagelayouts

□ Zip Frontend-Code

Project name.zwt

For processing in PC WORX, the project files are unpacked and saved to selected storage location on the hard disk of the PC. In addition to a header file of the extension *.mwt, these files include a directory of the same name.

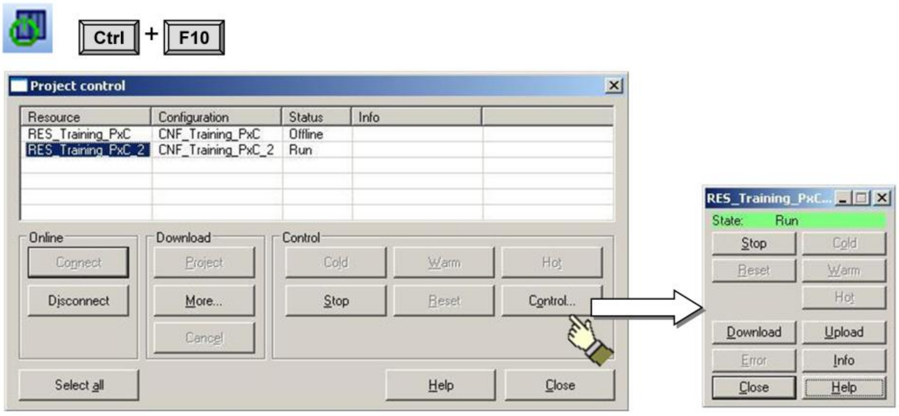
By selecting the Option *Save as...*, the project currently being processed in PC WORX is saved with the selected name and these files are then used for further processing.

By contrast, when archiving, the header file and directory are combined in a zwt-file (with new name if necessary). Work in PC WORX, however, is continued using the original project (and name).

Notes:

CPU Control Dialog

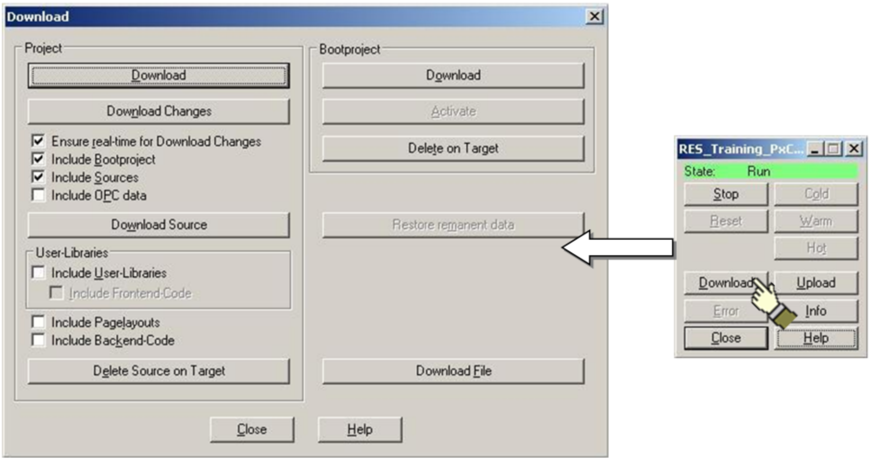
CPU Selection



Resource	Configuration	Status	Info
RES_Training_PxC	CNF_Training_PxC	Offline	
RES_Training_PxC_2	CNF_Training_PxC_2	Run	

After successful compilation of the control project (bus configurations and control program), the code can be downloaded to the control system via the project control dialog. In the case of a project, in which more than one control system has been configured, the control system is to be selected via above dialog. Thus, the download is not carried out to all systems at the same time. The individual control dialog can only be called after the connection to the control system has been established via *Connect*.

Temporary Project - Bootproject



☑ Ensure real-time for Download Changes
☑ Include Bootproject
☑ Include Sources
☐ Include OPC data
☐ Include User-Libraries
☐ Include Frontend-Code
☐ Include Pagelayouts
☐ Include Backend-Code

State: Run

Project

Transmit
The compiled machine code (bus configuration and application program) is written to a volatile main memory.

Transmit changes
Changes to the programming (not to the bus configuration or the hardware structure incl. tasks) are transmitted to the main memory during operation (if necessary, in real time if the corresponding option is activated)

Include project
After downloading to the main memory, the bootproject is written to the parameterization memory in a separate transmission.

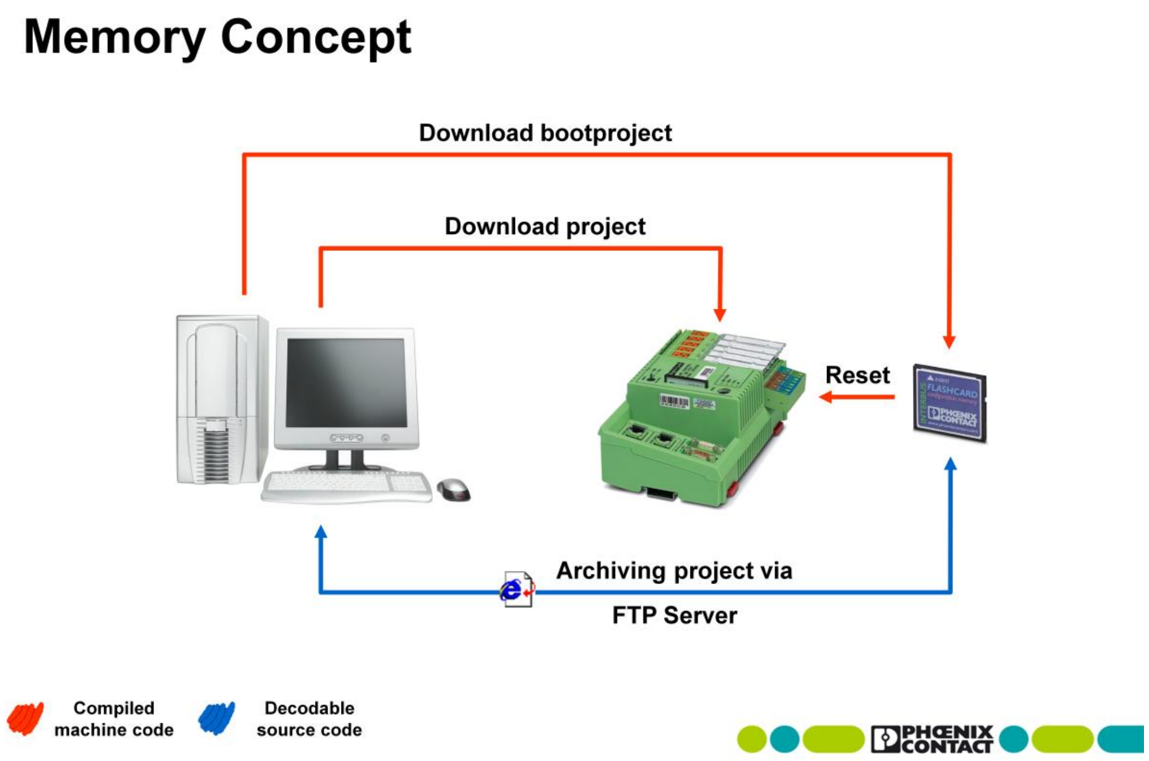
Include project sources
After downloading to the main memory, the compressed source files are written to the parameterization memory.

Include OPC data
With suitable control systems, after downloading to the main memory, the OPC configuration can be written to the parameterization memory.

⚠ Only after the successful download of the source files, can the project be read back from the parameterization memory. Decoding the machine code is not possible.

Notes:

Project Codes



Memory Concept

Download bootproject

Download project

Reset

Archiving project via

FTP Server

🟥 Compiled machine code 🟦 Decodable source code

When handling codes you differentiate between two code types:

Compiled machine code

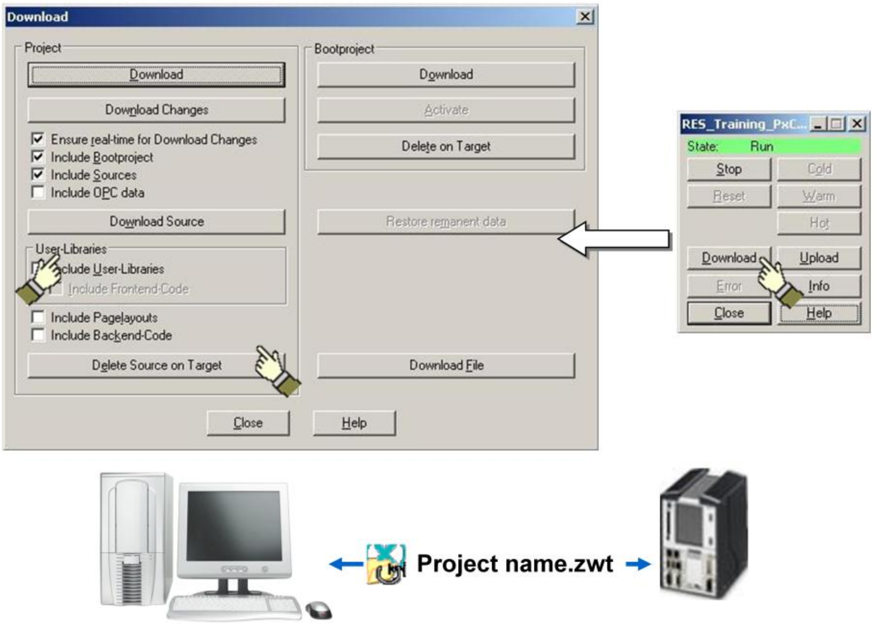
This code can be interpreted by control systems or directly be executed. With PC WORX it cannot be decoded. In the form of a boot simulation, it can be downloaded directly to a control system as an executable code using *Transmit project*. Furthermore, it can be saved as a bootproject in the parameterization memory of the control system. If the control system is reset or the machine code is manually activated using PC WORX, the machine code is loaded from the parameterization memory and executed on the control system.

Decodable source code

This code corresponds to the archived project (<Projectname>.zwt) saved on the hard disk of the programming device. This code cannot be interpreted by control systems, but only serves as a project backup on the control system. With control systems of the youngest generation, the up- and download of these files can be done via an FTP server.

If older control systems are used, the procedure described on the next page for saving the source code using a standardized archive name has to be followed.

Standard Archive



Project

☑ Ensure real-time for Download Changes
☑ Include Bootproject
☑ Include Sources
☐ Include OPC data

User-Libraries

☐ Include User-Libraries
☐ Include Frontend-Code

☐ Include Pagelayouts
☐ Include Backend-Code

State: Run

The standard archive can be used for saving the source code to all control systems (even to those without an integrated FTP server).

Via the *Download source* command in PC WORX, the current project is automatically archived and written to the parameterization memory with the name *ZipFile.zwt*.

If the archive file *ZipFile.zwt* already exists, the *Delete source on target* button is enabled for deleting the standard archive and the *Upload* button is enabled for uploading and opening the project on the programming device.

⚠ Under earlier PX WORX version/earlier controls, the archive used to be downloaded as ZipFile.zwt.

Notes:

Libraries

Appendix A2

Notes:

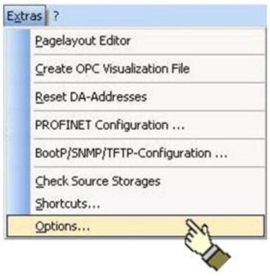
Contents

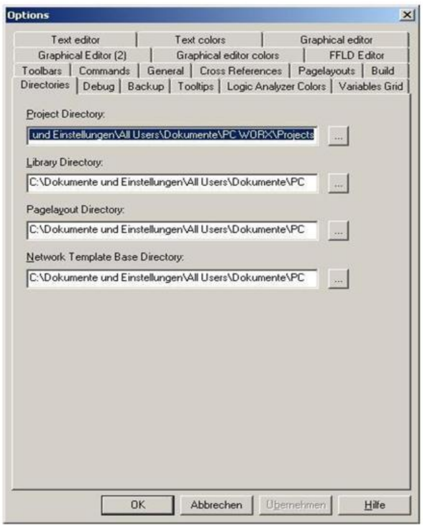
This chapter informs you about the importance of libraries in PC WORX, how they are created, which elements are part of a library and how libraries can later be used again in new projects.

⚠	Attention!
ℹ	Information
💡	Tip

Notes:

Library Directory


Extras
?
Pagelayout Editor
Create OPC Visualization File
Reset DA-Addresses
PROFINET Configuration ...
BootP/SNMP/TFTP-Configuration ...
Check Source Storages
Shortcuts...
Options...



Project Directory: und Einstellungen\All Users\Dokumente\PC WORX\Projects

Library Directory: C:\Dokumente und Einstellungen\All Users\Dokumente\PC

Pagelayout Directory: C:\Dokumente und Einstellungen\All Users\Dokumente\PC

Network Template Base Directory: C:\Dokumente und Einstellungen\All Users\Dokumente\PC

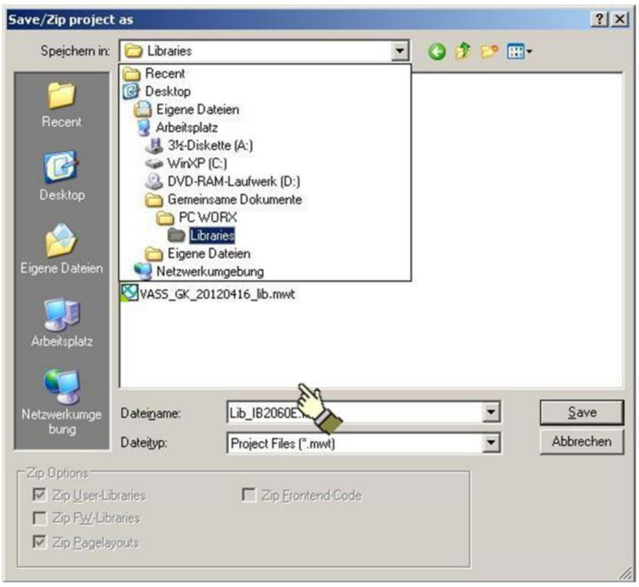
In PC WORX the term library refers to a project that contains program components which can be used in future projects. Every PC WORX project can be used as a library, i.e., integrated in new projects.

Libraries are managed automatically by PC WORX and saved to a directory on the hard disk. Via the options box, this directory can be adapted.

Notes:

Creating Libraries

Saving in the Library Folder



Save/Zip project as

Speichern in: Libraries

Dateiname: Lib_IB2060E

Dateityp: Project Files (*.mwt)

Zip Options

☑ Zip User-Libraries

□ Zip FW-Libraries

☑ Zip Pagelayouts

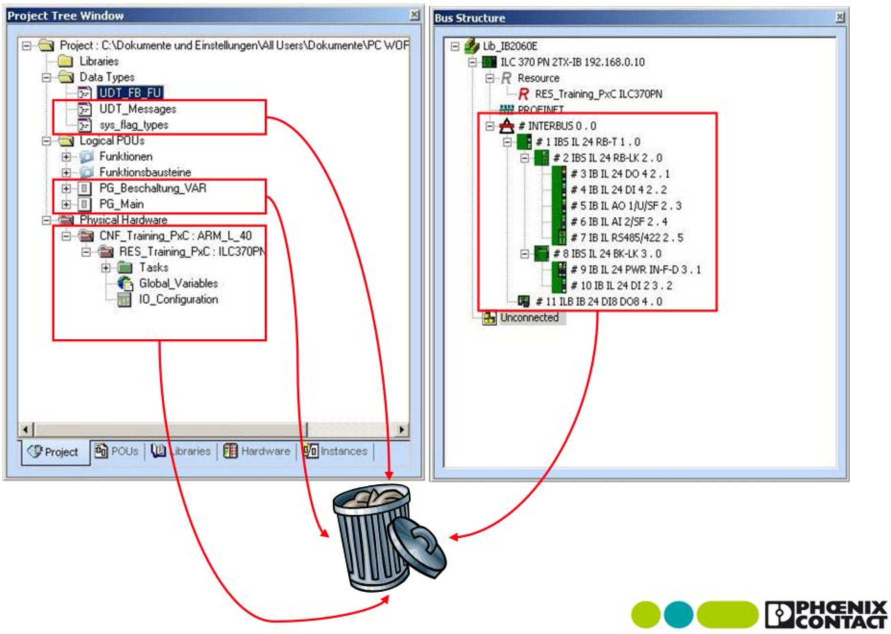
□ Zip Frontend-Code

If functions and function blocks of a current project are to be made available for use in other projects via a library, the project must first be saved with a descriptive file name in the library folder. As libraries are standard projects, the standard procedure for the saving of projects is used.

In order to be able to access the libraries later on via the insert dialog, the directory selected earlier via the option box should be used as the storage location.

⚠ If changes are implemented on the project before it was saved with the new name, the deleted elements are irreversibly lost.

Removal and Compilation



In principle, a library does not have to be explicitly created. Every PC WORX project can be used as a library. In many cases, however, it is useful to remove non-library-specific components from a project. These include the bus configuration as well as the controller and hardware structure of the project tree.

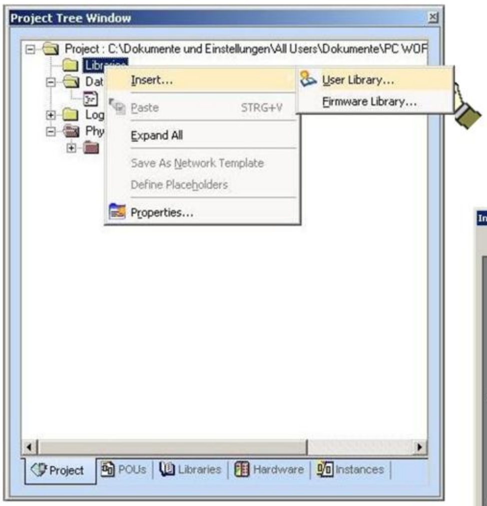
As the main task of libraries is to provide functions, function blocks, and corresponding user-defined data types, it is also useful to remove program POUs.

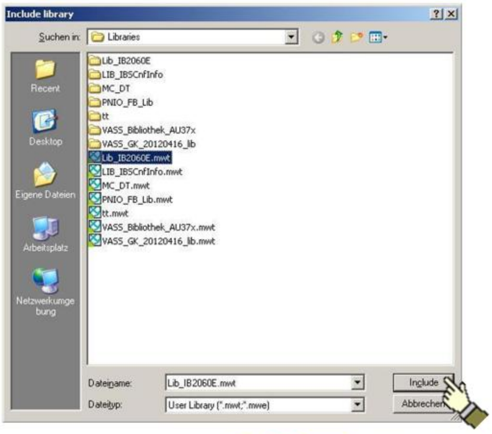
After removal you should be able to compile the project without receiving error messages or warnings. The project using the library can then be compiled.

Notes:

Using Libraries

Including User Libraries

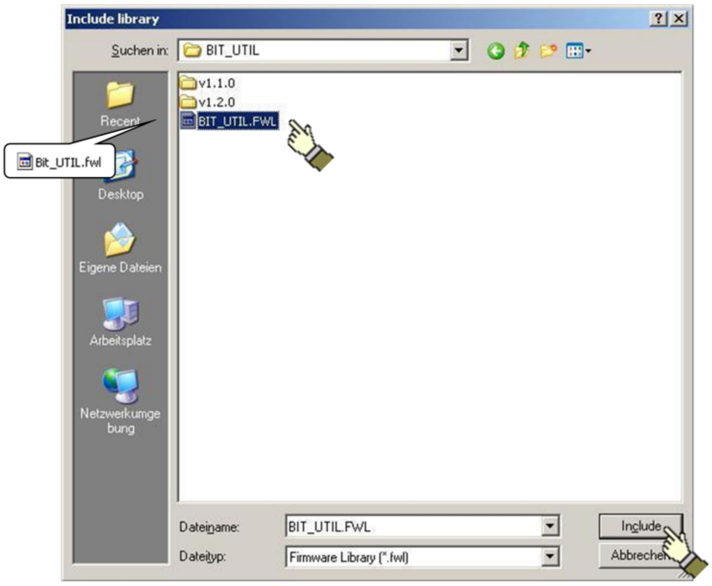




Via the context menu of the project tree, a library can be included in the current project that is to be worked on. You can include user-defined libraries and firmware libraries.

When inserting a user library, the system accesses the library directory where the extracted projects are located (*.mwt).

Including Firmware Libraries



Include library

Suchen in: BIT_UTIL

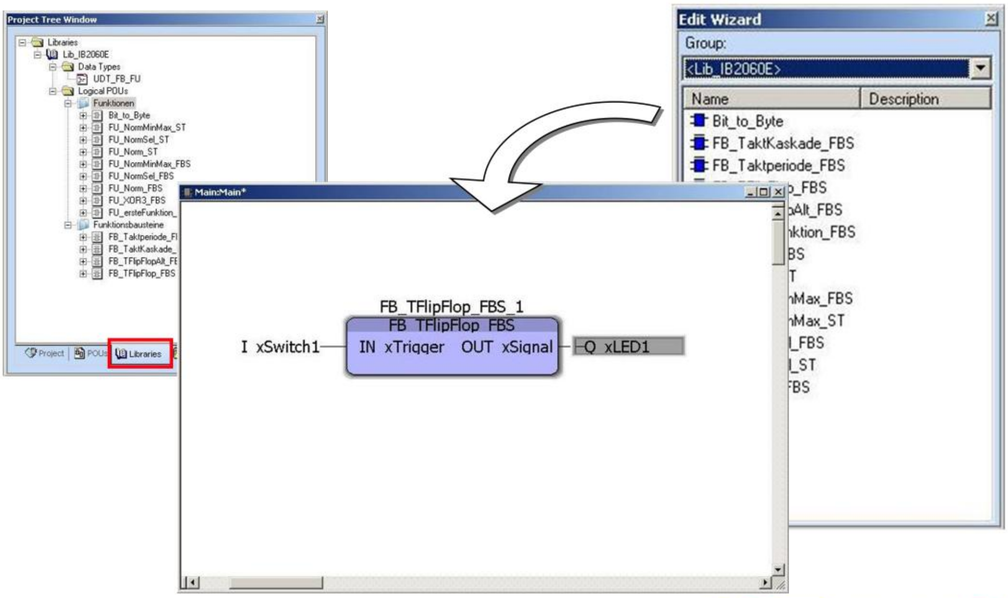
v1.1.0
v1.2.0
BIT_UTIL.FWL

Dateiname: BIT_UTIL.FWL

Dateityp: Firmware Library (*.fwl)

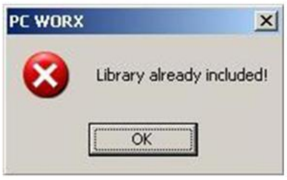
When including a firmware library, PC WORX accesses the fixed directory where additional directories for firmware libraries are located.

The Library in a User Project

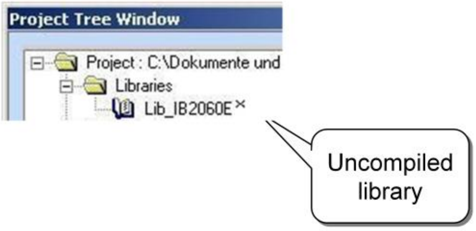


Included libraries can be viewed via the *Libraries* tab in the project tree. Each library is available as a group in the edit wizard. As a standard, blocks from a library that are used in a project will appear in blue.

Using Projects With Libraries


PC WORX

Library already included!


Project Tree Window

Project : C:\Dokumente und
Libraries
Lib_IB2060E *

Uncompiled library

A library can only be included in a project following successful compilation of the library project. If necessary, the corresponding project in the library directory must be opened and compiled.

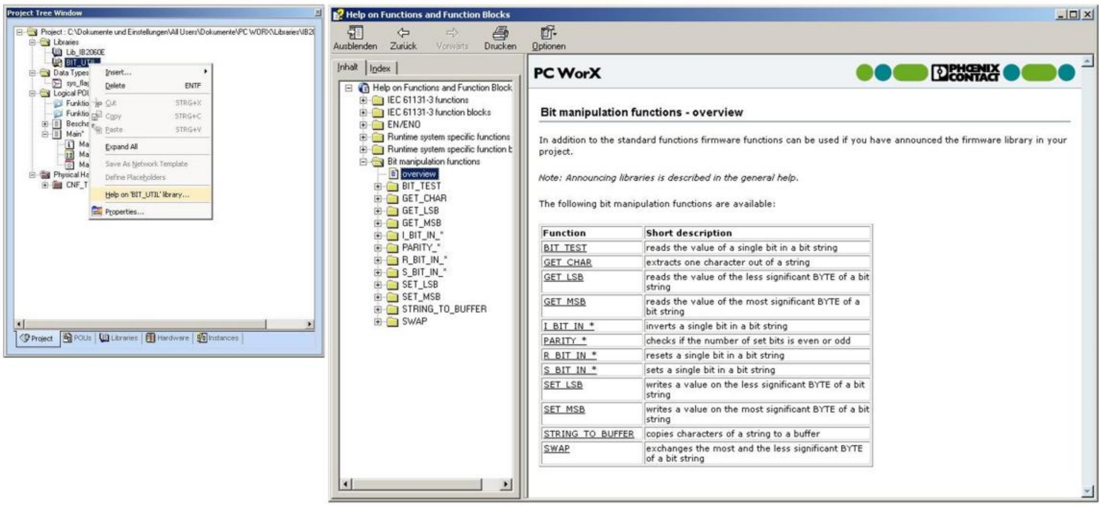
The same procedure must be selected when a project archive is opened in which libraries have already been integrated. An asterisk after an element in the project tree generally indicates that the element has not yet been compiled. This also holds true for libraries.

If on opening a project archive, one of its libraries already exists as a project in the library directory, you must decide how to proceed with the two elements via a selection dialog box.

You can use the properties dialog of the library to check which library is included in the project.

Notes:

Help on Firmware Libraries



Bit manipulation functions - overview

In addition to the standard functions firmware functions can be used if you have announced the firmware library in your project.

Note: Announcing libraries is described in the general help.

The following bit manipulation functions are available:

Function	Short description
BIT_TEST	reads the value of a single bit in a bit string
GET_CHAR	extracts one character out of a string
GET_LSB	reads the value of the less significant BYTE of a bit string
GET_MSB	reads the value of the most significant BYTE of a bit string
I_BIT_IN_*	inverts a single bit in a bit string
PARITY_*	checks if the number of set bits is even or odd
R_BIT_IN_*	resets a single bit in a bit string
S_BIT_IN_*	sets a single bit in a bit string
SET_LSB	writes a value on the less significant BYTE of a bit string
SET_MSB	writes a value on the most significant BYTE of a bit string
STRING_TO_BUFFER	copies characters of a string to a buffer
SWAP	exchanges the most and the less significant BYTE of a bit string

Unlike user-defined libraries, firmware libraries have a central help function, which can be called via the context menu.

Notes:

Testing & Debugging

Appendix A3

Notes:

Contents

This chapter offers you an overview of the possibilities to observe and control a PX WORX project and write data.

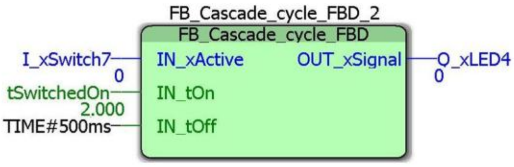
⚠ Attention!

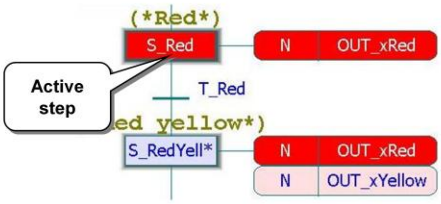
ℹ Information

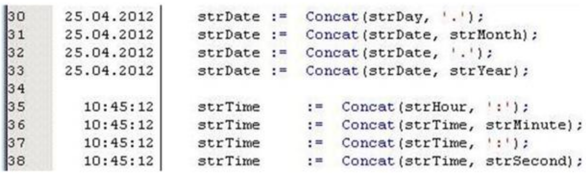
💡 Tip

Notes:

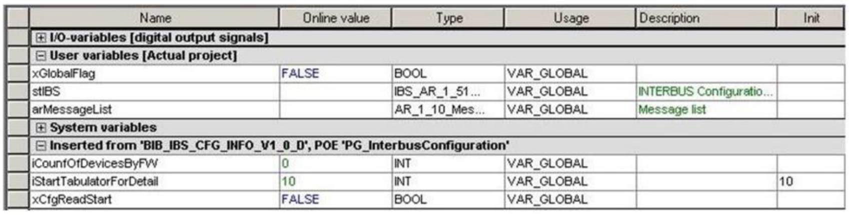
Status in Worksheets







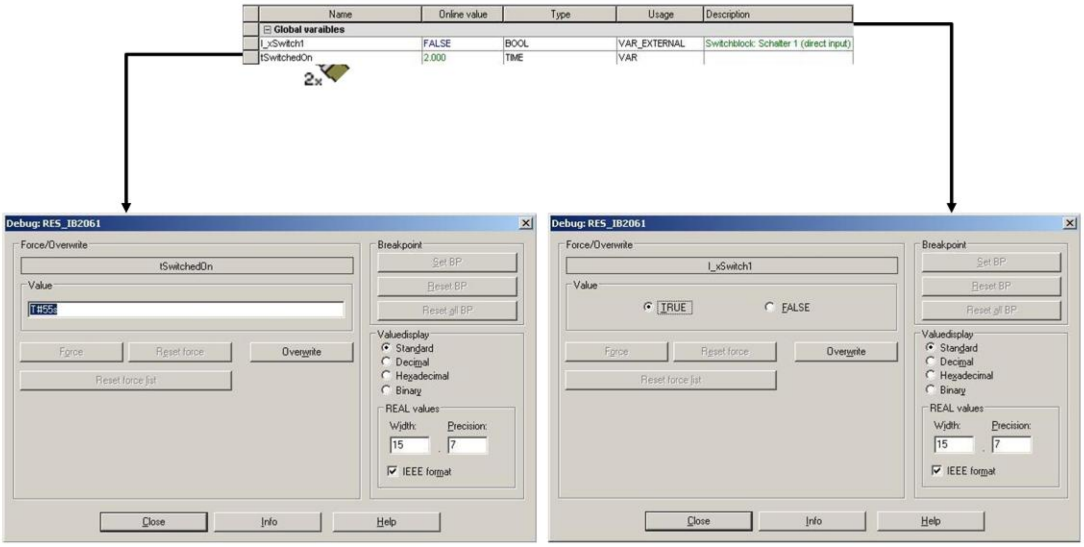
```
25.04.2012    strDate :=  Concat(strDay, '.');
25.04.2012    strDate :=  Concat(strDate, strMonth);
25.04.2012    strDate :=  Concat(strDate, '.');
25.04.2012    strDate :=  Concat(strDate, strYear);

  10:45:12    strTime     :=  Concat(strHour, ':');
  10:45:12    strTime     :=  Concat(strTime, strMinute);
  10:45:12    strTime     :=  Concat(strTime, ':');
  10:45:12    strTime     :=  Concat(strTime, strSecond);
```



Name	Online value	Type	Usage	Description	Init
⊞ **I/O-variables [digital output signals]**					
⊟ **User variables [Actual project]**					
xGlobalFlag	FALSE	BOOL	VAR_GLOBAL		
stIBS		IBS_AR_1_51...	VAR_GLOBAL	INTERBUS Configuratio...	
arMessageList		AR_1_10_Mes...	VAR_GLOBAL	Message list	
⊞ **System variables**					
⊟ **Inserted from 'BIB_IBS_CFG_INFO_V1_0_D', POE 'PG_InterbusConfiguration'**					
iCounfOfDevicesByFW	0	INT	VAR_GLOBAL		
iStartTabulatorForDetail	10	INT	VAR_GLOBAL		10
xCfgReadStart	FALSE	BOOL	VAR_GLOBAL		

In the status view you can select the display format of all worksheets via the debug dialog. There is a differentiation between numeric, string- and bit-based data types.

Overwriting/Forcing



	Name	Online value	Type	Usage	Description
	⊟ **Global varaibles**				
	I_xSwitch1	FALSE	BOOL	VAR_EXTERNAL	Switchblock: Schalter 1 (direct input)
	tSwitchedOn	2.000	TIME	VAR	

Debug: RES_IB2061

Force/Overwrite: tSwitchedOn

Value: T#55s

Valuedisplay: ● Standard ○ Decimal ○ Hexadecimal ○ Binary

REAL values: Width: 15 Precision: 7

☑ IEEE format

Debug: RES_IB2061

Force/Overwrite: I_xSwitch1

Value: ● TRUE ○ FALSE

Valuedisplay: ● Standard ○ Decimal ○ Hexadecimal ○ Binary

REAL values: Width: 15 Precision: 7

☑ IEEE format

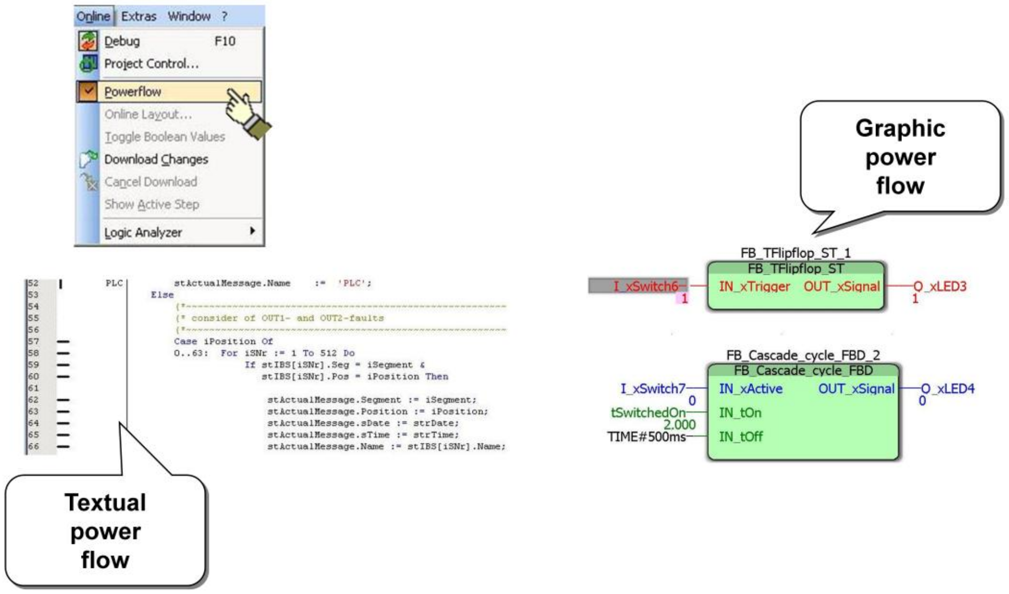
The overwriting of a variable is only effective if the variable does not have a pre-connected logic and is not connected to input process data. Overwriting results in writing a value to a variable just once. With this procedure, according to formal standards, the entry format for literals is to be used.

The forcing of variables is only supported for variables connected to process data objects.

Forcing overwrites the values transferred by the sensor technology and represents a serious intervention in the control program.

The state of forced variables on a control system can be read out via the PLC_DEBUG_FORCE system variable and via the control dialog of the CPU.

Powerflow (via Address Status)



```
 PLC          stActualMessage.Name    :=  'PLC';
          Else
              (*-----------------------------------------------------------
              (* consider of OUT1- and OUT2-faults
              (*-----------------------------------------------------------
              Case iPosition Of
              0..63:  For iSNr := 1 To 512 Do
                            If stIBS[iSNr].Seg = iSegment &
                               stIBS[iSNr].Pos = iPosition Then

                                stActualMessage.Segment := iSegment;
                                stActualMessage.Position := iPosition;
                                stActualMessage.sDate := strDate;
                                stActualMessage.sTime := strTime;
                                stActualMessage.Name := stIBS[iSNr].Name;
```

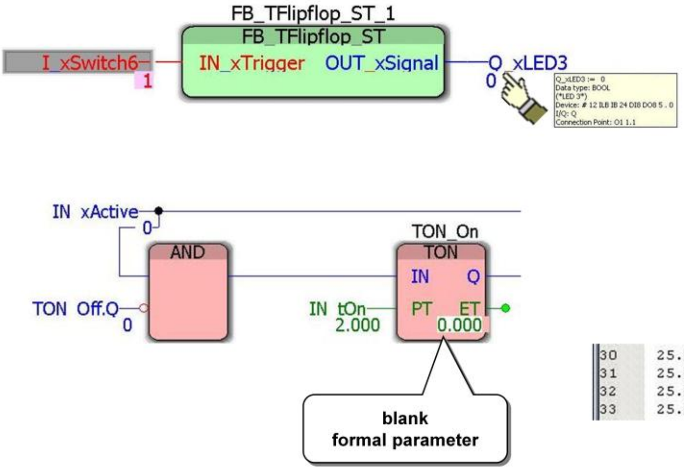
By activating the address status, the powerflow can be implemented in graphical and textual code worksheets that are online. A horizontal or vertical bar shows if the corresponding program part is being processed. Furthermore, the interim values of variables that are written to by various data sources ("scratchpad memory") are displayed.

Enabling the address status can lead to a significantly longer cycle time and thus also to an increased processor load. During operation it is absolutely necessary to observe the warning message as activating the address status can lead to a PLC stop.

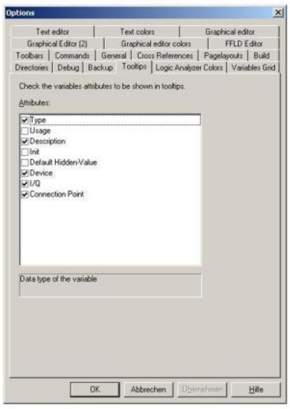
Notes:

Information on Mouse Pointer Tooltips

Tooltip



Q_xLED3 := 0
Data type: BOOL
(*LED 3*)
Device: # 12 ILB IB 24 DI8 DO8 5 . 0
I/Q: Q
Connection Point: O1 1.1

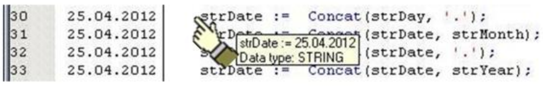


Check the variables attributes to be shown in tooltips.

Attributes:

☑ Type
☐ Usage
☑ Description
☐ Init
☐ Default Hidden-Value
☑ Device
☑ I/Q
☑ Connection Point

Data type of the variable



```
30    25.04.2012    strDate :=  Concat(strDay, '.');
31    25.04.2012    strDate :=  Concat(strDate, strMonth);
32    25.04.2012    strDate :=  Concat(strDate, '.');
33    25.04.2012    strDate :=  Concat(strDate, strYear);
```

strDate := 25.04.2012
Data type: STRING

In addition to the displays offered by the graphic and textual editors in their worksheet status, further information can be displayed when the mouse pointer is moved over elements. Via the option box you can select the details for a variable or an FB instance that are to be displayed.

In order to be able to display all selected details, it might be necessary to update the list of cross-references (standard: function button F12).

Notes:

Tasks for the
PC WORX 6 - IEC 61131 Programming Course

Reference: PC WORX 6.00.25 SP3.73

Overview of tasks:

	POU-Type_Function_Language	FBD	LD	IL	SFC	ST	UDT/BIB
1	`FU_Xor3_*`	▣	▣	▣		▣	
2	`FB_Tflipflop_*`	▤	▤	▤		▤	
3	`FB_Cascade_cycle_*`	▤		▤		▤	
4[a]	`FU_Norm_*`	▣		▣		▣	
4[b]	`FU_NormSel_*`	▣		▣		▣	
4[c]	`FU_NormMinMax_*`	▣		▣		▣	
5	`FB_Traff_light_*`				▤	▤	
6[a]	`UDT declaration`						▦
6[b]	`PG_Messages`					▯	
6[c]	`Library integration`						📚

Symbol key:

▣	Function
▤	Function block
▯	Program
▦	Data type worksheet
📚	Library

Tasks regarding communication path setting: TCP/IP

Com 1 Checking the IP address

To establish a communication connection via TCP/IP, it is required that both communication partners (i.e., PC WORX and the controller used) are assigned a valid IP address (including the subnet mask). If this is not (yet) the case or not known for the controller used, the IP address can **always** be checked **via the serial interface located on the controller** and be set, if required.

First, set the IP address and the subnet mask on your controller according to the specifications given by your speaker.

Documentation: Page 3-11

After transmitting the desired IP address, ensure that it will be “enabled” on the controller.

Com 2 Setting up the communication path

To exchange data between PC WORX and the corresponding controller, it is required to select a communication path for each controller. Even though the serial interface is available for all controllers, it is recommended to set the **TCP/IP** communication path, ensuring in this way a faster data exchange.
Therefore, select **TCP/IP** as the communication path for the controller **within the project**. Enter the IP address manually by selecting “Manual Input” from the “Connection Name” dropdown menu (see *COM 1* task) and test the connection.

Documentation: Pages 3-7, 3-16

Com 3 Using symbolic connection names

Once the connection has been established successfully, create a new symbolic connection name in the “ibethdef.dat” assignment file for communication with your controller and select it afterwards. Then test the connection to your controller again.

Documentation: Page 3-17

Tasks regarding INTERBUS bus configuration

IBS1 Bus configuration (online)

Use the controller to view the connected bus configuration via the *Connected Bus* dialog box. Copy this configuration (bus devices on basis installation platform) by selecting the corresponding devices from the range of available devices.

Documentation: Pages 4-6ff.

IBS2 Extending the bus configuration (offline) (optional)

After connecting the devices that are directly connected to the controller to the remaining remote bus devices, use the device catalog to also extend the bus configuration in the software. The devices to be added are shown in the screenshot below.

Documentation: Page 4-10

Tasks regarding PROFINET bus configuration

PN1 Bus configuration (online)

Read in the connected PROFINET devices using the training computer. Copy the devices shown in the dialog box and make the configuration settings as specified by your speaker.

Documentation: Pages 5-6ff.

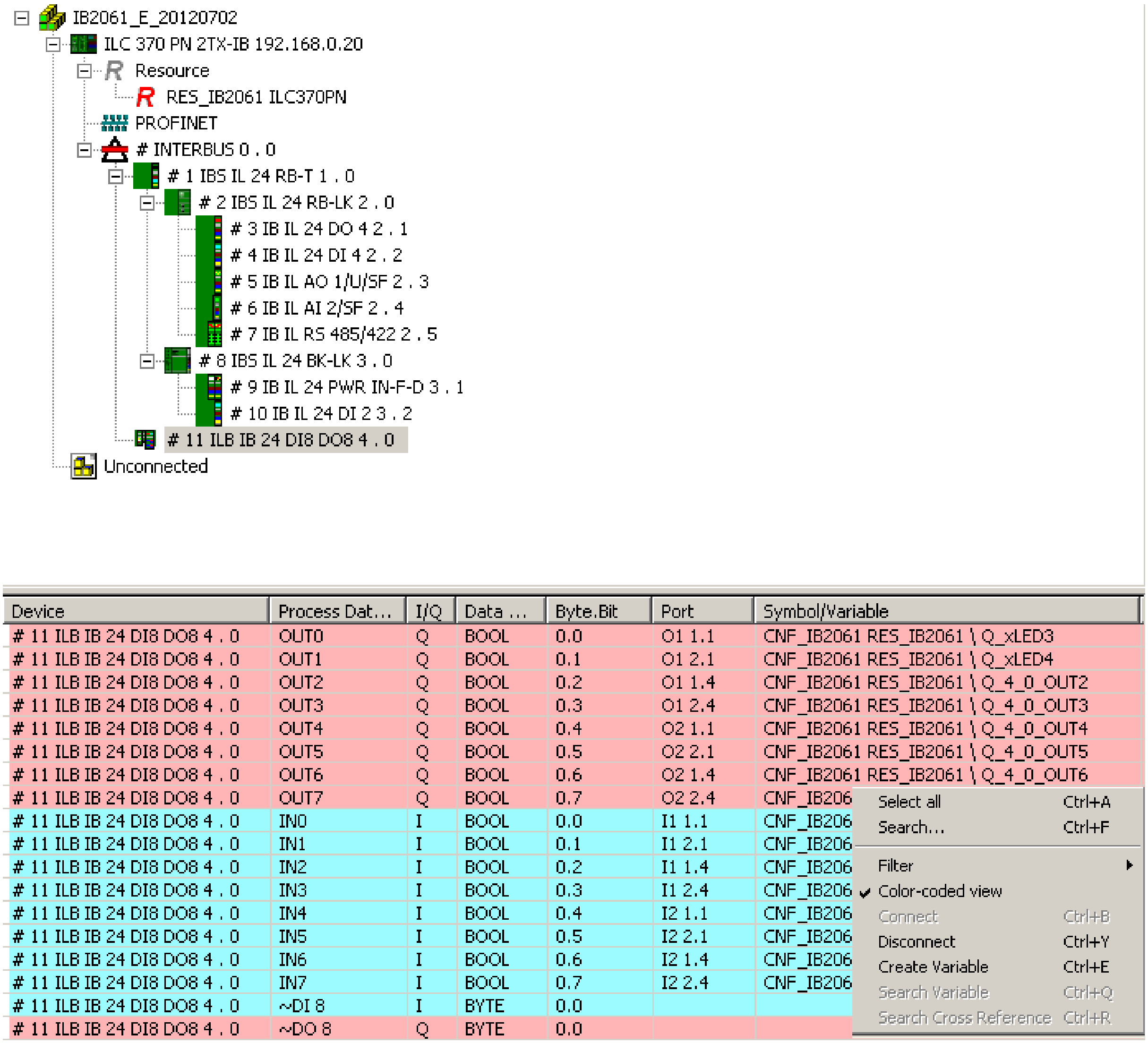
Tasks regarding configuration and variables

PV1 Creating process data variables

In the *Process Data Assignment* view, you will create global variables based on the process data objects that are provided by the configured devices. These automatically created variables should be saved to “Resource”.

Documentation: Page 6-6
Context menu for the process data objects of the modules

What is the relation between the automatically generated variable name and the process data object used?

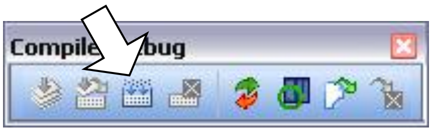


- IB2061_E_20120702
  - ILC 370 PN 2TX-IB 192.168.0.20
    - Resource
      - RES_IB2061 ILC370PN
    - PROFINET
    - # INTERBUS 0 . 0
      - # 1 IBS IL 24 RB-T 1 . 0
        - # 2 IBS IL 24 RB-LK 2 . 0
          - # 3 IB IL 24 DO 4 2 . 1
          - # 4 IB IL 24 DI 4 2 . 2
          - # 5 IB IL AO 1/U/SF 2 . 3
          - # 6 IB IL AI 2/SF 2 . 4
          - # 7 IB IL RS 485/422 2 . 5
        - # 8 IBS IL 24 BK-LK 3 . 0
          - # 9 IB IL 24 PWR IN-F-D 3 . 1
          - # 10 IB IL 24 DI 2 3 . 2
      - # 11 ILB IB 24 DI8 DO8 4 . 0
  - Unconnected

Device	Process Dat...	I/Q	Data ...	Byte.Bit	Port	Symbol/Variable
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT0	Q	BOOL	0.0	O1 1.1	CNF_IB2061 RES_IB2061 \ Q_xLED3
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT1	Q	BOOL	0.1	O1 2.1	CNF_IB2061 RES_IB2061 \ Q_xLED4
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT2	Q	BOOL	0.2	O1 1.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT2
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT3	Q	BOOL	0.3	O1 2.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT3
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT4	Q	BOOL	0.4	O2 1.1	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT4
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT5	Q	BOOL	0.5	O2 2.1	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT5
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT6	Q	BOOL	0.6	O2 1.4	CNF_IB2061 RES_IB2061 \ Q_4_0_OUT6
# 11 ILB IB 24 DI8 DO8 4 . 0	OUT7	Q	BOOL	0.7	O2 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN0	I	BOOL	0.0	I1 1.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN1	I	BOOL	0.1	I1 2.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN2	I	BOOL	0.2	I1 1.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN3	I	BOOL	0.3	I1 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN4	I	BOOL	0.4	I2 1.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN5	I	BOOL	0.5	I2 2.1	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN6	I	BOOL	0.6	I2 1.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	IN7	I	BOOL	0.7	I2 2.4	CNF_IB206
# 11 ILB IB 24 DI8 DO8 4 . 0	~DI 8	I	BYTE	0.0		
# 11 ILB IB 24 DI8 DO8 4 . 0	~DO 8	Q	BYTE	0.0		

PV2 I/O control for process data variables

Once you have created variables for the process data objects to be used, compile the project:

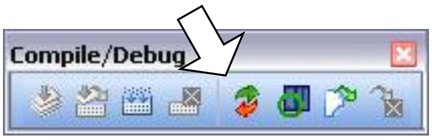


Click the icon above or the F9 function key.

After successful compilation, transmit the project to the main memory of your controller.

Documentation: Page A1-10

Change to *Programming View* (if not already done) and open the global variables table. The variables that you created yourself have automatically been added to the *Auto* group. Minimize all other variable groups (*Default*, *System Variables*) and activate the debug mode (status indicator) of the controller.



Click the icon above or the F10 function key.

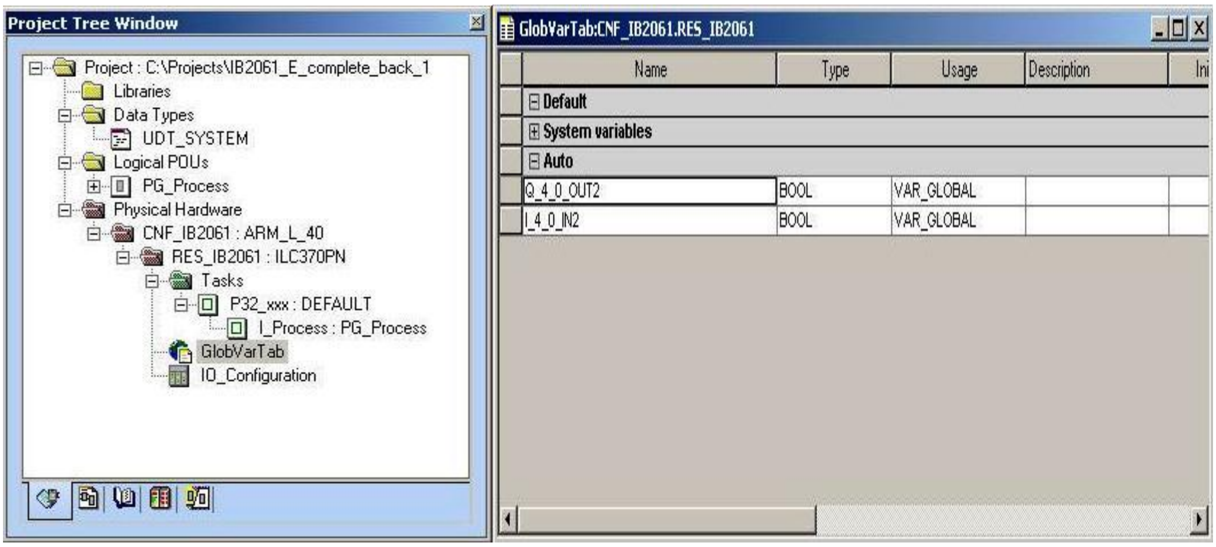
Check the states of your input variables and click the button to the left of the variable names to open the debug dialog box for individual variables. Check the functionality of your outputs by controlling them.

PV3 Adapting variable names

In the global variables table, modify the names of the automatically generated variables so that they can easily be related to hardware connection.

Create a new *Process Variables* variables group and shift the variables to this group without losing the connection to process data.

Context menu for a variables group



Project Tree Window

- Project : C:\Projects\IB2061_E_complete_back_1
  - Libraries
  - Data Types
    - UDT_SYSTEM
  - Logical POUs
    - PG_Process
  - Physical Hardware
    - CNF_IB2061 : ARM_L_40
      - RES_IB2061 : ILC370PN
        - Tasks
          - P32_xxx : DEFAULT
            - I_Process : PG_Process
        - GlobVarTab
        - IO_Configuration

GlobVarTab:CNF_IB2061.RES_IB2061

Name	Type	Usage	Description	Ini
⊟ Default				
⊞ System variables				
⊟ Auto				
Q_4_0_OUT2	BOOL	VAR_GLOBAL		
I_4_0_IN2	BOOL	VAR_GLOBAL		

In variables tables, contents can simply be overwritten after selecting the cell. When a cell is selected, the cursor jumps to position 1 of the text when using the *Pos1* key, and to the end when using the *End* key.

To mark a row, the gray button in variables tables located left to the variable name can be used. If a row is marked, it can be moved exactly using a red target line (without losing process data connection).

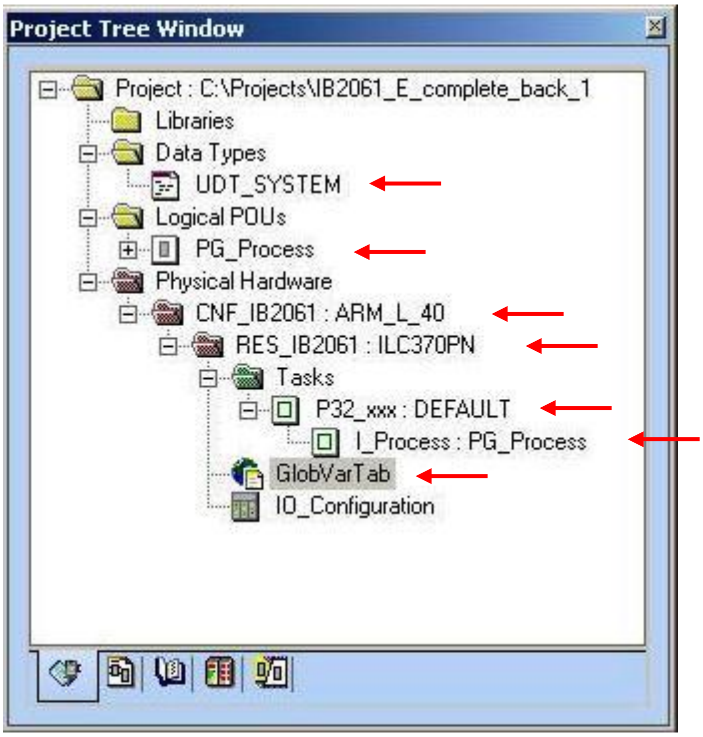
You will find a complete list of all key functions when searching for “Shortcuts” in the help system and selecting the “Shortcuts” hit in the variables worksheet given in tabular form.

PV4 Use of direct inputs (optional)

In the global variables table, direct controller inputs and outputs are available as system variables. For reasons of compatibility, these variables must not be renamed. However, to make them available to the project in a customized manner, it makes sense to copy these signals to individually named variables in the program called first within a PLC cycle.

PV5 Customization of the project

If a project template is used, the elements in the project tree always receive the same standard names. Adapt these to the course standard using the Element Properties dialog box.



The Element Pproperties dialog box of the project tree can be called via the key combination *Alt+Enter* or via the context menu.

Tasks regarding programming in function block diagram

FBDa *First programming*

In the worksheet of the **PG_Course** program, develop programming in such a way that the requirements below will be met.

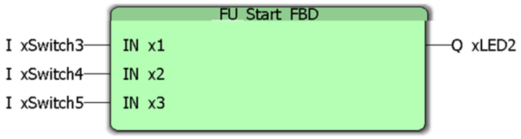
	Switch1	Switch2	Switch3	Light1
Combination 1	True	True	True	True
Combination 2	False	True	True	True
Combination 3	True	False	True	True
Combination 4	False	False	True	True
Combination 5	True	True	False	False
Combination 6	False	True	False	True
Combination 7	True	False	False	False
Combination 8	False	False	False	False

To parameterize your logic connection, use your automatically created and renamed variables instead of the names predefined in the table.

Documentation: Page 12-6
Documentation: Section 10
Block help from the context menu/edit wizard/appendix

Instead of using a NOT block, some blocks (e.g., those the Boolean logic can be used for) allow input and output parameters to be inverted.

FBDb First function



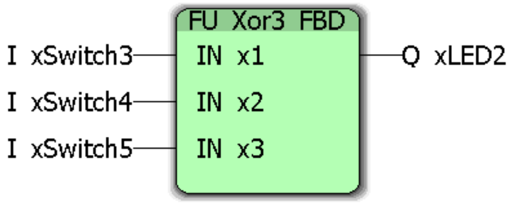
Add a **Functions** POU group to your project tree. To this group, add a new function with the name **FU_Start_FBD** using function block diagram as the programming language.

This function should provide the same logic as the previous programming and should be called in the program (instead of the previous programming).
It is now required to internally use variables of the "VAR_INPUT usage" and the output parameters of the function instead of global variables.

When assigning names to input parameters of function and function blocks, use more general names such as IN1 and IN2 instead of Switch1 and Switch2. The selection of a name should be based on the parameter function and not on the process variable that is accidentally connected to the parameterizable block within a project.

Documentation: Section 10 and pages 13-5ff.

FBD1 Function FU_Xor3_FBD



Add a function with the name **FU_Xor3_FBD** to your project tree. This function should meet the following requirements:

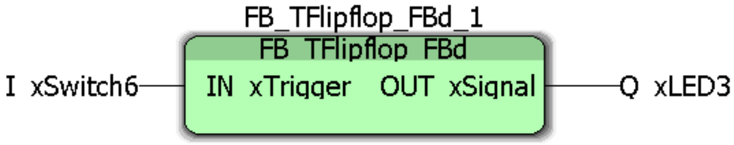
	IN1	IN2	IN3	FU_Xor3_FBD
Combination 1	True	True	True	False
Combination 2	False	True	True	False
Combination 3	True	False	True	False
Combination 4	False	False	True	True
Combination 5	True	True	False	False
Combination 6	False	True	False	True
Combination 7	True	False	False	True
Combination 8	False	False	False	False

Block help from the context menu/edit wizard/appendix

Instead of using a NOT block, some blocks (e.g., those the Boolean logic can be used for) allow input and output parameters to be inverted.

For a systematic approach, check the cases where the output parameter should have the value `True`. Represent these cases separately.

FBD2 Function block FB_TFlipflop_FBD



Add a **Function blocks** POU group to your project tree. To this goup, add a new function block with the name **FB_TFlipflop_FBD**
using function block diagram (FBD) as the programming language.

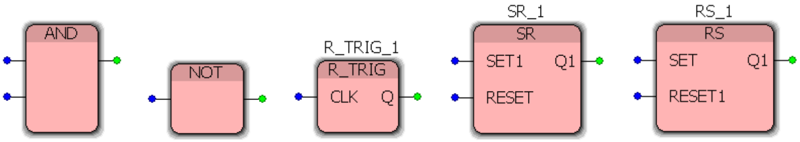
The function block should behave as follows:

If *a rising edge is detected at the IN input parameter,* **then** *the OUT output parameter is to be inverted.*

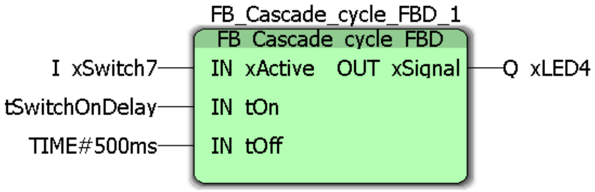
Block help from the context menu/edit wizard/appendix

Divide the task in two subtasks and combine the solutions afterwards. Formulate the task for yourself.

Blocks that can be helpful for creation (not all of them must be used):



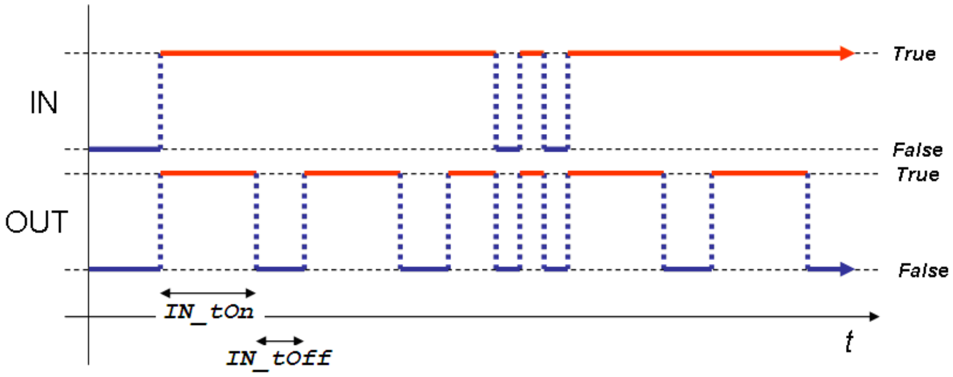
FBD3 Function block FB_Cascade_cycle_FBD



Add a function block with the name **FB_Cascade_cycle_FBD** to your project tree using function block diagram as the programming language.

The function block should behave as follows:
If the IN_xActive input parameter is set to True, the OUT_xSignal output parameter should immediately follow and be set to True. As long as IN_xActive is set to True, the OUT_xSignal should be set to True for the time specified for IN_tOn and then switch to False for the time specified for IN_tOff. This switching shall continue until IN_xActive is set to False. In this case, the OUT_xSignal should also change to False immediately.

This behavior is shown in the following diagram:

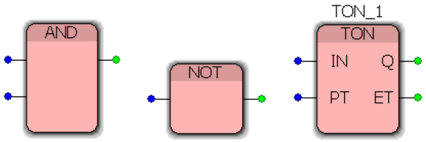


Block help from the context menu/edit wizard/appendix

Use the time diagram to develop programming step by step, i.e. section by section.

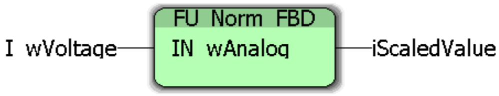
An alternative option is based on the idea of two overlapping signals: the continuous activity signal and an alternating interference signal.

Blocks that can be helpful for creation:

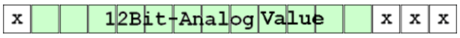


FBD4 Functions for analog value processing

[a] Basic functions: FU_Norm_FBD



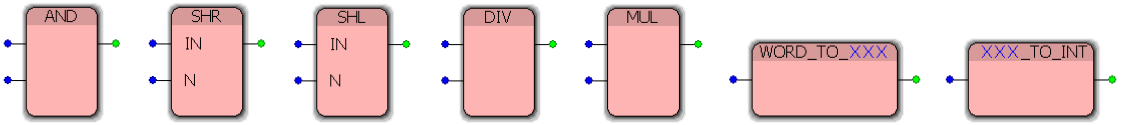
Add a function with the name **FU_Norm_FBD** to your project tree. This function is to scale an analog value provided by an analog input module according to the following scheme:

Analog input value	Representation	Scaled value
0-10V		$0..100_{decimal}$

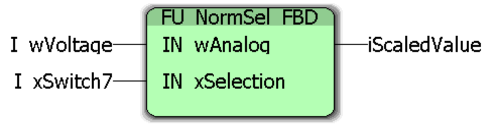
Block help from the context menu/edit wizard/appendix
Device data sheet

Please note that a conversion in data types is required for scaling. These data types are intended for parameterizing arithmetic basic functions and provide a value range sufficient for calculations.

Blocks that can be helpful for creation (not all of them must be used):



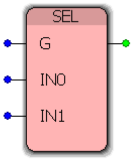
[b] Extension by binary range selection: FU_NormSel_FBD



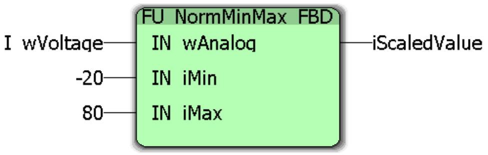
In the project tree, create a copy of the function programmed for task [a]. Change the POU name to **FU_NormSel_FBD** and add the `IN_xSelection` parameter to the programming. This parameter causes the scaled value to be indicated in percent (with `IN_xSelection` = `False`) or in per mil (with `IN_xSelection` = `True`).

Analog input value	Scale	Representation	Scaled value
0-10V	`False`	x 12Bit-Analog Value x x x	$0..100_{decimal}$
0-10V	`True`	x 12Bit-Analog Value x x x	$0..1000_{decimal}$

Additional blocks that can be helpful for creation:



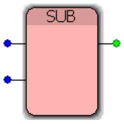
[c] Extension for user-defined scaling: FU_NormMinMax_FBD

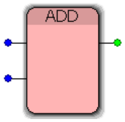


In the project tree, create a copy of the function programmed for task [a]. Change the POU name to **FU_NormMinMax_FBD** and add two INT input parameters - `IN_iMax` and `IN_iMin` - to the programming. These allow the user to dynamically adjust the upper and lower value of the scaled value via the defined values. Scaling should be linear.

Analog input value	Min..Max	Representation	Scaled value
0-10V	`-20..80`	x 12Bit-Analog Value x x x	$-20..80_{decimal}$

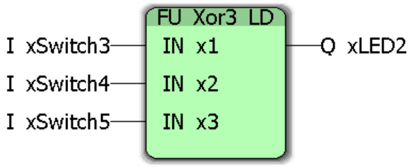
Additional blocks that can be helpful for creation:





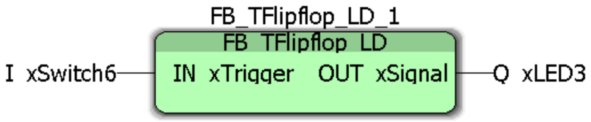
Tasks regarding programming in ladder diagram

LD1 Function FU_Xor3_LD



Add a function with the name **FU_Xor3_LD** to your project tree. This function shall perform the same task as the function of task FBD1.

LD2 Function block FB_TFlipflop_LD

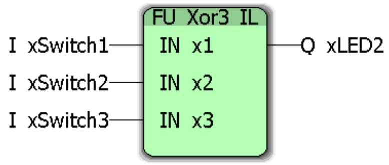


Add a function block with the name **FB_TFlipflop_LD** to your project tree. This block shall perform the same task as the function block of task FBD2.

Consider how you can realize and program an edge evaluation using the elements of ladder diagram and, in addition to this, implement the inversion via contacts and coils, in order to not require the use of function blocks.

Tasks regarding programming in instruction list

IL1 *Function FU_Xor3_IL*

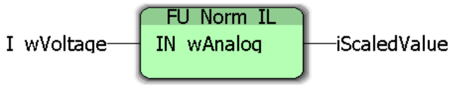


Add a function with the name **FU_Xor3_IL** to your project tree. This function shall perform the same task as the function of task FBD1.

Documentation: Page 14-6

IL4 Functions for analog value processing

[a] Basic functions: FU_Norm_IL

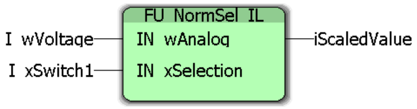


Add a function with the name **FU_Norm_IL** to your project tree. This function shall perform the same task as the function of task FBD4[a].

Documentation: Page 14-10

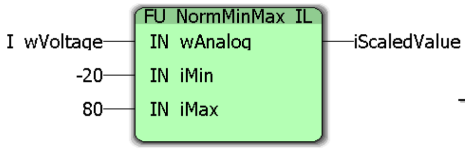
For branches to be executed in parallel, use brackets or local variables (intermediate markers) in instruction list.

[b] Extension by binary range selection: FU_NormSel_IL



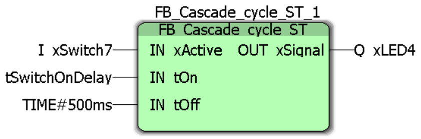
Add a function with the name **FU_NormSel_IL** to your project tree. This function shall perform the same task as the function of task FBD4[b].

[c] Extension for user-defined scaling: FU_NormMinMax_IL



Add a function with the name **FU_NormMinMax_IL** to your project tree. This function shall perform the same task as the function of task FBD4[c].

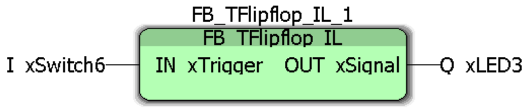
IL3 Function block FB_Cascade_cycle_IL



Add a function block with the name **FB_Cascade_cycle_IL** to your project tree. This block shall perform the same task as the function block of task FBD3.

Documentation: Page 14-12

IL2 Function block FB_TFlipflop_IL



Add a function block with the name **FB_TFlipflop_IL** to your project tree. This block shall perform the same task as the function block of task FBD2.

Consider how you can realize and program an edge evaluation using the elements of instruction list and, in addition to this, implement the inversion via conditional execution, in order to not require the use of function blocks.

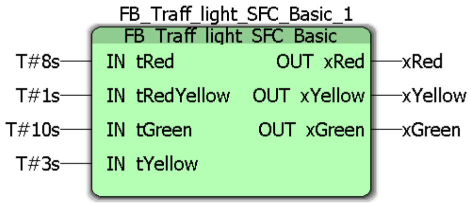
Documentation: Page 14-15

Tasks regarding programming in sequential function chart

SFC5 Function block FB_Traff_light_SFC

[a] Basic functions

Add a function block with the name **FB_Traff_light_SFC** to your project tree. This function block is to be programmed in sequential function chart (SFC).



In the first step, the basic functions of a single signal device are implemented, i.e. the sequence *Red* | *Red-Yellow* | *Green* | *Yellow*.

The input parameters (*Time* data type) are used to provide the block with the phase lengths. The individual light controls are called via the output parameters.

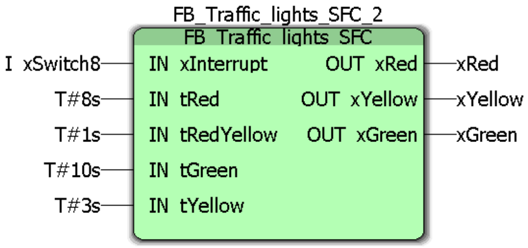
The output parameters should be created as action variables (without using action zooms). The transitions, however, should be programmed as transition zooms (any IEC 61131 language).

The sequence of the corresponding light phase is used as a transition condition.

Documentation: Section 16

[b] Extended functions with error flashing

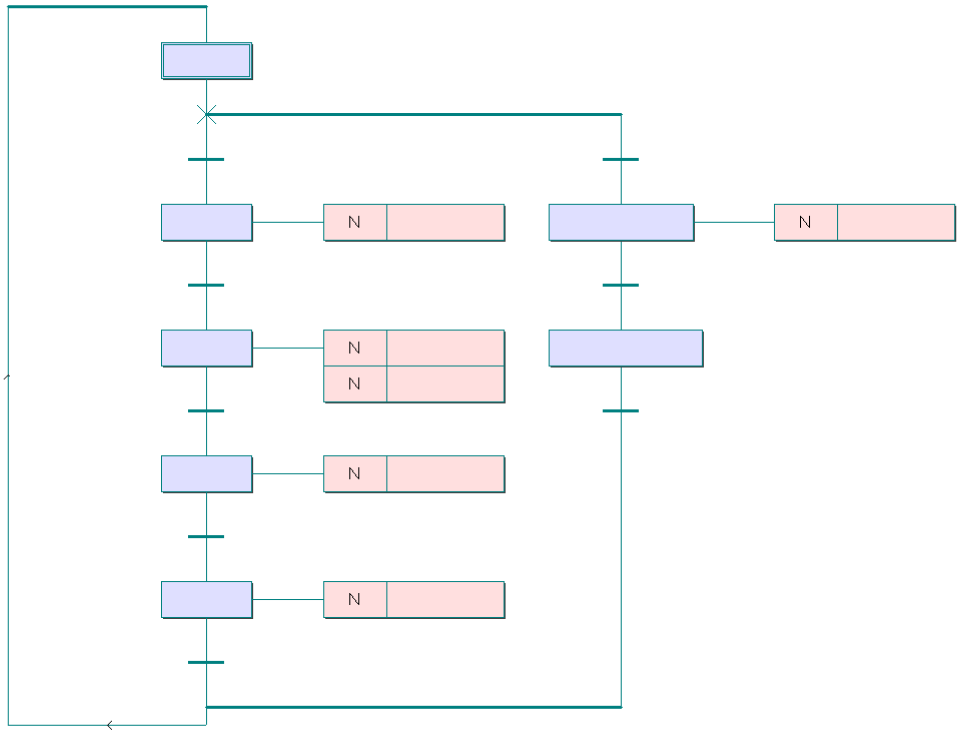
Add the IN_xInterrupt input parameter to the function block created for task [a]. Controlling this input should enable **direct** jumping to an alternative branch which has been inserted in parallel to the four previous action steps. Its sequence triggers a flashing yellow light (f = 1 Hz). This error mode is to be replaced by normal operation after resetting the IN_xInterrupt input parameter.



The deactivation of the currently active step (all steps except for the error mode steps) and the activation of the first error step is to be implemented via the **StepName.x** step flag (e.g. **S_RedYellow.x**) of the steps.

As each transition zoom is executed in each cycle, the required error logging can be implemented in each zoom. It would make sense to implement this programming in the zoom of the initial step.

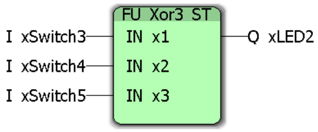
Use the following step scheme as an orientation help for programming:



The step chain shown above uses four steps for traffic light control in normal operation, two steps for error flashing and an additional step as the initial step.

Tasks regarding programming in structured text

ST1 *Function FU_Xor3_ST*

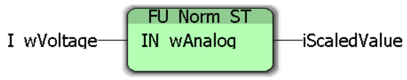


Add a function with the name **FU_Xor3_ST** to your project tree. This function shall perform the same task as the function of task FBD1.

Documentation: Page 18-6

ST4 Functions for analog value processing

[a] Basic functions: FU_Norm_ST

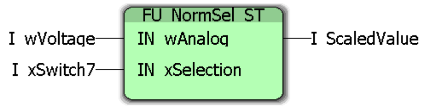


Add a function with the name **FU_Norm_ST** to your project tree. This function shall perform the same task as the function of task FBD4[a].

Documentation: Page 18-8

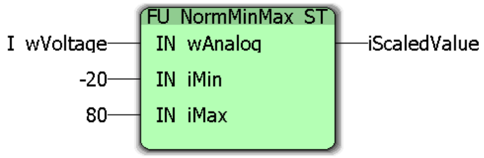
For functions to be executed in parallel and serially, you have to use brackets or local variables (intemediate markers) in structured text.

[b] Extension by binary range selection: FU_NormSel_ST



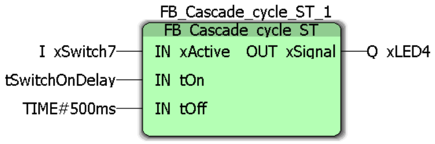
Add a function with the name **FU_NormSel_ST** to your project tree. This function shall perform the same task as the function of task FBD4[b].

[c] Extension for user-defined scaling: FU_NormMinMax_ST



Add a function with the name **FU_NormMinMax_ST** to your project tree. This function shall perform the same task as the function of task FBD4[c].

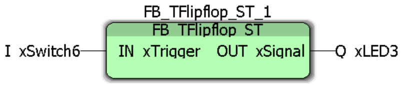
ST3 Function block FB_Cascade_cycle_ST



Add a function block with the name **FB_Cascade_cycle_ST** to your project tree. This block shall perform the same task as the function block of task FBD3.

Documentation: Page 18-10

ST2 Function block FB_TFlipflop_ST (optional)



Add a function block with the name **FB_TFlipflop_ST** to your project tree. This function shall perform the same task as the function block of task FBD2.

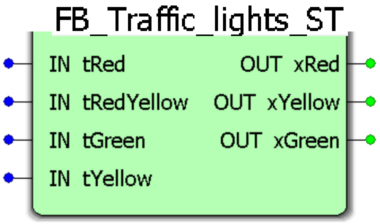
Consider how you can do without function blocks when using high-level language structures.

Documentation: Page 18-17

Re-read the formulation of functionality in task FBD2.

ST5 Function block FB_Traffic_lights_ST (optional)

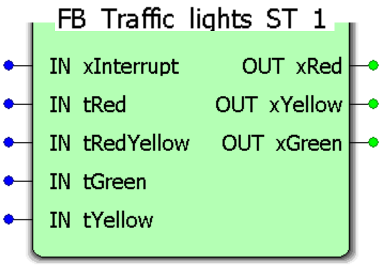
[a] Add a function block with the name **FB_Traffic_lights_ST** to your project tree. This block shall perform the same task as the function block of task SFC5.



Consider how you can implement the step chain following the example of the alternative solution for FBD3.

Documentation: Page 18-23

[b] Extend programming by an IN_xInterrupt input parameter, in the same way as with sequential function chart.



ST6 Logging of INTERBUS errors with FIFO memory

The target of the complex task ST6 is to log bus errors reported by the INTERBUS controller (including date, time, segment, position and plain text information on the respective device). This error message should be recorded in a FIFO (First In First Out) memory using ten elements. Element 1 should always be used to output the current error message.

[a] Data type definition

1. Definition of a structure data type according to the message requirements

ST_Message

└ Date	Type: `String`
└ Time	Type: `String`
└ Segment	Type: `Int`
└ Position	Type: `Int`

2. Definition of an array data type according to the message list requirements

AR_1_10_Message

Type: `Array [1..10] Of ST_Message`

[b] Program for error logging:

Add a program with the name **PG_Messages** in structured text to your project tree. Once completed, instantiate the program by calling the cyclic task available in your project under the name **I_Messages**.

For information on functions required for the following tasks, please refer to the last pages of the task in hand.

The program should perform the following tasks in five steps:

1. Processing values provided by the system variables and saving them to local variables.

Use two-digit values for day, month, hour, minute, and second. Use a four-digit value for the year.

Variable	Usage	Data type
strDay	VAR	String
strMonth	VAR	String
strYear	VAR	String
strHour	VAR	String
strMinute	VAR	String
strSecond	VAR	String
iSegment	VAR	String
iPosition	VAR	String

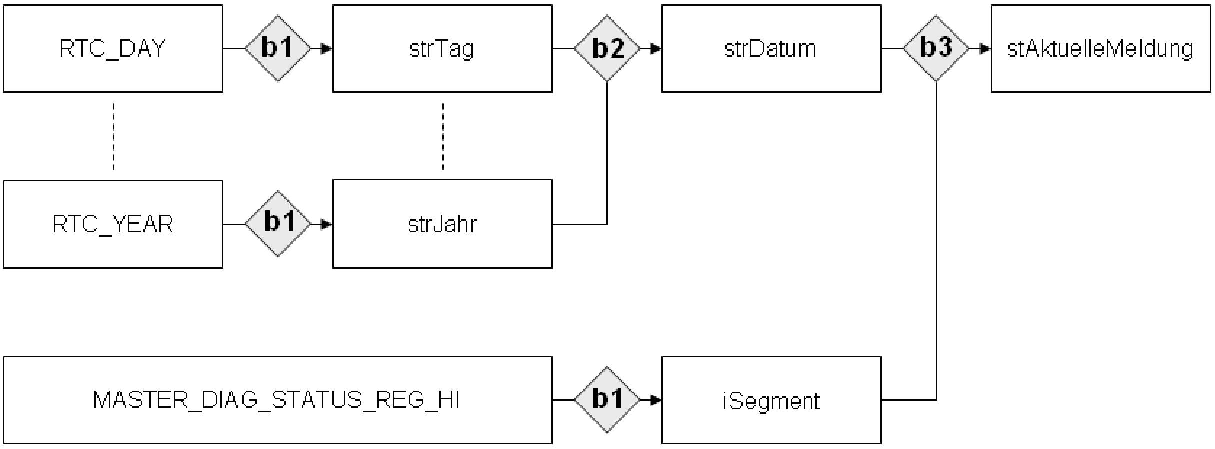
2. Combining time and date values to string variables of the following format:

Use the standard formats for date and time:
Example for date: `’26.01.2007’`
Example for time: `’14:17:52’`

Variable	Usage	Data type
strDate	VAR	String
strTime	VAR	String

3. Entering the compiled information of the previous steps in a local variable.

Variable	Usage	Data type
stCurrentMessage	VAR	ST_Message

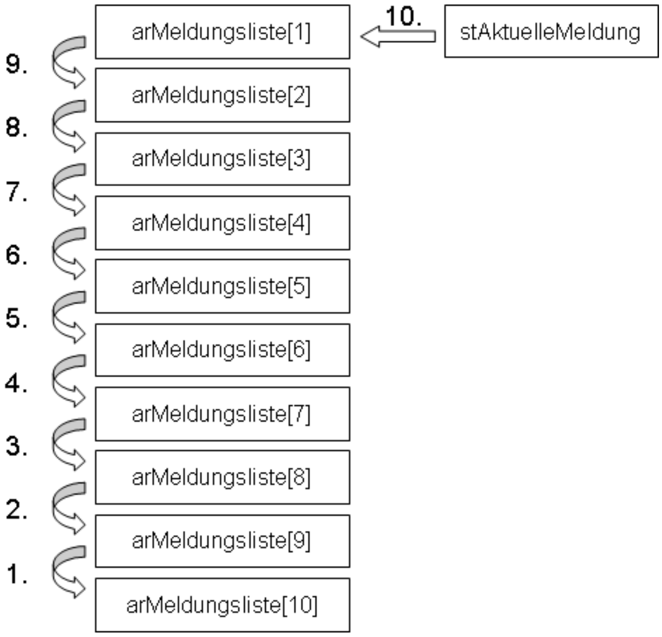


Graphical overview of conversions and assignments (selection)

4. Creating a global variable for entering the current message. This variable represents the message list.

Variable	Usage	Data type
arMessageList	VAR_GLOBAL	AR_1_10_Message

To update the message list, first copy the content of element 9 to element 10, then from element 8 to element 9, etc. Finally, the current message is entered in element 1.



This update can be done manually by means of ten separate statements or implemented via a For loop and an additional assignment. The For loop is not able to count down the control variable. This represents a special challenge when programming the For loop.

5. Conditional execution of the entry routine

The entry routine should only be executed, if the falling edge of the error-detecting bit and the pending bus error bit of the INTERBUS master are detected by the program. With regard to detection, use the system variables provided by the controller (additional information can be found in the appendix).

[c] Device name in plain text

The following tasks enable the respective device name to be shown in plain text.

1. Extending the ST_Message data type by one parameter.

ST_Message

└ Date	Type: `String`
└ Time	Type: `String`
└ Segment	Type: `Int`
└ Position	Type: `Int`
└ Name	Type: `String`

2. Integrating a user library.

Integrate the *BIB_IBS_CFG_INFO_V1_0_D.zwt* library in the current project.

The library provides the following elements:

Data types:

IBS_ST_DevInfo

└ Name	Type: `IBS_String20`
└ Segment	Type: `Int`
└ Position	Type: `Int`
└ Active	Type: `Bool`
└ Jumpered	Type: `Bool`
└ PF	Type: `Bool`

IBS_AR_1_512_DevInfo

└ `ARRAY [1..512] OF IBS_ST_DevInfo`

IBS_ST_CfgInfo

└ DevCnt	Type: `Int`
└ DevInfoList	Type: `IBS_AR_1_512_DevInfo`

Programs:

PG_InterbusConfiguration

The program reads the active INTERBUS configuration from the INTERBUS master using the *Read_Complete_Configuration* firmware service and determines the number of devices within the configuration. The data acquired is entered in the **stIBS** structure variables (module segment and position, active status and jumpered status, peripheral fault occurred). The structure variables are sorted according to the consecutive number of the modules. In addition, the program reads the Diag32 file saved on the memory card of the controller by sending the boot project. In the **stIBS** variable, the station names for the IBS modules which are saved in this file (with the initial value for the variable `iStartTabulatorForDetail` = 10) are added to the already acquired module information.

3. Program call and parameterization

Instantiate the program provided by the library and call the program via the cyclic task available in your program.

In the context menu of the project tree, select the function for transforming VAR_EXTERNAL into VAR_GLOBAL.

For the `iStartTabulatorForDetail` variable, assign the value specified in the description.

4. Using the device name from the configuration structure variable.

Check the stIBS variable for entries with regard to segment and position and use the entered name from the variable, on condition that both parameters match the values currently provided by the IBS master.

⚠ Consider the following situations: The IBS master is reported as a faulty segment (segment: 0) or an OUT1 or OUT2 device interface error has occurred. In the event of interface errors, the segment variable indicates the segment of the reporting remote bus device, the position variable has the value 80_{hex} for OUT1 or 81_{hex} for OUT2.

Support for task ST6

For [b] 1. System variables, realtime clock, INTERBUS diagnostics

The following system variables provide information on the system time of the ILC:

RTC_Year	INT (four-digit)
RTC_Month	INT
RTC_Day	INT
RTC_Hours	INT (24h-indication)
RTC_Minutes	INT
RTC_Seconds	INT

The following variables provide information on the bus error, active bus error diagnostics and the respective device:

MASTER_DIAG_PARAM_REG_HI	BYTE (Segment)
MASTER_DIAG_PARAM_REG_LOW	BYTE (Position)
MASTER_DIAG_STATUS_REG_BUS	BOOL (IBS bus error)
MASTER_DIAG_STATUS_REG_DTC	BOOL (IBS diagnostics active)

For [b] 1. Data type conversion Word_To_Int

When converting a byte variable into an integer value, please note that the most significant bit is the sign bit. If a byte without sign bit is to be converted, it is recommended to use the word data type in a roundabout way. The call structure is then as follows:

```
IntVar := Word_to_Int(Byte_To_Word(ByteVar));
```

This is the nested call of functions, as it is typical for structured text. Example for conversion with sign bit:

```
Byte#2#1001_0101 → Byte_To_Int →                    -107
Byte#2#1001_0101 → Byte_To_Word → Word_To_Int →     149
```

For [b] 1. Data type conversion Int_To_String

INT_TO_STRING is the function for converting an integer variable in a string variable. This function requires two input parameters, i.e. the integer to be converted (of integer type) and the format (of string type) into which the string is to be converted. The call in structured text is as follows:

```
String := Int_To_String(INT_Variable, Format);
```

Example:

INT_Value :	4
Format:	'%02d'
String:	'04'

(The number is indicated with two digits and leading zeros.)

INT_Value:	2007
Format:	'%04d'
String:	'2007'

(The number is indicated with four digits and leading zeros.)

For [b] 2. Concat string concatenation

CONCAT is the function for combining two strings to a new string. It is used as follows:

```
New string := Concat(string1, string2);
```

Example:

String 1 :	'PC WORX'
String 2:	' is awesome!'
New string:	'PC WORX is awesome!'

Spaces in the string (as in front of 'is') are also used. Please consider that only two values can be combined using CONCAT. To set up date and time, CONCAT has to be called four times in each case.

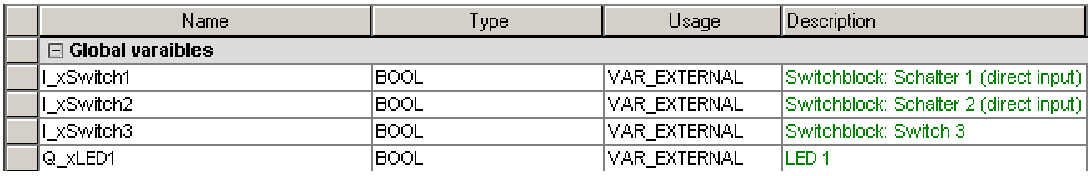
Solutions for the
PC WORX IEC 61131 Programming Course

Latest version: 5. Februar 2014
Reference: PC WORX 6.00.25 SP3.73

Solutions regarding programming in function block diagram

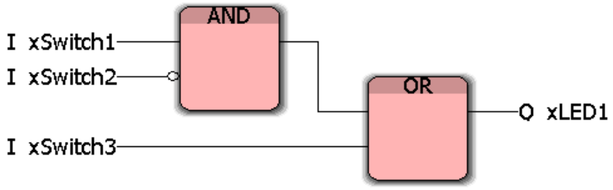
FBDa First programming

Variables table



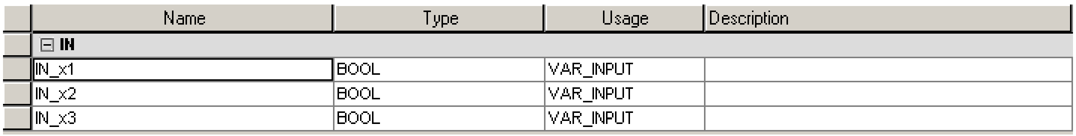
Name	Type	Usage	Description
⊟ **Global varaibles**			
I_xSwitch1	BOOL	VAR_EXTERNAL	Switchblock: Schalter 1 (direct input)
I_xSwitch2	BOOL	VAR_EXTERNAL	Switchblock: Schalter 2 (direct input)
I_xSwitch3	BOOL	VAR_EXTERNAL	Switchblock: Switch 3
Q_xLED1	BOOL	VAR_EXTERNAL	LED 1

Programming



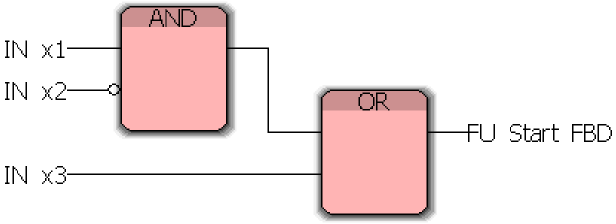
FBDb First function

Variables table



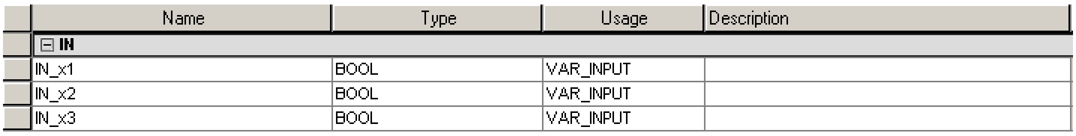
Name	Type	Usage	Description
⊟ **IN**			
IN_x1	BOOL	VAR_INPUT	
IN_x2	BOOL	VAR_INPUT	
IN_x3	BOOL	VAR_INPUT	

Programming



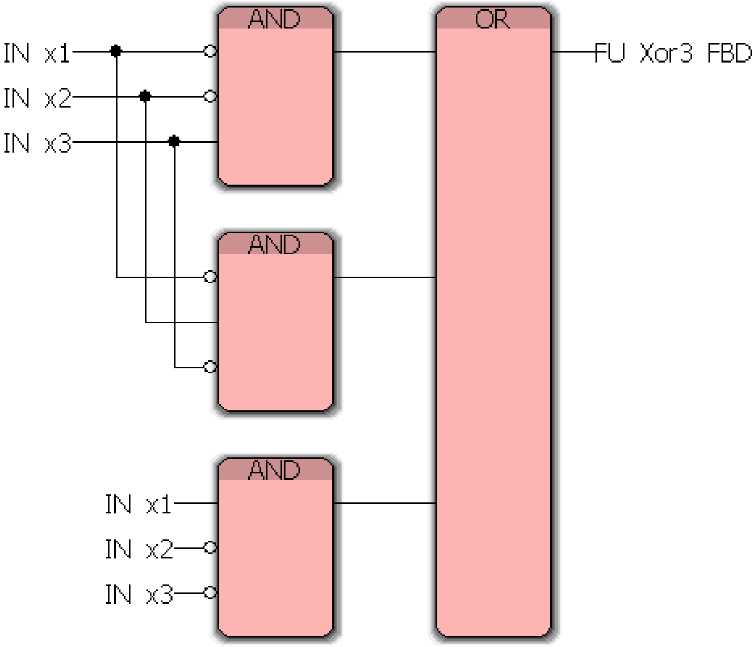
FBD1 *FU_Xor3_FBD*

Variables table



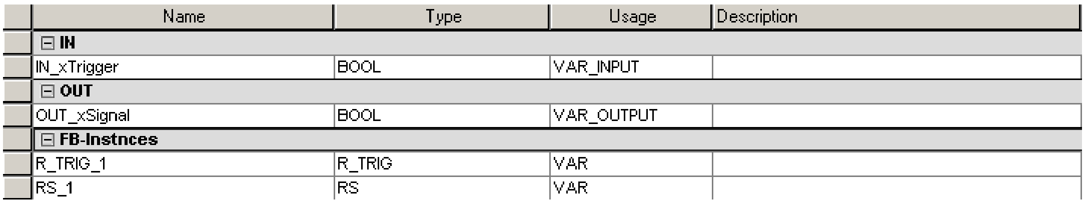
Name	Type	Usage	Description
⊟ **IN**			
IN_x1	BOOL	VAR_INPUT	
IN_x2	BOOL	VAR_INPUT	
IN_x3	BOOL	VAR_INPUT	

Programming



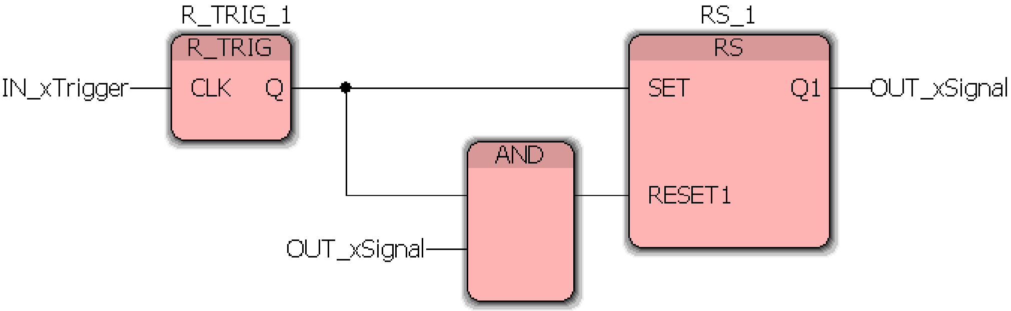
FBD2 FB_TFlipflop_FBD

Variables table

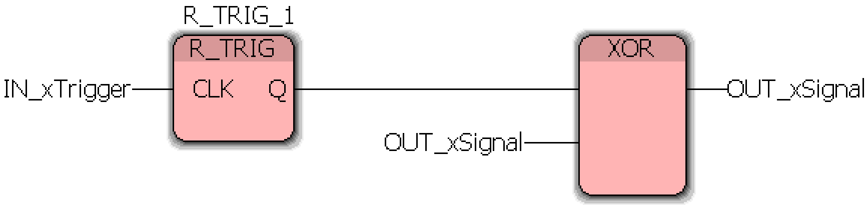


Name	Type	Usage	Description
⊟ **IN**			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **FB-Instnces**			
R_TRIG_1	R_TRIG	VAR	
RS_1	RS	VAR	

Programming

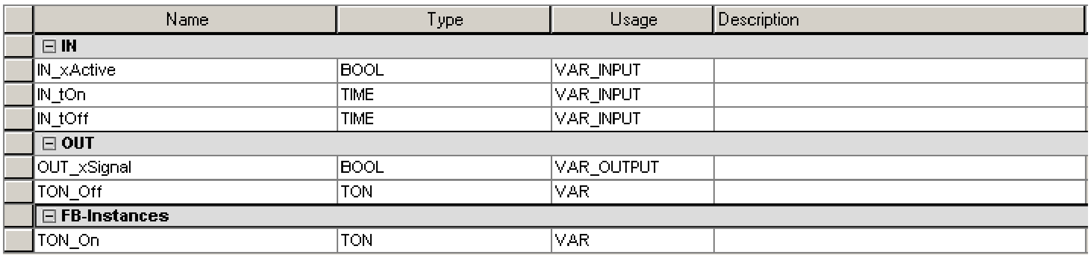


Alternative programming



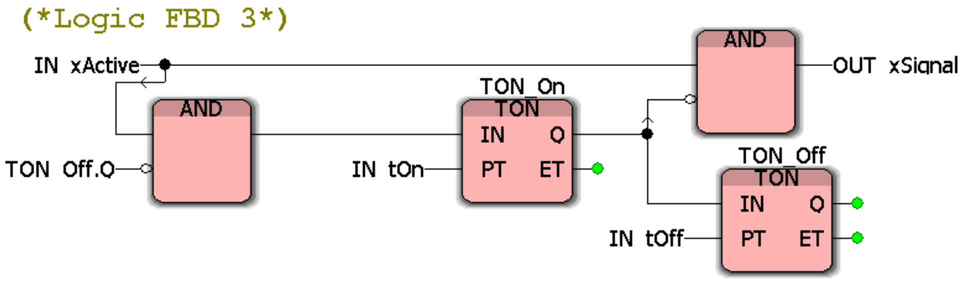
FBD3 FB_Cascade_cycle_FBD

Variables table



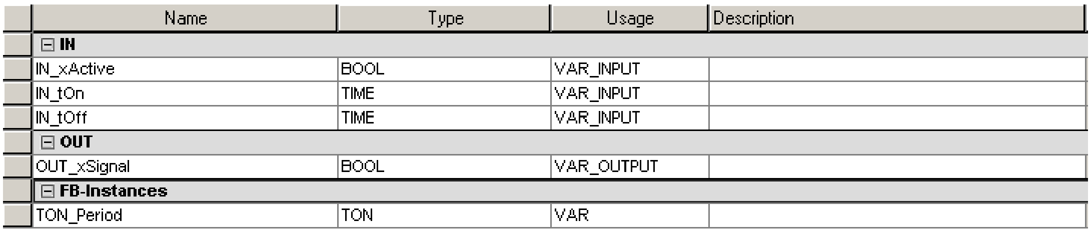
Name	Type	Usage	Description
⊟ **IN**			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
TON_Off	TON	VAR	
⊟ **FB-Instances**			
TON_On	TON	VAR	

Programming



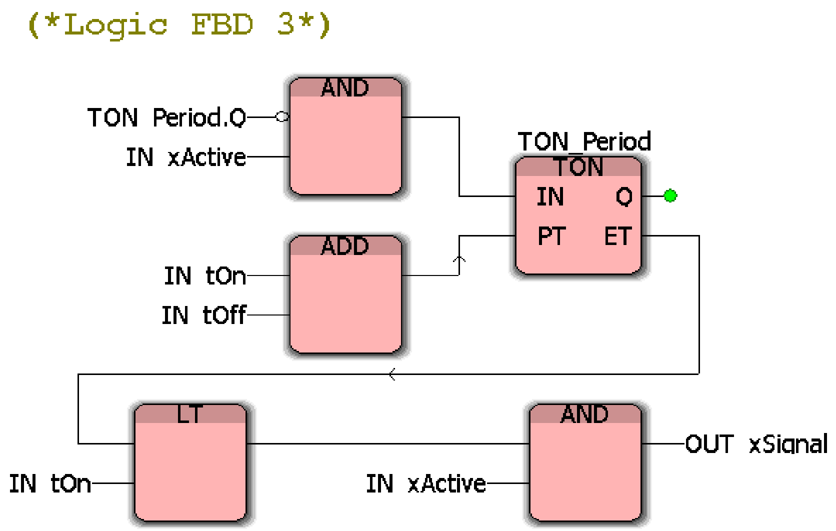
FBD3 FB_Cascade_cycle_FBD (alternative solution)

Variables table



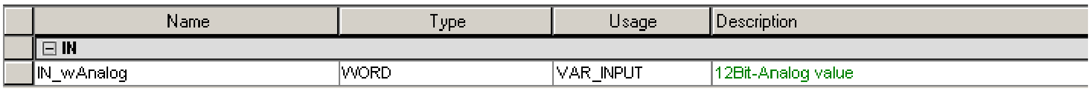
Name	Type	Usage	Description
⊟ **IN**			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **FB-Instances**			
TON_Period	TON	VAR	

Programming



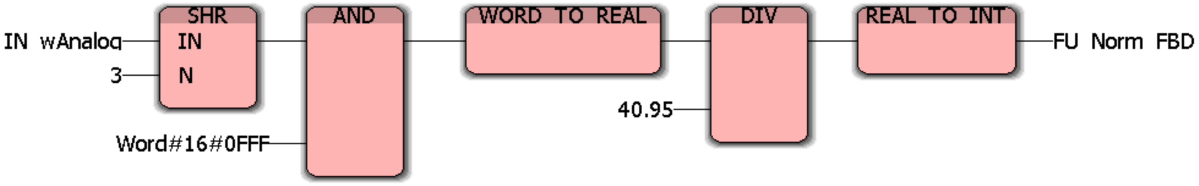
FBD4[a] FU_Norm_FBD

Variables table



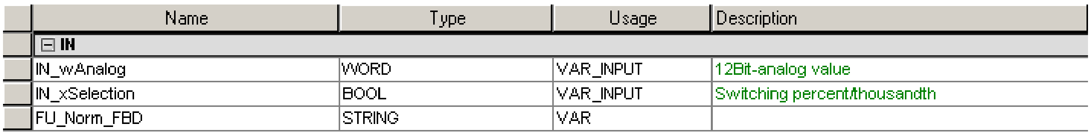
Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-Analog value

Programming



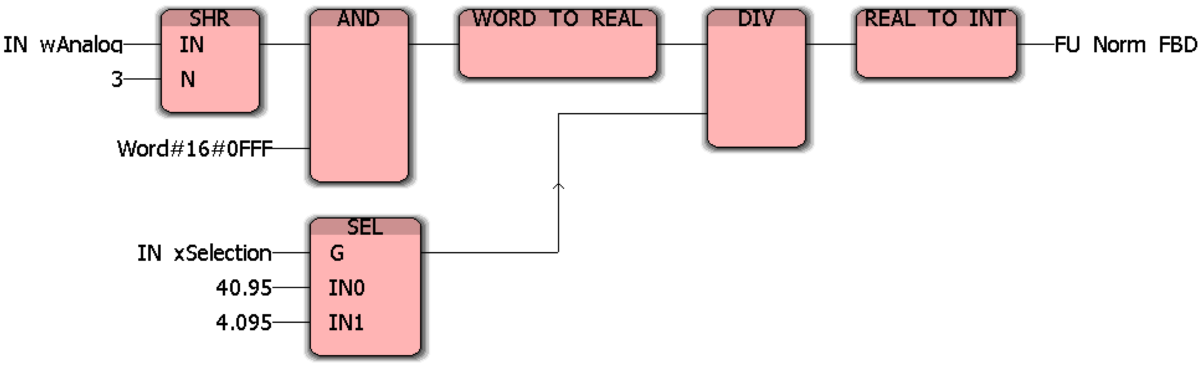
FBD4[b] FU_NormSel_FBD

Variables table



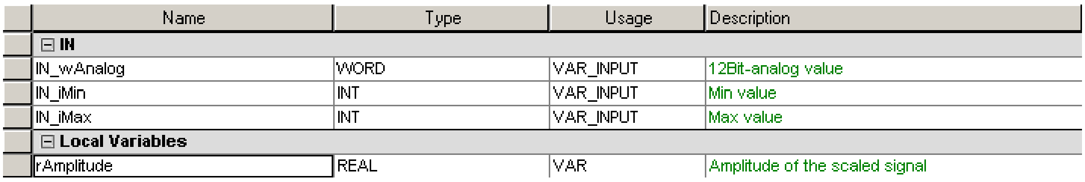
Name	Type	Usage	Description
⊟ IN			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth
FU_Norm_FBD	STRING	VAR	

Programming



FBD4[c] *FU_NormMinMax_FBD*

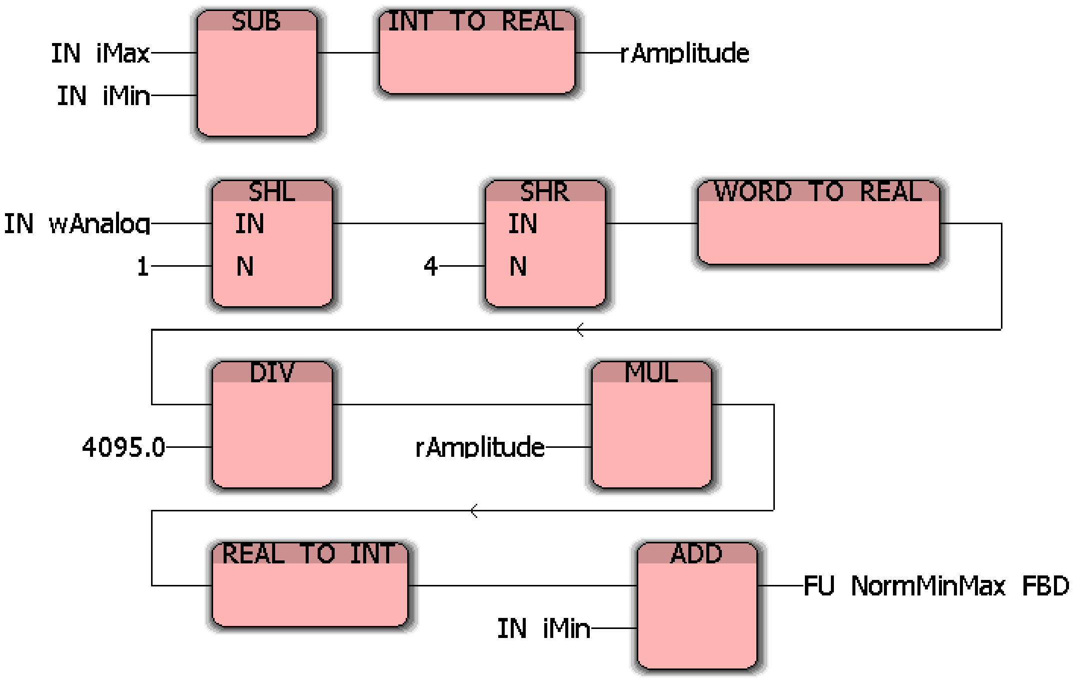
Variables table



	Name	Type	Usage	Description
	⊟ **IN**			
	IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
	IN_iMin	INT	VAR_INPUT	Min value
	IN_iMax	INT	VAR_INPUT	Max value
	⊟ **Local Variables**			
	rAmplitude	REAL	VAR	Amplitude of the scaled signal

Programming

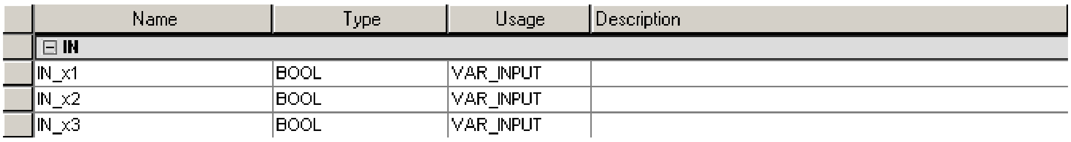
(*Logik FBS 4c*)



Solutions regarding programming in ladder diagram

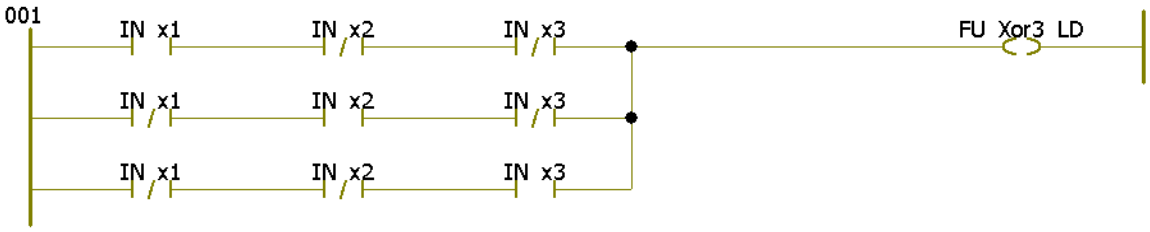
LD1 FU_Xor3_LD

Variables table



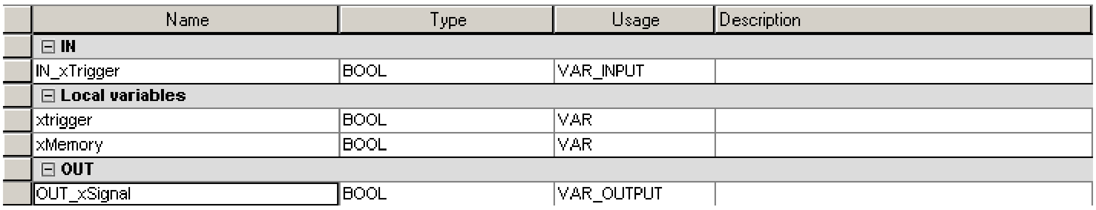
	Name	Type	Usage	Description
	⊟ **IN**			
	IN_x1	BOOL	VAR_INPUT	
	IN_x2	BOOL	VAR_INPUT	
	IN_x3	BOOL	VAR_INPUT	

Programming



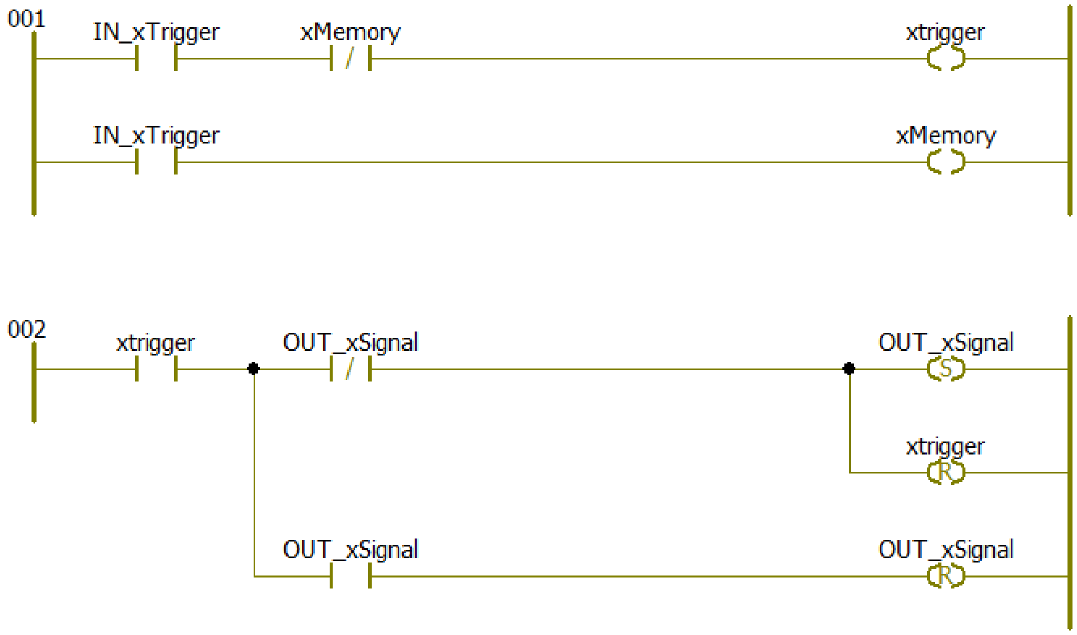
LD2 FB_TFlipflop_LD

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ **Local variables**			
xtrigger	BOOL	VAR	
xMemory	BOOL	VAR	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	

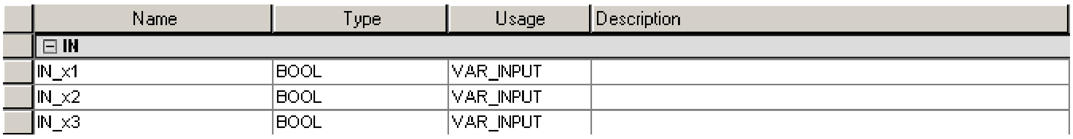
Programming



Solutions regarding programming in instruction list

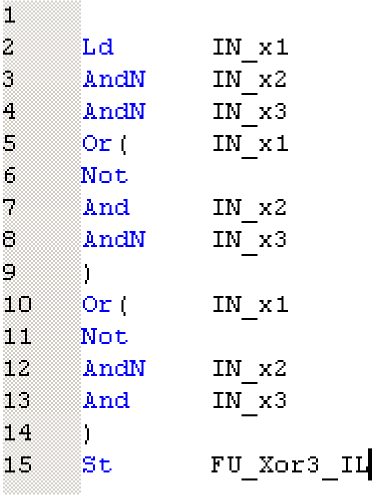
IL1 FU_Xor3_IL

Variables table



Name	Type	Usage	Description
⊟ IN			
IN_x1	BOOL	VAR_INPUT	
IN_x2	BOOL	VAR_INPUT	
IN_x3	BOOL	VAR_INPUT	

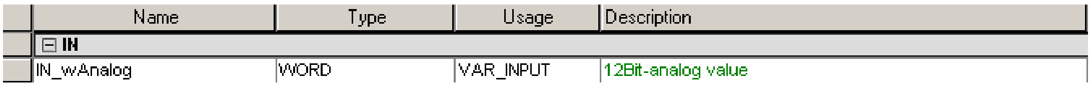
Programming



```

Ld      IN_x1
AndN    IN_x2
AndN    IN_x3
Or(     IN_x1
Not
And     IN_x2
AndN    IN_x3
)
Or(     IN_x1
Not
AndN    IN_x2
And     IN_x3
)
St      FU_Xor3_IL
```

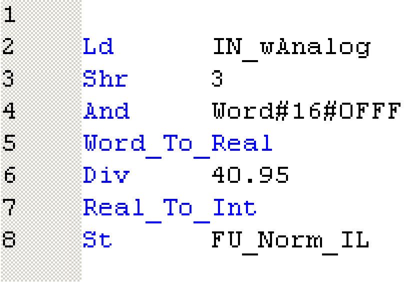
IL4[a] FU_Norm_IL

Variables table



Name	Type	Usage	Description
⊟ IN			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value

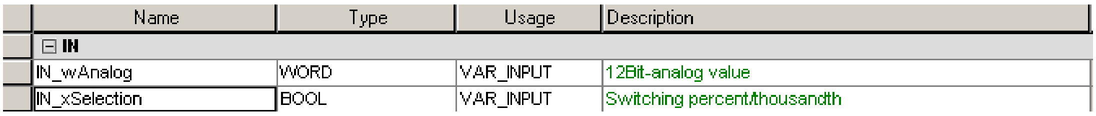
Programming



```

Ld        IN_wAnalog
Shr       3
And       Word#16#0FFF
Word_To_Real
Div       40.95
Real_To_Int
St        FU_Norm_IL
```

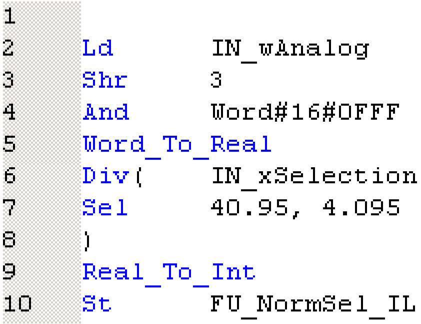
IL4[b] FU_NormSel_IL

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth

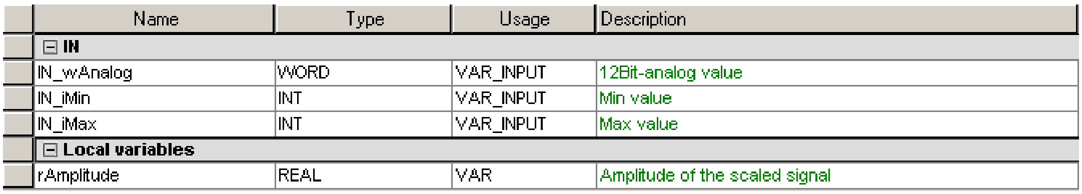
Programming



```

Ld        IN_wAnalog
Shr       3
And       Word#16#0FFF
Word_To_Real
Div(      IN_xSelection
Sel       40.95, 4.095
)
Real_To_Int
St        FU_NormSel_IL
```

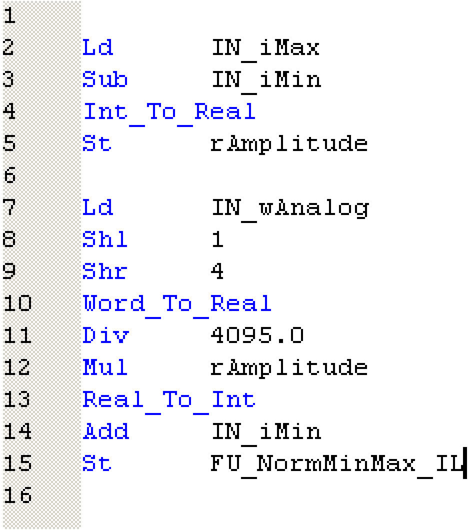
IL4[c] FU_NormMinMax_IL

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_iMin	INT	VAR_INPUT	Min value
IN_iMax	INT	VAR_INPUT	Max value
⊟ **Local variables**			
rAmplitude	REAL	VAR	Amplitude of the scaled signal

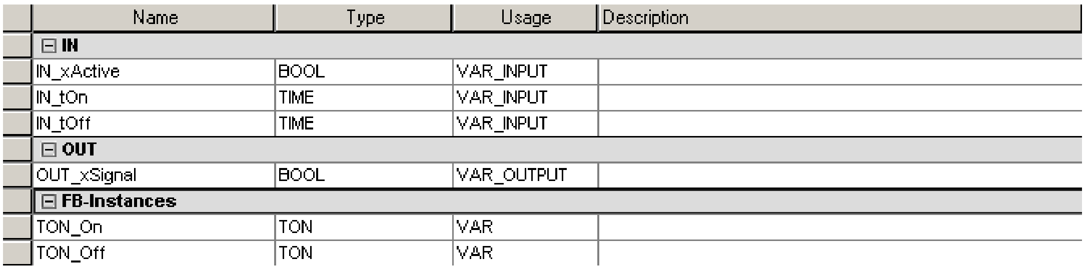
Programming



```

Ld        IN_iMax
Sub       IN_iMin
Int_To_Real
St        rAmplitude

Ld        IN_wAnalog
Shl       1
Shr       4
Word_To_Real
Div       4095.0
Mul       rAmplitude
Real_To_Int
Add       IN_iMin
St        FU_NormMinMax_IL

```

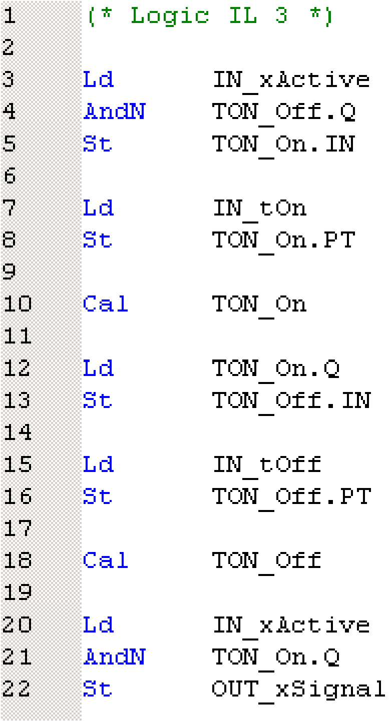
IL3 FB_Cascade_cycle_IL

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **FB-Instances**			
TON_On	TON	VAR	
TON_Off	TON	VAR	

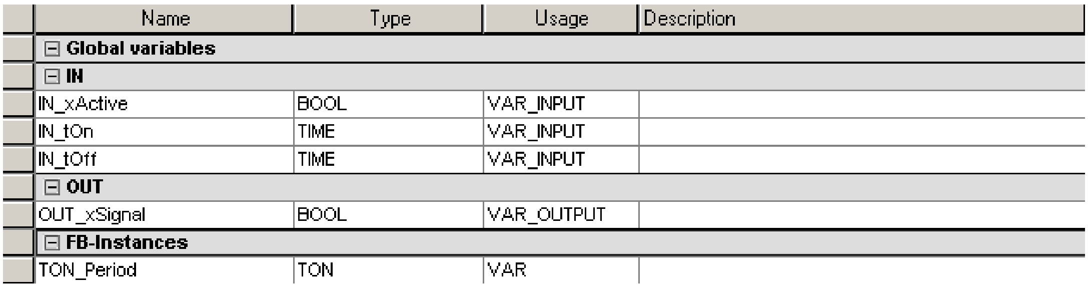
Programming



```
(* Logic IL 3 *)

Ld      IN_xActive
AndN    TON_Off.Q
St      TON_On.IN

Ld      IN_tOn
St      TON_On.PT

Cal     TON_On

Ld      TON_On.Q
St      TON_Off.IN

Ld      IN_tOff
St      TON_Off.PT

Cal     TON_Off

Ld      IN_xActive
AndN    TON_On.Q
St      OUT_xSignal
```

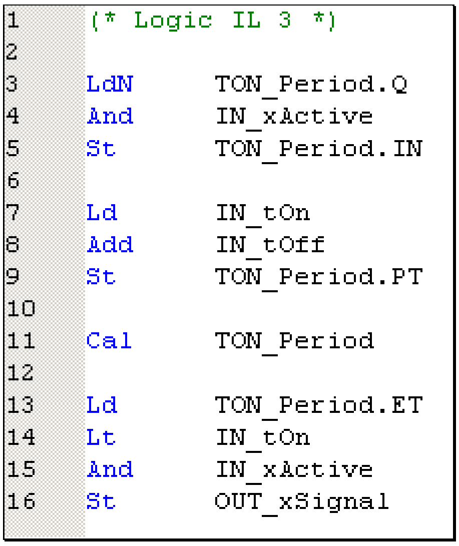
IL3 FB_Period_cycle_IL (alternative solution)

Variables table



Name	Type	Usage	Description
⊟ **Global variables**			
⊟ **IN**			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **FB-Instances**			
TON_Period	TON	VAR	

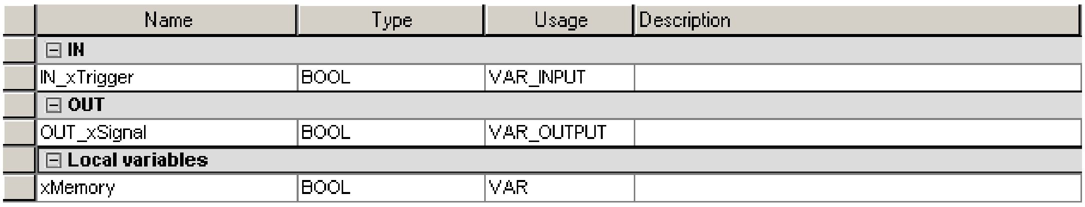
Programming



```
(* Logic IL 3 *)

LdN      TON_Period.Q
And      IN_xActive
St       TON_Period.IN

Ld       IN_tOn
Add      IN_tOff
St       TON_Period.PT

Cal      TON_Period

Ld       TON_Period.ET
Lt       IN_tOn
And      IN_xActive
St       OUT_xSignal
```

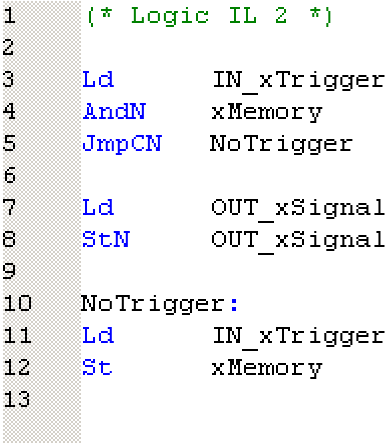
IL2 FB_TFlipflop_IL

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **Local variables**			
xMemory	BOOL	VAR	

Programming

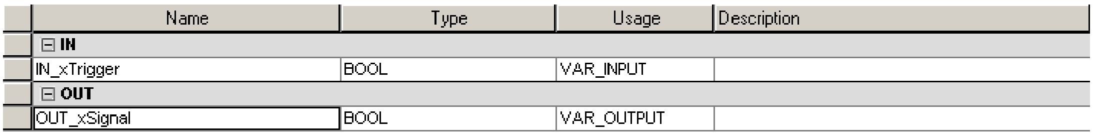


```
 (* Logic IL 2 *)

Ld       IN_xTrigger
AndN     xMemory
JmpCN    NoTrigger

Ld       OUT_xSignal
StN      OUT_xSignal

NoTrigger:
Ld       IN_xTrigger
St       xMemory

```

Solutions regarding programming in sequential function chart

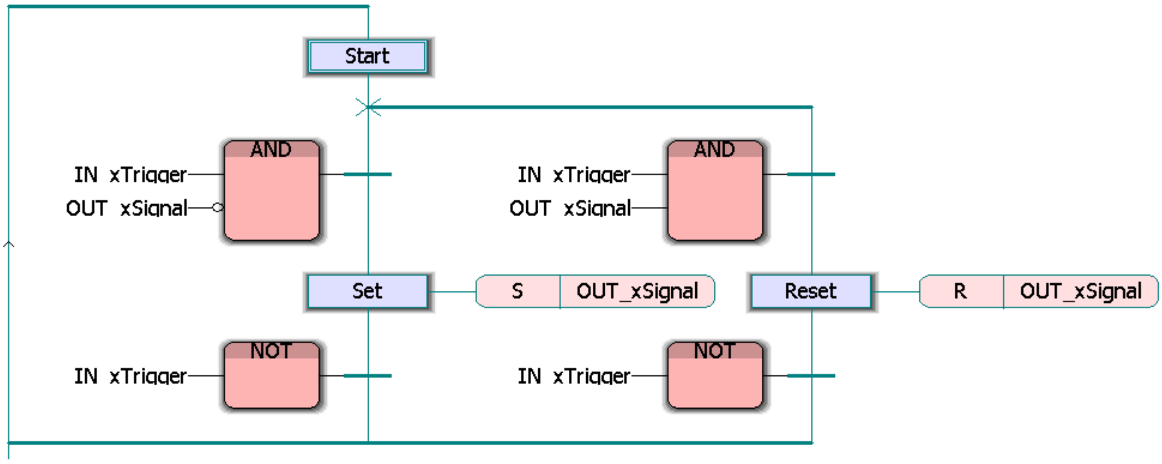
SFC2 FB_TFlipflop_SFC (no task assigned)

Variables table



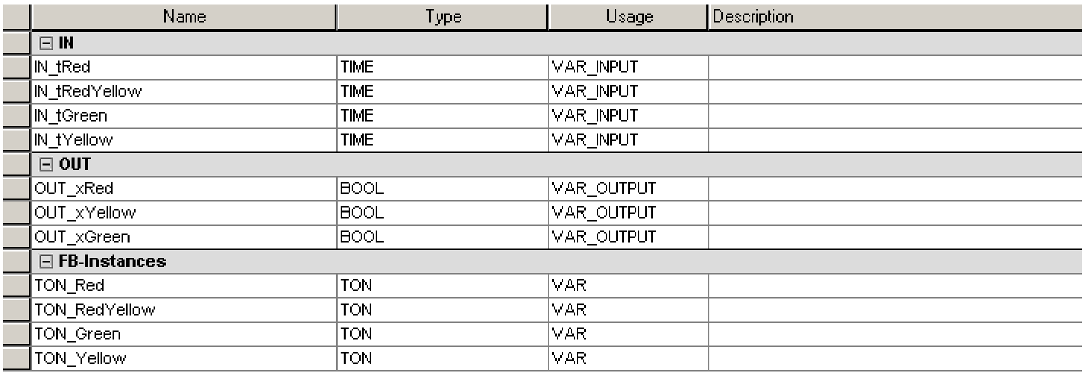
Name	Type	Usage	Description
⊟ **IN**			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	

Programming



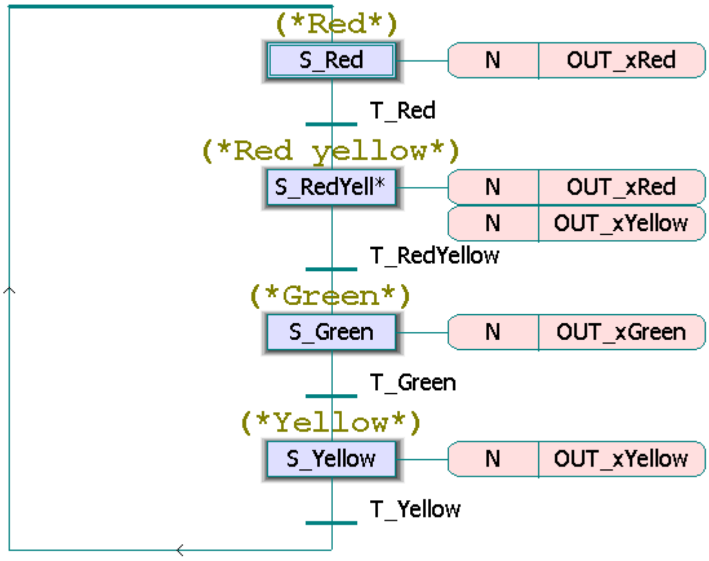
SFC5[a] FB_Traff_light_SFC

Variables table

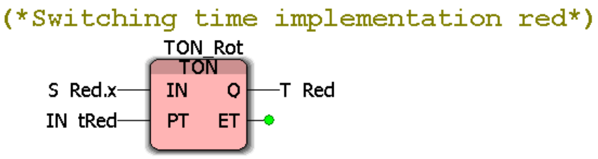


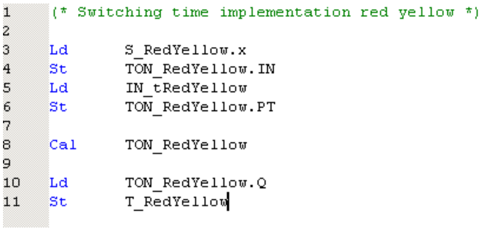
Name	Type	Usage	Description
⊟ IN			
IN_tRed	TIME	VAR_INPUT	
IN_tRedYellow	TIME	VAR_INPUT	
IN_tGreen	TIME	VAR_INPUT	
IN_tYellow	TIME	VAR_INPUT	
⊟ OUT			
OUT_xRed	BOOL	VAR_OUTPUT	
OUT_xYellow	BOOL	VAR_OUTPUT	
OUT_xGreen	BOOL	VAR_OUTPUT	
⊟ FB-Instances			
TON_Red	TON	VAR	
TON_RedYellow	TON	VAR	
TON_Green	TON	VAR	
TON_Yellow	TON	VAR	

Programming

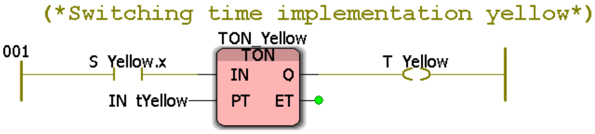


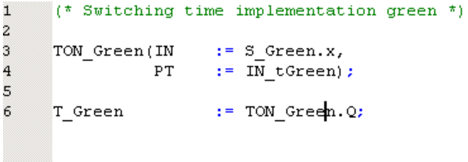
Detail programming





```
(* Switching time implementation red yellow *)

Ld      S_RedYellow.x
St      TON_RedYellow.IN
Ld      IN_tRedYellow
St      TON_RedYellow.PT

Cal     TON_RedYellow

Ld      TON_RedYellow.Q
St      T_RedYellow
```

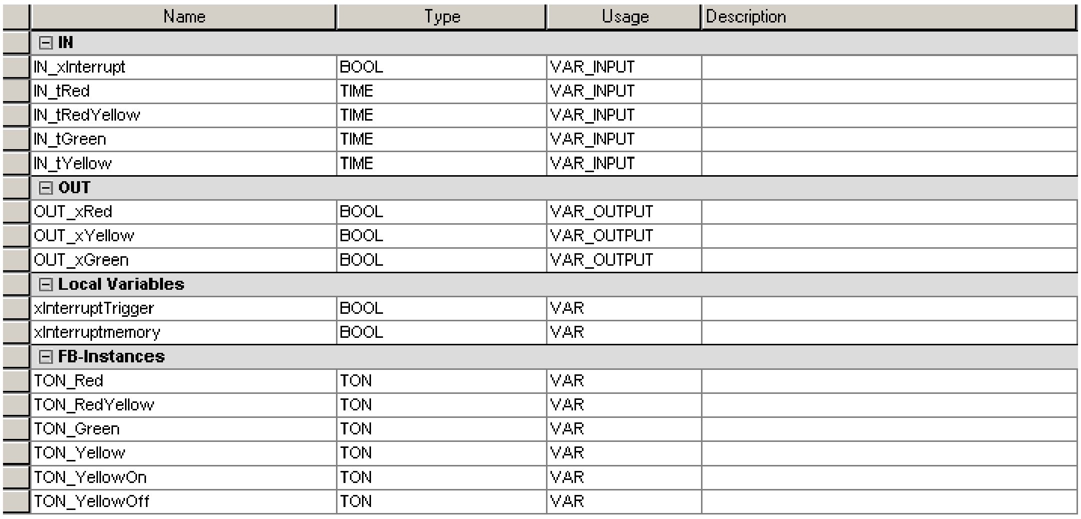




```
(* Switching time implementation green *)

TON_Green(IN    := S_Green.x,
          PT    := IN_tGreen);

T_Green         := TON_Green.Q;
```

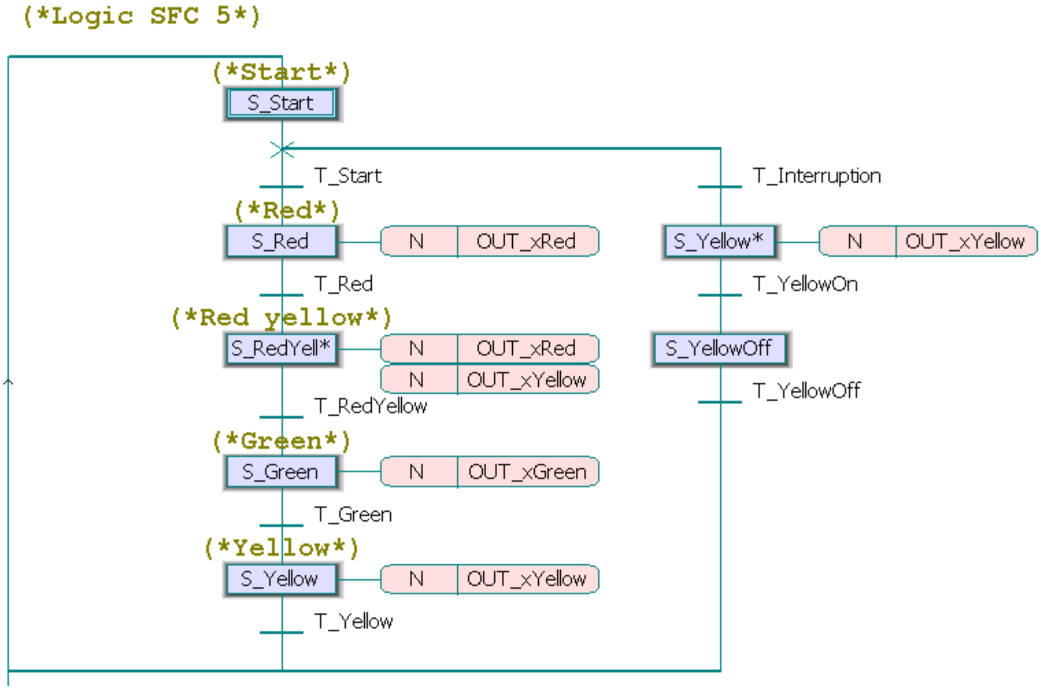
SFC[b] FB_Traffic_lights_SFC

Variables table



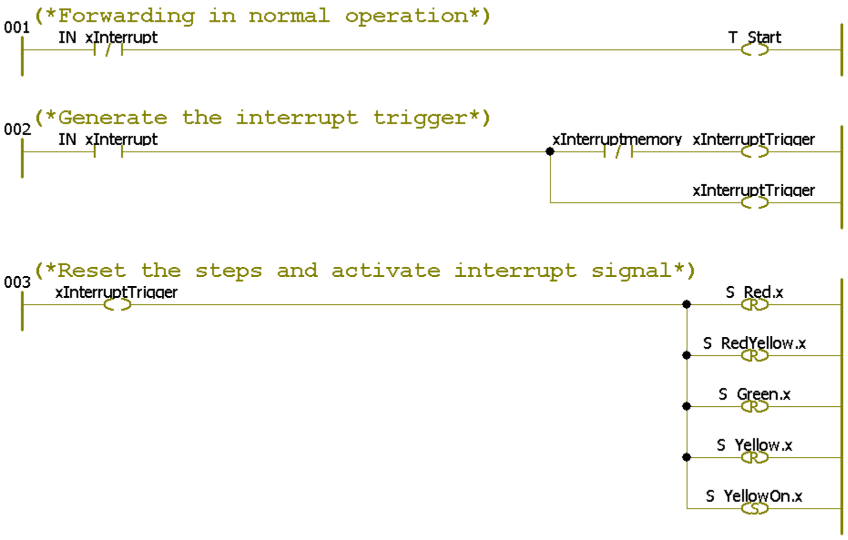
Name	Type	Usage	Description
⊟ **IN**			
IN_xInterrupt	BOOL	VAR_INPUT	
IN_tRed	TIME	VAR_INPUT	
IN_tRedYellow	TIME	VAR_INPUT	
IN_tGreen	TIME	VAR_INPUT	
IN_tYellow	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xRed	BOOL	VAR_OUTPUT	
OUT_xYellow	BOOL	VAR_OUTPUT	
OUT_xGreen	BOOL	VAR_OUTPUT	
⊟ **Local Variables**			
xInterruptTrigger	BOOL	VAR	
xInterruptmemory	BOOL	VAR	
⊟ **FB-Instances**			
TON_Red	TON	VAR	
TON_RedYellow	TON	VAR	
TON_Green	TON	VAR	
TON_Yellow	TON	VAR	
TON_YellowOn	TON	VAR	
TON_YellowOff	TON	VAR	

Programming

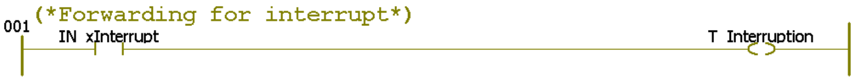


Detail programming (in addition to part [a])

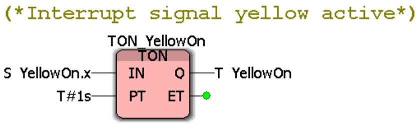
T_Start

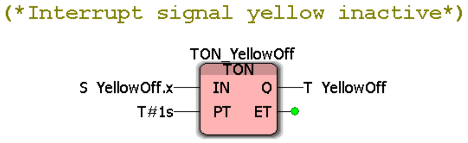


T_Interruption



T_YellowOn/YellowOff

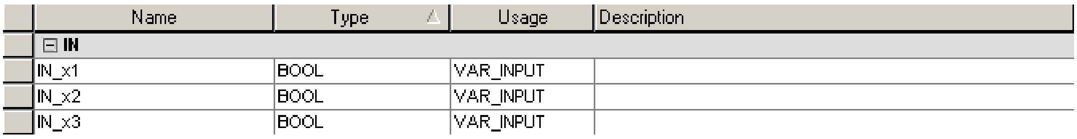




Solutions regarding programming in structured text

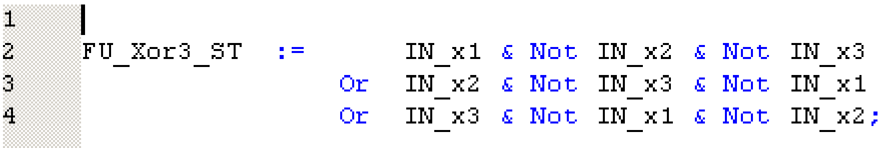
ST1 FU_Xor3_ST

Variables table



Name	Type	Usage	Description
⊟ IN			
IN_x1	BOOL	VAR_INPUT	
IN_x2	BOOL	VAR_INPUT	
IN_x3	BOOL	VAR_INPUT	

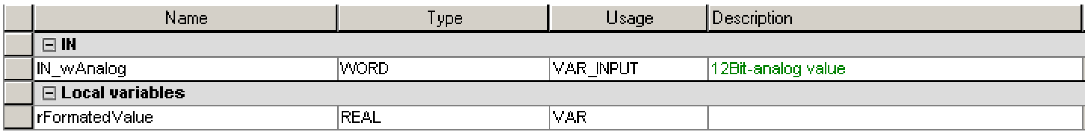
Programming



```

FU_Xor3_ST  :=      IN_x1 & Not IN_x2 & Not IN_x3
                 Or IN_x2 & Not IN_x3 & Not IN_x1
                 Or IN_x3 & Not IN_x1 & Not IN_x2;
```

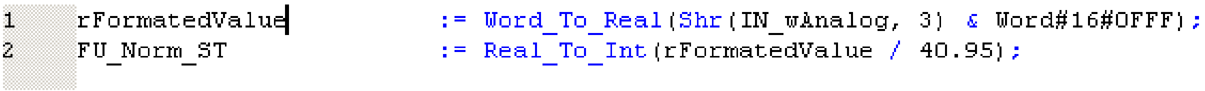
ST4[a] FU_Norm_ST

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
⊟ **Local variables**			
rFormatedValue	REAL	VAR	

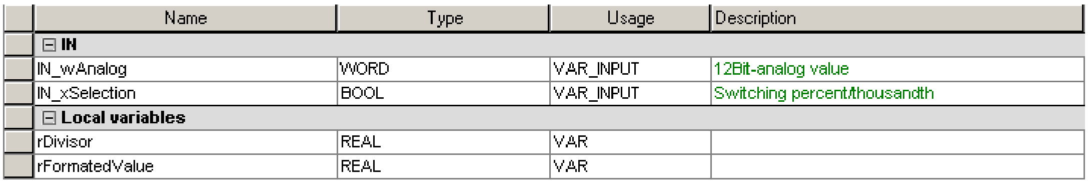
Programming



```
rFormatedValue              := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
FU_Norm_ST                  := Real_To_Int(rFormatedValue / 40.95);
```

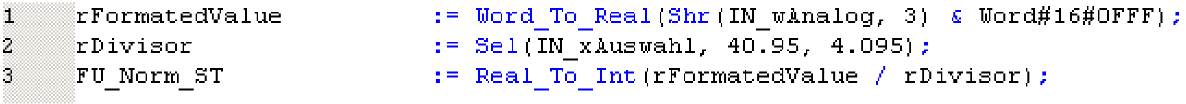
ST4[b] FU_NormSel_ST

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_xSelection	BOOL	VAR_INPUT	Switching percent/thousandth
⊟ **Local variables**			
rDivisor	REAL	VAR	
rFormatedValue	REAL	VAR	

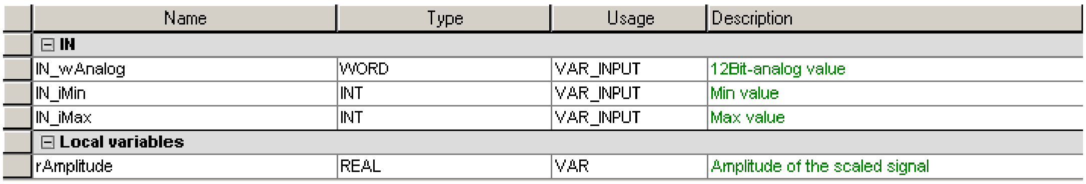
Programming



```
rFormatedValue              := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
rDivisor                    := Sel(IN_xAuswahl, 40.95, 4.095);
FU_Norm_ST                  := Real_To_Int(rFormatedValue / rDivisor);
```

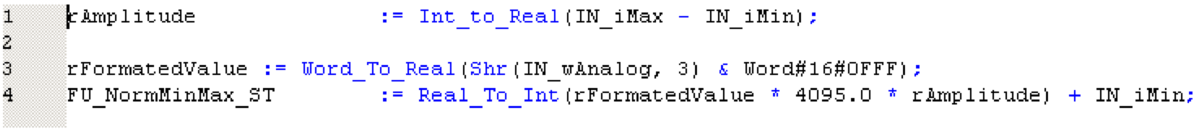
ST4[c] FU_NormMinMax_ST

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_wAnalog	WORD	VAR_INPUT	12Bit-analog value
IN_iMin	INT	VAR_INPUT	Min value
IN_iMax	INT	VAR_INPUT	Max value
⊟ **Local variables**			
rAmplitude	REAL	VAR	Amplitude of the scaled signal

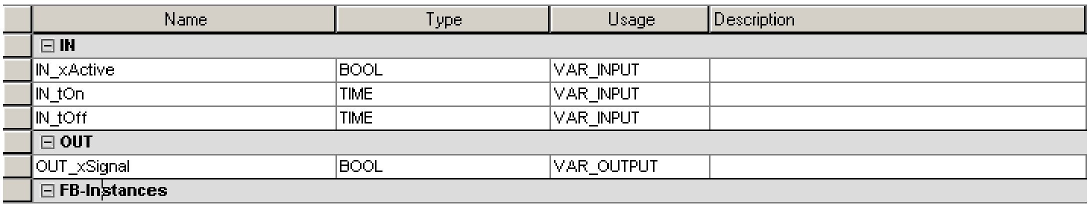
Programming



```
rAmplitude              := Int_to_Real(IN_iMax - IN_iMin);

rFormatedValue := Word_To_Real(Shr(IN_wAnalog, 3) & Word#16#0FFF);
FU_NormMinMax_ST        := Real_To_Int(rFormatedValue * 4095.0 * rAmplitude) + IN_iMin;
```

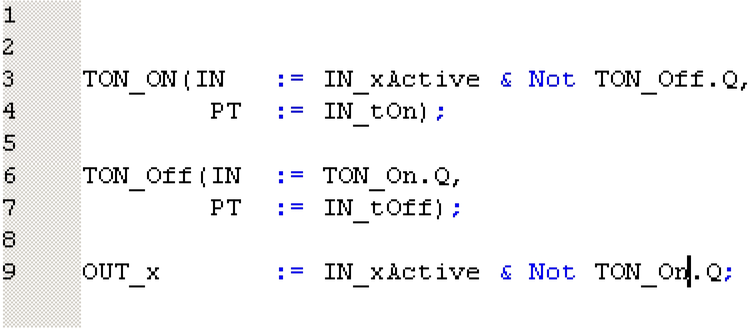
ST3 FB_Cascade_cycle_ST

Variables table



Name	Type	Usage	Description
⊟ IN			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ FB-Instances			

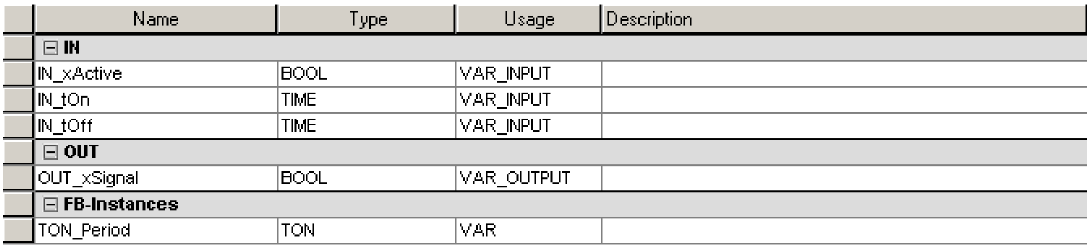
Programming



```


TON_ON(IN    := IN_xActive & Not TON_Off.Q,
        PT   := IN_tOn);

TON_Off(IN   := TON_On.Q,
        PT   := IN_tOff);

OUT_x        := IN_xActive & Not TON_On.Q;
```

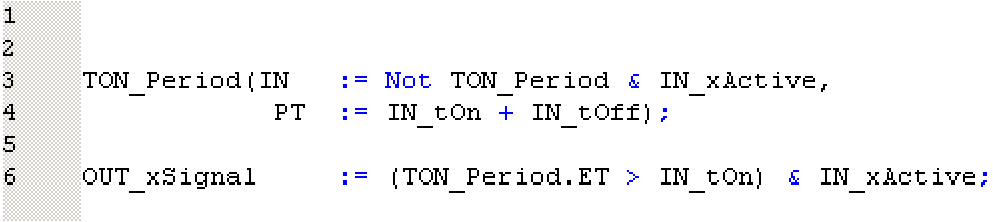
ST3 FB_Period_cycle_ST (alternative solution)

Variables table



Name	Type	Usage	Description
⊟ IN			
IN_xActive	BOOL	VAR_INPUT	
IN_tOn	TIME	VAR_INPUT	
IN_tOff	TIME	VAR_INPUT	
⊟ OUT			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ FB-Instances			
TON_Period	TON	VAR	

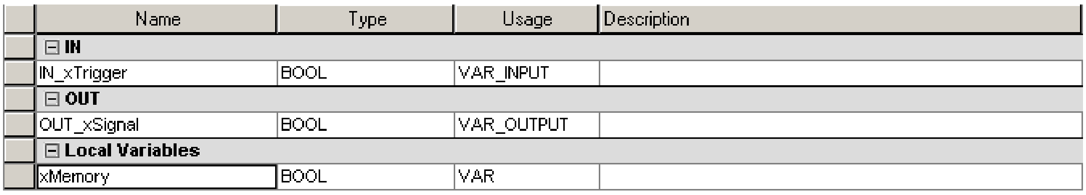
Programming



```


TON_Period(IN    := Not TON_Period & IN_xActive,
            PT   := IN_tOn + IN_tOff);

OUT_xSignal      := (TON_Period.ET > IN_tOn) & IN_xActive;
```

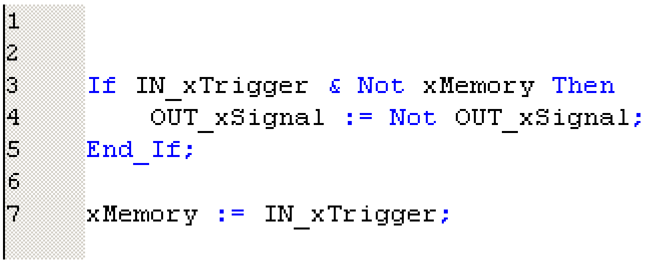
ST2 FB_TFlipflop_ST

Variables table



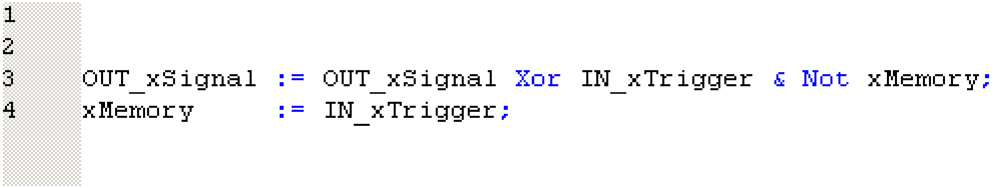
Name	Type	Usage	Description
⊟ **IN**			
IN_xTrigger	BOOL	VAR_INPUT	
⊟ **OUT**			
OUT_xSignal	BOOL	VAR_OUTPUT	
⊟ **Local Variables**			
xMemory	BOOL	VAR	

Programming



```


If IN_xTrigger & Not xMemory Then
    OUT_xSignal := Not OUT_xSignal;
End_If;

xMemory := IN_xTrigger;
```

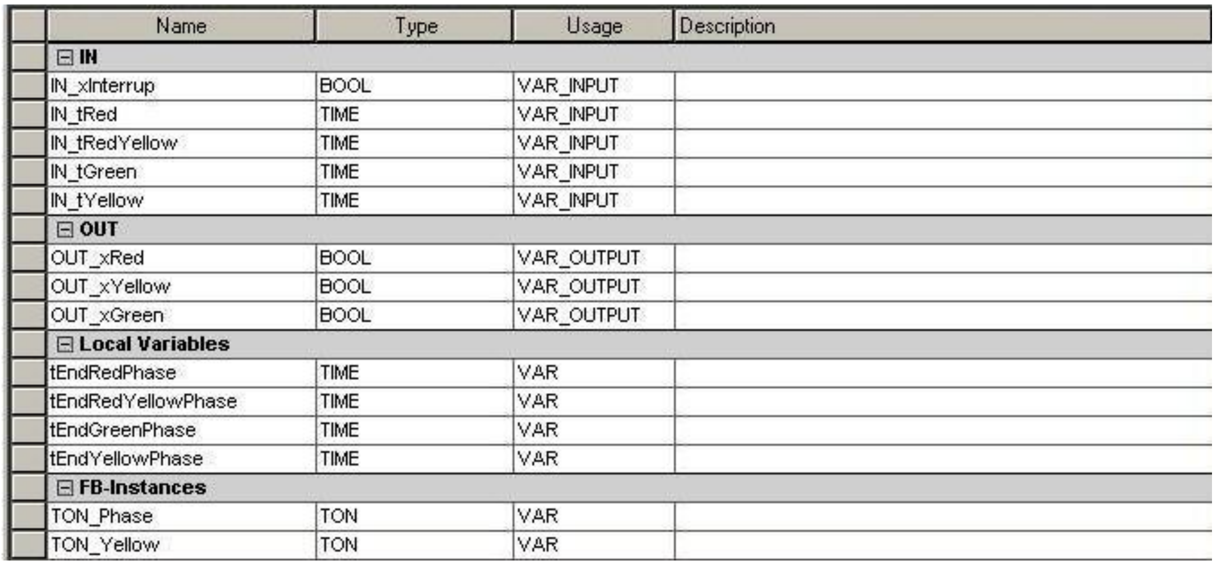
Alternative solution



```


OUT_xSignal := OUT_xSignal Xor IN_xTrigger & Not xMemory;
xMemory     := IN_xTrigger;
```

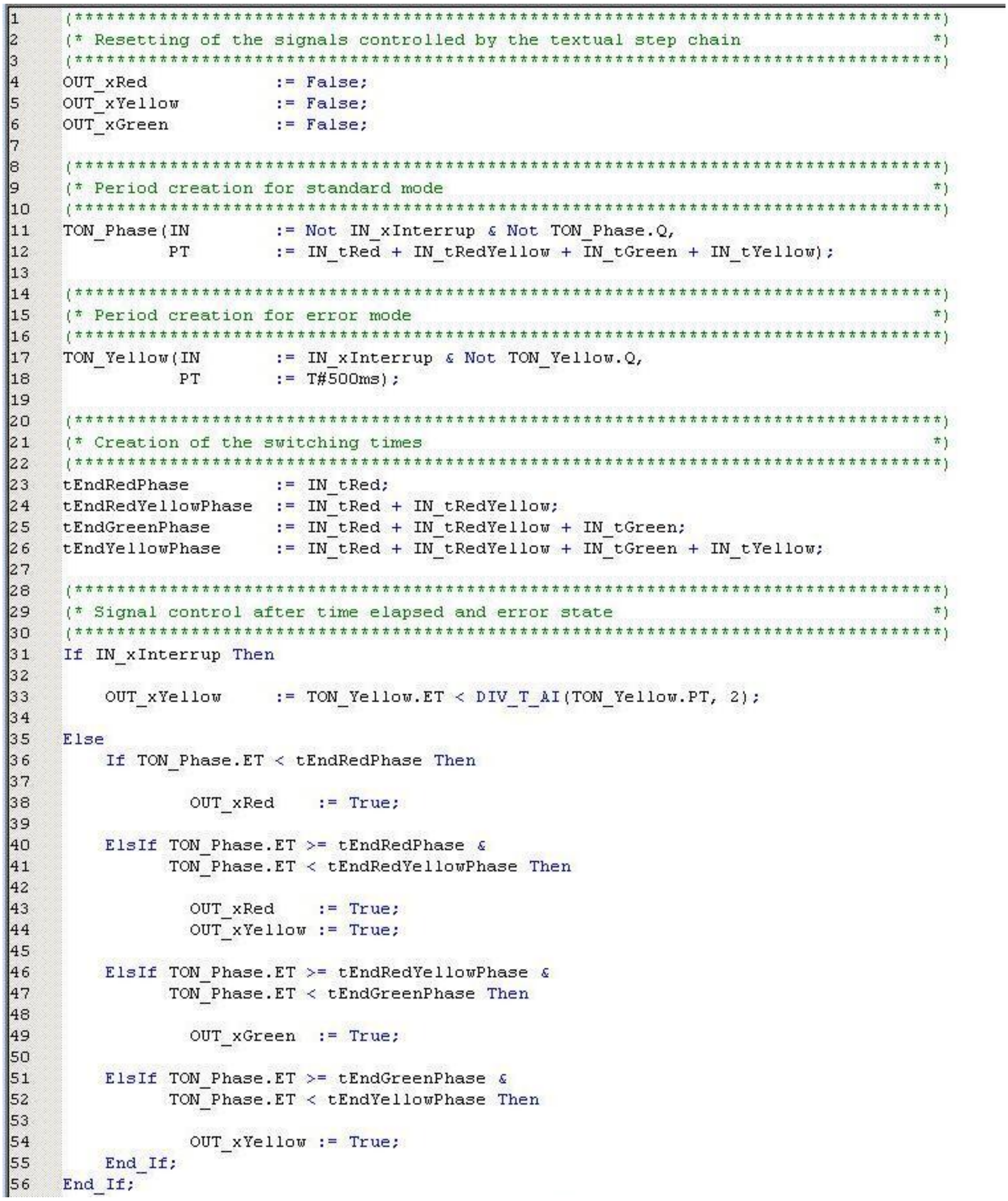
ST5 FB_Traffic_lights_ST

Variables table



Name	Type	Usage	Description
⊟ **IN**			
IN_xInterrup	BOOL	VAR_INPUT	
IN_tRed	TIME	VAR_INPUT	
IN_tRedYellow	TIME	VAR_INPUT	
IN_tGreen	TIME	VAR_INPUT	
IN_tYellow	TIME	VAR_INPUT	
⊟ **OUT**			
OUT_xRed	BOOL	VAR_OUTPUT	
OUT_xYellow	BOOL	VAR_OUTPUT	
OUT_xGreen	BOOL	VAR_OUTPUT	
⊟ **Local Variables**			
tEndRedPhase	TIME	VAR	
tEndRedYellowPhase	TIME	VAR	
tEndGreenPhase	TIME	VAR	
tEndYellowPhase	TIME	VAR	
⊟ **FB-Instances**			
TON_Phase	TON	VAR	
TON_Yellow	TON	VAR	

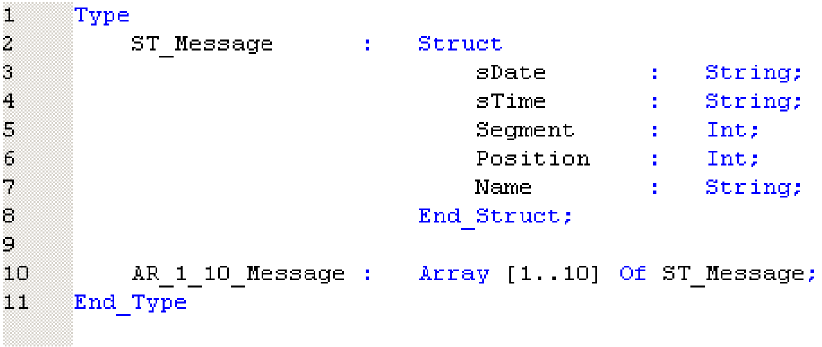
Programming



```
(***************************************************************************)
(* Resetting of the signals controlled by the textual step chain          *)
(***************************************************************************)
OUT_xRed            := False;
OUT_xYellow         := False;
OUT_xGreen          := False;

(***************************************************************************)
(* Period creation for standard mode                                       *)
(***************************************************************************)
TON_Phase(IN        := Not IN_xInterrup & Not TON_Phase.Q,
          PT        := IN_tRed + IN_tRedYellow + IN_tGreen + IN_tYellow);

(***************************************************************************)
(* Period creation for error mode                                          *)
(***************************************************************************)
TON_Yellow(IN       := IN_xInterrup & Not TON_Yellow.Q,
           PT       := T#500ms);

(***************************************************************************)
(* Creation of the switching times                                         *)
(***************************************************************************)
tEndRedPhase        := IN_tRed;
tEndRedYellowPhase  := IN_tRed + IN_tRedYellow;
tEndGreenPhase      := IN_tRed + IN_tRedYellow + IN_tGreen;
tEndYellowPhase     := IN_tRed + IN_tRedYellow + IN_tGreen + IN_tYellow;

(***************************************************************************)
(* Signal control after time elapsed and error state                       *)
(***************************************************************************)
If IN_xInterrup Then

    OUT_xYellow     := TON_Yellow.ET < DIV_T_AI(TON_Yellow.PT, 2);

Else
    If TON_Phase.ET < tEndRedPhase Then

            OUT_xRed    := True;

    ElsIf TON_Phase.ET >= tEndRedPhase &
          TON_Phase.ET < tEndRedYellowPhase Then

            OUT_xRed    := True;
            OUT_xYellow := True;

    ElsIf TON_Phase.ET >= tEndRedYellowPhase &
          TON_Phase.ET < tEndGreenPhase Then

            OUT_xGreen  := True;

    ElsIf TON_Phase.ET >= tEndGreenPhase &
          TON_Phase.ET < tEndYellowPhase Then

            OUT_xYellow := True;
    End_If;
End_If;
```

ST6[a] Data type declaration

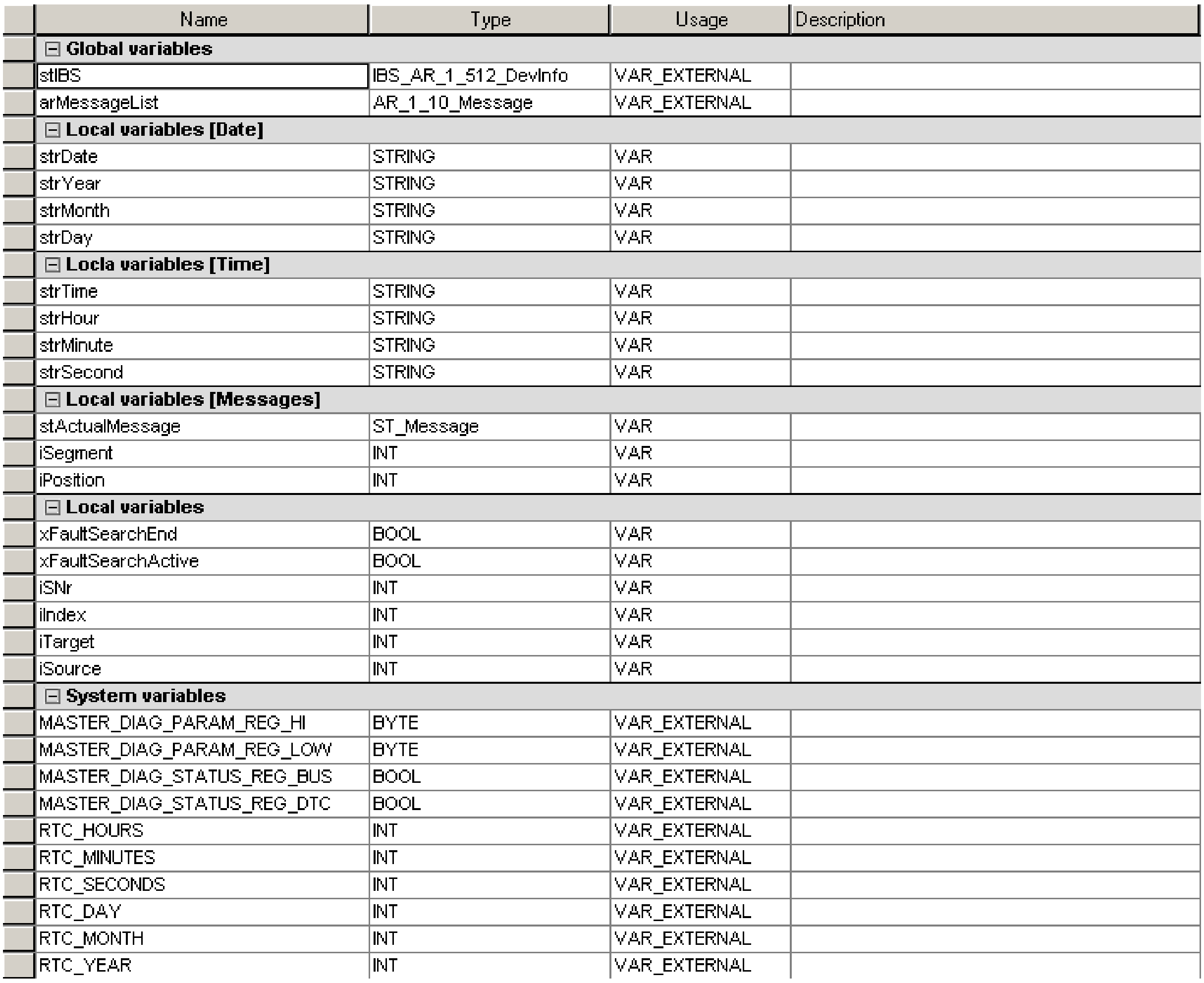
Worksheet for user-defined data types



```
Type
    ST_Message      :   Struct
                            sDate       :   String;
                            sTime       :   String;
                            Segment     :   Int;
                            Position    :   Int;
                            Name        :   String;
                        End_Struct;

    AR_1_10_Message :   Array [1..10] Of ST_Message;
End_Type
```

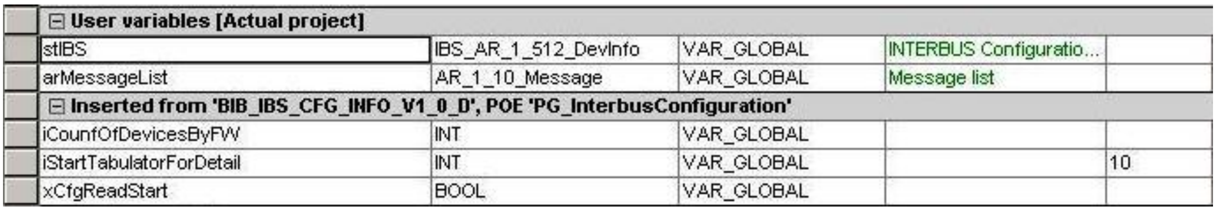
ST6[b+c] ▣ PG_Messages

Local variables table



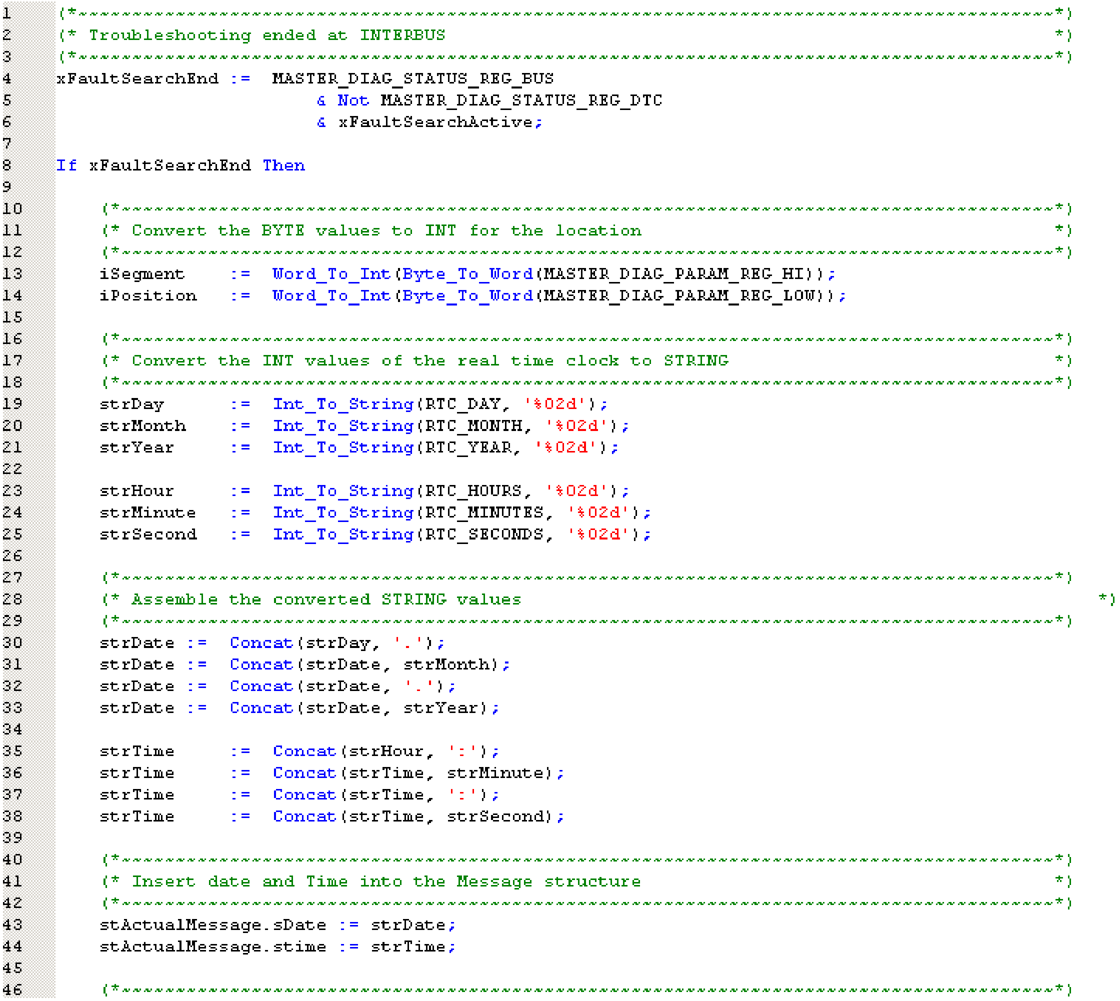
Name	Type	Usage	Description
⊟ **Global variables**			
stIBS	IBS_AR_1_512_DevInfo	VAR_EXTERNAL	
arMessageList	AR_1_10_Message	VAR_EXTERNAL	
⊟ **Local variables [Date]**			
strDate	STRING	VAR	
strYear	STRING	VAR	
strMonth	STRING	VAR	
strDay	STRING	VAR	
⊟ **Locla variables [Time]**			
strTime	STRING	VAR	
strHour	STRING	VAR	
strMinute	STRING	VAR	
strSecond	STRING	VAR	
⊟ **Local variables [Messages]**			
stActualMessage	ST_Message	VAR	
iSegment	INT	VAR	
iPosition	INT	VAR	
⊟ **Local variables**			
xFaultSearchEnd	BOOL	VAR	
xFaultSearchActive	BOOL	VAR	
iSNr	INT	VAR	
iIndex	INT	VAR	
iTarget	INT	VAR	
iSource	INT	VAR	
⊟ **System variables**			
MASTER_DIAG_PARAM_REG_HI	BYTE	VAR_EXTERNAL	
MASTER_DIAG_PARAM_REG_LOW	BYTE	VAR_EXTERNAL	
MASTER_DIAG_STATUS_REG_BUS	BOOL	VAR_EXTERNAL	
MASTER_DIAG_STATUS_REG_DTC	BOOL	VAR_EXTERNAL	
RTC_HOURS	INT	VAR_EXTERNAL	
RTC_MINUTES	INT	VAR_EXTERNAL	
RTC_SECONDS	INT	VAR_EXTERNAL	
RTC_DAY	INT	VAR_EXTERNAL	
RTC_MONTH	INT	VAR_EXTERNAL	
RTC_YEAR	INT	VAR_EXTERNAL	

Global variables table (extract)

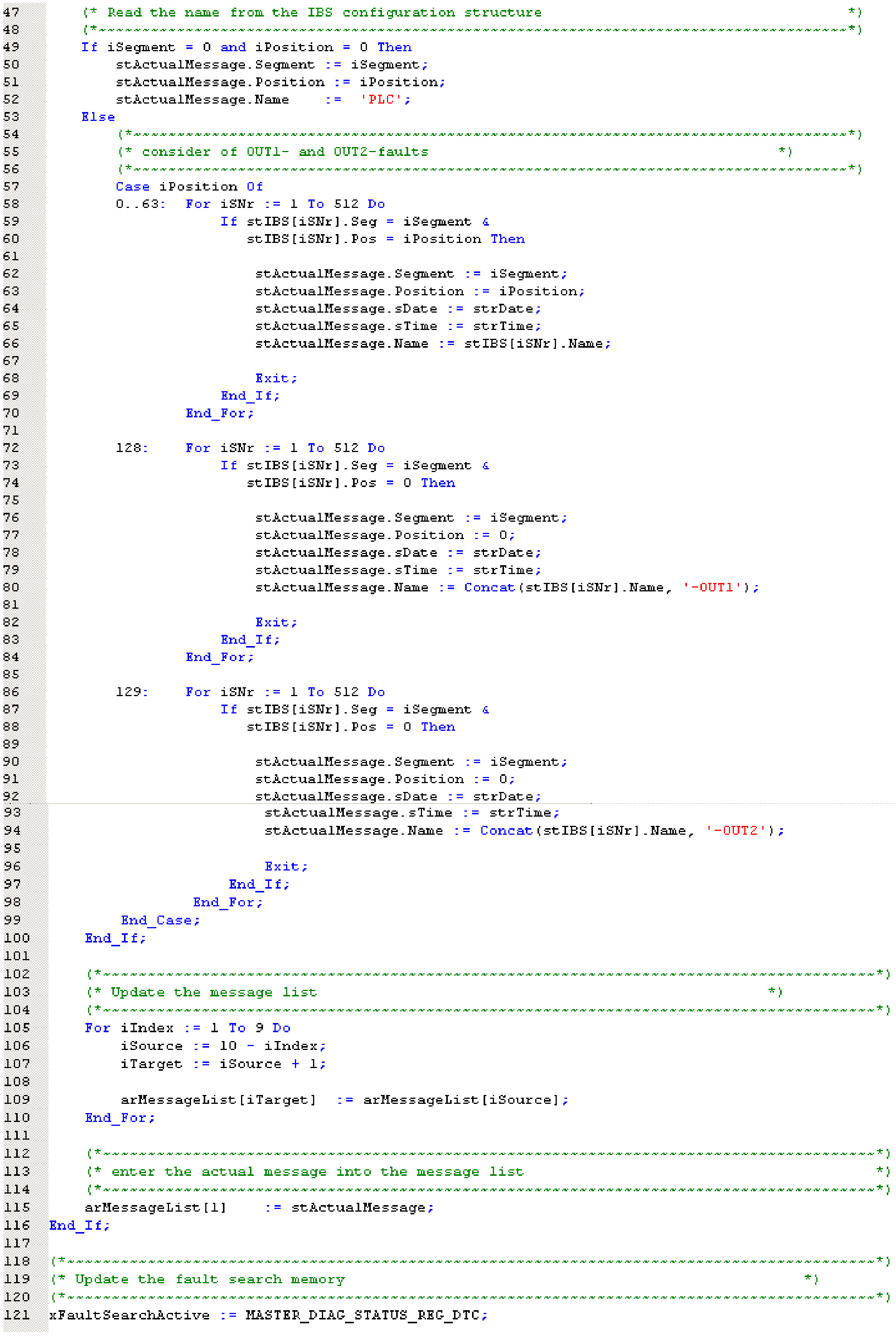


⊟ **User variables [Actual project]**				
stIBS	IBS_AR_1_512_DevInfo	VAR_GLOBAL	INTERBUS Configuratio...	
arMessageList	AR_1_10_Message	VAR_GLOBAL	Message list	
⊟ **Inserted from 'BIB_IBS_CFG_INFO_V1_0_D', POE 'PG_InterbusConfiguration'**				
iCounfOfDevicesByFW	INT	VAR_GLOBAL		
iStartTabulatorForDetail	INT	VAR_GLOBAL		10
xCfgReadStart	BOOL	VAR_GLOBAL		

Programming



```
(*******************************************************************************************)
(* Troubleshooting ended at INTERBUS                                                       *)
(*******************************************************************************************)
xFaultSearchEnd :=  MASTER_DIAG_STATUS_REG_BUS
                        & Not MASTER_DIAG_STATUS_REG_DTC
                        & xFaultSearchActive;

If xFaultSearchEnd Then

    (***************************************************************************************)
    (* Convert the BYTE values to INT for the location                                     *)
    (***************************************************************************************)
    iSegment    :=  Word_To_Int(Byte_To_Word(MASTER_DIAG_PARAM_REG_HI));
    iPosition   :=  Word_To_Int(Byte_To_Word(MASTER_DIAG_PARAM_REG_LOW));

    (***************************************************************************************)
    (* Convert the INT values of the real time clock to STRING                             *)
    (***************************************************************************************)
    strDay      :=  Int_To_String(RTC_DAY, '%02d');
    strMonth    :=  Int_To_String(RTC_MONTH, '%02d');
    strYear     :=  Int_To_String(RTC_YEAR, '%02d');

    strHour     :=  Int_To_String(RTC_HOURS, '%02d');
    strMinute   :=  Int_To_String(RTC_MINUTES, '%02d');
    strSecond   :=  Int_To_String(RTC_SECONDS, '%02d');

    (***************************************************************************************)
    (* Assemble the converted STRING values                                                     *)
    (***************************************************************************************)
    strDate := Concat(strDay, '.');
    strDate := Concat(strDate, strMonth);
    strDate := Concat(strDate, '.');
    strDate := Concat(strDate, strYear);

    strTime     :=  Concat(strHour, ':');
    strTime     :=  Concat(strTime, strMinute);
    strTime     :=  Concat(strTime, ':');
    strTime     :=  Concat(strTime, strSecond);

    (***************************************************************************************)
    (* Insert date and Time into the Message structure                                     *)
    (***************************************************************************************)
    stActualMessage.sDate := strDate;
    stActualMessage.stime := strTime;

    (***************************************************************************************)
```



```
    (* Read the name from the IBS configuration structure                                  *)
    (*************************************************************************************)
    If iSegment = 0 and iPosition = 0 Then
        stActualMessage.Segment := iSegment;
        stActualMessage.Position := iPosition;
        stActualMessage.Name    :=  'PLC';
    Else
        (*********************************************************************************)
        (* consider of OUT1- and OUT2-faults                                     *)
        (*********************************************************************************)
        Case iPosition Of
        0..63:  For iSNr := 1 To 512 Do
                    If stIBS[iSNr].Seg = iSegment &
                       stIBS[iSNr].Pos = iPosition Then

                        stActualMessage.Segment := iSegment;
                        stActualMessage.Position := iPosition;
                        stActualMessage.sDate := strDate;
                        stActualMessage.sTime := strTime;
                        stActualMessage.Name := stIBS[iSNr].Name;

                        Exit;
                    End_If;
                End_For;

        128:    For iSNr := 1 To 512 Do
                    If stIBS[iSNr].Seg = iSegment &
                       stIBS[iSNr].Pos = 0 Then

                        stActualMessage.Segment := iSegment;
                        stActualMessage.Position := 0;
                        stActualMessage.sDate := strDate;
                        stActualMessage.sTime := strTime;
                        stActualMessage.Name := Concat(stIBS[iSNr].Name, '-OUT1');

                        Exit;
                    End_If;
                End_For;

        129:    For iSNr := 1 To 512 Do
                    If stIBS[iSNr].Seg = iSegment &
                       stIBS[iSNr].Pos = 0 Then

                        stActualMessage.Segment := iSegment;
                        stActualMessage.Position := 0;
                        stActualMessage.sDate := strDate;
                        stActualMessage.sTime := strTime;
                        stActualMessage.Name := Concat(stIBS[iSNr].Name, '-OUT2');

                        Exit;
                    End_If;
                End_For;
        End_Case;
    End_If;

    (*************************************************************************************)
    (* Update the message list                                                     *)
    (*************************************************************************************)
    For iIndex := 1 To 9 Do
        iSource := 10 - iIndex;
        iTarget := iSource + 1;

        arMessageList[iTarget]  := arMessageList[iSource];
    End_For;

    (*************************************************************************************)
    (* enter the actual message into the message list                                     *)
    (*************************************************************************************)
    arMessageList[1]    := stActualMessage;
End_If;

(*****************************************************************************************)
(* Update the fault search memory                                                   *)
(*****************************************************************************************)
xFaultSearchActive := MASTER_DIAG_STATUS_REG_DTC;
```