SIEMENS

SCE Training Curriculum

Siemens Automation Cooperates with Education (SCE) | 09/2015

PA Module P01-04

SIMATIC PCS 7 – Individual Drive Functions

Cooperates with Education

Automation

SIEMENS

Matching SCE Trainer Packages for these curriculum

- **SIMATIC PCS 7 Software block of 3 packages**
  Order No. 6ES7650-0XX18-0YS5
- **SIMATIC PCS 7 Software block of 6 packages**
  Order No. 6ES7650-0XX18-2YS5
- **SIMATIC PCS 7 Software Upgrade block of 3 packages**
  Order No. 6ES7650-0XX18-0YE5 (V8.0 → V8.1) or 6ES7650-0XX08-0YE5 (V7.1 → V8.0)
- **SIMATIC PCS 7 Hardware Set including RTX Box**
  Order No. 6ES7654-0UE13-0XS0

Please note that these trainer packages may be replaced with subsequent packages.
An overview of the available SCE packages is provided at: siemens.com/sce/tp

Continuing education

For regional Siemens SCE continuing education, contact your regional SCE contact partner.
siemens.com/sce/contact

Additional information relating to SIMATIC PCS 7 and SIMIT

In particular, Getting Started, videos, tutorials, manuals and programming guide.
siemens.com/sce/pcs7

Additional information relating to SCE

siemens.com/sce

Note on Usage

The training curriculum for the integrated automation solution Totally Integrated Automation (TIA) was prepared for the program "Siemens Automation Cooperates with Education (SCE)“ specifically for training purposes at public educational and R&D facilities. Siemens AG is not liable for the contents.

This document may only be used for initial training on Siemens products/systems. This means it may be copied entirely or partially and handed to trainees for use within the scope of their training. Passing on or copying this document and communicating its contents is permitted within public training and continuing education facilities for training purposes.

Exceptions require written permission by Siemens AG. Contact person: Roland Scheuerer roland.scheuerer@siemens.com.

Violators are subject to damages. All rights including translation rights are reserved, particularly in the event a patent is granted or a utility model or design is registered.

Usage for industrial customer courses is explicitly not permitted. We do not agree to the commercial utilization of these documents.

We would like to thank the Technical University Dresden, particularly Prof. Dr. Leon Urbas and Annett Krause, MS, as well as the Michael Dziallas Engineering Corporation and those who provided support in preparing this SCE training document.

INDIVIDUAL DRIVE FUNCTIONS

TRAINING OBJECTIVE

After working through this module, the students will be able to define and classify the term 'individual drive function' within the scope of object-oriented software structuring. They understand the concept, the structure as well as the functional method of individual drive functions; they know typical individual drive functions and their implementation in ***PCS 7***.

THEORY IN BRIEF

The objective of object-oriented software structuring is to simulate the structure of the real plant as clearly as possible by modularizing the user software accordingly. To this end, at least one function block is provided for each field device type. This function block in turn provides the entire control logic, the necessary protection and monitoring functions as well as suitable operator control and visualization options. The user program utilizes this block to implement the desired operating behavior of a machine or a process.

Motors and valves are control engineering equipment that is not controlled directly, in the sense of object-oriented automation, but is initially modeled as function block types. Such function block types are called ***Individual Drive Functions (IDF)***. They enable control, monitoring and operation of the control engineering equipment by providing corresponding connections for actuating and control signals as well as for parameter assignment and monitoring functions. The technical implementation of the control is achieved through an instance of the function block type and is hidden from the user. Figure 1 shows the transition from the real motor–in this case a pump–to a block of the corresponding individual drive function.

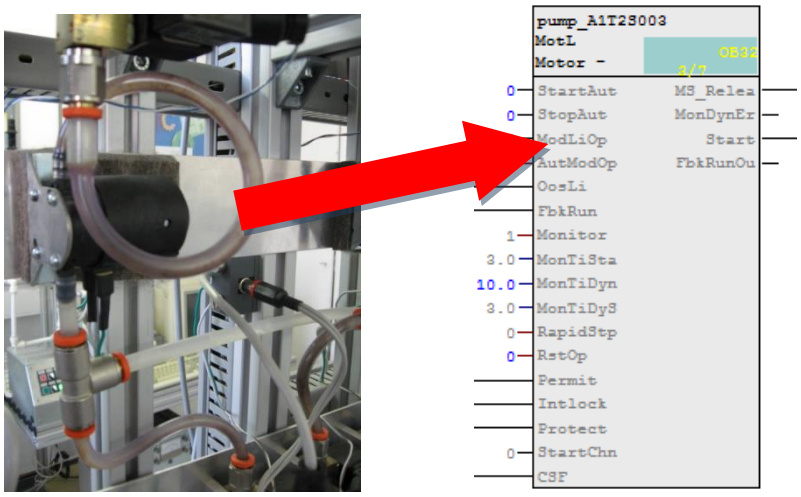


Figure 1: Transition from real motor to blocks of the individual drive function

In principle, control engineering equipment can be operated in four different operating modes. The device is in one of the following modes:

– ***Shut down***

– ***Manual mode***

– ***Automatic mode***

– ***Local mode***

The individual drive function must always be in exactly only one operating mode. The operating modes mentioned can either be equivalent or arranged hierarchically using priorities. In addition, individual drive functions provide functions that protect against device and process faults. To this end, different interlocks as well as execution time monitoring for the device and for the controlled process are implemented.

Function blocks, referred to as block types in ***PCS 7***, represent pre-assembled program parts for processing recurring functions. They can be inserted in CFCs where they can be parameterized as instances, interconnected, and adapted to the project requirements. The block type in this case specifies the characteristic for all instances of this type. In control engineering libraries, ***PCS 7*** offers a variety of high performance and tested individual drive functions as block types. They each model a control engineering device and make the entire control logic available. In addition, functions are offered for the following:

– ***Operator control*** and ***Monitoring*** of the individual drive function

– ***Controlling*** signals

– ***Monitoring*** and ***Alarming***

– ***Operating State Selection***

– ***Interlocks***.

Faceplates with different ***views*** allow for seamless integration into a corresponding process control system.

Individual drive functions enable the efficient development of high performance, high quality solutions. They modularize and type-define recurrent functionalities. This means functionalities can be reused and centrally modified which speeds up the development process considerably.

THEORY

OBJECT-ORIENTED SOFTWARE STRUCTURING

The objective of object-oriented software structuring is to simulate the structure of the real plant through corresponding modularization of the user software as clearly as possible. To this end, a separate program is created for each field device in the plant. At least one function block is provided for each field device type.

This block implements the entire control logic for this field device type. In addition, it makes necessary protection and monitoring functions available as well as suitable control and visualization options. This means it encapsulates the entire functionality that is necessary in connection with the corresponding field device type. The user program utilizes this block to implement a desired control for a machine or for a process without having to access the knowledge of the internal data and operations of the function block.

CHANNEL FUNCTIONS (DRIVERS)

In addition to handling the field devices through separate reusable blocks, it is often advisable to abstract IO interfacing also by using ***channel blocks*** (***drivers***). Although it is always possible to access the process image directly by using symbol names or addresses, the multitude of possible parameters for configuring the channel have to be set at a different location. This quickly results in confusing programs. ***PCS 7*** provides a number of drivers (channel blocks) that evaluate the status signals of the modules and support testing and commissioning of automation programs through simulation modes. In the analog drivers according to Table 1, internal digital variables are mapped to the physical addresses and display variables by means of the parameters VLRANGE and VHRANGE. ***PCS 7*** can generate the necessary drivers automatically by using channel blocks. Channel blocks are therefore often used in the templates of the ***PCS 7*** libraries.

Table 1: Listing of different channel blocks to abstract IO interfacing

Channel blocks	Block	Connector, Parameter
Digital output	PCS7DiOu	PV_Out
Digital input	PCS7DiIn	PV_In
Analog output	PCS7AnOu	PV_Out, Scale
Analog input	PCS7AnIn	PV_In, Scale

INDIVIDUAL DRIVE FUNCTIONS

As control engineering devices, motors and valves are of crucial importance in factory and process automation. A large number of commercial types with specific operational and signaling behavior are available. In the sense of object-oriented automation, such devices are not controlled directly but initially modeled as function block types. They are then always controlled indirectly by an instance of the corresponding function block type. Function blocks for motors and valves are called ***Individual Drive Functions (IDF)***. Individual drive functions enable control, monitoring and operation of control engineering devices by providing corresponding connections for actuating and control signals as well as for parameter assignment and monitoring functions. The technical implementation of the control, such as starting performance, activating the drive, or device monitoring, for example, is implemented through the function block instance and is hidden from the user. ***PCS 7*** provides a variety of efficient and tested individual drive functions as block types in the control engineering libraries. Table 2 summarizes the individual drive functions in the ***PCS 7 Advanced Process Library*** [2].

Table 2: Individual drive functions of the ***PCS 7 Advanced Process Library***

Individual drive function	Usage	Object name
MotL	Control of motors by means of a control signal (on/off) and a feedback signal	FB 1850
MotRevL	Control of reversible motors (clockwise/counterclockwise) and up to two feedback signals	FB 1851
MotSpedL	Control of two-speed motors (slow/rapid) and up to two feedback signals	FB 1856
VlvL	Operation of control valves with one control signal (open/close) and two position feedback signals (open/closed)	FB 1899
VlvMotL	Control of motor-driven valves with two control signals and two position feedback signals (open/closed)	FB 1900

PROTECTIVE MEASURES

When control engineering devices are activated, different protective measures have to be taken. The devices themselves have to be protected from faults, and the controlled process has to be taken to a safe state in case of a malfunction and must be maintained in this state until the fault is rectified.

Device faults (for example, cable break, and axis break) cannot be prevented by the control engineering side, but the effects can be minimized through redundancy concepts. ***Process faults*** (for example, container overflow, dry run of a pump), however, are to be prevented directly through the control. To this end, corresponding ***interlocks*** are implemented. If the individual drive function detects a dangerous process state based on the current input values, the controlled device is taken to a safe state (refer to the chapter Functional Safety). The device is kept in that state for the duration of the dangerous process state. Usually, interlocks are specified using an ***interlock matrix***.

To detect a device error that occurred, the individual drive function often performs an ***execution time check***. By using certain sensor information from limit sensors in valves, for example, the individual drive function checks whether the actuating signals that were read out have the required effect. If over a certain period of time the measured values contradict the actuating signals that were read out, there is a fault. If such a runtime error is detected, the higher level control system is alerted and the controlled device is deactivated. The device remains inactive until the runtime error is removed and the alarm was acknowledged. Simple, binary circuit breakers are often used to detect device faults.

OPERATING MODES

Control engineering devices are generally not operated exclusively automatically. From time to time it is necessary handle the control manually from the control desk, or to activate the device directly on site; for example, when repairs have to be made. For this reason, we distinguish between four basic operating modes

– ***Shut down***: The device is not active.

– ***Automatic mode***: The individual drive function is activated automatically by a higher level program.

– ***Manual mode***: The operator activates the individual drive function directly by means of a graphical user interface of the control system.

– ***Local mode***: The operator operates the device on site; for example, by using the operator panel.

The individual drive function must always be in exactly one operating mode. There are different concepts as to how switching the operating mode connected with this requirement can be realized safely and unambiguously. Basically, these concepts can be differentiated as the equality of the operating modes and an operating mode hierarchy. In the latter case, the possible operating modes are clearly prioritized in addition. A selected operating mode is changed in this case exactly when the device is not active (operating mode ***Shut down***) or when the requested new operating mode has a higher priority than the one that was selected previously.

FUNCTION BLOCK TYPES IN PCS 7

Function block types are referred to as block types In ***PCS 7*** and represent pre-assembled program parts for processing recurring functions. They can be inserted in CFCs where they can be parameterized, interconnected, and adapted to the project requirements.

The block type specifies the characteristic for all instances of this type. To this end, the block types used for a project are stored in a master data library. When the block type stored there is changed, the changes are accepted directly by all instances. This concept of type definition supports efficient engineering through reusability and central changeability of functions that recur frequently.

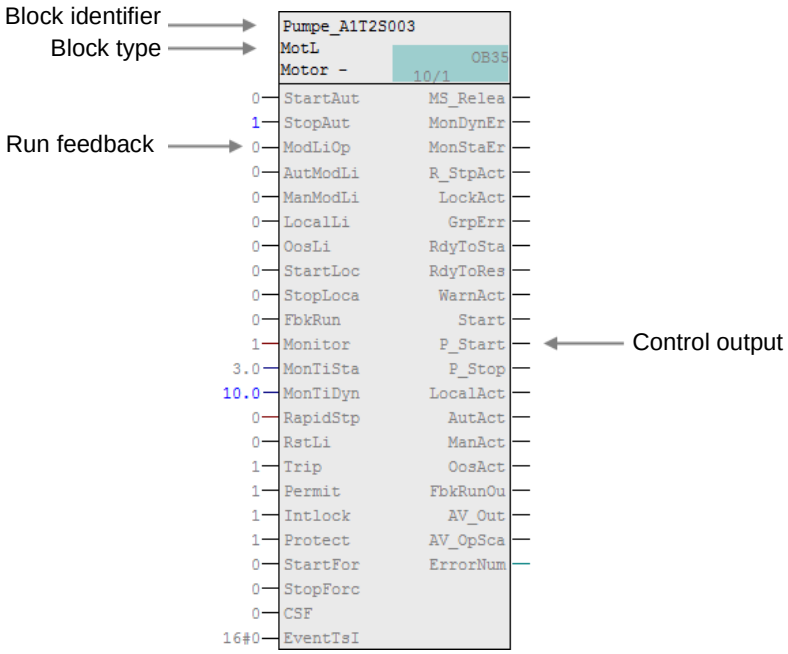


Figure 2: Block of the individual drive function MotL

An individual drive function in ***PCS 7*** models a control engineering device and provides the entire control logic. Figure 2 describes the basic structure of the corresponding motor block using the individual drive function ***Motor_Lean*** as an example.

In addition, the block provides the following functions:

Operating, Monitoring, Signaling

Process values and setpoints can be operated and monitored by means of the display and control area. Operator authorizations and maintenance releases can be controlled. General and instance-specific messages provide information about the device and process status.

Controlling Signals

Control signals can be read out in the static or the pulsed mode. The signal status, which means the quality of the actuating signal, is monitored. Internal and external setpoints as well as simulated values can be specified. In addition, ramps or dead zones can be set.

Monitoring

The block can monitor limits and generate corresponding warnings or alarms if the limits are violated. In addition, feedback from actuating signals can be monitored.

Interlock Functions

The block enables a simple switch-on release, interlock without reset as well as interlock with reset. It implements a motor protection function that can switch off the motor in case of thermal overload. In addition, a quick stop is available for motors; it has the highest priority in all operating modes and states. If there is an interlock, the device is automatically taken to a de-energized state and thus to a defined safety position.

Operating State Selection

The operating modes mentioned above: local mode, automatic mode, manual mode and shutdown are available for all individual drive functions in ***PCS 7***. They are prioritized in the descending mode in the sequence mentioned. Automatic and manual mode have the same priority. In addition, it is possible to take the block into another operating mode by means of configurable input parameters, regardless of the currently pending control (***forcing*** of operating modes).

Display Blocks with Different Views

Display blocks provide for each block type a corresponding block symbol and, depending on the use case, corresponding views. Typical display blocks are, for example, the block symbol itself, the parameter view of motors and valves, or the limit view of motors.

This list clearly shows the complexity and the functional scope of a customary individual drive function. The number of available inputs and outputs is correspondingly large regarding these blocks. For example, the individual drive function ***MotL*** has more than 53 connections. To keep the complexity of the program design low nevertheless, it is possible to hide inputs or outputs that are not needed. Moreover, the individual drive functions in ***PCS 7*** use a uniform and integrated scheme for designating the inputs and outputs.

The individual drive functions in ***PCS 7*** provide a large functional scope and guarantee constantly high quality and reliability of the algorithms used. All block types are tested extensively and have proven themselves industrially. This considerably reduces the effort for developing efficient high quality solutions.

LITERATURE

[1] Seitz, M. (2008): Speicherprogrammierbare Steuerungen (PLCs) Hanser Fachbuchverlag.

[2] SIEMENS (2014): Process Control System PCS 7: Advanced Process Library (V8.1). A5E33257529-AA. (http://support.automation.siemens.com/WW/view/en/90682917)

STEP BY STEP INSTRUCTIONS

TASK

As the first program, we are creating in the ***Continuous Function Chart (CFC)*** a pump motor for draining the liquid from Reactor R001. The pump motor has an output for activating the pump and an indication to check whether the pump is actually running.

Table 3: Assignment list

Symbol	Address	Data Type	Symbol Comment
A1.T2.A1T2S003.SO+.O+	I 1.3	BOOL	Pump outlet reactor R001 Feedback on
A1.T2.A1T2S003.SV.C	Q 3.4	BOOL	Pump outlet reactor R001 Actuator signal

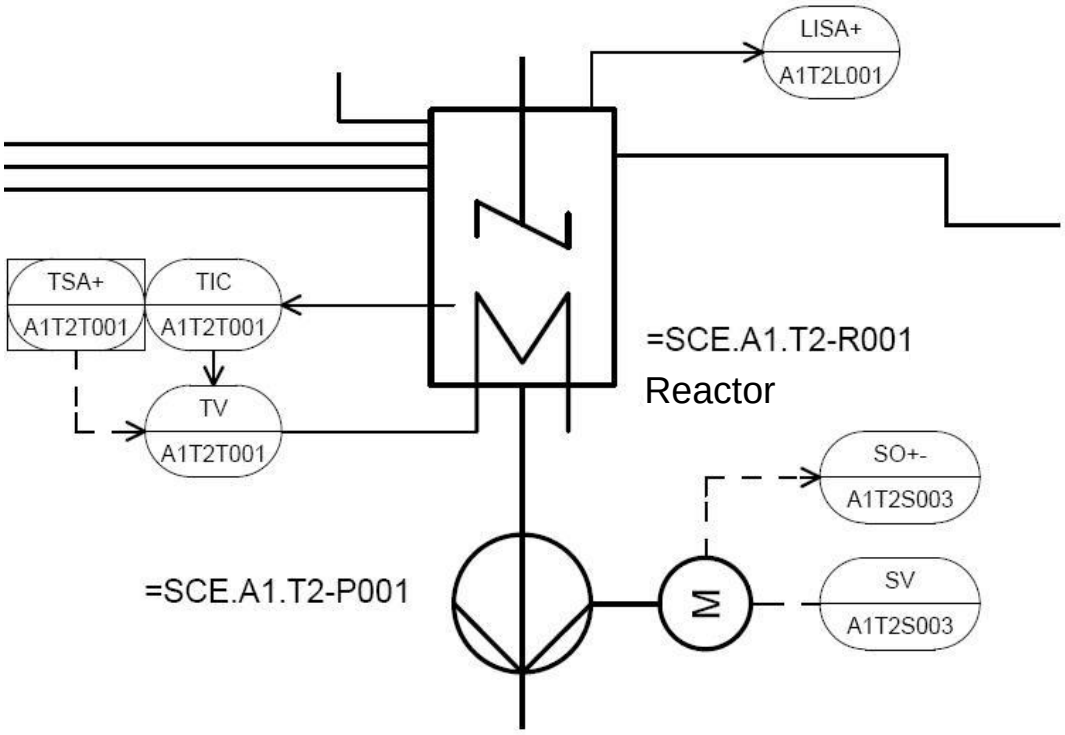


Figure 3: Excerpt from P&ID flowchart

When creating the program, a pre-assembled chart 'Motor_Lean' from a ***PCS 7*** library is used. It is copied to the master data library belonging to the project and adapted there. Then the program is loaded to the PLC simulation and tested.

TRAINING OBJECTIVE

In this chapter, the student will learn the following:

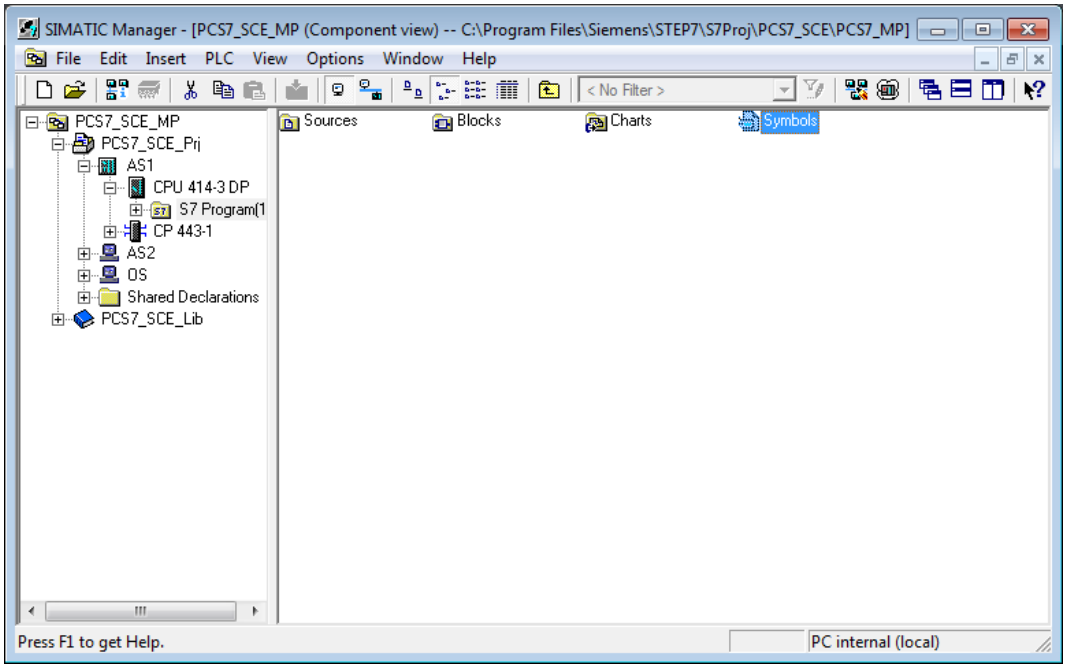
– Creating and importing symbols with the symbol table
– Using master data libraries
– Creating and editing CFCs
– Compiling and downloading the project centrally
– Testing the program by means of the control functions in the CFC

These instructions are based on project 'PCS7_SCE_0103_Ueb_R1503_en.zip'.

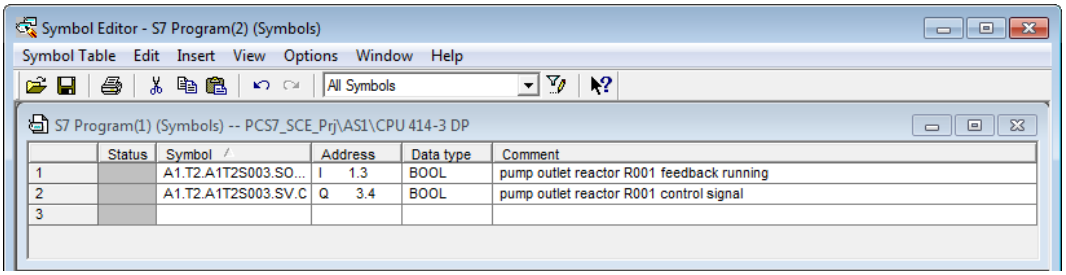
PROGRAMMING

1. Before starting with programming the individual drive functions for the pump motor, we have to create the symbols for the global variables. To this end, we select the Component view in SIMATIC Manager, highlight the folder 'S7-Program(1)' and open the symbols of the symbol table with a double click.

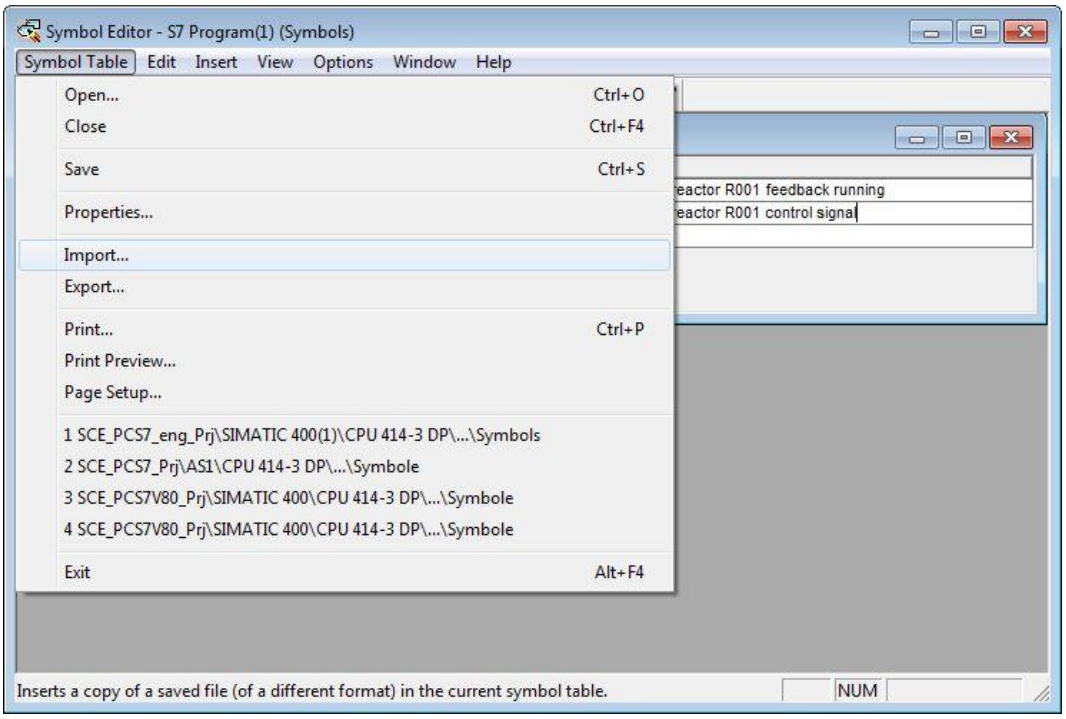
   (→ SIMATIC Manager → View→ Component view→ AS1 → CPU 414-3 DP → S7-Program(1) → Symbols)



2. We now can specify the symbol and the comment for each address in the symbol table.

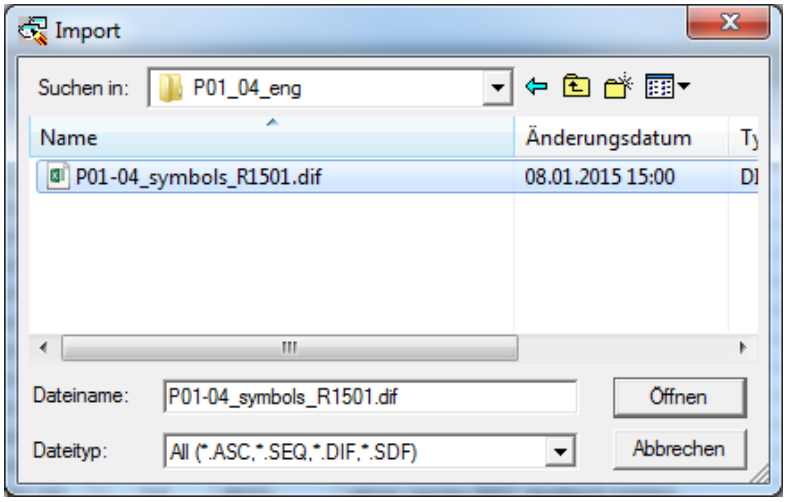


3. If available, the content of the entire symbol table can be imported in *.dif format. (→ Symbol Table → Import). In this case, the imported table is integrated in the existing table.



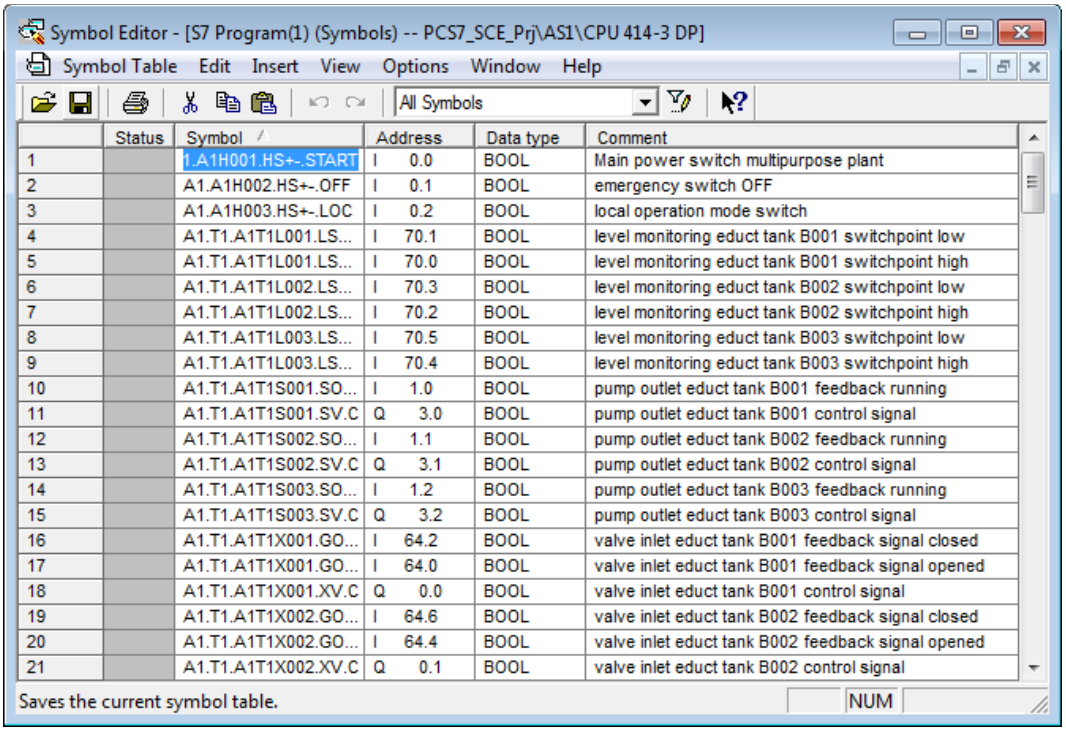
4. Now we select the source file in the 'Data Interchange Format' (*.DIF)

   (→ P01-04_symbols_R1501_en.dif → Open)



5. Before closing, the completed symbol table has to be saved.

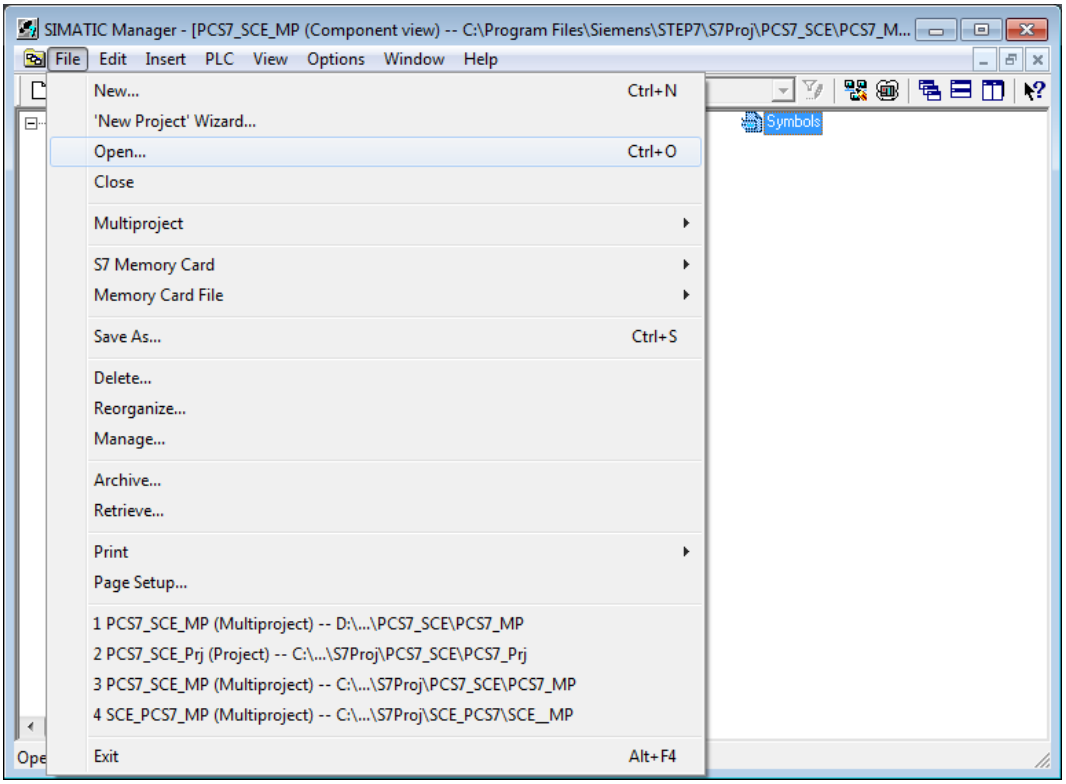
   (→ Save → )

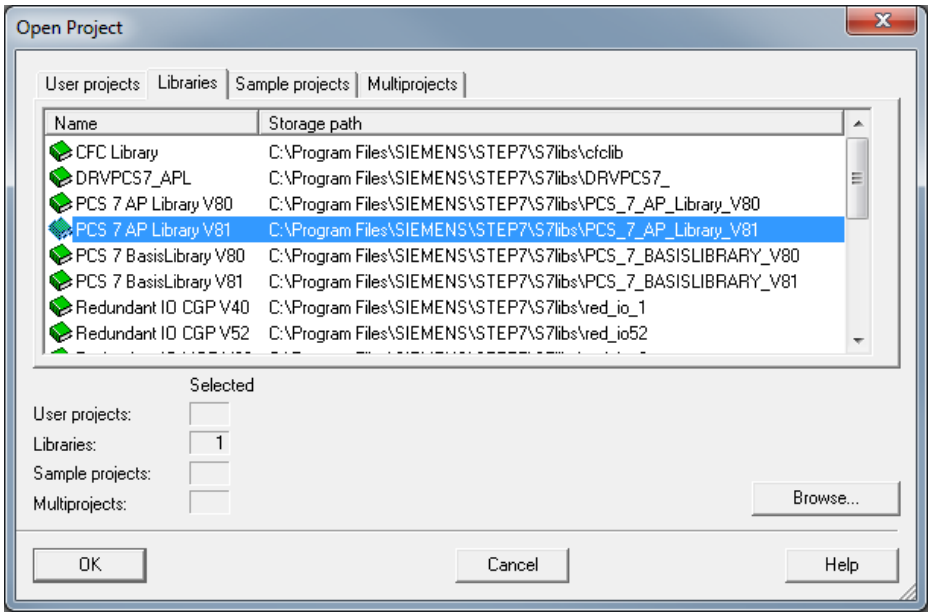


	Status	Symbol	Address	Data type	Comment
1		1.A1H001.HS+-.START	I 0.0	BOOL	Main power switch multipurpose plant
2		A1.A1H002.HS+-.OFF	I 0.1	BOOL	emergency switch OFF
3		A1.A1H003.HS+-.LOC	I 0.2	BOOL	local operation mode switch
4		A1.T1.A1T1L001.LS...	I 70.1	BOOL	level monitoring educt tank B001 switchpoint low
5		A1.T1.A1T1L001.LS...	I 70.0	BOOL	level monitoring educt tank B001 switchpoint high
6		A1.T1.A1T1L002.LS...	I 70.3	BOOL	level monitoring educt tank B002 switchpoint low
7		A1.T1.A1T1L002.LS...	I 70.2	BOOL	level monitoring educt tank B002 switchpoint high
8		A1.T1.A1T1L003.LS...	I 70.5	BOOL	level monitoring educt tank B003 switchpoint low
9		A1.T1.A1T1L003.LS...	I 70.4	BOOL	level monitoring educt tank B003 switchpoint high
10		A1.T1.A1T1S001.SO...	I 1.0	BOOL	pump outlet educt tank B001 feedback running
11		A1.T1.A1T1S001.SV.C	Q 3.0	BOOL	pump outlet educt tank B001 control signal
12		A1.T1.A1T1S002.SO...	I 1.1	BOOL	pump outlet educt tank B002 feedback running
13		A1.T1.A1T1S002.SV.C	Q 3.1	BOOL	pump outlet educt tank B002 control signal
14		A1.T1.A1T1S003.SO...	I 1.2	BOOL	pump outlet educt tank B003 feedback running
15		A1.T1.A1T1S003.SV.C	Q 3.2	BOOL	pump outlet educt tank B003 control signal
16		A1.T1.A1T1X001.GO...	I 64.2	BOOL	valve inlet educt tank B001 feedback signal closed
17		A1.T1.A1T1X001.GO...	I 64.0	BOOL	valve inlet educt tank B001 feedback signal opened
18		A1.T1.A1T1X001.XV.C	Q 0.0	BOOL	valve inlet educt tank B001 control signal
19		A1.T1.A1T1X002.GO...	I 64.6	BOOL	valve inlet educt tank B002 feedback signal closed
20		A1.T1.A1T1X002.GO...	I 64.4	BOOL	valve inlet educt tank B002 feedback signal opened
21		A1.T1.A1T1X002.XV.C	Q 0.1	BOOL	valve inlet educt tank B002 control signal

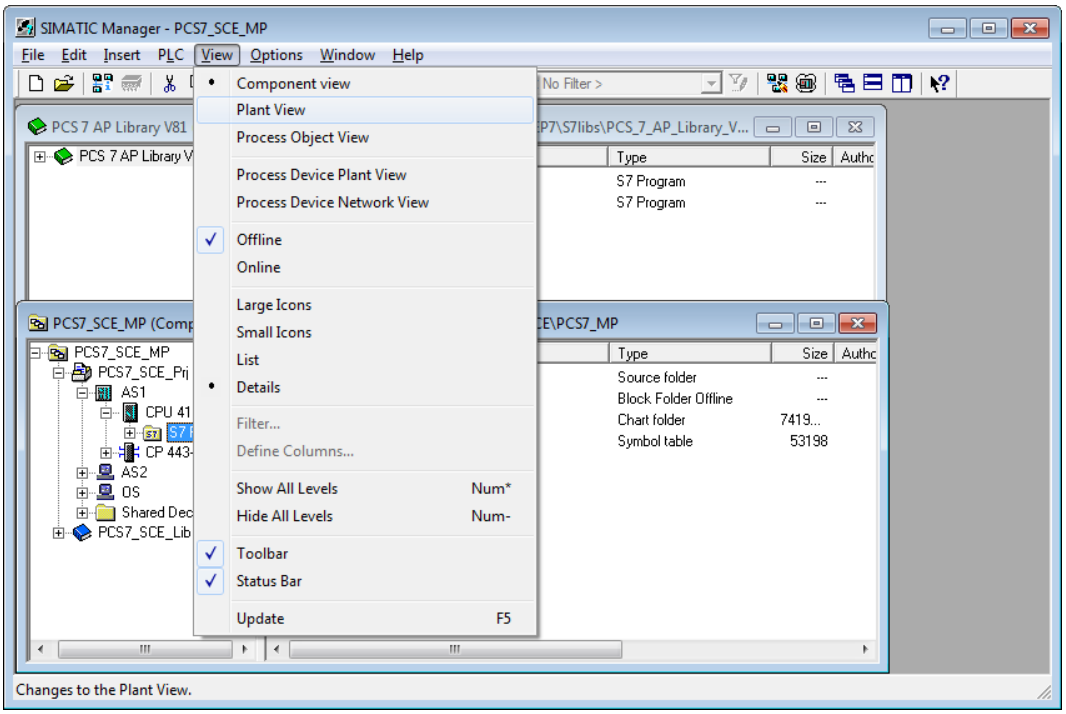
6. In extensive libraries, ***PCS 7*** provides a variety of prepared blocks and also pre-assembled charts, called templates. We want to use exactly such a template for the pump motor. To this end, we open the ***PCS 7 AP Library V81***.

   (→ File → Open → Libraries → PCS 7 AP Library V81 → OK)



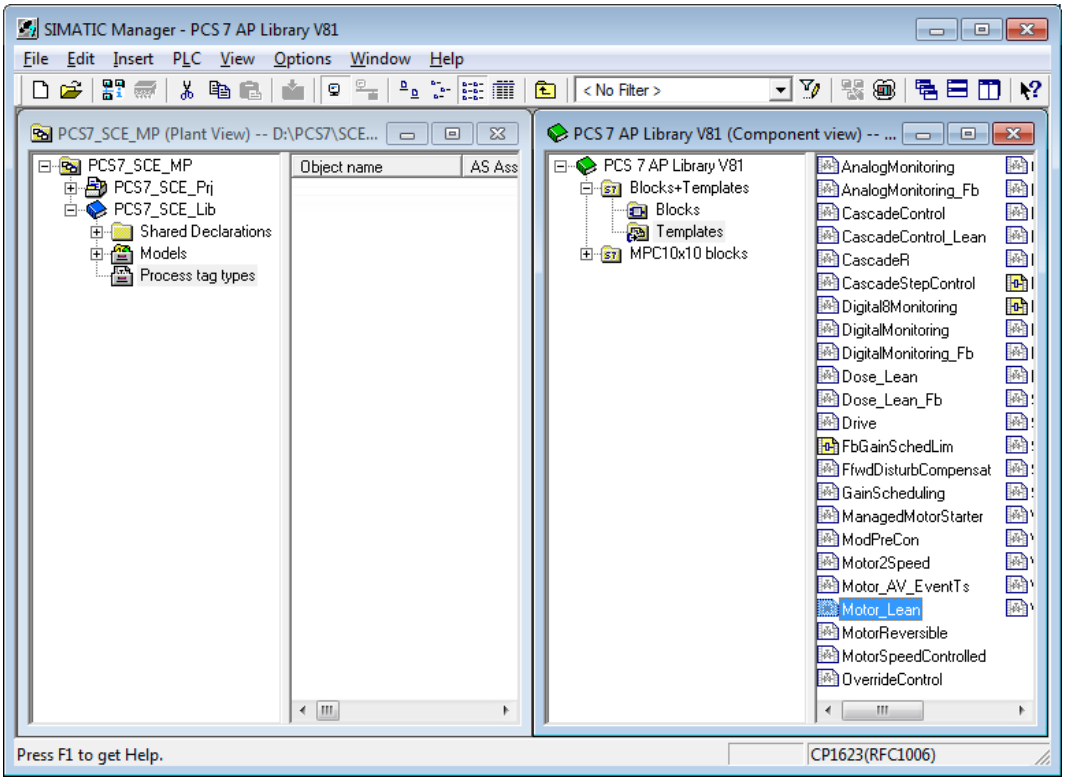


7. To drag this template from the library to the project, go to the Plant view of the project. (→ View → Plant View)

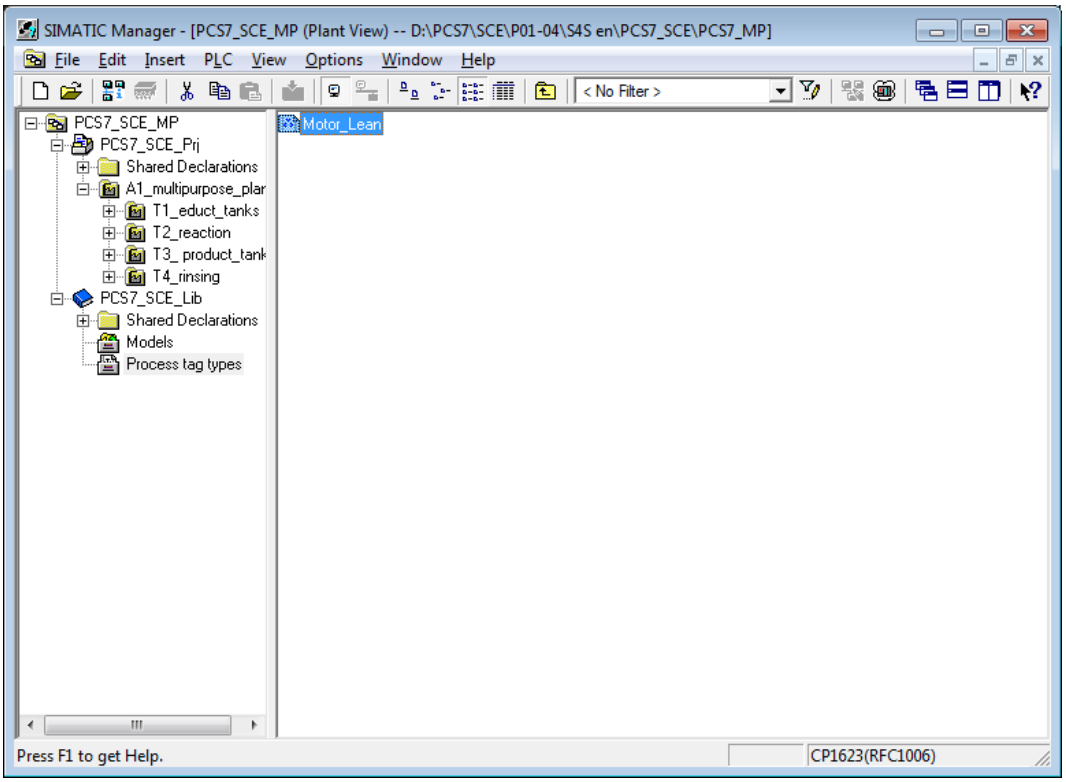


8. With Drag&Drop, Motor_Lean (library, templates) is moved to Process tag types (master data library).

   (→ PCS 7 Library → Blocks+Templates → Templates → drag Motor_Lean to Plant View → SCE_PCS7_Lib → Drag process tag types)



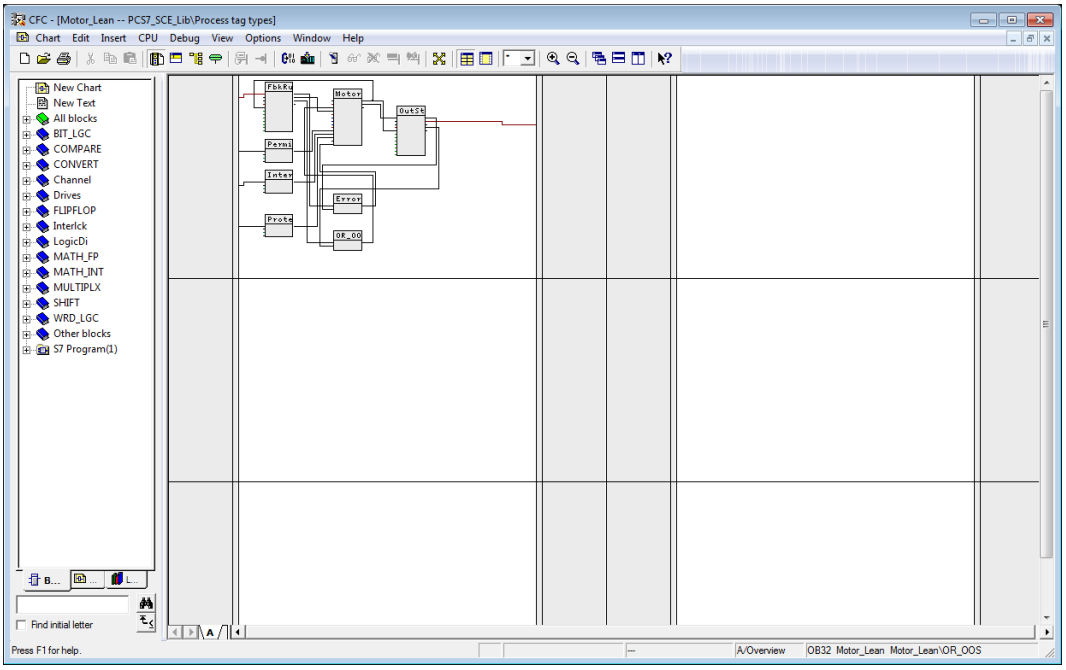
9. Now we are making a change centrally for all process tags of the type 'Motor_Lean'. This is done be opening the CFC 'Motor_Lean' in the master data library with a double click. (→ Motor_Lean)



Note: CFC stands for Continuous Function Chart and is a graphic programming language for describing continuous processes. In the CFC, pre-assembled blocks are placed, parameterized and interconnected. This way, the programmer creates an overall software structure for controlling a machine.

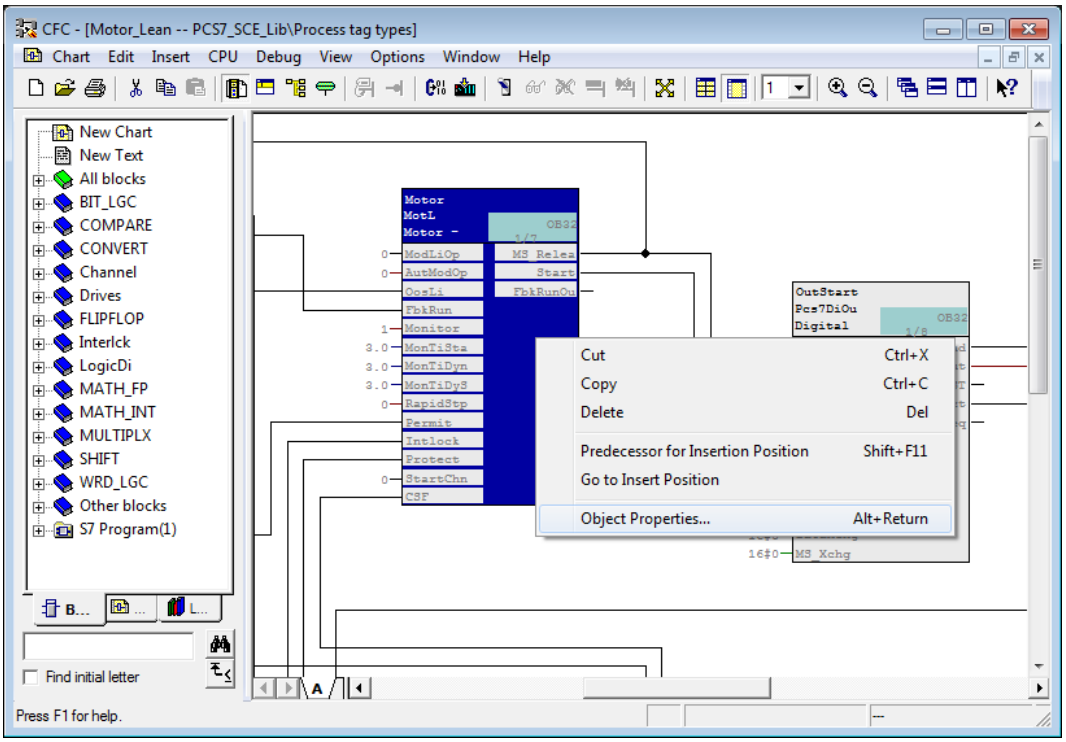
10. A CFC consists of chart partitions with 6 sheets each. In the overview, all six sheets are displayed with gray sheet bars. In the sheet bars, the incoming signals are shown on the left and the outgoing signals on the right of the sheet. By double-clicking a sheet, its view can be changed.

    (→ Overview → Double click on first sheet)

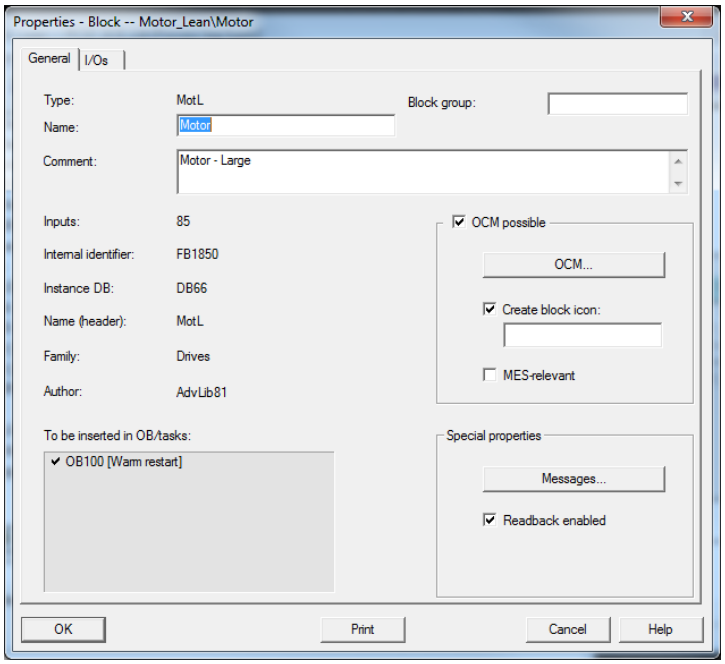


Note: Using the tabs in the lower bar, we can switch between the chart partitions (maximum A to Z). Here only chart partition A is initially provided.

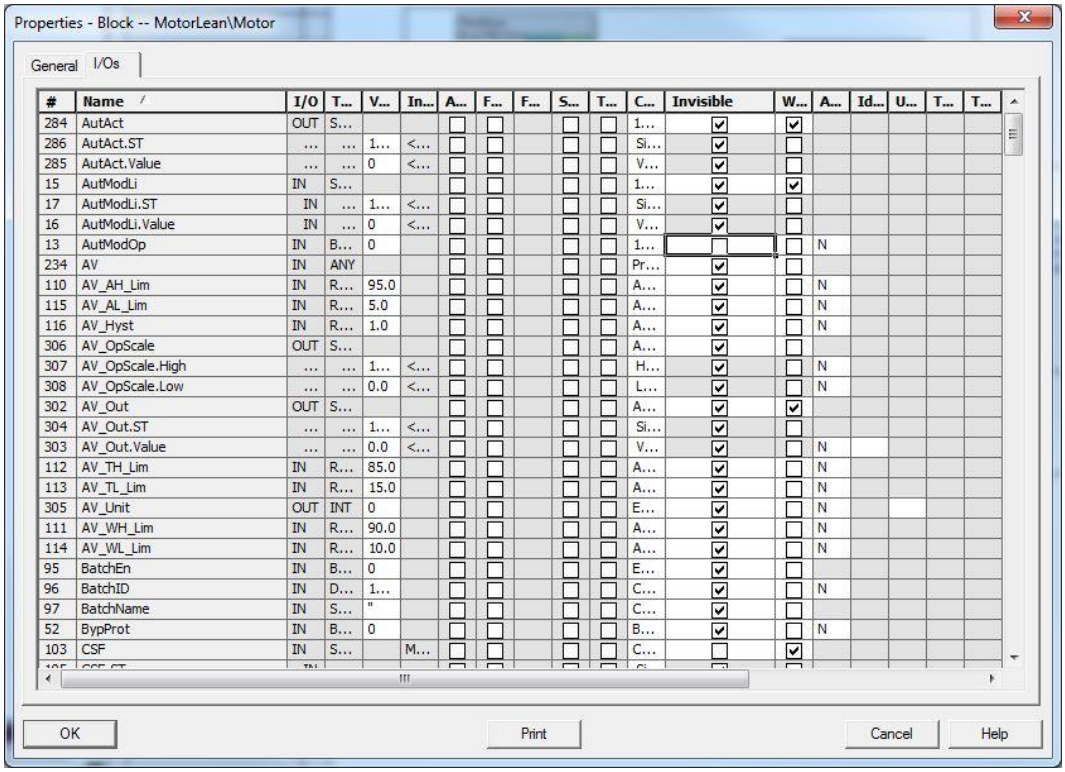
11. We now are making a change in the 'MotL' block with the process tag type 'Motor_Lean'. To this end, its properties are opened with Object Properties. (→ MotL-> right-click -> Object Properties)



12. You do not want to change the general properties such as operator control and monitoring; therefore, we change to the I/Os. (→ I/Os)



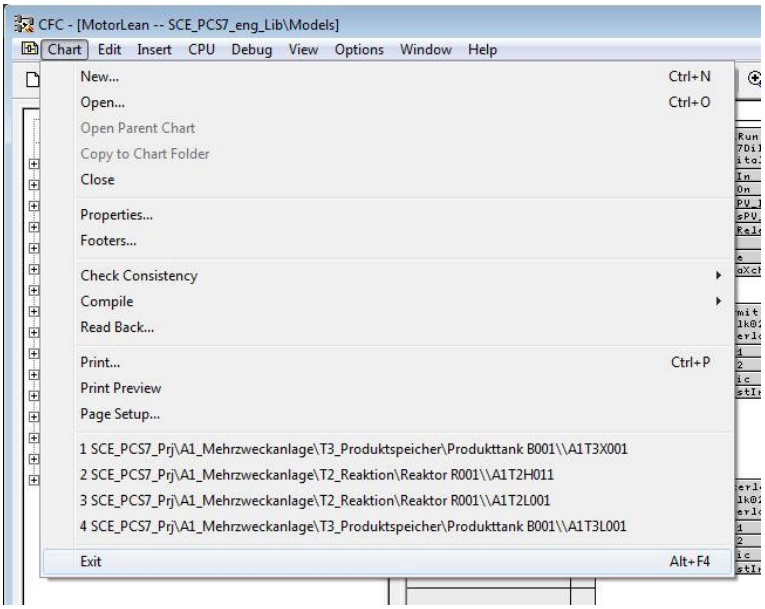
13. The connections are represented in a table together with a variety of properties that can be set. The most important properties are introduced below. In the 'MotL' block, we are only deleting the Invisible feature for 'AutModOP'. This makes the connection visible in the sheet. Exit the Properties with
(→ AutModOP → ☐ → OK)



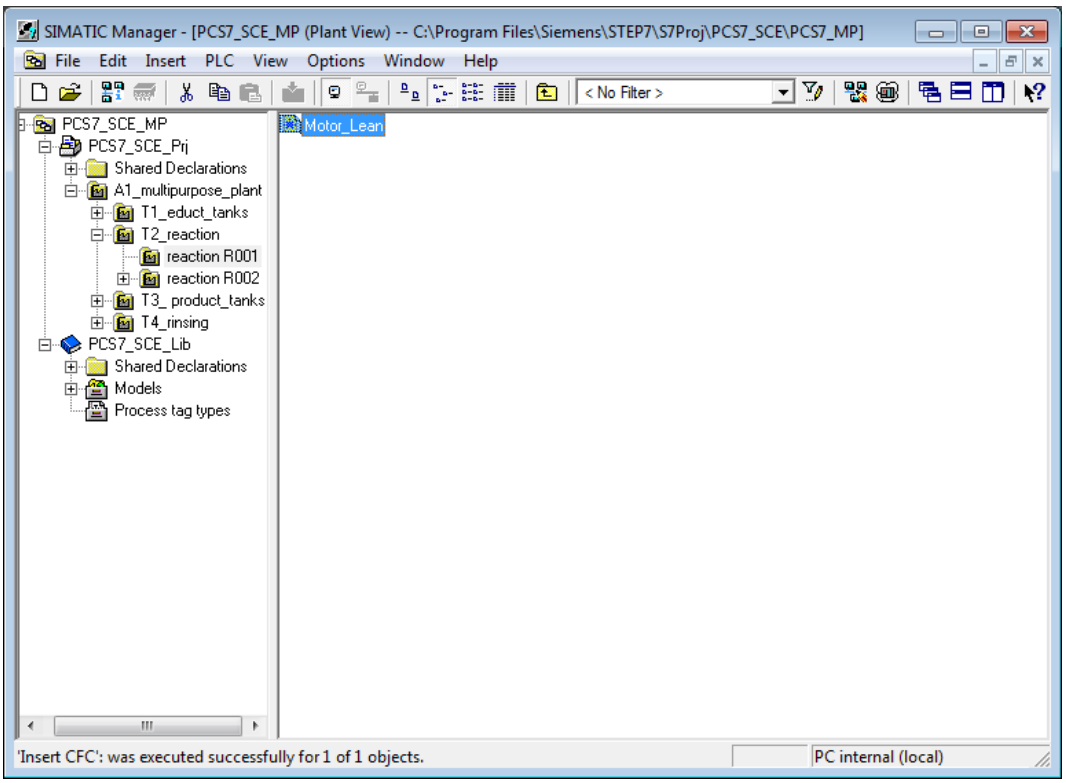
#	Name	I/O	T...	V...	In...	A...	F...	F...	S...	T...	C...	Invisible	W...	A...	Id...	U...	T...	T...
284	AutAct	OUT	S...			☐	☐		☐	☐	1...	☑	☑					
286	AutAct.ST	...	...	1...	<...	☐	☐		☐	☐	Si...	☑	☐					
285	AutAct.Value	...	...	0	<...	☐	☐		☐	☐	V...	☑	☐					
15	AutModLi	IN	S...			☐	☐		☐	☐	1...	☑	☑					
17	AutModLi.ST	IN	...	1...	<...	☐	☐		☐	☐	Si...	☑	☐					
16	AutModLi.Value	IN	...	0	<...	☐	☐		☐	☐	V...	☑	☐					
13	AutModOp	IN	B...	0		☐	☐		☐	☐	1...	☐	☐	N				
234	AV	IN	ANY			☐	☐		☐	☐	Pr...	☑	☐					
110	AV_AH_Lim	IN	R...	95.0		☐	☐		☐	☐	A...	☑	☐	N				
115	AV_AL_Lim	IN	R...	5.0		☐	☐		☐	☐	A...	☑	☐	N				
116	AV_Hyst	IN	R...	1.0		☐	☐		☐	☐	A...	☑	☐	N				
306	AV_OpScale	OUT	S...			☐	☐		☐	☐	A...	☑	☐					
307	AV_OpScale.High	...	...	1...	<...	☐	☐		☐	☐	H...	☑	☐	N				
308	AV_OpScale.Low	...	...	0.0	<...	☐	☐		☐	☐	L...	☑	☐	N				
302	AV_Out	OUT	S...			☐	☐		☐	☐	A...	☑	☑					
304	AV_Out.ST	...	...	1...	<...	☐	☐		☐	☐	Si...	☑	☐					
303	AV_Out.Value	...	...	0.0	<...	☐	☐		☐	☐	V...	☑	☐	N				
112	AV_TH_Lim	IN	R...	85.0		☐	☐		☐	☐	A...	☑	☐	N				
113	AV_TL_Lim	IN	R...	15.0		☐	☐		☐	☐	A...	☑	☐	N				
305	AV_Unit	OUT	INT	0		☐	☐		☐	☐	E...	☑	☐	N				
111	AV_WH_Lim	IN	R...	90.0		☐	☐		☐	☐	A...	☑	☐	N				
114	AV_WL_Lim	IN	R...	10.0		☐	☐		☐	☐	A...	☑	☐	N				
95	BatchEn	IN	B...	0		☐	☐		☐	☐	E...	☑	☐					
96	BatchID	IN	D...	1...		☐	☐		☐	☐	C...	☑	☐	N				
97	BatchName	IN	S...	"		☐	☐		☐	☐	C...	☑	☐					
52	BypProt	IN	B...	0		☐	☐		☐	☐	B...	☑	☐	N				
103	CSF	IN	S...		M...	☐	☐		☐	☐	C...	☐	☑					

Note: By clicking on the header of a property, the representation after this column can be sorted alphabetically.

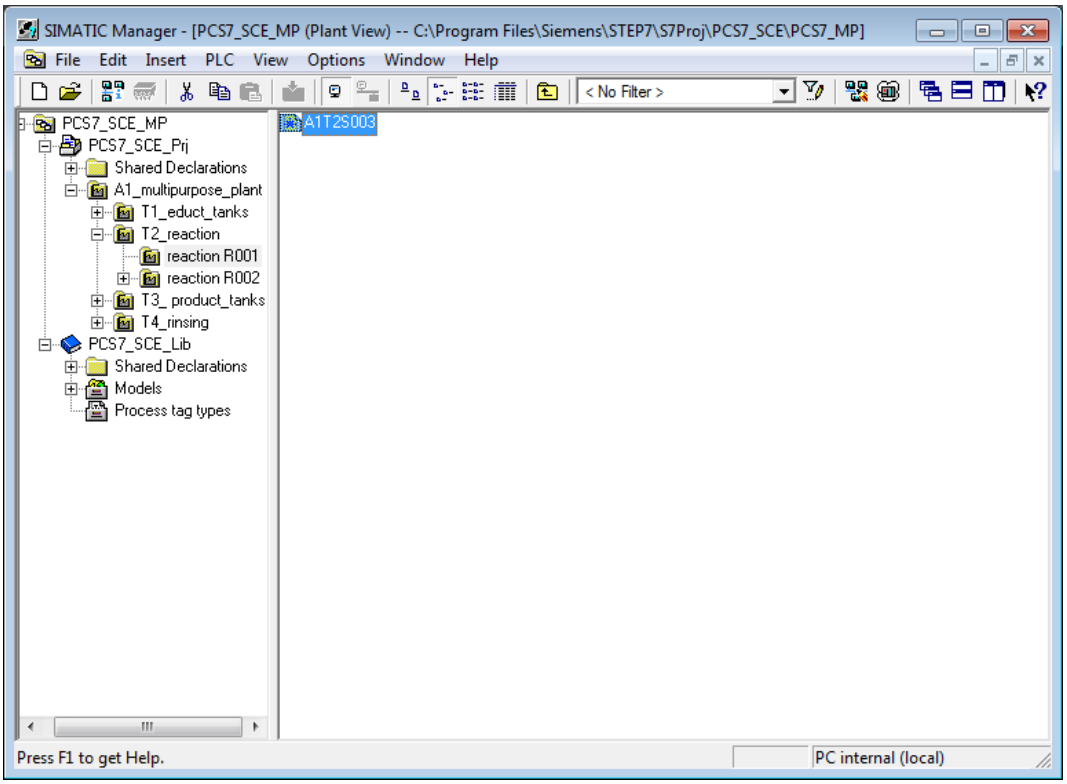
14. We can now close the process tag type. It is saved automatically after each change. (→ Chart → Exit)



15. To use the process tag type 'Motor_Lean', it is dragged to the chart folder 'Reactor R001'.
(→ Motor_Lean → Reactor R001')

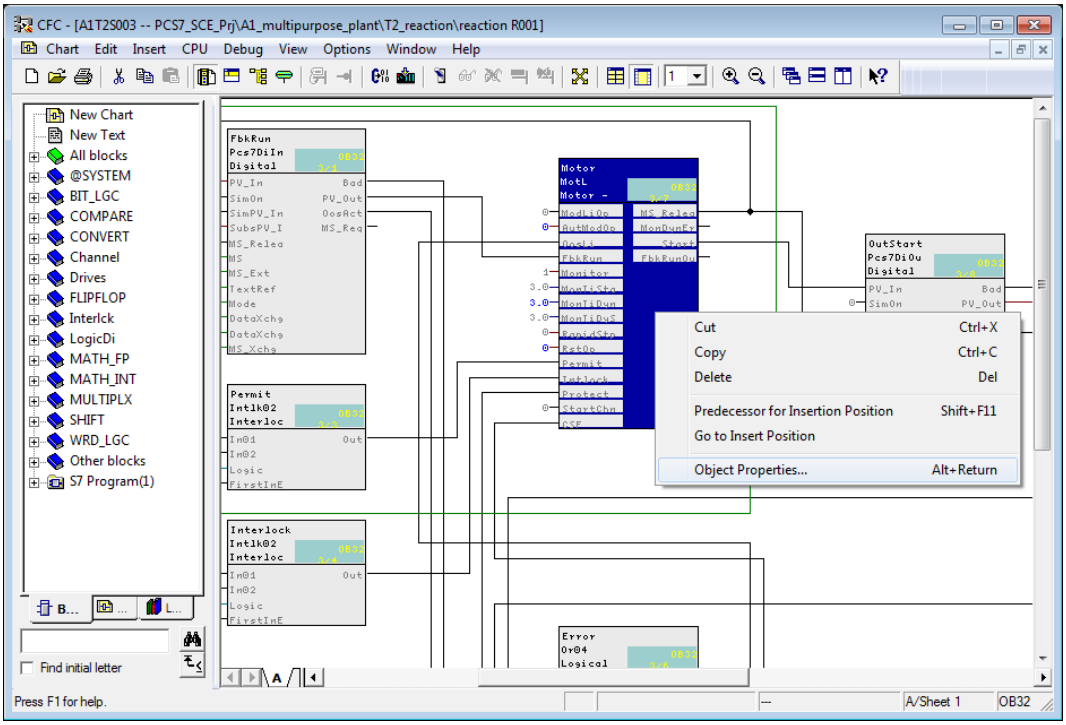


16. Because this chart will be used to activate the pump A1T2S003, it is now renamed A1T2S003 and opened with a double-click.
(→ Reactor R001 → A1T2S003 → A1T2S003)



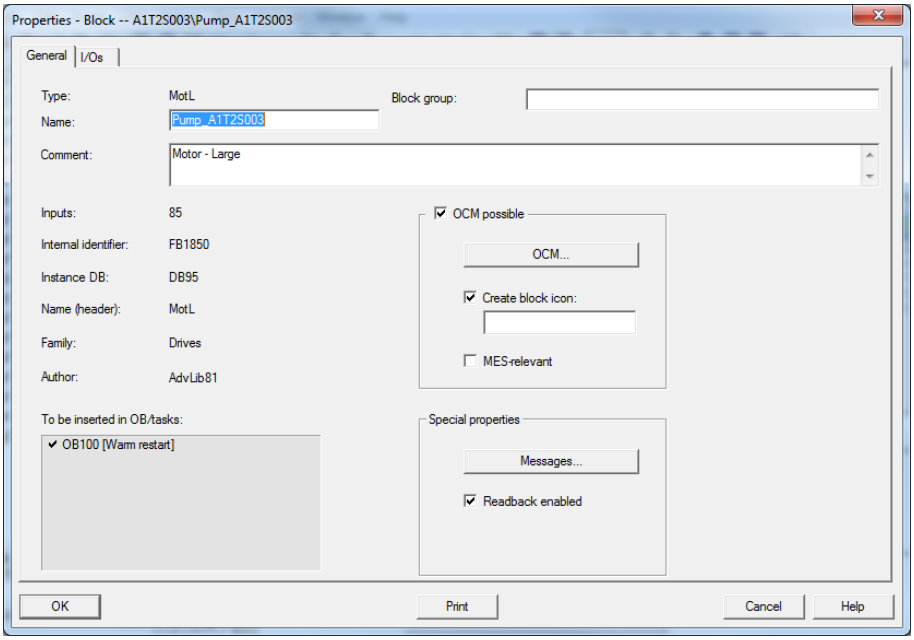
17. With a right-click, we open the properties of the 'MotL' block.

    (→ Motor_Lean → Object Properties)



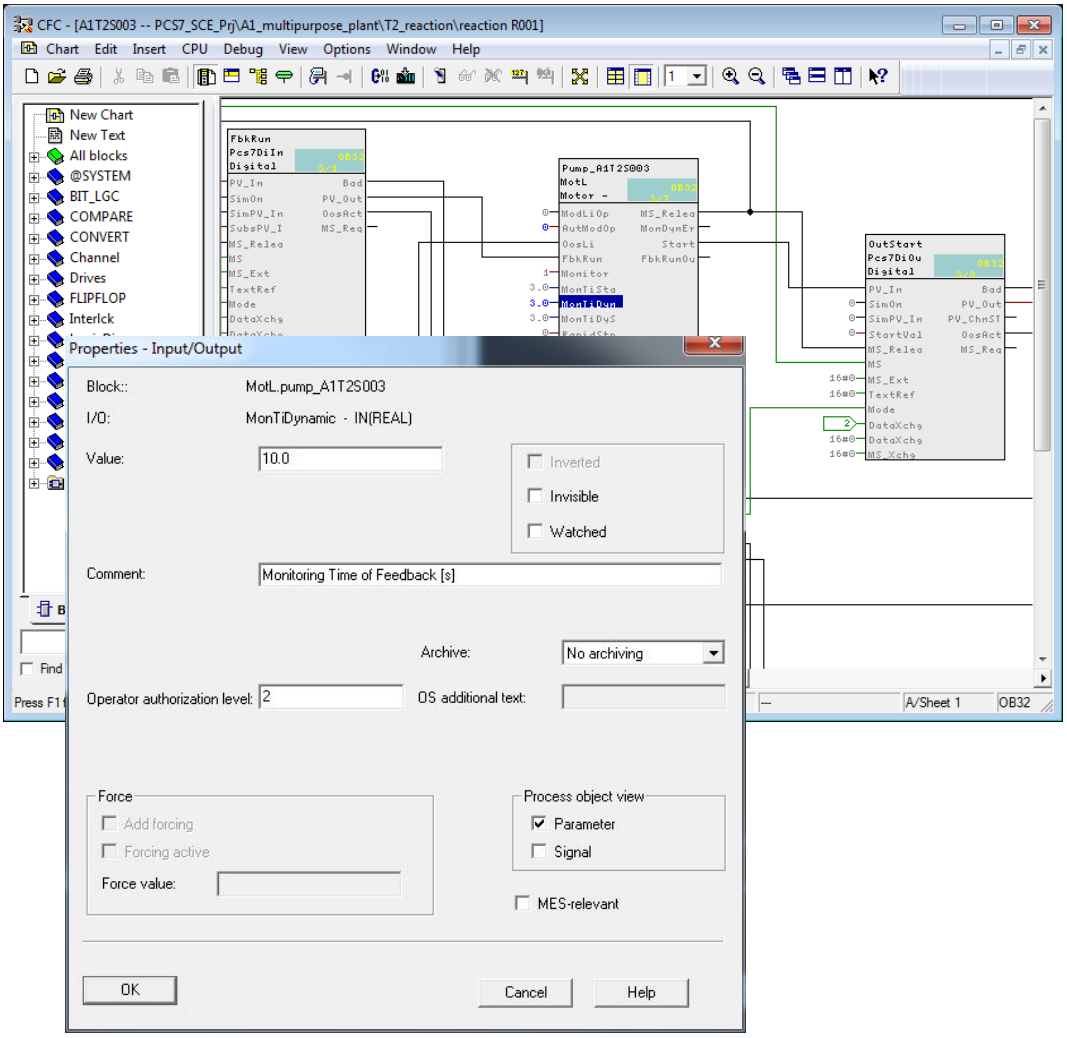
18. The name of the block is changed in the general properties.

    (→ Pump_A1T2S003 → OK)

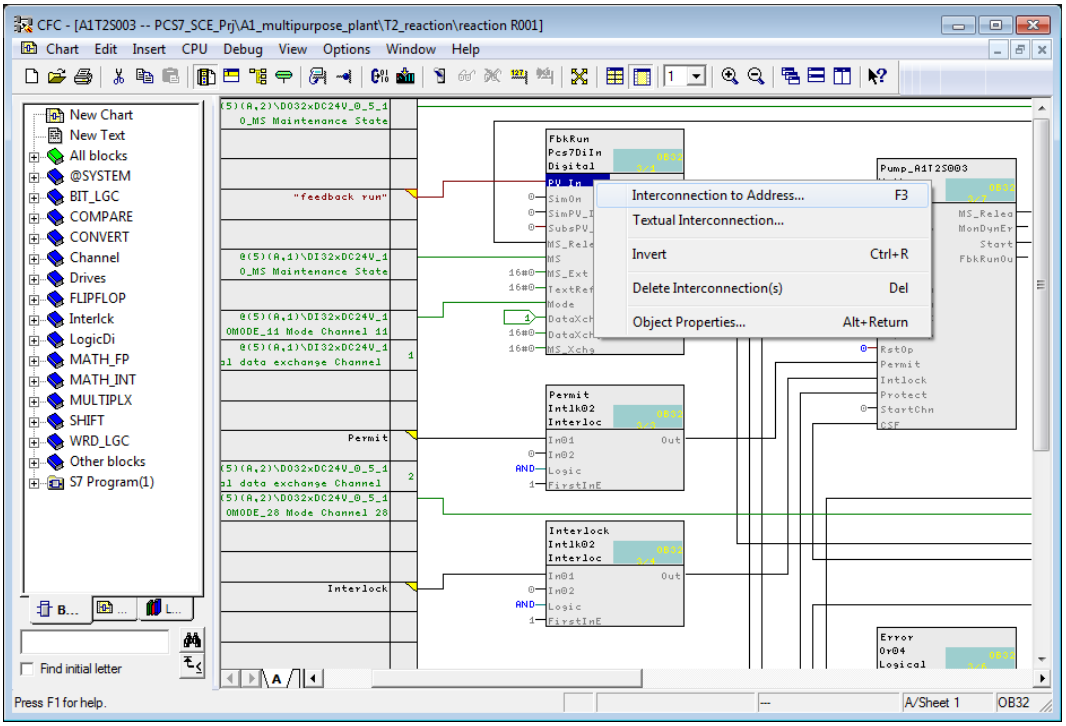


19. Now the time for feedback monitoring after the motor block was operated successfully is changed to 10.0 seconds. To do this, we open the Properties dialog for the input 'MonTiDyn' with a double-click and enter 10.0 as Value. We exit the dialog with 'OK'.

    (→ MonTiDyn → 10.0 → OK)



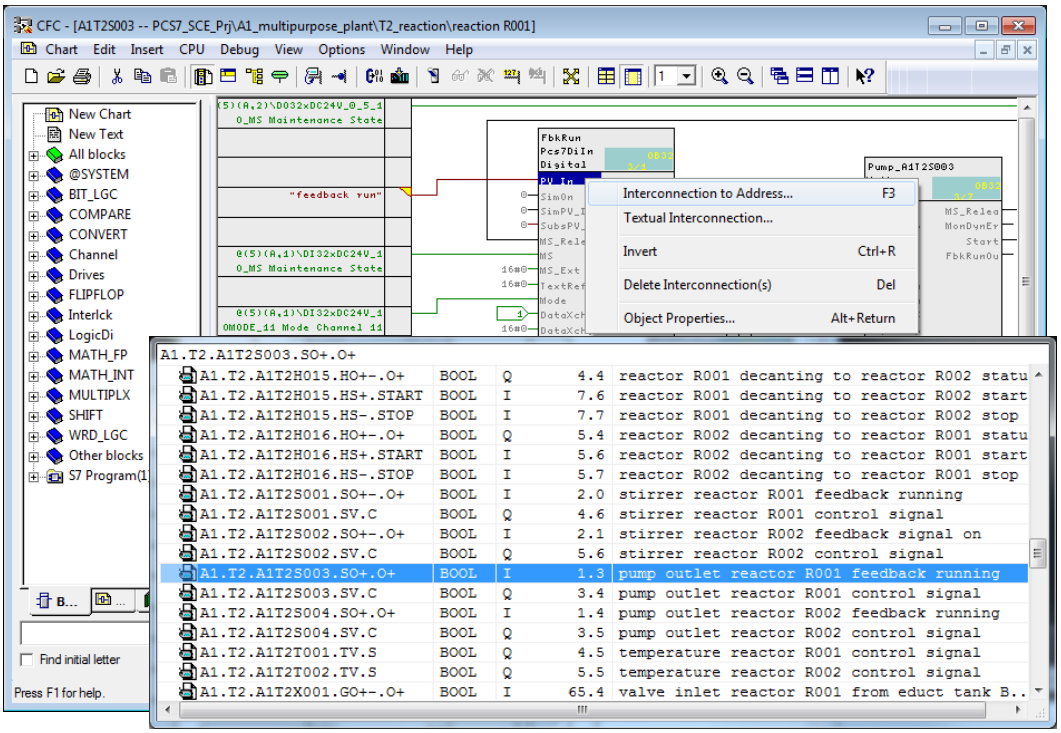
20. Next, the feedback is connected to the input address. This is done by right-clicking on the input 'PV_In' in the 'PCS7DiIn' block and then selecting the interconnection to the address.
(→ PV-In → Interconnection to Address)



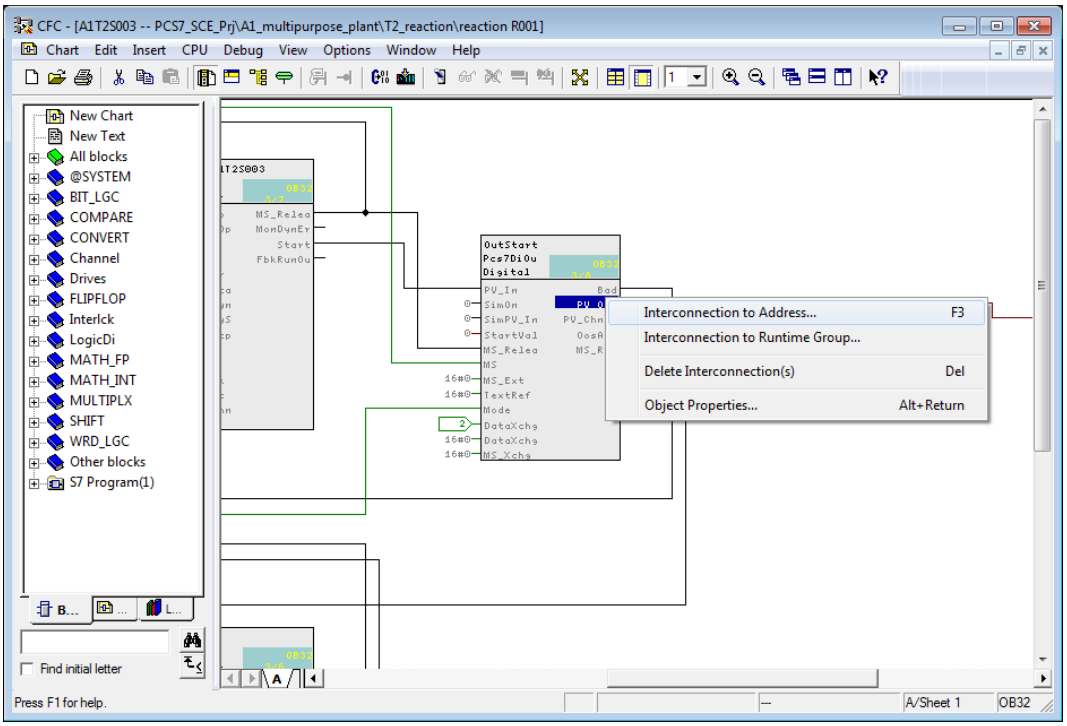
Note: Like the PCS7DiOu block, the PCS7DiIn block is a driver block for interfacing with the PLC IO. If the value 'PV_In' in one of these blocks is interconnected with an address that is also configured in the hardware configuration, the input MODE is supplied automatically with data during the compilation that is run later.

For this to happen later during the compilation run, 'Generate module driver' has to be selected.

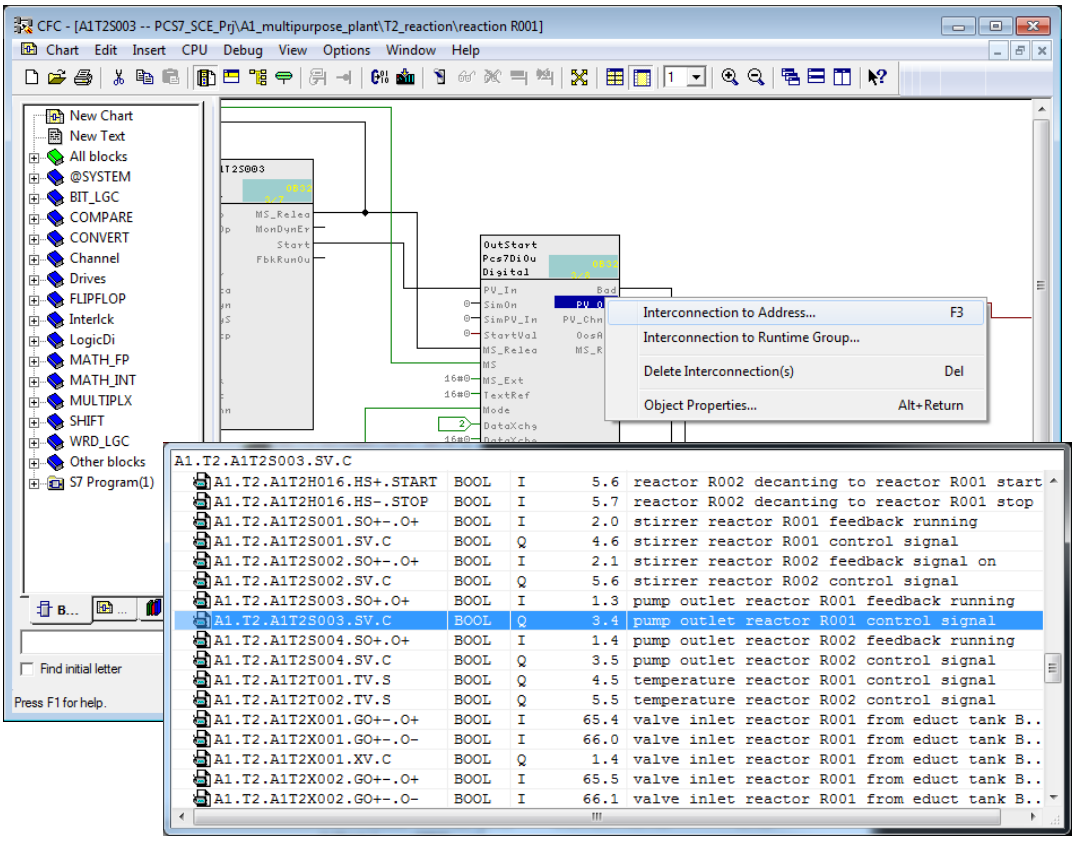
21. The address that signals whether the pump is running can then be selected conveniently directly from the symbol table.
(→ A1.T2.A1T2S003.S0+.0+)



22. Now the activation of the output address has to be interconnected. This is done by right-clicking the output 'PV_Out' in the block 'PCS7DiOu' and then selecting the interconnection to the address. (→ PV_Out → Interconnection to Address)

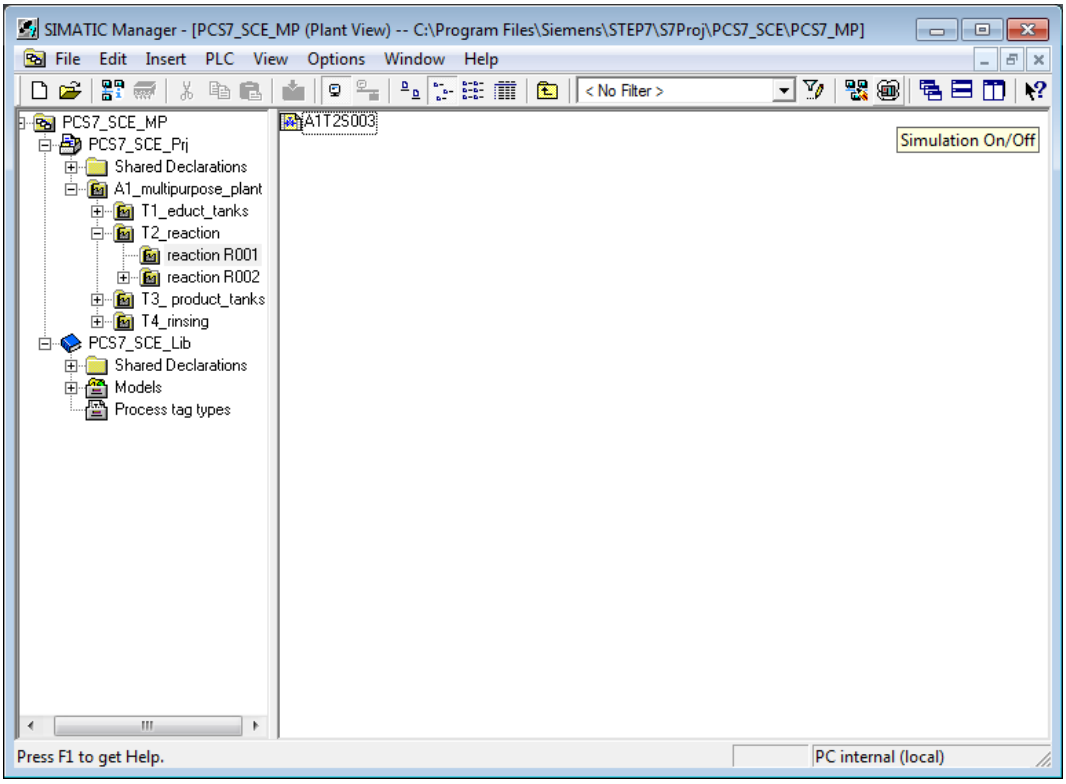


23. The address for activating the pump can once again be conveniently selected from the symbol table. (→ A1.T2.A1T2S003.SV.C)

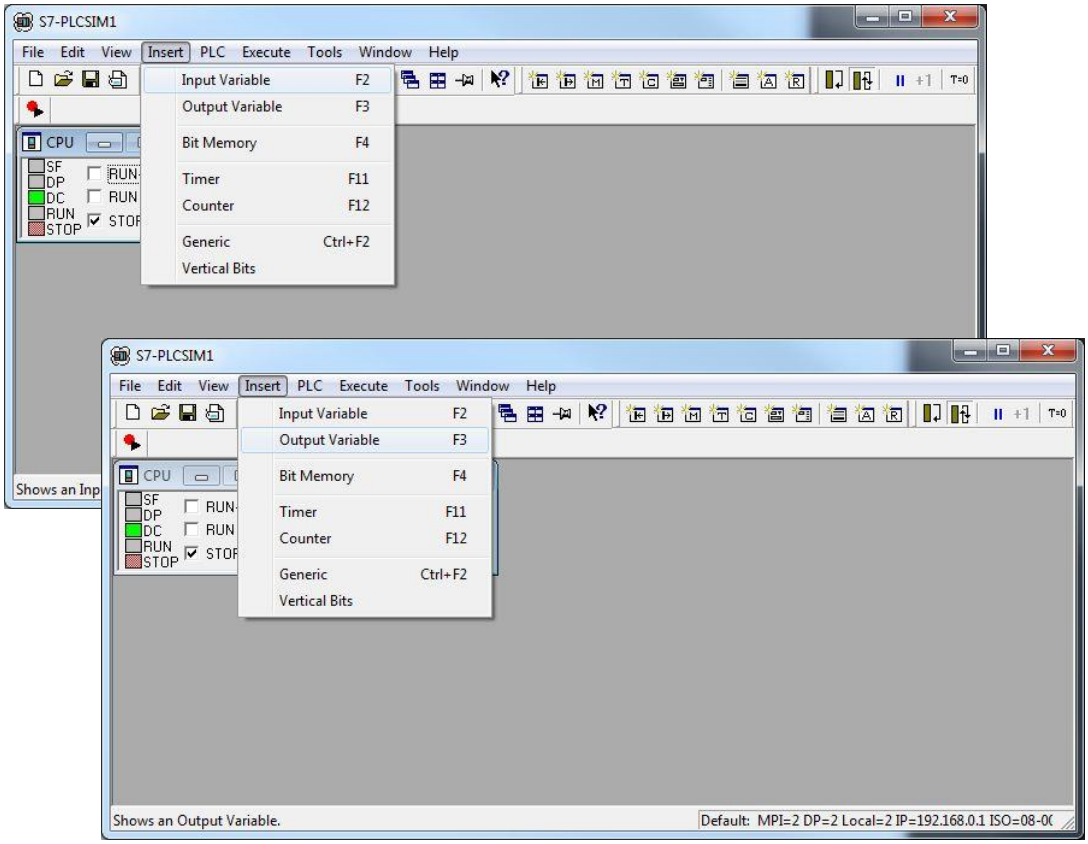


Note: The placeholder at the output of Pcs7DiOu 'output start' should be deleted; otherwise, there will be a warning during compilation!

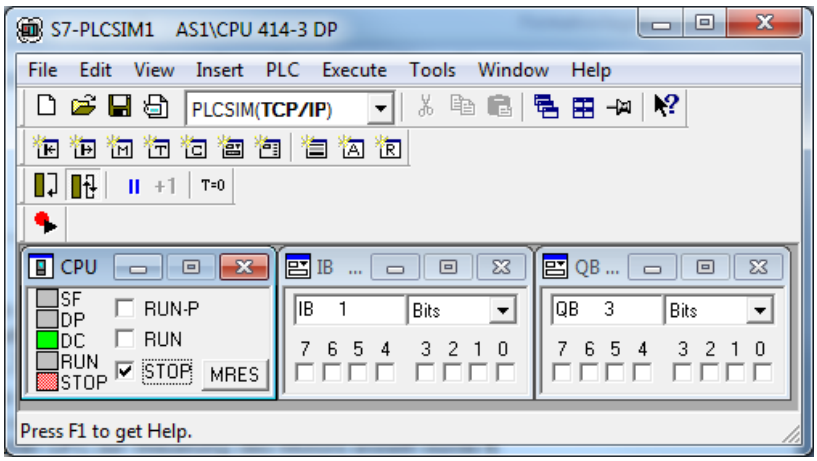
24. Before the program for the pump motor can be compiled and loaded, the PLC simulation ***S7-PLCSIM*** has to be started from the SIMATIC-Manager. (→ )



25. The PLC simulation acts like a real SIMATIC S7 CPU. However, the inputs and outputs have to be inserted first before they can be monitored and operated. (→ Insert → Input → Insert → Output)

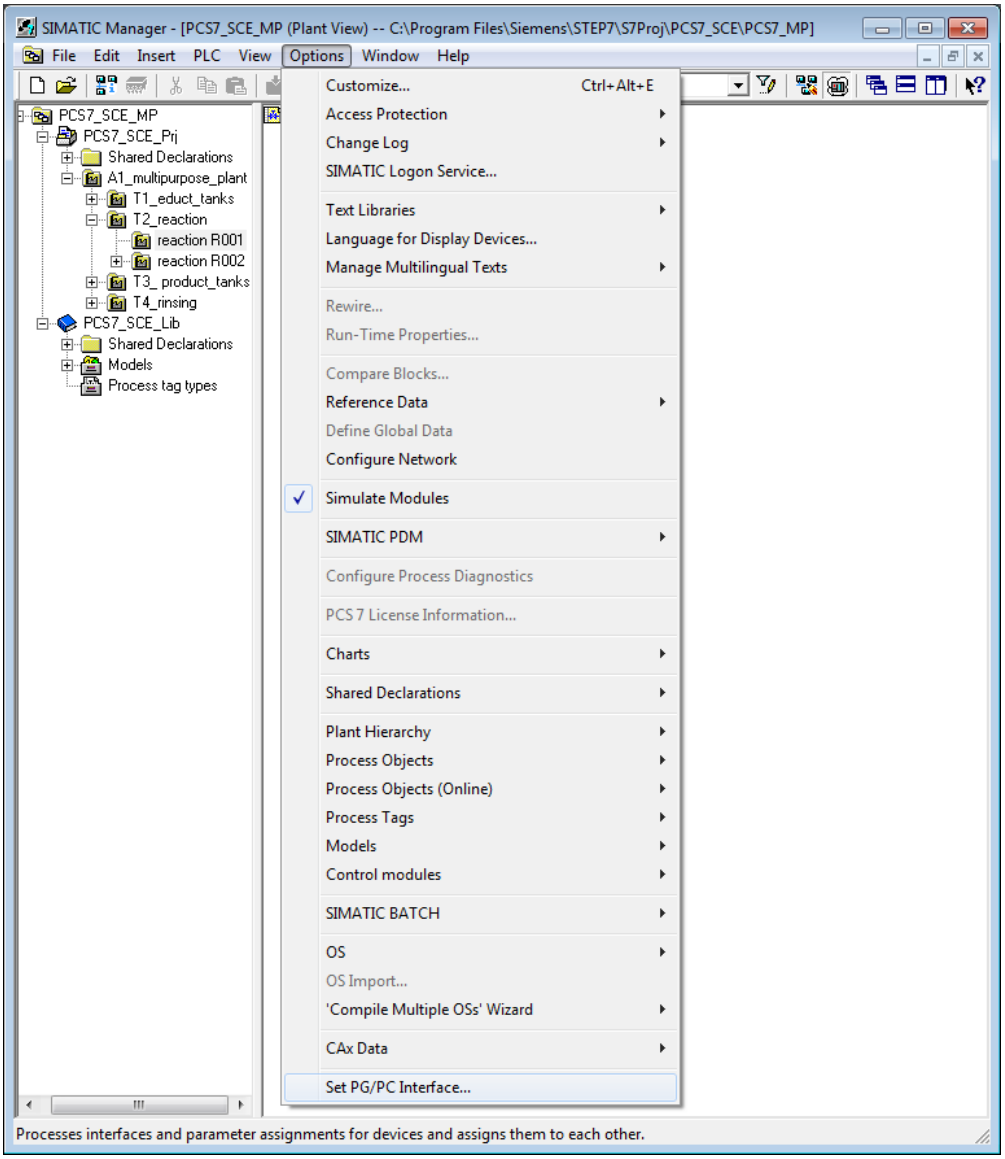


26. Next, the correct byte addresses have to be entered. (→ IB1 → QB3)

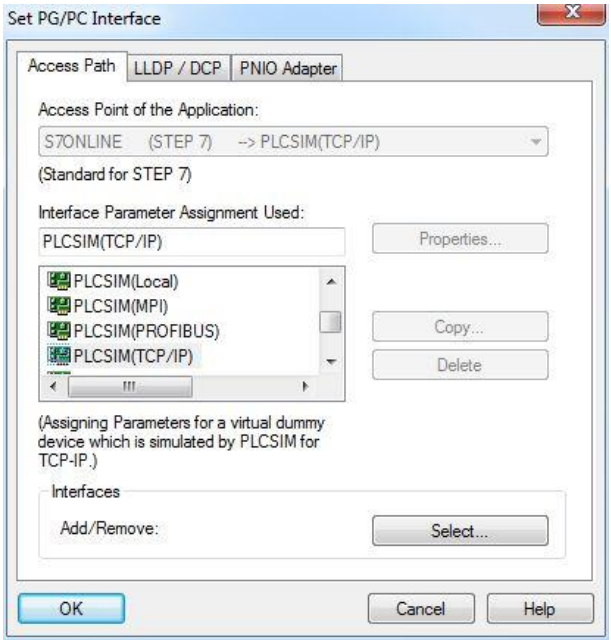


27. For loading from the SIMATIC Manager to ***S7-PLCSIM*** to take place using the correct interface, the PG/PC interface will be set correctly.

(→ SIMATIC Manager → Options → Set PG/PC Interface)

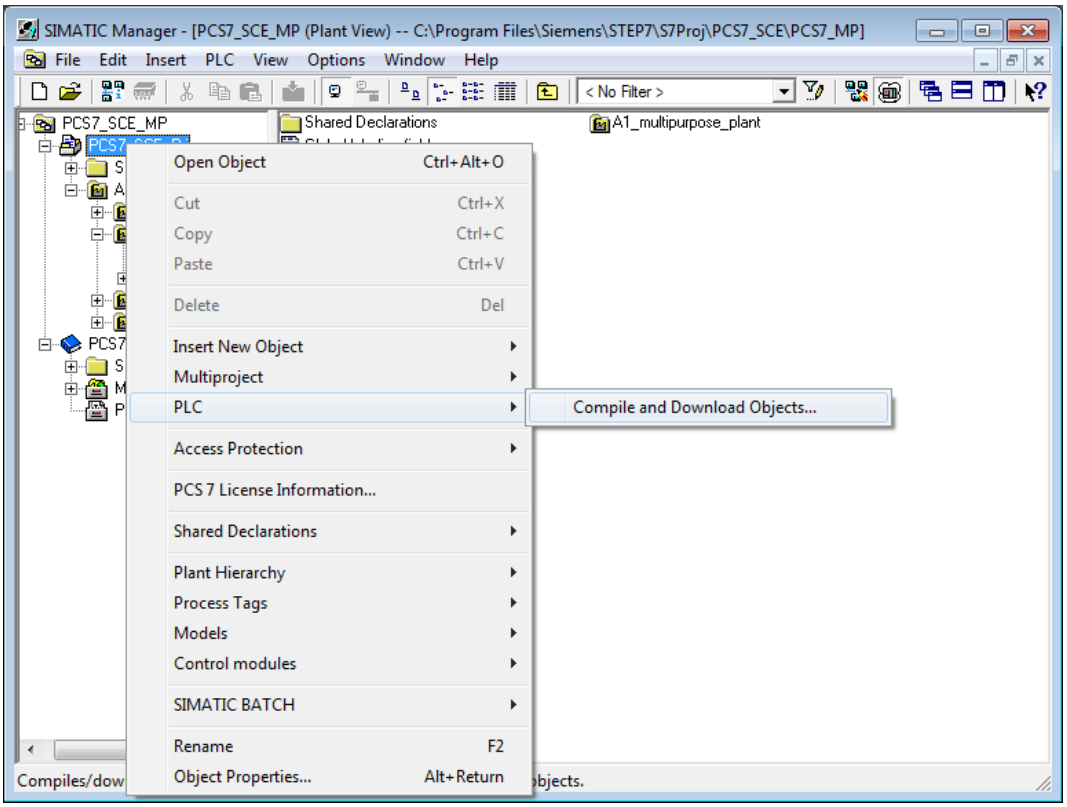


28. PLCSIM(TCP/IP) is set as interface here. (→ PLCSIM(TCP/IP) → OK)

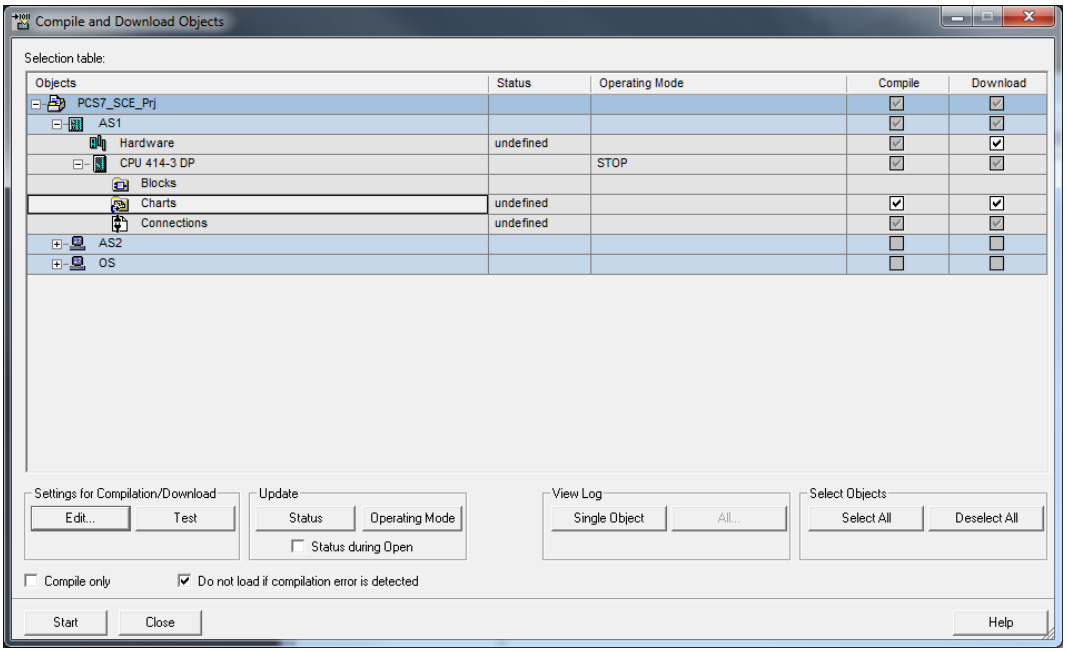


29. Now the project folder can be highlighted and compilation and download of the objects can start.

(→ Plant View → SCE_PCS7_Prj → PLC → Compile and Download Objects)

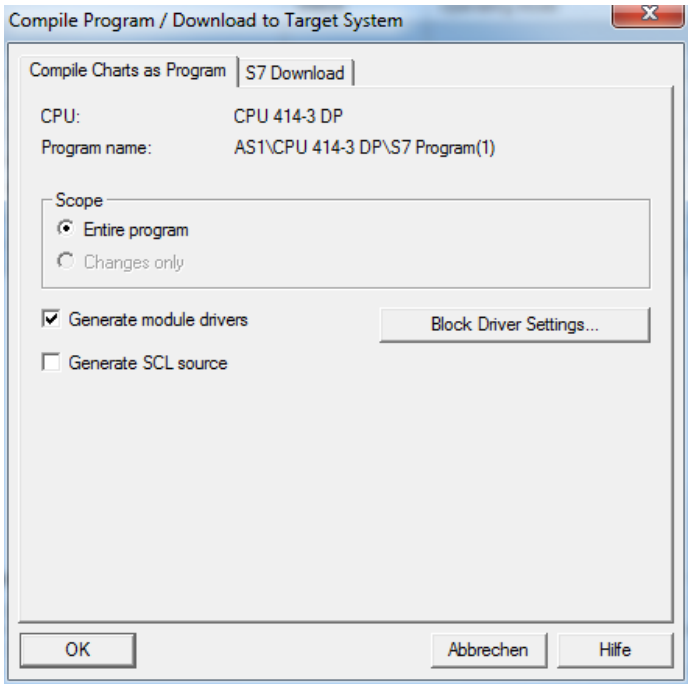


30. In the following selection, 'Compile and Download Objects' is selected for the hardware and the charts of the AS1. Then the folder 'Charts' is highlighted and its setting checked using 'Edit'.
(→ ☑ → ☑ → ☑ → Charts → Edit)

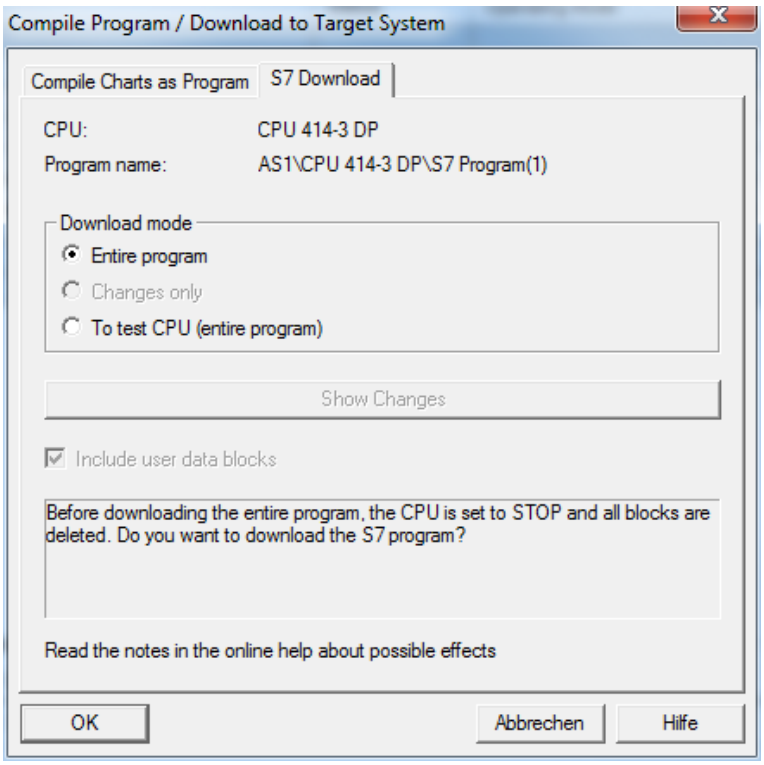


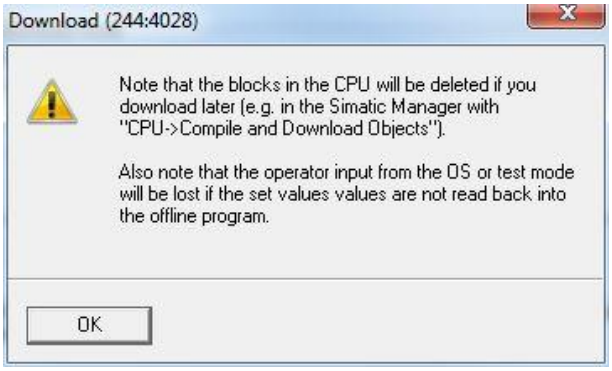
31. When compiling the charts, it is important to compile the entire program and to have the module drivers generated.

(→ Entire program → Generate module drivers → S7 Download)

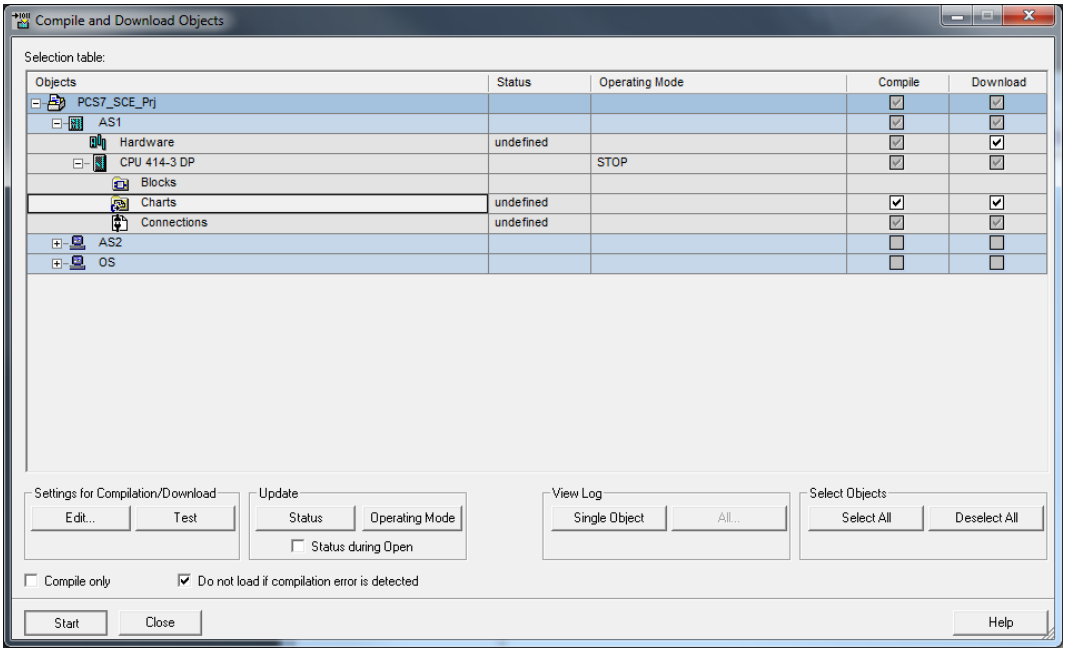


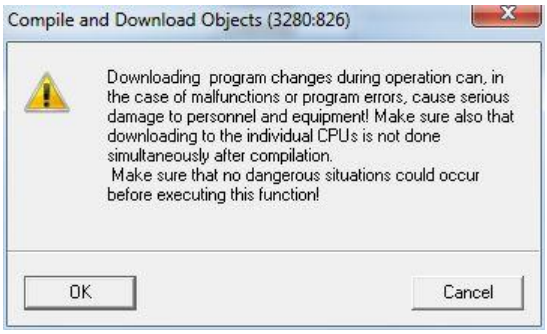
32. When downloading the charts, it is also important to download the entire program. (→ Entire program → OK → OK)

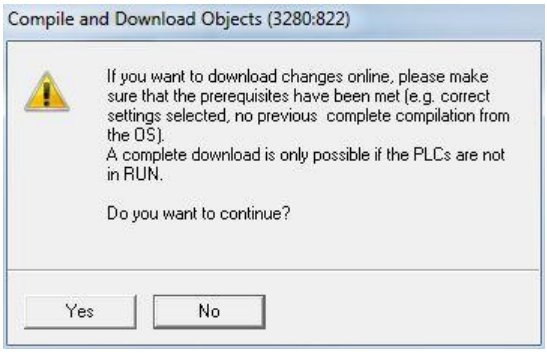




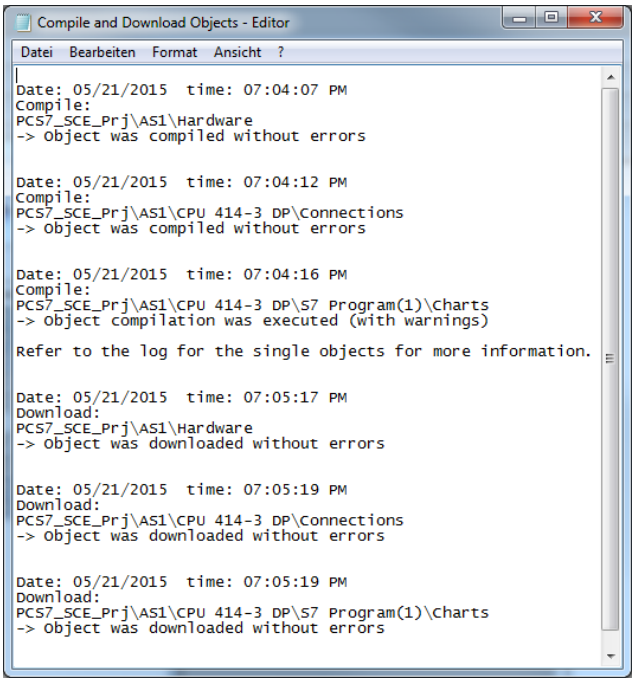
33. Finally, we can start 'Compile and Download Objects'. The warnings and instructions regarding plant safety should be read carefully. Prior to compile and download, the CPU has to be switched to STOP.
(→ Start → OK → Yes)



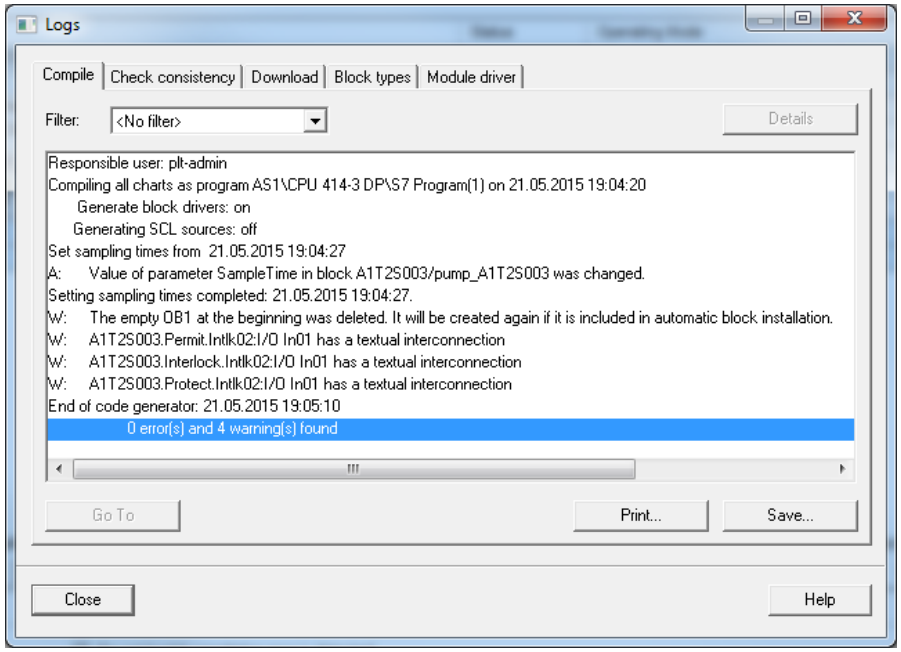




34. Ultimately, the errors and warnings are displayed in a log. Close the window. (→ )

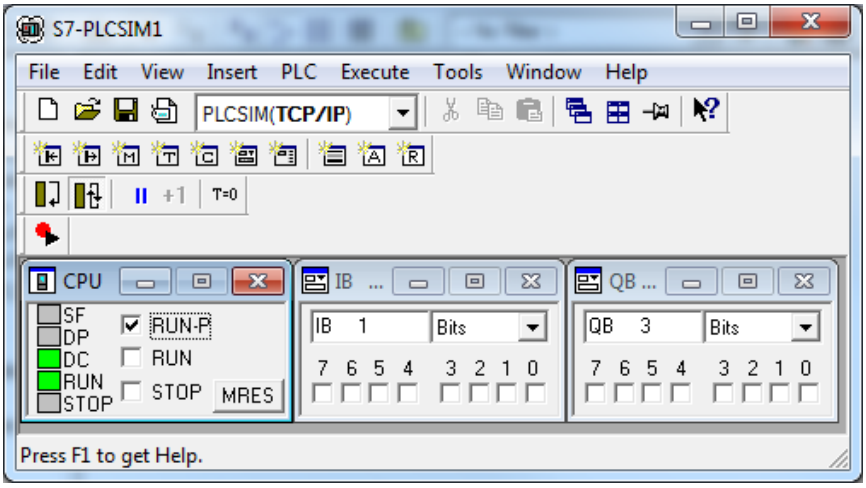


35. If you want to view log details, check at Logs and click on Single object. (→ Single Object → Close → Close)

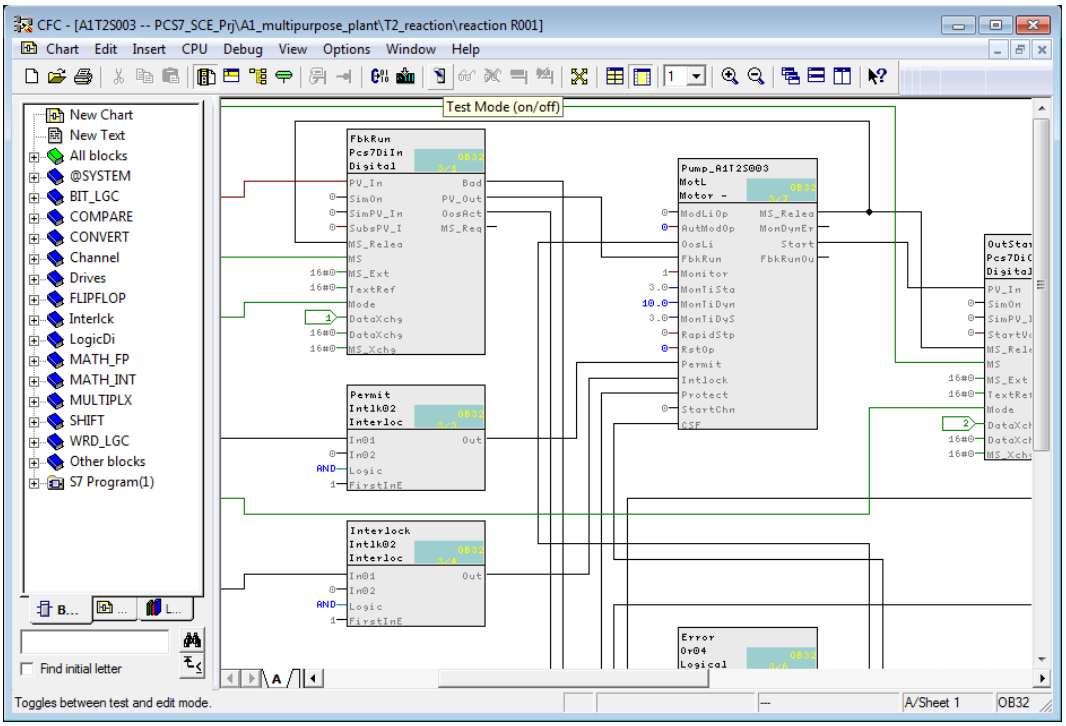


Note: Four warnings are indicated here. Delete the empty OB1 and existing textual interconnections. They occur because of the unconnected connections of the template. These connections are connected in chapter P01-05.

36. To test the program, the CPU in ***S7-PLCSIM*** is switched to 'RUN-P'. (→ S7-PLCSIM → RUN-P)

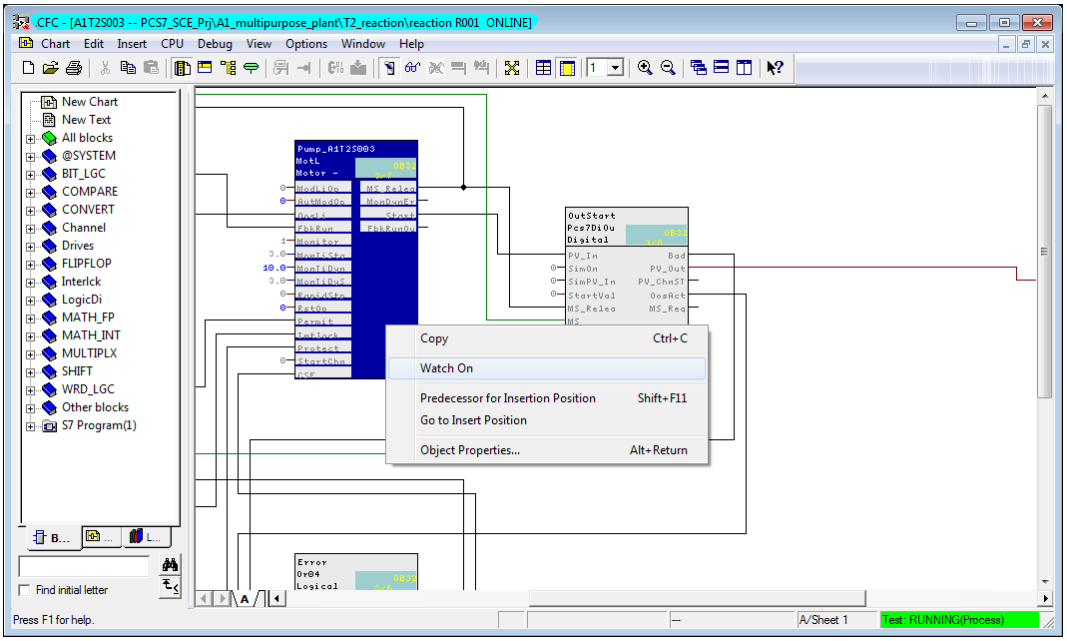


37. Before the individual blocks can be monitored in the CFC, the chart has to be switched to the test mode first. (→ CFC → )



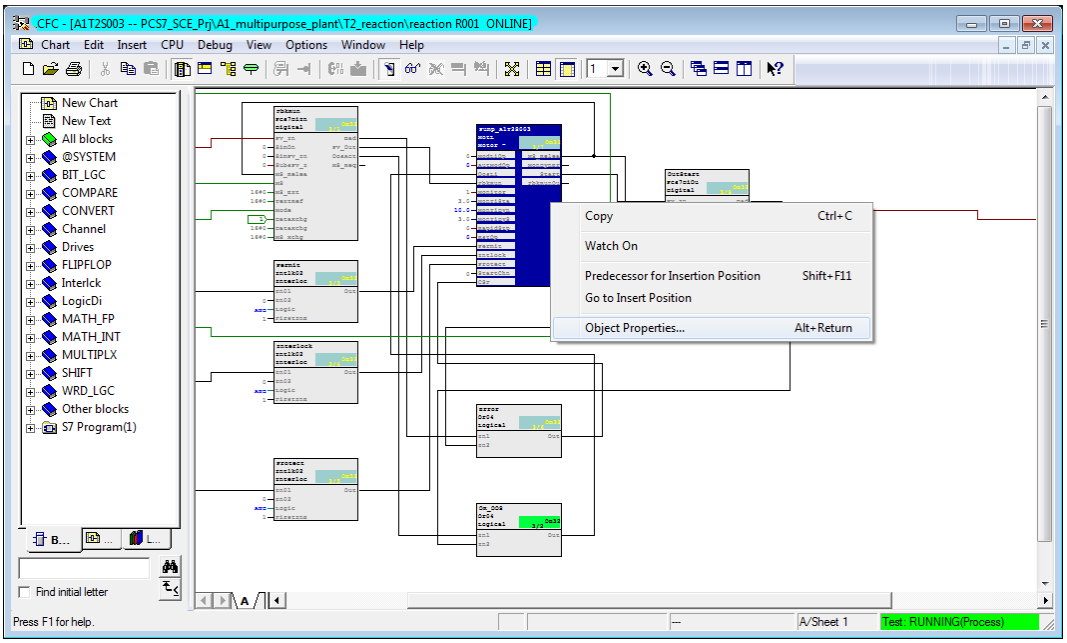
38. The blocks that are to be monitored must now be explicitly activated for monitoring. The same applies subsequently to the individual connections of the block.

    (→ Pump_A1T2S003 → Watch On)

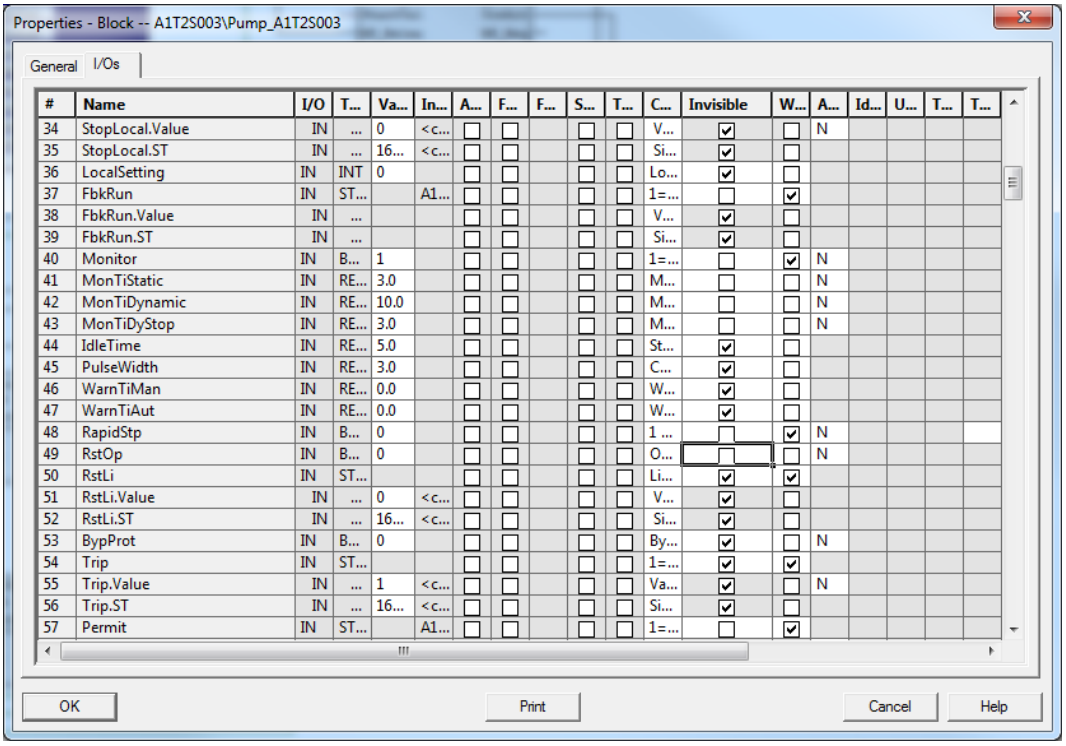


39. To continue, we have to make the connections for the automatic control 'StartAut' and 'StopAut' of the 'MotL' block visible. If there should be an error, 'RstOp' and ‘MonDynErr' should be made visible, too.

    (→ Object Properties)

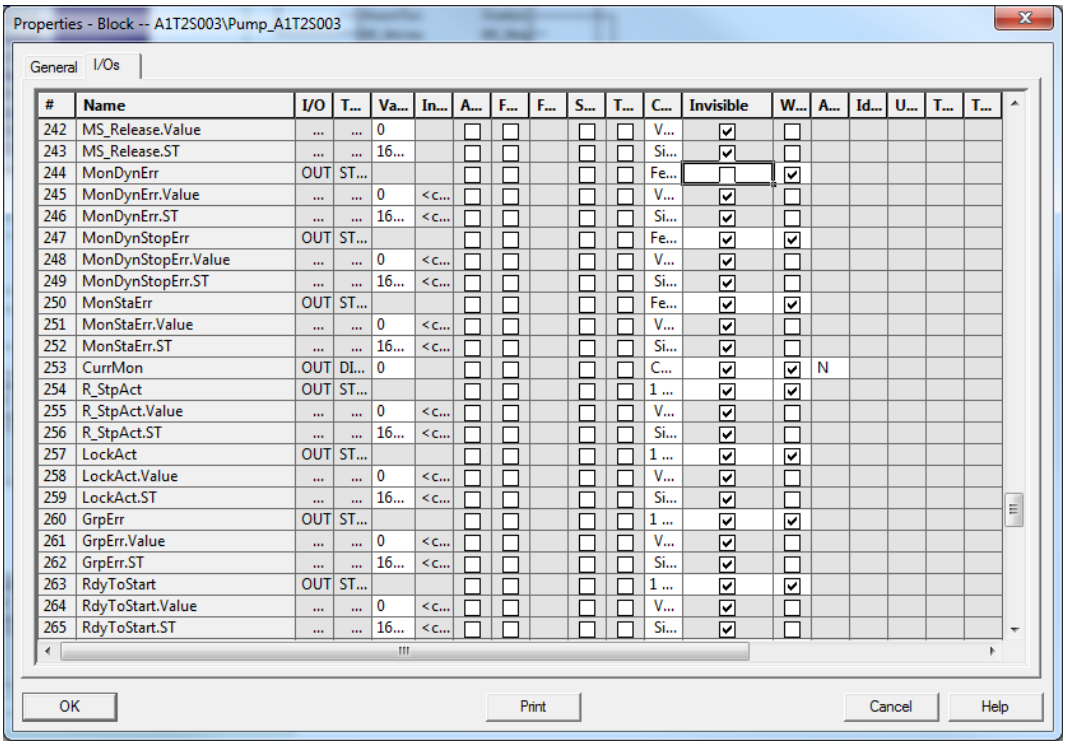


(→ RstOp)



#	Name	I/O	T...	Va...	In...	A...	F...	F...	S...	T...	C...	Invisible	W...	A...	Id...	U...	T...	T...
34	StopLocal.Value	IN	...	0	<c...	☐	☐		☐	☐	V...	☑	☐	N				
35	StopLocal.ST	IN	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
36	LocalSetting	IN	INT	0		☐	☐		☐	☐	Lo...	☑	☐					
37	FbkRun	IN	ST...		A1...	☐	☐		☐	☐	1=...	☐	☑					
38	FbkRun.Value	IN	...			☐	☐		☐	☐	V...	☑	☐					
39	FbkRun.ST	IN	...			☐	☐		☐	☐	Si...	☑	☐					
40	Monitor	IN	B...	1		☐	☐		☐	☐	1=...	☐	☑	N				
41	MonTiStatic	IN	RE...	3.0		☐	☐		☐	☐	M...	☐	☐	N				
42	MonTiDynamic	IN	RE...	10.0		☐	☐		☐	☐	M...	☐	☐	N				
43	MonTiDyStop	IN	RE...	3.0		☐	☐		☐	☐	M...	☐	☐	N				
44	IdleTime	IN	RE...	5.0		☐	☐		☐	☐	St...	☑	☐					
45	PulseWidth	IN	RE...	3.0		☐	☐		☐	☐	C...	☑	☐					
46	WarnTiMan	IN	RE...	0.0		☐	☐		☐	☐	W...	☑	☐					
47	WarnTiAut	IN	RE...	0.0		☐	☐		☐	☐	W...	☑	☐					
48	RapidStp	IN	B...	0		☐	☐		☐	☐	1 ...	☐	☑	N				
49	RstOp	IN	B...	0		☐	☐		☐	☐	O...	☐	☐	N				
50	RstLi	IN	ST...			☐	☐		☐	☐	Li...	☑	☑					
51	RstLi.Value	IN	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
52	RstLi.ST	IN	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
53	BypProt	IN	B...	0		☐	☐		☐	☐	By...	☑	☐	N				
54	Trip	IN	ST...			☐	☐		☐	☐	1=...	☑	☑					
55	Trip.Value	IN	...	1	<c...	☐	☐		☐	☐	Va...	☑	☐	N				
56	Trip.ST	IN	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
57	Permit	IN	ST...		A1...	☐	☐		☐	☐	1=...	☐	☑					

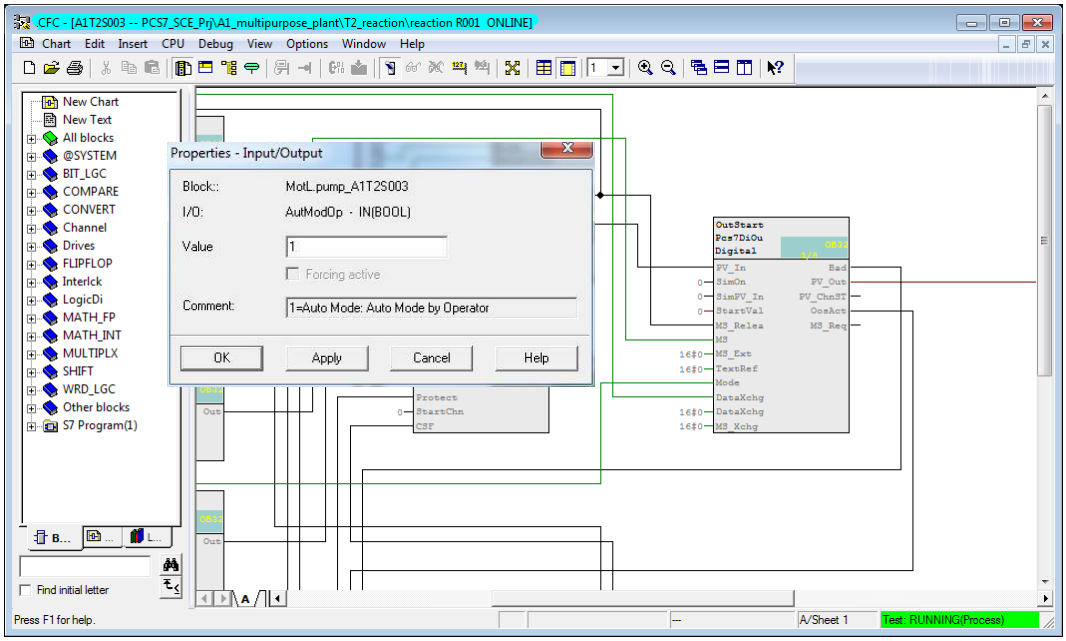
(→ MonDynErr)



#	Name	I/O	T...	Va...	In...	A...	F...	F...	S...	T...	C...	Invisible	W...	A...	Id...	U...	T...	T...
242	MS_Release.Value	...	...	0		☐	☐		☐	☐	V...	☑	☐					
243	MS_Release.ST	...	...	16...		☐	☐		☐	☐	Si...	☑	☐					
244	MonDynErr	OUT	ST...			☐	☐		☐	☐	Fe...	☐	☑					
245	MonDynErr.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
246	MonDynErr.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
247	MonDynStopErr	OUT	ST...			☐	☐		☐	☐	Fe...	☑	☑					
248	MonDynStopErr.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
249	MonDynStopErr.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
250	MonStaErr	OUT	ST...			☐	☐		☐	☐	Fe...	☑	☑					
251	MonStaErr.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
252	MonStaErr.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
253	CurrMon	OUT	DI...	0		☐	☐		☐	☐	C...	☑	☑	N				
254	R_StpAct	OUT	ST...			☐	☐		☐	☐	1 ...	☑	☑					
255	R_StpAct.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
256	R_StpAct.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
257	LockAct	OUT	ST...			☐	☐		☐	☐	1 ...	☑	☑					
258	LockAct.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
259	LockAct.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
260	GrpErr	OUT	ST...			☐	☐		☐	☐	1 ...	☑	☑					
261	GrpErr.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
262	GrpErr.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					
263	RdyToStart	OUT	ST...			☐	☐		☐	☐	1 ...	☑	☑					
264	RdyToStart.Value	...	...	0	<c...	☐	☐		☐	☐	V...	☑	☐					
265	RdyToStart.ST	...	...	16...	<c...	☐	☐		☐	☐	Si...	☑	☐					

40. Next, we switch to Automatic with 'AutModOp' == 1 for the manual/automatic mode.

    (→ AutModOp → Properties → "1")



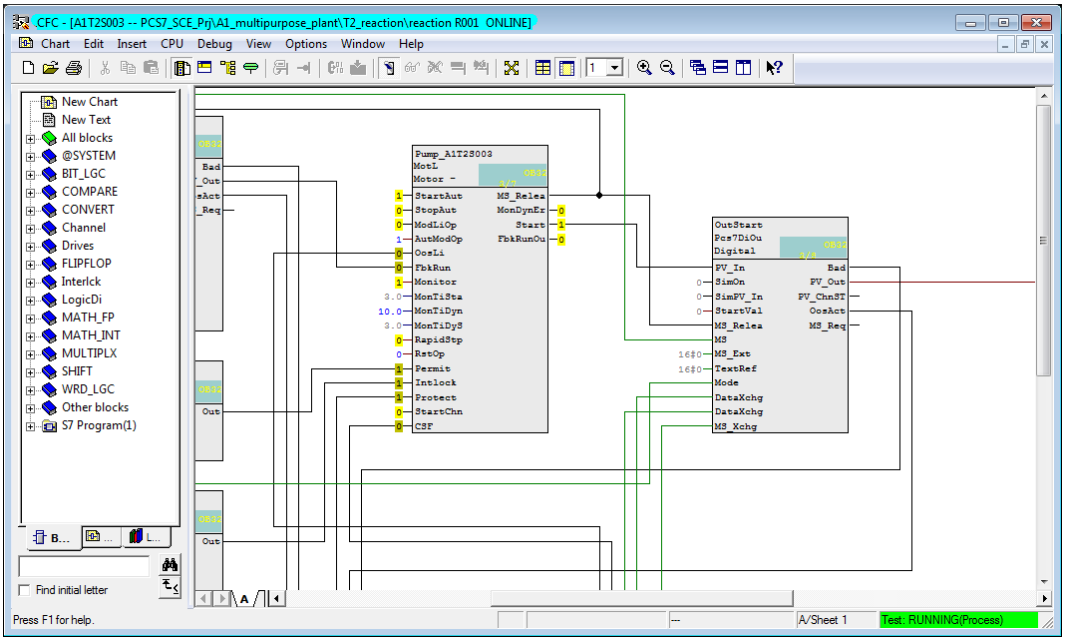
Note: During testing, we should not omit setting the feedback I1.3 within 10 seconds after activating the output Q 3.4 in ***S7-PLCSIM***. If this is not done, the block Pump_A1T2S003 shuts off and reads out an error.

Note: If there is no feedback, the pump block not only switches the actuating signal START to 0, but indicates by setting the output 'ModDynErr' to 1 that the run signal of the pump had not arrived on time. All connections have to be made visible. To prevent damage through repeated switch-on attempts, the pump block has to be reset before another attempt is made.

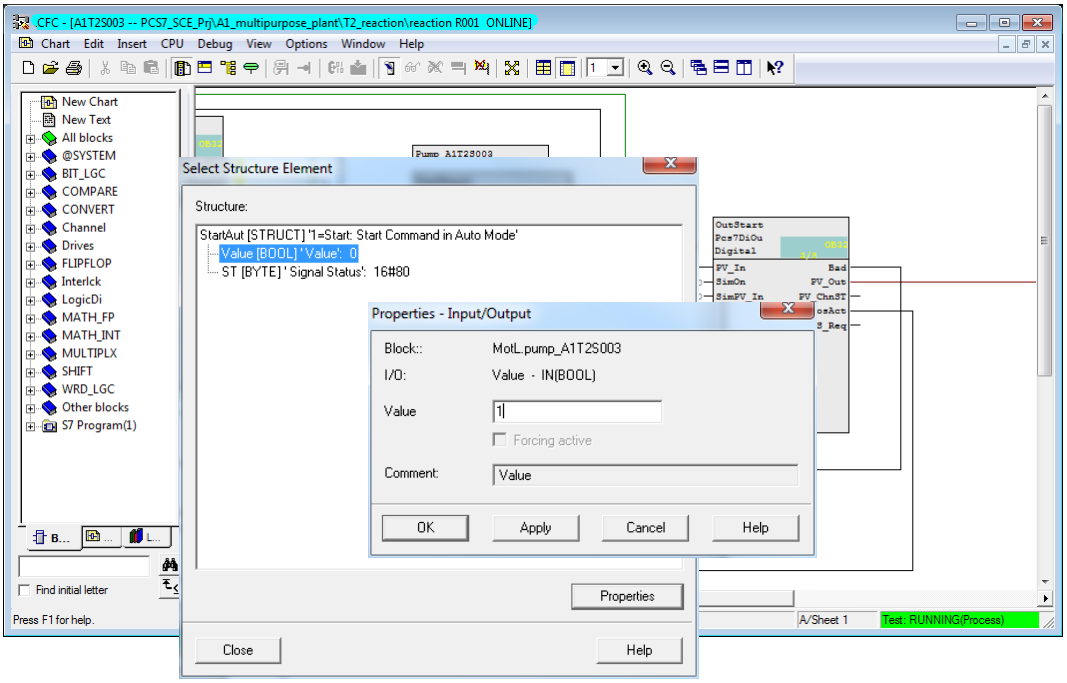
To do this, input 'RstOp' has to be briefly set to 1 and then again to 0!

By double-clicking on this input parameter of pump block Pump_A1T2S003, the dialog window shown above is opened. In the field Value, we first enter a 1; this value is transmitted to the control system by clicking on the button 'Apply' and the error outputs are set to 0. To return to the normal method of functioning, 'RstOp' must be reset to the original state by entering 0 and 'Apply' once more.

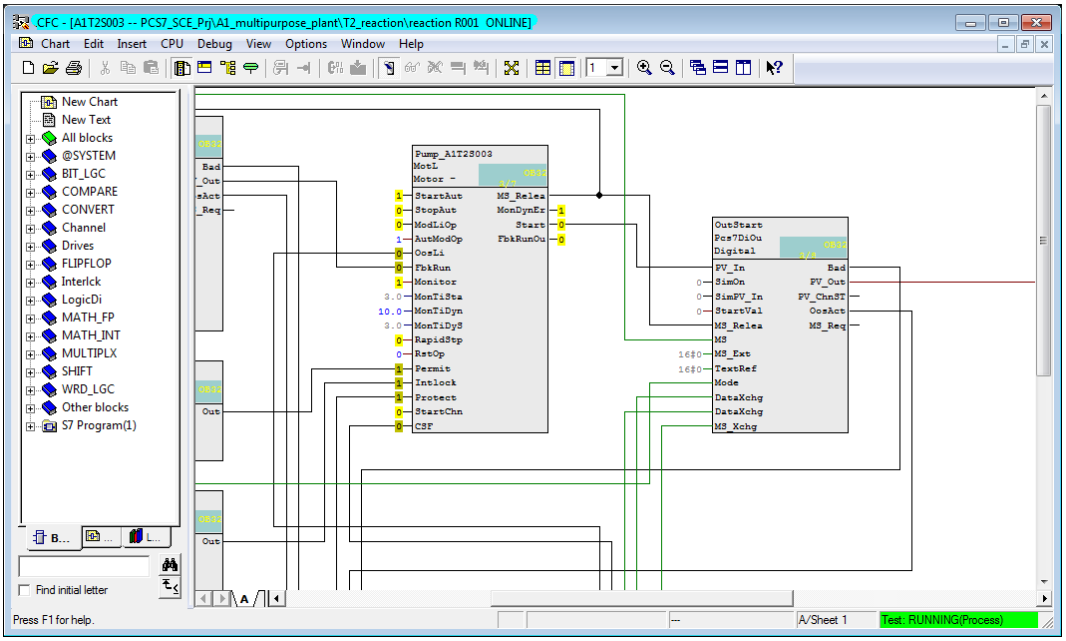
41. Next, the pump is started with (StartAut == 1 and StopAut == 0) and can be stopped again with (StartAut == 0 and StopAut == 1).



(→ StartAut → Properties → Value "1")



42. If the run feedback was not switched on in time in the PLCSIM, an error is displayed in 'MonDynErr'. It can be acknowledged with RstOp' == 1.



TESTING THE AUTOMATION LOGIC WITH THE SIMULATION

Entering process states manually in the simulated control system is still possible with justifiable effort for small functions. For more complicated runs with several dynamic process variables, the limits of what is doable are quickly reached. Here, using process simulation is recommended.

For this course, the essential relationships of the process that we are automating here were mapped with the simulation software ***SIMIT***. The model maps the dynamic behavior of the pumps, valves, containers, reactors as well as the local operator panel with main switch, EMERGENCY OFF, switch-over to local operation and the corresponding operator controls. The dynamic processes have been accelerated in comparison to reality by a factor 5 to 50 to keep waiting times short.

The simulator operator interface is shown in Figure 4. On the left side, it shows the process scheme as well as the signal levels of controlled and measured variables. On the right side, the ochre-shaded local operator panel is shown on top, below that a series of control elements is provided for the simulation.

Using the simulator is simple; we only have to make sure that the assignment of the inputs and outputs was not changed.

With the simulator, pump activation can now be checked very simply:

1. After ***S7-PLCSIM*** the simulation program is started.
2. The simulation starts with the 75% filled educt tanks; all others are drained. This state can be restored in the simulation control any time with the option 'RESET'. The option 'RESET 50%' fills all tanks as shown in Figure 4 to 50%.
3. For testing, the motor block is activated as described in the previous section under Step 41. In the simulation, the actuating signal of the pump is illuminated green.
4. The simulated motor startup takes about 2 seconds. Then, the motor's running indicator flashes green in the simulation, and the signal level for binary input I 1.3 of the control system is set.
5. To open the supply path, the valve for product tank T3.B001 has to be opened as well. In the exercise below, this valve is activated by a suitable individual drive function. If the

pump is switched on and the valve is open, we can watch in the simulation how the contents of reactor T2.R001 are pumped into product tank T3.B001.

6. Using the simulation controls, the product tanks can be drained and the educt tanks can be filled. When the tanks are filled using the simulation control, first the simulation has to be separated from the control system. To this end, the button with the horizontal line ( ) is clicked once. Then, the valve can be opened with the button to the right of it. The actuating signal flashes green. After approximately one second, the signal of the limit switch for the open position follows. After an additional 5 to 10 seconds, the first level changes are visible.
7. With 'RESET' and 'RESET 50%', the simulation can be returned to a defined state.

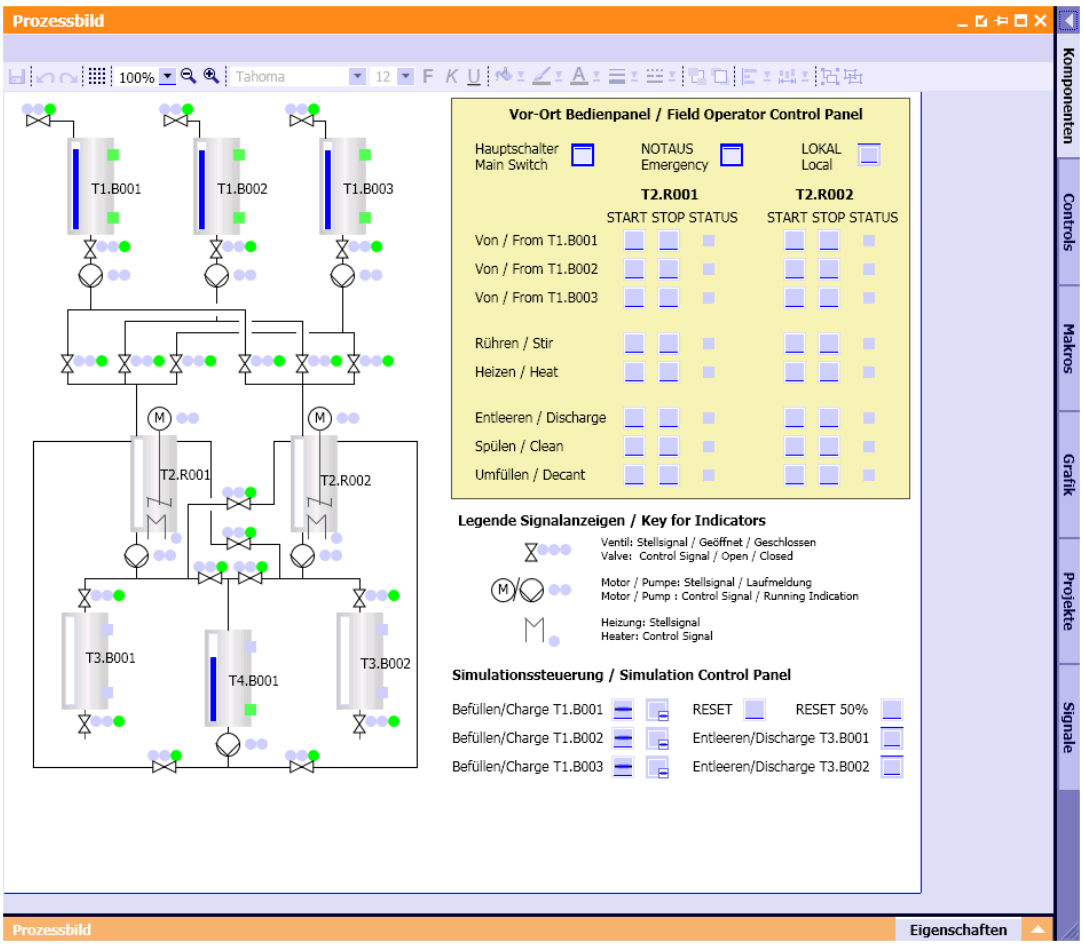


Figure 4: User interface of the process simulation

EXERCISES

In the exercises we apply what we have learned from theory and from the step by step instructions. To this end, we utilize and expand the available multi-project from the step by step instruction (PCS7_SCE_0104_R1503_en.zip).

The objective of this exercise is to create a CFC that can be used for controlling the valves of the plant. The exercise is based on the knowledge acquired through the step by step instructions where a similar CFC for controlling the motor was created.

In addition, a CFC is created for scaling the level, which means an analog input value, from the digitalized to the physical value. This task adds to our knowledge because the CFC is created without a template but still with a block from the library. This CFC is required for the chapter 'Functional Safety'.

TASKS

The following exercises are based on the step by step instructions. The corresponding steps of the instructions can be used for each exercise as an aid.

1. Insert the template 'ValveLean' as template in the process tag types (analogous to 'Motor_Lean' ‘). This template is used to implement the valves.
2. In the chart folder 'Product container B001' of unit T3_Product_memory, an object instance of the valve template is to be inserted and renamed to A1T3X001. Open the CFC and also adapt the name of the block 'VlvL'. Now, close the feedback and control signals (see Figure 5 and Table 4).
3. Download and test your implementation with the SIMIT model. The following connections should be visible for this: 'ModLiOp', 'AutModOp', 'ManModOp', 'OpenAut', 'CloseAut' and 'RstOp'.
4. To incorporate the analog level sensor A1T2L001 (see Figure 5), create a new CFC in the chart folder 'Reactor R001'. Name it A1T2L001 and open it. Drag the block 'Pcs7AnIn' (FB1869) from the catalog to the CFC. To do this, select the tab Libraries in the left frame and then use either the search function at the very bottom or open PCS 7 AP Library V81/Blocks+Templates\Blocks/Channel. As soon as the block has been inserted, rename it Level_A1T2L001.
5. Now assign parameters to the block 'Pcs7AnIn' by setting the input value 'Scale' to High = 0 and Low = 1158.0, and the 'PV_InUni' to 1040 (for the unit ml). Connect the input 'PV_In' of block 'Pcs7AnIn' with the symbol for the actual level value (see Table 4) of Reactor R001.
6. Now implement the high and low level sensor of product container B001. For this, set up a CFC in the chart folder 'Product_tank B001' and name it A1T3L001. Open the chart and insert the block 'Pcs7DiIn' twice from the catalog (analogous to exercise 5). Name one block A1T3L001_LSA+ and the other A1T3L001_LSA-. Interconnect 'PV_In' with the sensor signals.
7. Next, create the CFCs for the main switch, the EMERGENCY OFF switch and the switch for local operation. As in exercise 7, a CFC for A1H001, A1H002 and A1H003 is created in the chart folder 'Multi-purpose plant'; the block 'Pcs7DiIn' is added, named, and interconnected with the respective address to 'PV_In'.

Table 4: Symbols for implementation of the valve control and the fill sensor

		Symbol	Address	Data Type	Comment
Exercise	2	A1.T3.A1T3X001.XV.C	Q 0.6	BOOL	Open/close valve inflow Product tank B001 Actuating signal
		A1.T3.A1T3X001.GO+-.O+	I 15.4	BOOL	Open/close valve inflow Product tank B001 Feedback open/on
		A1.T3.A1T3X001.GO+-.O-	I 15.5	BOOL	Open/close valve inflow Product tank B001 Feedback closed
	5	A1.T2.A1T2L001.LISA+.M	IW 512	WORD	Actual level value Reactor R001
	6	A1.T3.A1T3L001.LSA+.SA+	I 18.6	BOOL	Level monitoring Product tank B001 Trip point H
		A1.T3.A1T3L001.LSA-.SA-	I 18.7	BOOL	Level monitoring Product tank B001 Trip point L
	7	A1.A1H001.HS+-.START	I 0.0	BOOL	Switch on multi-purpose plant
		A1.A1H002.HS+-.OFF	I 0.1	BOOL	Activate EMERGENCY OFF (NO contact)
		A1.A1H003.HS+-.LOC	I0.2	BOOL	Activate local operation

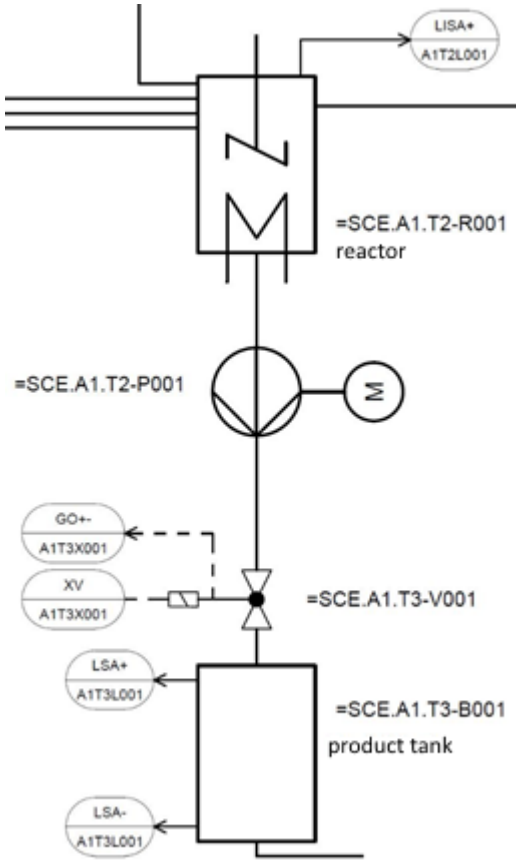


Figure 5: Excerpt from P&ID flowchart