MOELLER

Training Guide

Sucosoft S40
Programming Software

02/02 AWB27-1307-GB

1st edition 12/97
2nd edition 07/98
3rd edition 06/99
4th edition 02/02
See list of revisions on inside rear cover

© Moeller GmbH, Bonn

Author: Arno Dielmann, Jürgen Herrmann, Peter Stammerjohann
Editor: Annette Scholz, Thomas Kracht
Translator: David Long, Bonn

Contents

Introduction

Devices and software

The PS4 series compact controllers are a powerful range of multi-purpose devices for small to medium size automation tasks. The advantages are space-saving design, a comprehensive range of basic features as well as remote and with some types local expandability. These features enable PS4 series controllers to be used for almost all areas of automation engineering: ranging from simple custom applications via machine controllers installed directly on the machine, for intelligent pre-processing, through to networked systems e.g. for building automation applications or for control of remote stations, for example, in a sewage treatment plant.

The PS416 series controllers are a modular multi-purpose system for medium to large size automation tasks. The advantages are space-saving design, short processing times and the large memory. The modular concept with various CPUs, sizes and a wide range of plug-in cards provide more flexibility for custom automation applications. Due to the high processing speed, the PS416 system is often used where a lot of data needs to be processed quickly.

All programmable controllers are programmed with Sucosoft S40. The Sucosoft S40 is a programming software that meets the requirements of the international IEC/EN 61131-3 standard and provides a comprehensive instruction set in the four programming languages Instruction List (IL), Ladder Diagram (LD), Function Block Diagram (FBD) and Structured Text (ST). You can changeover between the AWL, KOP and FBS programming languages, allowing you to use and combine the most suitable language to suit your requirement and personal preference. ST can be combined with IL-sequences with completed instructions, but not with LD or FBD graphic elements. The programmable controllers process all standard elementary data types and derived data types. Besides the wide range of IEC standard functions and function blocks, Moeller functions and function blocks are also provided. These can be added to your own user-defined function blocks.

Documentation

This manual is designed to provide a fast introduction to the Sucosoft S40 programming package. You will learn to work with the Sucosoft S40 by means of a simple control task: You will be guided through all steps required from creating an individual project to testing the program which has been created. After code generation, the complete example program will be transferred to the controller and tested. You will thus get to know the different possibilities provided by the Test and Commissioning tool. A further programming example illustrates how to use the function blocks provided with the Sucosoft S40 programming software and how to program your own, user-defined function blocks.

You will need approximately two hours for the first programming example and approximately three hours for the second example.

☞ You can find a complete description of all the programming possibilities for PS4 and PS416 controllers in the reference manuals AWB2700-1305-GB “Programming Software S40, User Interface” and AWB2700-1306-GB “Programming Software S40, Language Elements for PS4-200, PS4-300 and PS416”.

Writing conventions

Select “Project → New” means choose the New command in the Project menu.

Italic lower-case letters indicate texts which you must enter exactly as shown.
Example: *c:\projects\example*

Instructions for actions you need to perform are marked with an arrow ▶. All other sections merely provide information and no action is required on your part.

☞ Information and tips provide you with additional notes concerning the topic and include references to other sections of interest.

Hardware requirements

PS4

For the first example you will require a PS4-141-MM1, PS4-151-MM1, PS4-201-MM1, PS4-271-MM1 or PS4-341-MM1 controller. For the second example you will also need an EM4-101-DD1 expansion module and a KPG1-PS3 connecting cable. A ZB4-303-KB1 programming cable (the cable connecting PC and PS4) is required for both examples.

PS416

For the example programs you will require a PS416 basic unit fitted with a power supply card, a CPU-300 or CPU-400, a digital input card and a PS416-OUT-400 digital output card. You will also need a PS416-ZBK-210 programming cable or a UM 1.5 interface converter. Other cards which may already be inserted in the rack are not needed for the programming example and will not be described here.

1 Installing and Wiring the PS4

Address coding

Set the station address of the EM4-101-DD1 with the address coding switch S2 before connecting the EM4-101-DD1 to the PS4-141-MM1. The program examples assume station address 1, which is selected as follows (see also AWB27-1257-GB):

S2	1	2	3	4	5	6	7	8
	1	0	1	1	1	1	0	1

1 = ON, 0 = OFF

Bus terminating resistors

The bus terminating resistors must be enabled for the physically first and last stations on the bus. Switch S1 of the EM4-101-DD1 must thus be set as follows:

S1: 1 = ON
2 = ON

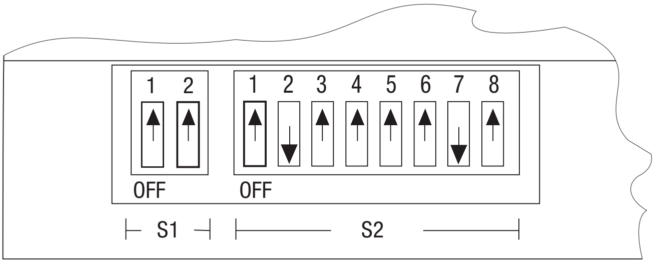


Figure 1: Switch on bus terminating resistors

The following figure shows the wiring of the PS4-201-MM1, PS4-141-MM1 or PS4-341 and the EM4-101-DD1.

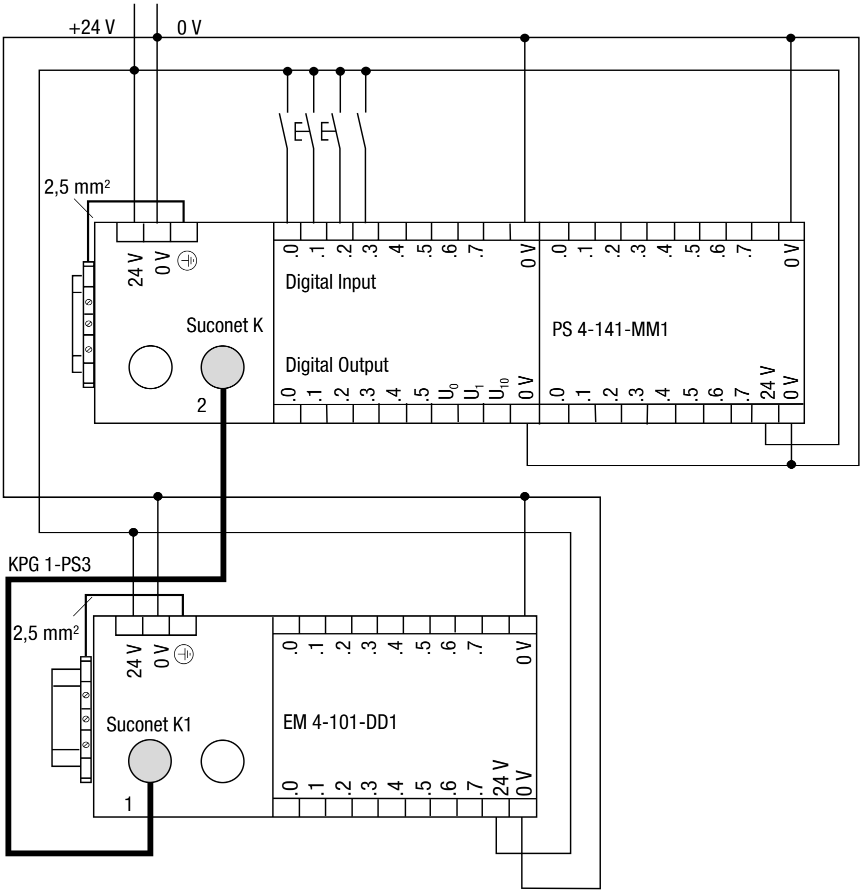

+24 V
0 V
2,5 mm²
24 V
0 V
.0 .1 .2 .3 .4 .5 .6 .7 0 V
Digital Input
Suconet K
Digital Output
.0 .1 .2 .3 .4 .5 U0 U1 U10 0 V
2
.0 .1 .2 .3 .4 .5 .6 .7 0 V
PS 4-141-MM1
.0 .1 .2 .3 .4 .5 .6 .7 24 V 0 V
KPG 1-PS3
2,5 mm²
24 V
0 V
Suconet K1
1
.0 .1 .2 .3 .4 .5 .6 .7 0 V
EM 4-101-DD1
.0 .1 .2 .3 .4 .5 .6 .7 24 V 0 V

Figure 2: PS4 – EM4 wiring example

2 Installing and Wiring the PS416

Address coding

Before inserting the digital input card and the PS416-OUT-400 digital output card in the rack, you must set the card addresses with the DIP switches. For the examples in this manual the digital cards assume a byte address of “0” for digital I/O for both inputs and outputs, which is selected as follows (see also AWA 27-1304):

S1	S2	S3	S4	S5	S6	S7	S8
1	1	1	1	1	1	x	x

1 = ON, 0 = OFF, x = not used

Inserting the cards

Insert the power supply card in slots 0 and 1 on the extreme left of the rack. The PS416-CPU should be inserted adjacently, in slots 2 and 3. The digital I/O cards PS416-INP-40x and PS416-OUT-400 should be inserted in slots 4 and 5.

The following figure shows the wiring of the PS416. You can use any PS416-BGT series rack for the example.

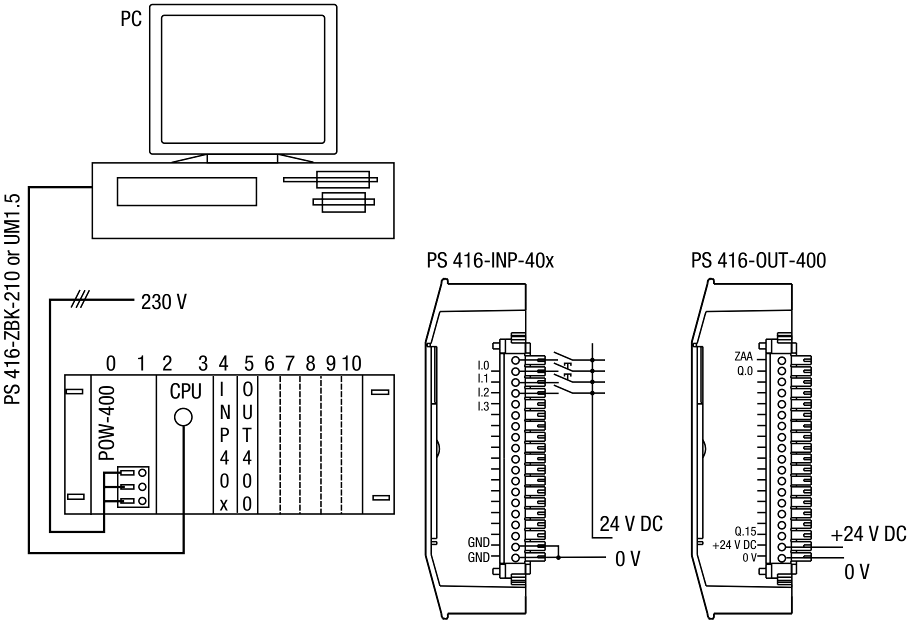


Figure 3: PS416 wiring example

Connect up the 230 V AC or 24 V DC power supply depending on the power supply card used. The LEDs to indicate the status of the outputs are provided with 24 V DC through the plug-in screw terminals on the PS416-OUT-400. Inputs 1 and 2 of the PS416-INP-40x are connected to buttons, and inputs 0 and 3 are connected to switches which are all supplied with 24 V DC.

Set the switch of the CPU programming interface to RS 232 if you want to use the PS416-ZBK-210 programming cable. If you want to use the UM1.5 interface converter, set the switch to the RS 485 position.

3 Programming Task 1

After studying the automation task, you will be shown how to write it in an IL program. You will then be guided through the Sucosoft S40 steps required to:

Create a project,

Create the program file,

Specify the configuration,

Generate the program code,

and carry out test and commissioning.

Automation task

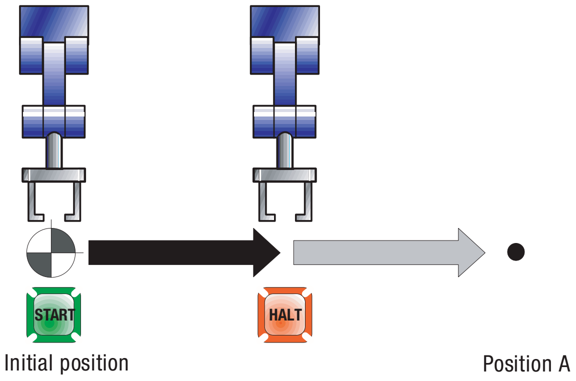


Figure 4: Automation task

When the system is in the initial position, pressing the START button should move the robot arm to position A. The HALT button is used to interrupt the action.

Basic information on programming to IEC/EN 61131-3

As specified by IEC/EN 61131-3, the variables used in the program must be declared between two keywords. Different keywords are used depending on the scope. The local variables which are used in the first programming task should be declared between the keywords VAR and END_VAR.

The instruction section of a program organisation unit (POU) is always placed after the declaration section. The entries required in the declaration section depend on the scope. For local variables, you must specify the variable name,
the data type and, if appropriate, the physical address, an initial value, an attribute and a comment. Variables used as intermediate markers are declared symbolically, i.e. without entering an address.

Binary input and output operands, i.e. variables assigned to physical addresses, are called to EC/EN 61131-3. Conventions exist for directly represented variables which must be observed when specifying the address:

Each directly represented variable must be prefixed with the keyword AT which is followed by a percent sign, an identifier (I = Input, Q = Output) and a 5-digit address. The first three digits of the address are the device address which is specified in the topology configuration. With a control system which only consists of a controller without further stations, the first three digits are always 0. The last two digits identify the byte and bit address.

Example: AT %I0.0.0.0.3 identifies the input operand bit 3 in byte 0.

The example task only uses binary input/output variables. The following addresses are used:

Initial position:	Input 0.0
Start button:	Input 0.1
Stop button:	Input 0.2
Position A:	Input 0.3
Motor:	Output 0.0

The variable names are freely selectable. The variable declaration appears as follows:

```
VAR
  initial_pos     AT %I0.0.0.0.0:BOOL;     (*Initial position of machine*)
  start_button    AT %I0.0.0.0.1:BOOL;     (*Start button*)
  stop_button     AT %I0.0.0.0.2:BOOL;
  position_A      AT %I0.0.0.0.3:BOOL;     (*Position A reached*)
  motor           AT %Q0.0.0.0.0:BOOL;
END_VAR
```

The binary input operand “initial_pos” is read and ANDed with the “start_button.

Read command: LD
AND sequence: AND

If the current result (CR) = 1, the “Motor” output operand is set to “1”. (The LED on the “motor” output is lit).
Set command if CR = 1: S

The input operands “stop_button” and “position_A” are read and ORed. If the current result = 1, the output operand “motor” is reset

(The LED on the output “motor” goes out.)
Reset command if CR = 1: R

The instruction list then appears as follows:

```
LD   initial_pos
AND  start_button
S    motor       (*switches motor on*)
LD   stop_button
OR   position_A
R    motor       (*switches motor off*)
```

To identify the program, it must have the PROGRAM keyword and the program name in front of the declaration block. In addition, the end of the instruction section must be followed by the END_PROGRAM keyword to indicate the end of the program.

The program example has the following form:

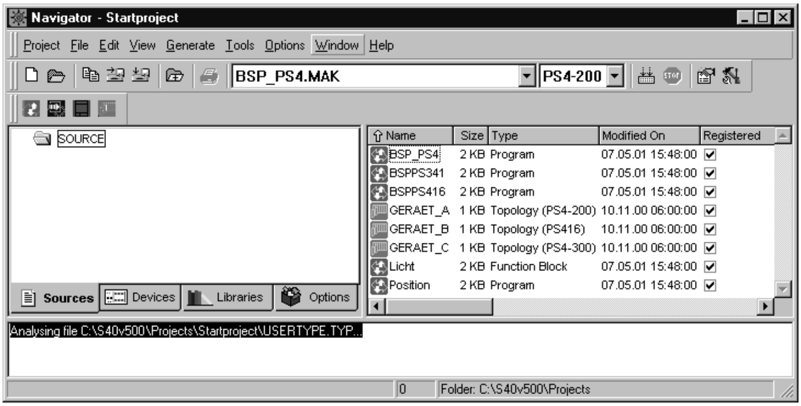
```
PROGRAM position
VAR
  initial_pos     AT %I0.0.0.0.0:BOOL;     (*Initial position of machine*)
  start_button    AT %I0.0.0.0.1:BOOL;     (*Start button*)
  stop_button     AT %I0.0.0.0.2:BOOL;
  position_A      AT %I0.0.0.0.3:BOOL;     (*Position A reached*)
  motor           AT %Q0.0.0.0.0:BOOL;
END_VAR
  LD   initial_pos
  AND  start_button
  S    motor         (*Switches motor on*)
  LD   stop_key
  OR   position_A
  R    motor         (*Switches motor off*)
END_PROGRAM
```

Do not enter the first and last lines of the program containing the PROGRAM and END_PROGRAM keywords. This is done automatically be the POU-EDITOR.

4 NAVIGATOR

Overview

Start Windows and double-click on the Navigator icon. The Sucosoft S40 starts and the NAVIGATOR window opens:



Name	Size	Type	Modified On	Registered
BSP_PS4	2 KB	Program	07.05.01 15:48:00	✓
BSPPS341	2 KB	Program	07.05.01 15:48:00	✓
BSPPS416	2 KB	Program	07.05.01 15:48:00	✓
GERAET_A	1 KB	Topology (PS4-200)	10.11.00 06:00:00	✓
GERAET_B	1 KB	Topology (PS416)	10.11.00 06:00:00	✓
GERAET_C	1 KB	Topology (PS4-300)	10.11.00 06:00:00	✓
Licht	2 KB	Function Block	07.05.01 15:48:00	✓
Position	2 KB	Program	07.05.01 15:48:00	✓

```
Analysing file C:\S40v500\Projects\Startproject\USERTYPE.TYP...
```

```
0    Folder: C:\S40v500\Projects
```

Figure 5: NAVIGATOR window

The Navigator window contains the icons used to activate the individual tools.

The Navigator helps you perform all of the steps you need to take, from creation of a user program through to the program's execution in the PLC:

It includes the project management tool which you use to create and structure projects.

It allows you to access the POU Editor so that you can edit your POUs (programs, function blocks and functions). It allows you to access the topology configurator which you use to define the hardware configuration.

It includes the Code Generator which is used to compile your program into an executable program for the specified controller.

You can directly access the Test & Commissioning tool which allows you to transfer your program to the PLC and carry out tests.

It provides you with the Form Editor which is used to view and edit standard forms.

In the main window of the Navigator, the title bar appears at the top and the status bar at the bottom. Below the title bar is the menu bar, followed by the icon bar and then the toolbar next to it. The area between is divided into three windows:

The top left window (browser window) contains the “tree structure” with the respective folders. The three tabs “Sources”, “Devices” and “Libraries” form the bottom edge of this window.

In the right hand window is the “File” window.

Beneath this window is the “Output” window for status and error messages, e.g. during generation of the program code.

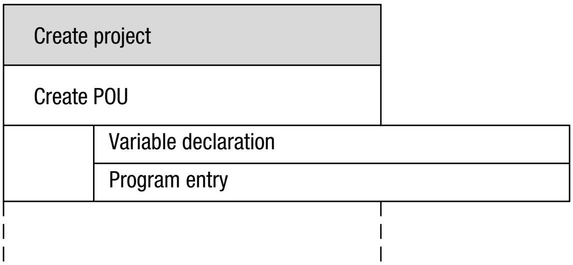
You can change the size of the window by dragging the parting line with the mouse.

Overview

The diagram below shows an overview of the steps you will follow in program task 1, from program entry to the test run of the complete program.
The overview will be helpful for the following sections.

- Create project
- Create POU
  - Variable declaration
  - Program entry
- Topology configuration of the system components
- Program code generation
- Test and commissioning
  - Transfer program to PLC
  - Start program
  - Test program

5 Creating a New Project



In our example, the following path will be used:

C:\PROJECTS\EXAMPLE

To create a new project, you select ‹ Project → New...› in the menu or click on the corresponding toolbar button.

Creating a New Project

The “Create New Project” dialog box opens:

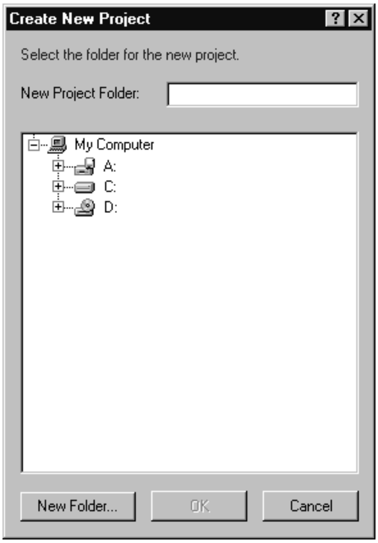


Figure 6: Create new project

▶ You first select the drive, in this case “C”.

In this example, you will store your project in the folder “Projects”. You create this folder as follows:

▶ Click on the “New folder” button then enter the new folder name in the “New folder” dialog box. Confirm with OK.

▶ Now enter “Example” for the name of the new project in the “New project folder” input field. Confirm with OK.

Sucosoft S40 then creates a project structure with the folders “Devices” and “Source”.

The title bar displays “Example” as the name of the new project and the status bar displays “C:\Projects”, indicating the drive and path of the project.

☞ Before you start creating your POU, make sure the right type of PLC is selected. The PLC type is displayed in a selection window in the toolbar of the Navigator. The data types and manufacturer function blocks or functions the POU Editor provides you with depend on the PLC type you select.

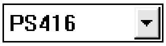

PS416

Figure 7: Selection window for the PLC type

Select PLC type
The following controllers are grouped under the three types of PLC you can select:

PS4-200:

PS4-141-MM1,
PS4-151-MM1,
PS4-201-MM1,
PS4-271-MM1

PS4-300:

PS4-341-MM1

PS416:

PS416-CPU-200,
PS416-CPU-300,
PS416-CPU-400

▶ Select the required PLC type in the selection window within the toolbar

6 Creating a Program

User programs, which you create with the POU Editor, can consist of one or more POUs (files). The term “POU” according to the international standard IEC/EN 61131-3 represents Program Organisation Unit and designates the three POU types, which are “Program”, “Function module” and “Function”. “Function” or “function module” is selected for the program section which is used most often.

All of the POUs (files) you store are automatically registered under the current project by the Navigator and placed in the “Source” folder. Also stored in this folder are the topology configuration files you create with the topology configurator. All files of type Topology and POU (program, functions and function blocks) are then displayed in the “File view” window of the Navigator.

You can enter the basic settings of the POU Editor when creating a new POU, including the programming language you wish to use in the POU Editor when it opens. You enter these settings under Options → Settings... → Instruction Section.

In the description below, the buttons of the standard toolbar are used. Activate this toolbar – if necessary – via ‹ View → Toolbar› .

Context-sensitive menus
Context-sensitive menus help to make the programming task easier for you. These are short menus containing the most important commands for a specific Sucosoft S40 function.

You open context-sensitive menus by clicking with the right mouse button on the object or window you are dealing with. The content of the menu depends on the environment, the context where the mouse pointer is positioned and on the selected element.

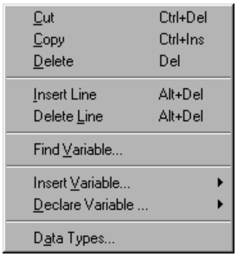


Figure 8: Context-sensitive menu for declaration section in syntax mode

For our example, you will need an executable program. The POU type Program must therefore be selected.

Use the Navigator menu to create the user program:

▶ Select ‹ Tools → POU editor› or the respective button in the toolbar.

POU Editor in Offline mode

The POU Editor opens.

▶ Select the – required POU type “Program” corresponding with – the ‹ File → POU new → Program› or the “P” button in the standard toolbar.

“Program”

When you create a new POU program, the POU Editor asks if you want to create variables or declare variables from a topology. The physical PLC addresses which you have defined in the topology configurator are accepted in the declaration section of the POU program and assigned with the “Global” scope. You only need to assign the variable names.

As this function reduces the editing effort required, it is useful to read Section 7 now. Generate your topology example and answer the question with “yes”.

The two windows “Declaration and Instruction section” open, arranged on the screen as specified in the basic settings of the POU Editor.

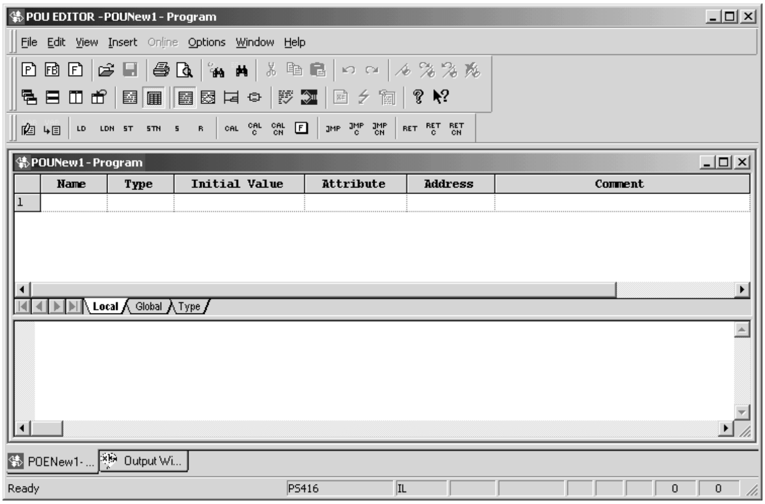

POU EDITOR -POUNew1 - Program
File Edit View Insert Online Options Window Help
LD LDN ST STN S R CAL CALC CALCN JMP JMPC JMPCN RET RETC RETCN
POUNew1 - Program
Name Type Initial Value Attribute Address Comment
Local Global Type
POENew1-... Output Wi...
Ready PS416 IL 0 0

Figure 9: POU Editor in Syntax mode

You enter the program in two stages:

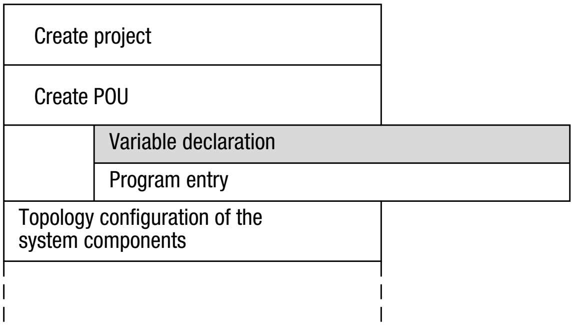
- In the declaration section, you declare the variables you want to use in the instruction section.
- In the instruction section, you create your user program.

To enter your variables in the declaration section, Sucosoft S40 provides you with a table-oriented, Syntax-controlled Variable Editor Syntax-controlled Variable Editor (Syntax mode). Declaration of the variables in predefined input fields is made easier for you with user guidance. There is also a pure text-based, free variable editor (Free mode) without user guidance for experienced PLC programmers. Beginners are advised to declare the variables in Syntax-controlled mode since you will not have to worry about entering keywords and the syntax.

For creating your user program, you can choose between the programming languages Instruction List (IL), Structured Text (ST) or one of the graphic programming languages, ladder diagram (LD) or function block diagram (FBD). We will create the example program in IL.

You are free to decide whether to first declare the variables or write the instruction list section. You can also carry out both at the same time. For this example, we will first declare the variables and then enter the instruction section.

Declaring the variables



You enter the variables, separately for each scope, under different tabs. With the selected “Program” POU type, only the tabs of the permissible scope, i.e. “Type”, “Local” and “Global” are available.

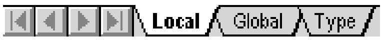


Figure 10: Scope for the “Program” POU type

You select the scope by clicking on one of the tabs visible at the bottom of the window.

Commence the variable declaration with the “Local” preset scope. You do not need to click on the tab in this case. The window only displays cells which are permissible for this scope. Now declare all of the variables of this type.

Should you create a POU that contains other scopes, you select the relevant tab for the other types and declare these variables. The list always displays variables of the same type.

You enter the variables by completing the cells in the editing line. You can switch between cells with

Cursor left/right,
TAB/Shift + TAB,
Cursor up/down,
Page up/Page down or
by mouse click.

Errors in variable declarations are determined by the syntax test or when saving the program and displayed in the error protocol window. In this case, double click on the error message and you will be transferred automatically to the error. Correct the error.

As mentioned earlier, the program example only uses "local" scope variables.

▶ To declare the first variable in the program example make the following entries in the order shown.

Name: initial_pos

Type: Select the variables of the data type under ‹ Insert → Variable declaration› or use the context-sensitive menu via the right-hand mouse button. Mark the "data type" group and select the required type from the list which appears below.

Initial value: An entry is not necessary – so that the "initial_pos" variable is assigned the value "0".

Attribute: No entry.
You can choose RETAIN for a retentive variable or CONSTANT for a constant. These entries are not meaningful for input/output variables (I/Q).

Address: I0.0.0.0.0
Physical address of the “initial_pos” variable.

Comment: Can be entered if required. Enter “System initial position” in this case.

▶ Finish the entry in this line by pressing the Enter key.

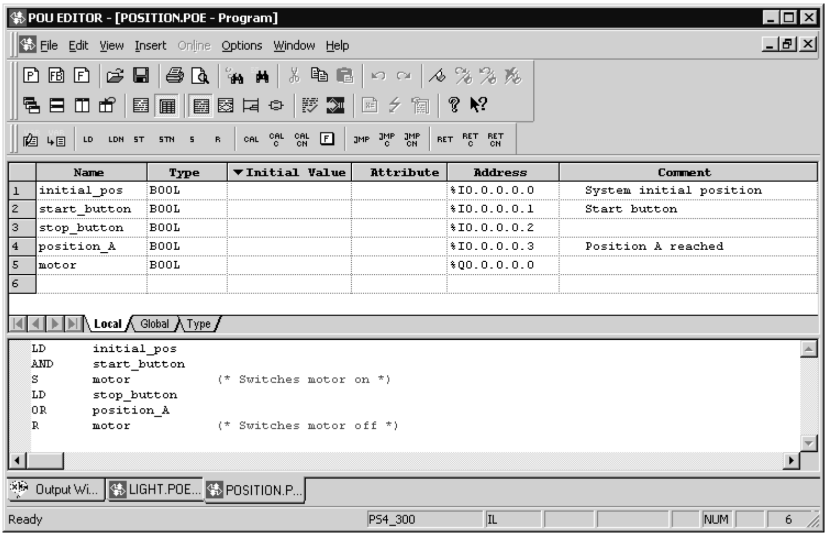
The fully defined variable will appear as follows:

Name	Type	Address	Comment
initial_pos	BOOL	I0.0.0.0.0	System initial position

▶ Use the same method to declare the remaining variables of the example program.

Name	Type	Address	Comment
start_button	BOOL	I0.0.0.0.1	Start_button
stop_button	BOOL	I0.0.0.0.2	
position_A	BOOL	I0.0.0.0.3	Position A reached
motor	BOOL	Q0.0.0.0.0	

The variable list is then complete.


POU EDITOR - [POSITION.POE - Program]
File Edit View Insert Online Options Window Help
Local Global Type
Output Wi... LIGHT.POE... POSITION.P...
Ready PS4_300 IL NUM 6

	Name	Type	Initial Value	Attribute	Address	Comment
1	initial_pos	BOOL			%I0.0.0.0.0	System initial position
2	start_button	BOOL			%I0.0.0.0.1	Start button
3	stop_button	BOOL			%I0.0.0.0.2	
4	position_A	BOOL			%I0.0.0.0.3	Position A reached
5	motor	BOOL			%Q0.0.0.0.0	
6						

```
LD      initial_pos
AND     start_button
S       motor           (* Switches motor on *)
LD      stop_button
OR      position_A
R       motor           (* Switches motor off *)
```

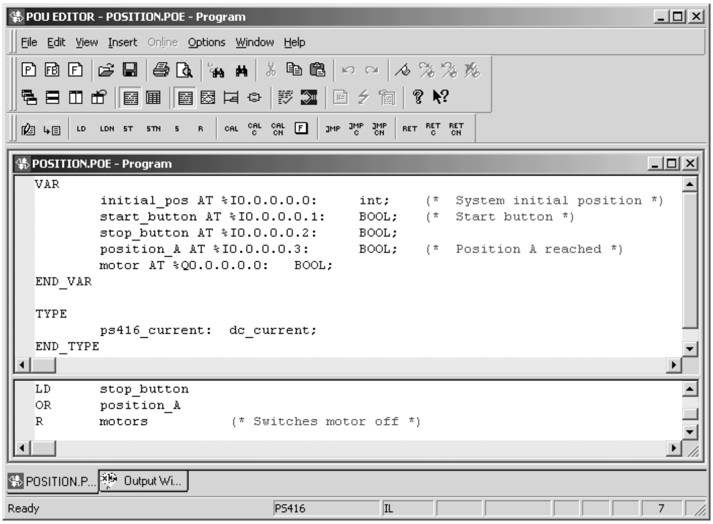
Figure 11: Declared variables in Syntax Mode

The completed variable declaration can be viewed under “Free mode”:

▶ Select ‹ Options → Variable Editor → Free Mode› or the button in the standard toolbar.

“Free Mode”

The Editor window appears on the screen with the variable declaration which you created in the Syntax-Controlled Variable Editor:


POU EDITOR - POSITION.POE - Program
File Edit View Insert Online Options Window Help
POSITION.POE - Program
POSITION.P...
Output Wi...
Ready
PS416
IL
7

```
VAR
        initial_pos AT %I0.0.0.0.0:     int;    (*  System initial position *)
        start_button AT %I0.0.0.0.1:    BOOL;   (*  Start button *)
        stop_button AT %I0.0.0.0.2:     BOOL;
        position_A AT %I0.0.0.0.3:      BOOL;   (*  Position A reached *)
        motor AT %Q0.0.0.0.0:   BOOL;
END_VAR

TYPE
        ps416_current:  dc_current;
END_TYPE
```

```
LD      stop_button
OR      position_A
R       motors          (* Switches motor off *)
```

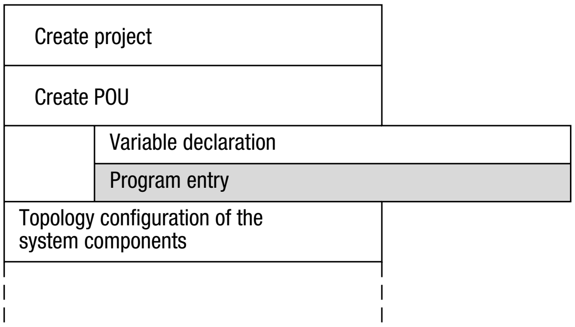
Figure 12: Declared variables in Free Mode

In the Free Variable Editor, the declared variables are represented in the form of declaration blocks as described in the IEC Standard, with the keywords VAR, END_VAR and AT, with the separator ":" and the appropriate characters for the comment (*Comment*).

This formal structure is created automatically when declaring the variables in the Syntax-Mode. The correct order of declaration of the individual ranges of validity is also controlled automatically. The declaration procedure in the Syntax-Mode is thus very easy, but it has the disadvantage that it shows a variable list with only one scope at a time, for example only input or only local variables.

The variables can also be declared in the Free Variable Editor, but you must know the keywords, separators and syntax rules in advance. The Free Variable Editor provides you with a complete overview of all declared variables of each of the ranges of validity, and the entries can be made faster if you have the required programming knowledge. In addition, corrections can be made more easily in the Free Variable Editor.

Program entry in the IL Editor



After the variable declaration is complete, you can commence the entry of the instruction section of the program.

▶ You can change to the IL section by clicking with the left mouse button in that section.

The IL programming language was preset earlier but you can also select it in the toolbar or via the POU Editor menu item Options → Programming Language → IL.

“Use of the IL Editor”

The programming language IL is a text-based, line-oriented language with the following structure:

```
Label (optional) – Operator – Operand – Comment (optional)
```

You must enter at least one tab or space between the individual elements and start each instruction with a new line. The comment is written between the combinations of characters Open Bracket/Asterisk [(*] and Asterisk/Close Bracket [*)]. Enter a tab before each comment so that the comment is separated from the command line. The row and column number of the current cursor position are displayed in the status bar at the bottom of the window.

The operators and comments are displayed in different colours to simplify editing of the instruction list.

Insert operators

The available operators are represented in the toolbar as icons. They change to suit the programing language which you set.

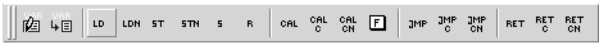


Figure 13: IL Toolbar

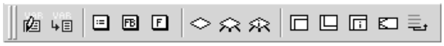


Figure 14: ST Toolbar

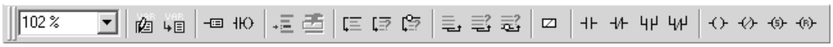


Figure 15: LD Toolbar

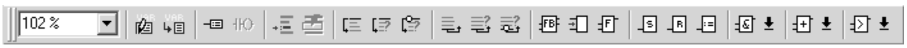


Figure 16: FBD Toolbar

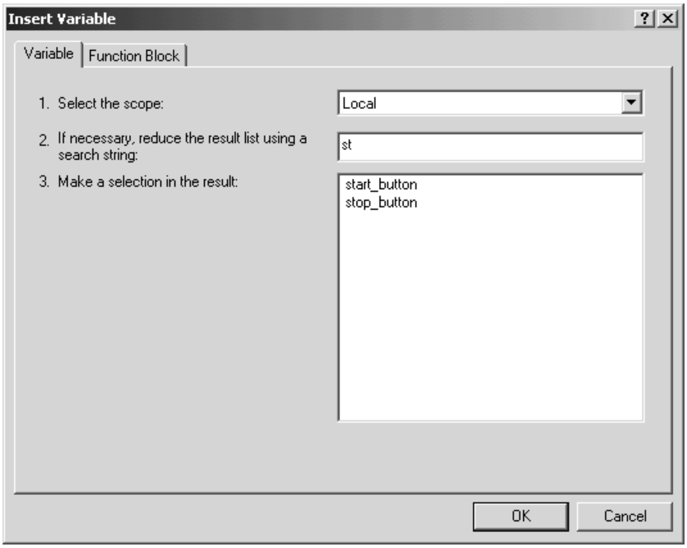
A tool tip (appears when you move the mouse pointer over the button) provides a short description of the operator.

You can also use the keypad to enter the commands.

Enter the variables

Variables which have already been declared can be inserted in the instruction section via the right hand mouse button and "Insert variable..." or with the respective menu listing under "Insert".

A dialog opens in which the variables, and if applicable, the function modules with their respective ranges of validity are displayed.



1. Select the scope: Local
2. If necessary, reduce the result list using a search string: st
3. Make a selection in the result:

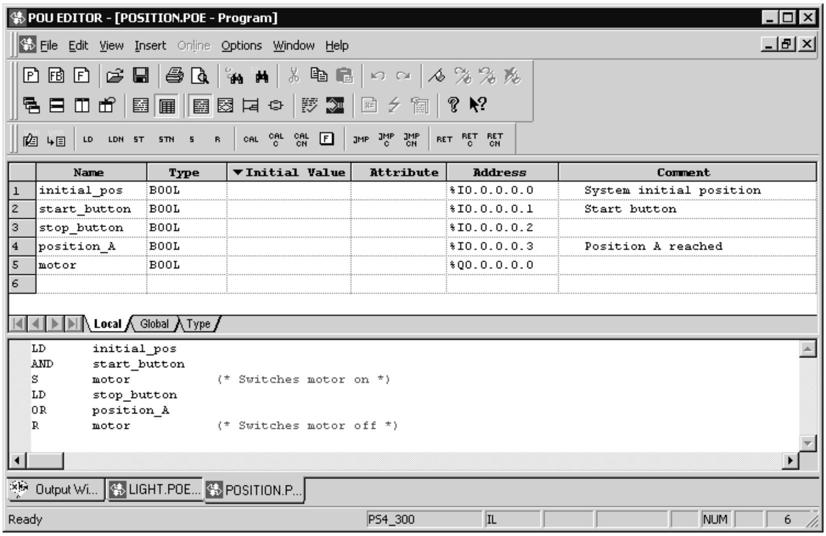
start_button
stop_button

Figure 17: Inserting variables

▶ Enter the example program as shown, making sure you adhere to the stated rules.

```
LD      initial_pos
AND     start_button
S       motor        (*Switches motor on*)
LD      stop_button
OR      position_A
R       motor        (*Switches motor off*)
```

When you have finished entering the program, the Editor window appears as follows:


POU EDITOR - [POSITION.POE - Program]
File Edit View Insert Online Options Window Help
LD LDN ST STN S R CAL CALC CALCN JMP JMPC JMPCN RET RETC RETCN
Local Global Type
Output Wi... LIGHT.POE... POSITION.P...
Ready PS4_300 IL NUM 6

	Name	Type	Initial Value	Attribute	Address	Comment
1	initial_pos	BOOL			%I0.0.0.0.0	System initial position
2	start_button	BOOL			%I0.0.0.0.1	Start button
3	stop_button	BOOL			%I0.0.0.0.2	
4	position_A	BOOL			%I0.0.0.0.3	Position A reached
5	motor	BOOL			%Q0.0.0.0.0	
6						

```
LD      initial_pos
AND     start_button
S       motor           (* Switches motor on *)
LD      stop_button
OR      position_A
R       motor           (* Switches motor off *)
```

Figure 18: The finished example program

Saving the program

POUs within a project are always stored in the “source” folder (default) or its sub-folders. Should you need to store a POU such as a function block in a sub-folder, you have to create this first in the Navigator via the Edit menu or the context-sensitive menu for the source file tree in the “Tree” window.

▶ Save the finished example program with ‹ File → Save As...› or by clicking on the button below:

“Save current POU”

In the window that opens, the “SOURCE” folder which is displayed is the folder for the current project and the destination folder for the POU.

▶ Enter the name “position” for the POU and confirm this by clicking on the OK button.

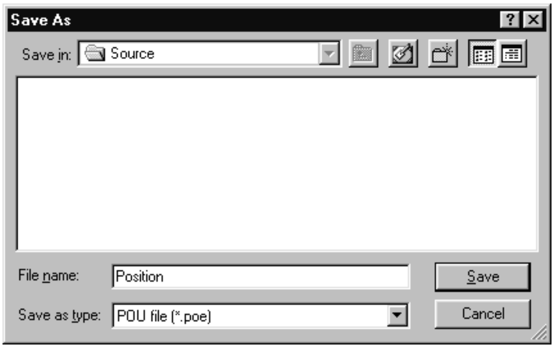


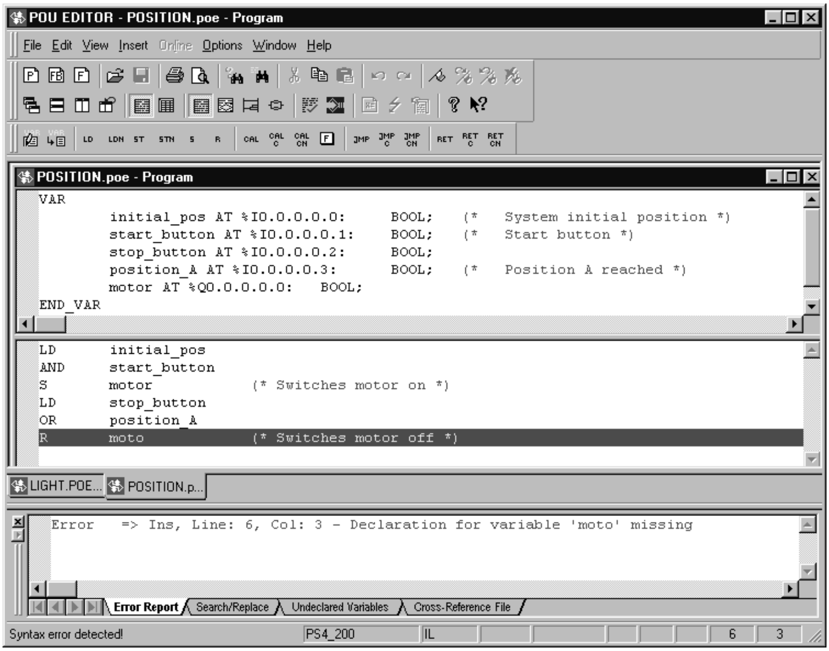
Figure 19: “Save as...” dialog for saving the POU

Syntax check
With the syntax check, the current POU is saved automatically. If the POU has just been created, you are requested to save the POU with a new name.

▶ Select ‹ File → Syntax check› or the respective button

“Syntax check”

If you have entered the program correctly, a message should appear in the status bar informing you that the syntax check has been completed without errors. Otherwise the POU Editor output window will open and display an error list:


POU EDITOR - POSITION.poe - Program
File Edit View Insert Online Options Window Help
LD LDN ST STN S R CAL CALC CALCN JMP JMPC JMPCN RET RETC RETCN
POSITION.poe - Program
LIGHT.POE... POSITION.p...
Error Report Search/Replace Undeclared Variables Cross-Reference File

```
VAR
        initial_pos AT %I0.0.0.0.0:     BOOL;   (*  System initial position *)
        start_button AT %I0.0.0.0.1:    BOOL;   (*  Start button *)
        stop_button AT %I0.0.0.0.2:     BOOL;
        position_A AT %I0.0.0.0.3:      BOOL;   (*  Position A reached *)
        motor AT %Q0.0.0.0.0:   BOOL;
END_VAR
```

```
LD      initial_pos
AND     start_button
S       motor           (* Switches motor on *)
LD      stop_button
OR      position_A
R       moto            (* Switches motor off *)
```

```
Error   => Ins, Line: 6, Col: 3 - Declaration for variable 'moto' missing
```

```
Syntax error detected!          PS4_200    IL          6    3
```

Figure 20: Error messages during program code generation

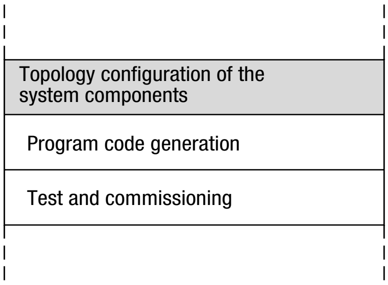
Displayed from left to right are:

- Error location (Declaration section or Instruction section)
- Line number
- Column number and
- a brief description of the error.

If you cannot identify the error through the brief description, select the line with the error and press F1 to get further information.

Trouble shooting: Double click the line with the error or mark it and then press the Enter key and the cursor will jump to the fault location.

7 Topology Configuration



Before generating the program code, you must specify your hardware system with the Topology Configurator tool. You need to enter the required information on the system (topology) and to configure each system component.

▶ Start the Topology Configurator in the menu under "Tools" or use the respective icon on the NAVIGATOR.

Topology configurator

The Topology configurator is started.

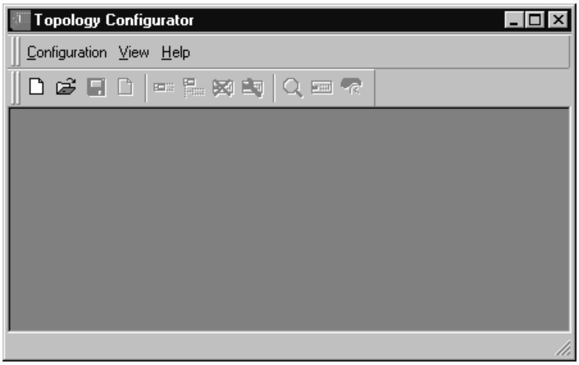


Figure 21: Topology configurator

Creating the topology configuration for the PS4

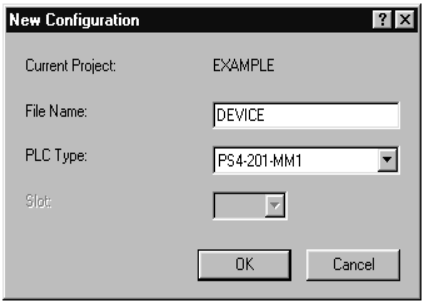
How to create a topology configuration is explained with the following example. For the example, you will need a PS4-141-MM1, PS4-151-MM1, PS4-201-MM1, PS4-271-MM1 or PS4-341.

To specify the topology configuration, you must define all modules of the automation system and enter information about each component. However, only the PS4 compact controller is relevant to the example topology in the following. Other modules which may be connected to your controller are not considered in the example.

All actions required for the configuration of a topology can be accessed via the menu or the toolbar of the Topology Configurator tool. Explanations of the individual icons can be found under the tool tips.

First define the CPU:

▶ Create a new topology configuration:


New Configuration

Current Project: EXAMPLE

File Name: DEVICE

PLC Type: PS4-201-MM1

Slot:

Figure 22: "New Configuration" dialog

▶ Enter the name DEVICE into the File Name box. Select the PS4-141-MM1 (or the PS4 controller type which you are using) in the PLC Type list box and confirm with OK.

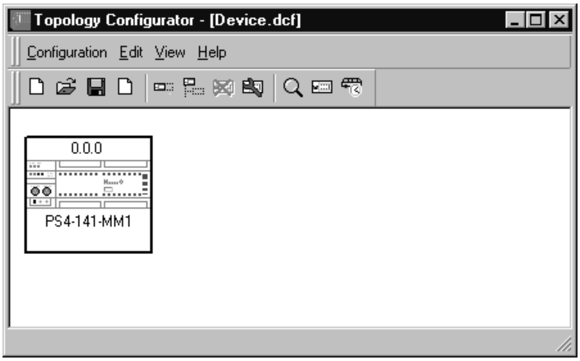

Topology Configurator - [Device.dcf]

Figure 23: Topology with PS4-141-MM1

☞ In the present example, you do not need to specify slave or master parameters for the PLC. You only need to do this if the controller is connected to a networked controller system via the Suconet K interface. You define the settings for the PLC in the parameters window. Open the window via the context-sensitive menu, or ‹ Edit → Setting Parameters› or via the Set Parameters button.

▶ Save the configuration.

Creating the topology configuration for the PS416

You will need a PS416-CPU-400 controller for this example.

To specify the topology configuration, you must define all modules of the automation system and enter information about each component. However, only the CPU and the PS416-OUT-400 digital output card as well as the input card are considered in the following example topology.

Proceed in the same manner with the creation of the PS4 configuration as with the creation of the PS416 configuration:

▶ Create a new topology configuration:
▶ Select a PS416-CPU-200/300/400 to suit your hardware.

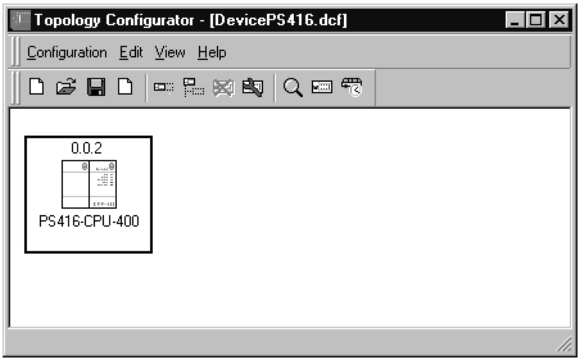

Topology Configurator - [DevicePS416.dcf]
Configuration Edit View Help
0.0.2
PS416-CPU-400

Figure 24: Topology with PS416

- ▶ Click on the Local Expansion button and select the input card PS416-INP-400 or PS416-INP-401 from the list and confirm with OK.
- ▶ Follow the same procedure to choose the output card PS 416-OUT-400.
- ▶ Select the two cards one after the other in the graphical view and after clicking on the Set Parameters button, enter in the dialog box the byte addresses previously configured on the card by DIP switch.

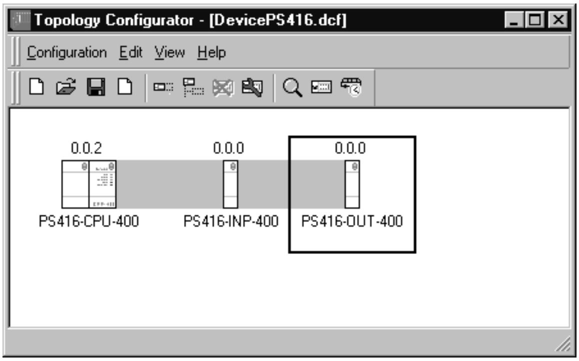
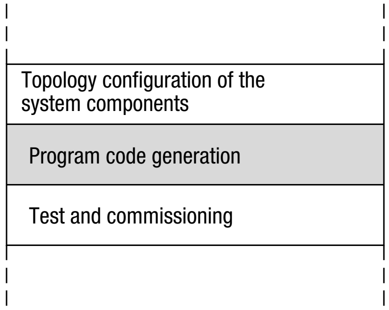

Topology Configurator - [DevicePS416.dcf]
Configuration Edit View Help
0.0.2
0.0.0
0.0.0
PS416-CPU-400
PS416-INP-400
PS416-OUT-400

Figure 25: Extended configuration

▶ Save the configuration.

8 Program Code Generation



Program code generation is carried out in two steps:

Create a make file list, which specifies all the elements (files) for consideration during program code generation: The program POU (with the respective function modules and functions), the topology file and the program parameter definition. One generated list can be created per program POU. It is automatically given the name of this program POU. Sucosoft S40 then updates the make file automatically as soon as you modify a project POU, the program parameters or the topology.

Generation of the program code according to the files specified in the make file.

All commands required to create an executable user program can be activated with the edit buttons in the toolbar.

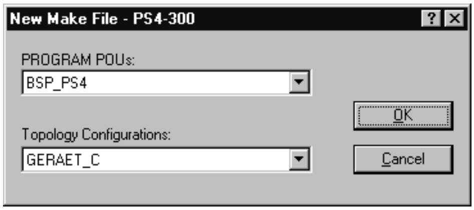
Creating the make file

▶ Use the NAVIGATOR toolbar to define the PLC for which the program code is to be generated.

▶ Select ‹ Generate → New make file...› or click on the respective button.

“Create make file”

The New Make File dialog box opens:


New Make File - PS4-300

PROGRAM POUs: BSP_PS4

Topology Configurations: GERAET_C

Figure 26: New Make File dialog box

In this dialog you select the Program POU and the topology. Only topologies which suit the PLC which you have selected in the NAVIGATOR will be displayed.

▶ Click the OK button.

The make list is created.

Undertaking program parameterization

Program parameters such as compiler options for the PLC type selected or conditions relevant to the execution of the program, such as maximum cycle time, password and marker range, have to be specified by you. You use the corresponding button to do this

"Setting the program parameters"

In our example, however, the preset standard parameters are sufficient.

Generating the program code

▶ Select ‹ Generate → Generate program code› or click on the respective button.

"Generate program code"

The executable program is generated according to the currently selected make file. The Output window shows the progress of the generation process. The respective message wil appear if compilation completes without errors.

If errors occur while the program code is being generated, they are displayed in the Output window with a corresponding message.

You can print out these errors via Project → Print Output.

Trouble shooting

Messages state from left to right:

Error location (declaration section or instruction section)

Line number

Column number and

Name and path of the POU

▶ Select an error line and confirm with the Enter key or double click on an error line.

The POU Editor opens. The cursor is located at the position of the POU at which the error was found. The POU line with the error is marked red in accordance with the basic setting made earlier.

▶ Fix the error and re-start generation of the program code.

9 Test and Commissioning

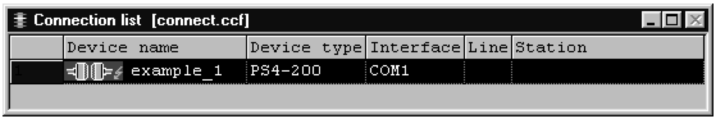
The Test and Commissioning (T & C) tool is used to transfer the program to the controller and test it. The following steps are necessary:

- Specify and establish the connection between the programming device and PLC
- For the PS416 or PS4-300 PLC's, transfer the operating system to the PLC.
- Transfer program to the PLC
- Start program
- Test program.

▶ Start the T & C.

"T & C"

The main test and commissioning (T & C) window opens with the "Connection list":



Connection list [connect.ccf]

	Device name	Device type	Interface	Line	Station
1	example_1	PS4-200	COM1		

Figure 27: "Connection list" window

Connecting the programming device to the controller

Specifying the connection

First of all you must define the connection between the programming device (PC) and the controller. The Connection List window automatically displays some default values which you can add to or modify as required.

▶ Click the Device Name box with the mouse and enter the PLC name “example_1” for the PLC to be connected.

If your programming cable is not connected to the default serial interface COM1, you can set another serial interface. Click in the “interface” field and select another interface from the pop-up list which appears.

The Line and Station fields in the Connection List window are not relevant for this example and are left empty.

▶ Save the data you entered with ‹ File → Save› . The file is automatically assigned the name CONNECT.CCF.

Establishing the connection

This is how you connect your programming device with the PLC.

Prerequisite: Only the “Connection list” is open in the T & C window.

▶ Connect the PLC via ‹ Device → Connect› or the respective icon.

“Connect device”

Successful connection establishment is indicated by display of a modified device icon.

Figure 28: Representation of successful connection

Transferring the operating system and program

Test and commissioning	
	Transfer program to PLC
	Start Program
	Test program

If the PLC contains no operating system or has an old operating system version, you must first transfer a new version of the operating system into the controller. The current version of the operating system is shipped with Sucosoft S40.

Transferring the operating system from Sucosoft to the controller

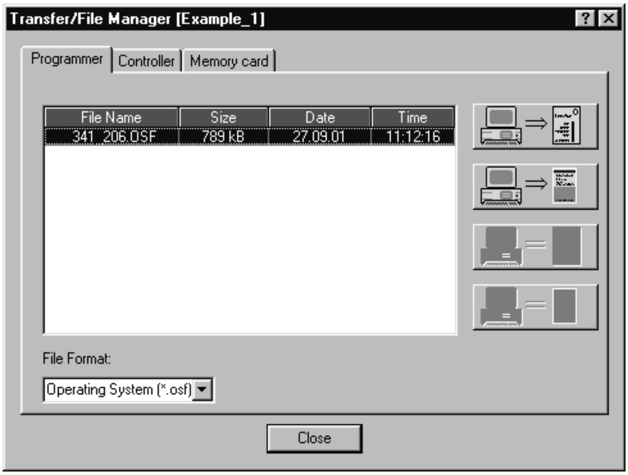
The transfer of an operating system is unnecessary with the PS4-200 type PLC. This section only relates to the PS4-300 and PS416 type PLC's.

▶ Select ‹ Device → Transfer/File Manager› or the respective icon.

“Transfer/File Manager”

The “Transfer/File Manager” dialog field opens. The “Programmer (PRG)” tab is the preset default. Names of the executable files of the “.PCD” type are shown in a list.

▶ In oder to transfer the operating system to the PLC, select the “operating system (*.OSF)” format in the File Name list and mark the required operating system.


Transfer/File Manager [Example_1]

File Name	Size	Date	Time
341_206.OSF	789 kB	27.09.01	11:12:16

File Format: Operating System (*.osf)

Figure 29: Selected operating system for transfer

▶ Select the “Transfer → PLC” button.

“Transfer to PLC”

Caution!
When you replace an existing operating system in the controller with a new version, all user programs and data are deleted at the same time!

The transfer can take several minutes, depending on the baud rate of the connection. The Transfer progress indicator is displayed to show the progress of the transfer.

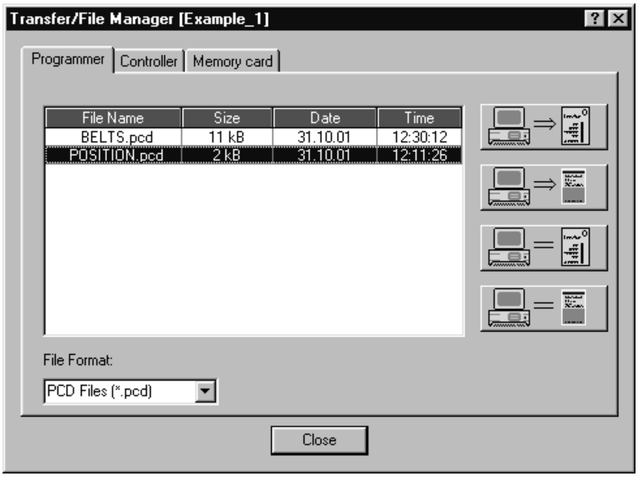
Transfer of the program from the Sucosoft to the PLC.

This section applies for all controller types.

▶ Access the “Transfer/File Manager” as previously described.

The “Programmer (PRG)” tab is the preset default. Names of the executable files of the “.PCD” type are shown in a list.

▶ Select the “Program (*.PCD)” file format in the list.


Transfer/File Manager [Example_1]

File Name	Size	Date	Time
BELTS.pcd	11 kB	31.10.01	12:30:12
POSITION.pcd	2 kB	31.10.01	12:11:26

File Format: PCD Files (*.pcd)

Figure 30: Program file selection

- ▶ Select the program code file you wish to transfer to the PLC (i.e. POSITION.PCD) and click Transfer → PLC to start the transfer.

“Transfer to PLC”

The Transfer progress indicator is displayed to show the progress of the transfer.

Starting the program

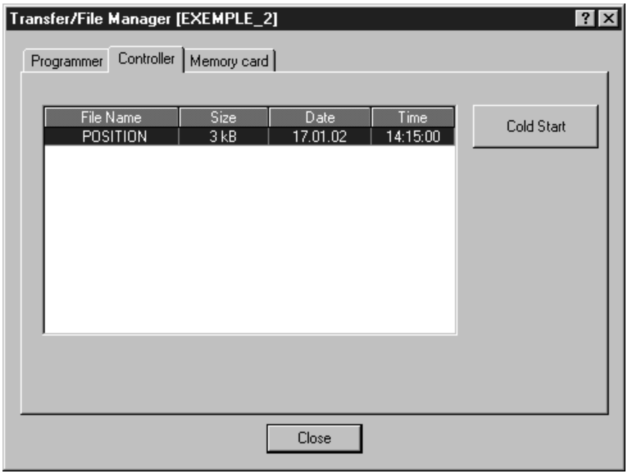
Test and commissioning

Transfer program to PLC

Start Program

Test program

- ▶ Click the “Controller” tab in the “Transfer/File Manager” dialog and click on the “Cold Start” button to start the program.


Transfer/File Manager [EXEMPLE_2]

File Name	Size	Date	Time
POSITION	3 kB	17.01.02	14:15:00

Figure 31: Start program

After successful transfer, the green “READY” lamp lights on the CPU card to show that it is ready. If the program was unable to start, you will get a system message indicating the possible reason.

You can get further information concerning the state of the program and the CPU in the “Program Status” dialog described in the following section.

Status and diagnostics

▶ Select ‹ Generate → CPU Status› or click on the respective button.

"CPU Status"

The "Program Status" dialog apears. Here you can check

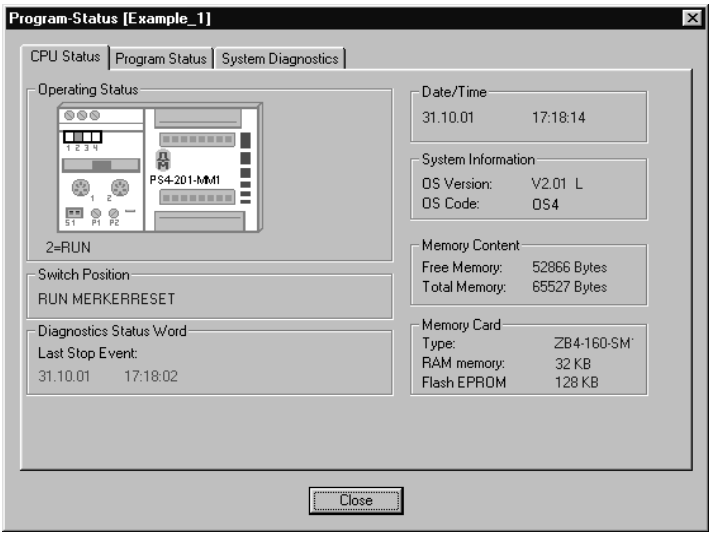
- if the user program is correctly executed (CPU Status),
- the possible causes of a fault user program (CPU Diagnostics),
- which properties are assigned to your user program (program status).

Detailed information concerning the "Status and diagnostics" dialog can be found in the "S40 User Interface" manual (AWB2700-1305-GB). The following is just a brief description.

CPU Status

▶ Click on the “CPU Status” tab.

The following dialogs will appear depending on the PLC selected:


Program-Status [Example_1]

Operating Status
2=RUN

Switch Position
RUN MERKERRESET

Diagnostics Status Word
Last Stop Event:
31.10.01 17:18:02

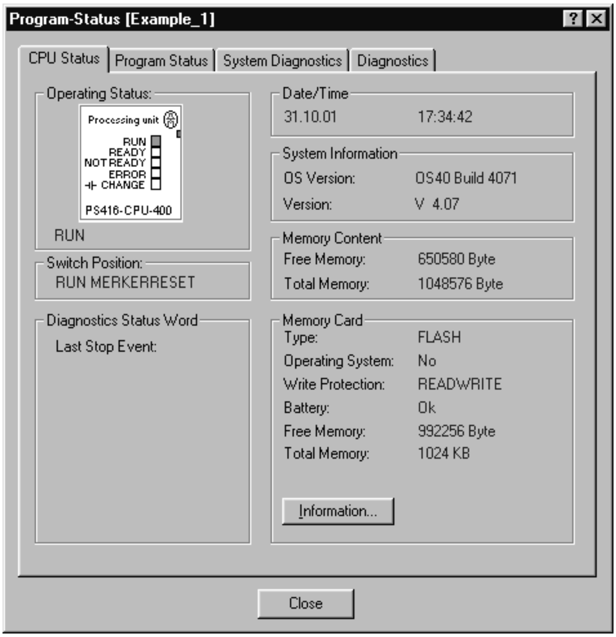
Date/Time
31.10.01 17:18:14

System Information
OS Version: V2.01 L
OS Code: OS4

Memory Content
Free Memory: 52866 Bytes
Total Memory: 65527 Bytes

Memory Card
Type: ZB4-160-SM'
RAM memory: 32 KB
Flash EPROM 128 KB

Figure 32: CPU Status for PS4-200 (PS4-300 analog)



Operating Status: RUN

Switch Position: RUN MERKERRESET

Diagnostics Status Word
Last Stop Event:

Date/Time
31.10.01 17:34:42

System Information

OS Version:	OS40 Build 4071
Version:	V 4.07

Memory Content

Free Memory:	650580 Byte
Total Memory:	1048576 Byte

Memory Card

Type:	FLASH
Operating System:	No
Write Protection:	READWRITE
Battery:	Ok
Free Memory:	992256 Byte
Total Memory:	1024 KB

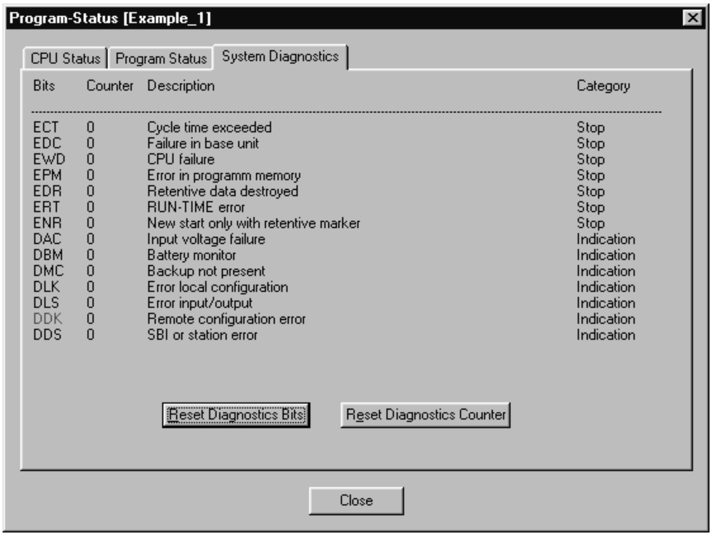
Figure 33: CPU Status for PS416

The RUN operating status indicates that the user program is being executed. If the NOT READY operating status is displayed, an error has occured. Possible causes for this error can be found in the CPU diagnostics.

CPU diagnostics

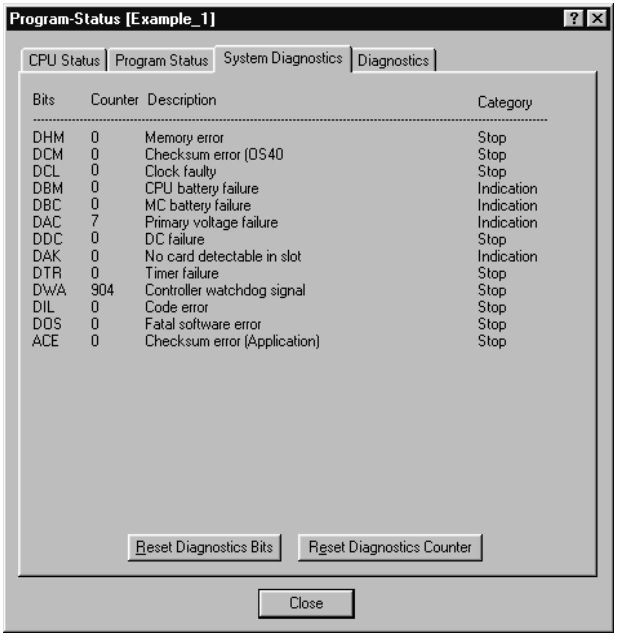
▶ Click on the “System Diagnostics” tab in the “Program Status” dialog.

Depending on the controller selected, the following dialog will appear with a listing of the diagnostics bits.


Program-Status [Example_1]

Bits	Counter	Description	Category
ECT	0	Cycle time exceeded	Stop
EDC	0	Failure in base unit	Stop
EWD	0	CPU failure	Stop
EPM	0	Error in programm memory	Stop
EDR	0	Retentive data destroyed	Stop
ERT	0	RUN-TIME error	Stop
ENR	0	New start only with retentive marker	Stop
DAC	0	Input voltage failure	Indication
DBM	0	Battery monitor	Indication
DMC	0	Backup not present	Indication
DLK	0	Error local configuration	Indication
DLS	0	Error input/output	Indication
DDK	0	Remote configuration error	Indication
DDS	0	SBI or station error	Indication

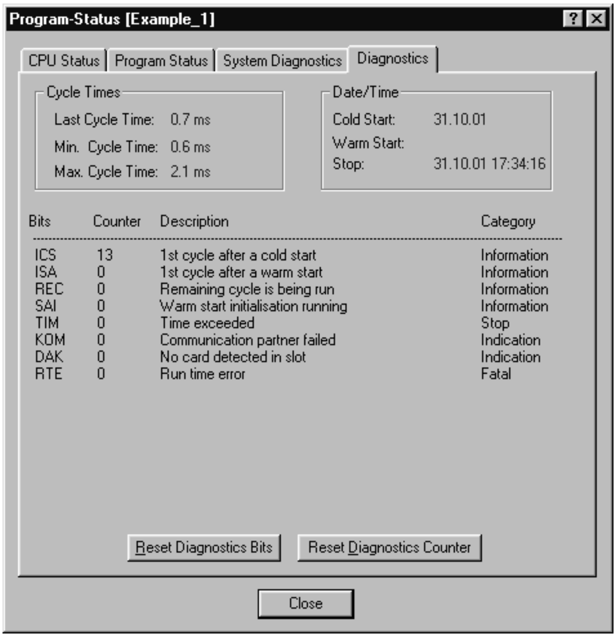
Figure 34: CPU diagnostics for PS4-200 (PS4-300 analog)



Bits	Counter	Description	Category
DHM	0	Memory error	Stop
DCM	0	Checksum error (OS40	Stop
DCL	0	Clock faulty	Stop
DBM	0	CPU battery failure	Indication
DBC	0	MC battery failure	Indication
DAC	7	Primary voltage failure	Indication
DDC	0	DC failure	Stop
DAK	0	No card detectable in slot	Indication
DTR	0	Timer failure	Stop
DWA	904	Controller watchdog signal	Stop
DIL	0	Code error	Stop
DOS	0	Fatal software error	Stop
ACE	0	Checksum error (Application)	Stop

Figure 35: CPU diagnostics for PS416

With the PS416, you can access more information with the “Diagnostics” tab:



Cycle Times

Last Cycle Time:	0.7 ms
Min. Cycle Time:	0.6 ms
Max. Cycle Time:	2.1 ms

Date/Time

Cold Start:	31.10.01
Warm Start:	
Stop:	31.10.01 17:34:16

Bits	Counter	Description	Category
ICS	13	1st cycle after a cold start	Information
ISA	0	1st cycle after a warm start	Information
REC	0	Remaining cycle is being run	Information
SAI	0	Warm start initialisation running	Information
TIM	0	Time exceeded	Stop
KOM	0	Communication partner failed	Indication
DAK	0	No card detected in slot	Indication
RTE	0	Run time error	Fatal

Figure 36: Program diagnostics for PS416

Program status

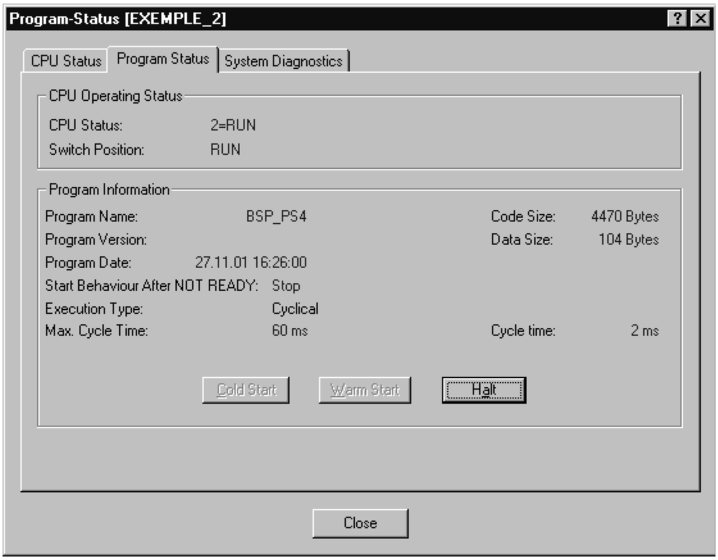
▶ Click on the “Program Status” tab.

The following dialogs will appear depending on the PLC selected. Here you can for example, do the following:

perform a cold or warm start – depending on the position of the CPU operating selection switch – or,

stop the PLC with a mouse click or

read off how long your program requires for execution by examining the cycle time.


Program-Status [EXEMPLE_2]

CPU Status | Program Status | System Diagnostics

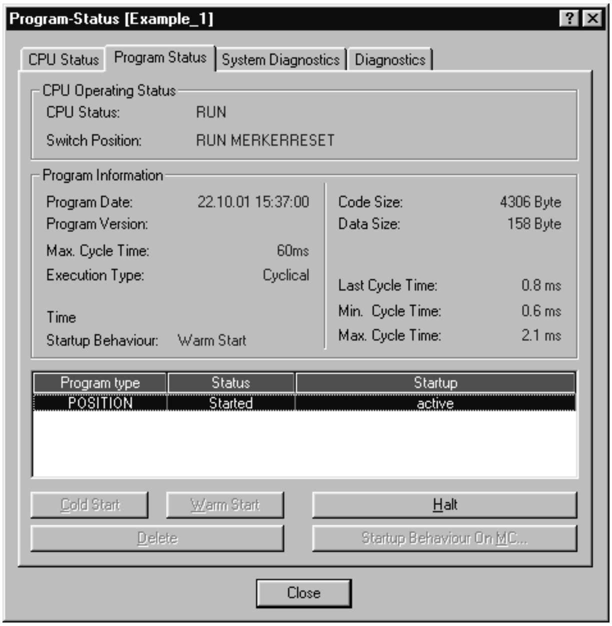
CPU Operating Status

CPU Status: 2=RUN
Switch Position: RUN

Program Information

Program Name: BSP_PS4
Program Version:
Program Date: 27.11.01 16:26:00
Start Behaviour After NOT READY: Stop
Execution Type: Cyclical
Max. Cycle Time: 60 ms
Code Size: 4470 Bytes
Data Size: 104 Bytes
Cycle time: 2 ms

Figure 37: Program status of PS4-200



Program-Status [Example_1]

CPU Operating Status

CPU Status:	RUN
Switch Position:	RUN MERKERRESET

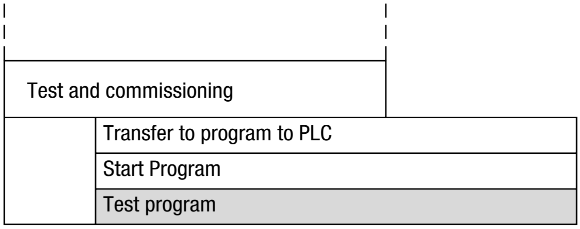
Program Information

Program Date:	22.10.01 15:37:00	Code Size:	4306 Byte
Program Version:		Data Size:	158 Byte
Max. Cycle Time:	60ms		
Execution Type:	Cyclical	Last Cycle Time:	0.8 ms
Time		Min. Cycle Time:	0.6 ms
Startup Behaviour:	Warm Start	Max. Cycle Time:	2.1 ms

Program type	Status	Startup
POSITION	Started	active

Figure 38: Program status of PS416

Testing the program

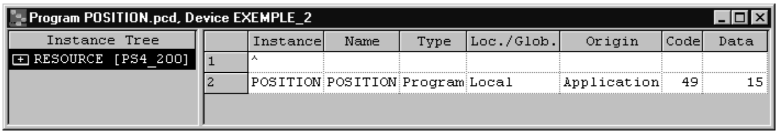


Prerequisite: Only the “Connection list” is open in the T & C window.

▶ Select ‹ Device → Program› or click on the respective button.

Program

The menu and toolbar are adapted accordingly.


Program POSITION.pcd, Device EXEMPLE_2

Instance Tree
⊞ RESOURCE [PS4_200]

	Instance	Name	Type	Loc./Glob.	Origin	Code	Data
1	^						
2	POSITION	POSITION	Program	Local	Application	49	15

Figure 39: T & C with open Program POU

The Program window consists of two parts. The left panel “Instance Tree” contains RESOURCE which indicates the first level of the program structure and represents the CPU. The right panel of the dialog box contains the name of the example program POU (POSITION).

▶ Double-click the RESOURCE line to display the further program structure, in this case only the POSITION program POU.

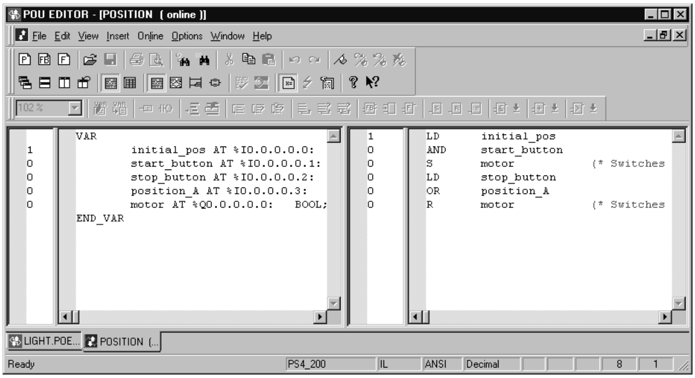
If further POUs (function blocks and functions) are called by the program POU, this is indicated by a plus sign next to the program name. This allows you to list and select all components of a program structure and carry out modifications.

☞ If you want to edit the program online, the current project must first be loaded in the Project Manager. In addition, the contents of the individual POUs and the program code must match the program version in the controller. In our example, we assume that you have followed our instructions above, in which case both requirements are fulfilled.

▶ In order to view or change the POU online: Mark the POU names "POSITION" in the left panel and select the menu point ‹ Program → View/Change POU› or the respective button.

View/Change POU

The POU Editor opens in Online mode, in which your POU "POSITION" is opened and presented in the foreground.


POU EDITOR - [POSITION (online)]
File Edit View Insert Online Options Window Help
102 %
LIGHT.POE...
POSITION (...
Ready
PS4_200
IL
ANSI
Decimal
8
1

```
        VAR
1               initial_pos AT %I0.0.0.0.0:
0               start_button AT %I0.0.0.0.1:
0               stop_button AT %I0.0.0.0.2:
0               position_A AT %I0.0.0.0.3:
0               motor AT %Q0.0.0.0.0:   BOOL;
        END_VAR
```

```
1   LD      initial_pos
0   AND     start_button
0   S       motor              (* Switches
0   LD      stop_button
0   OR      position_A
0   R       motor              (* Switches
```

Figure 40: POU Editor in Online mode

The “Online” menu and the buttons for the following actions are availabe:

status indication of variables,

activation of changes made in the instruction section

transfer of variables to the variables window.

Checking POU - diaplay variable state

▶ Select the POU Editor window ‹ Online → Status display› or click on the “Status display” button to switch the status indication on and off.

“Status display”

The variable states in the declaration and instruction section appear. In our IL-program, the variable states in the status column are indicated on the left of the declaration and instruction section.

▶ Cancel the update of the variable states by choosing the Status Display command again or the Status Display button.

The previously read states are then displayed as “frozen” states. You can use this option for troubleshooting. For our example, however, this is not necessary.

If, in a complex program, you wish to display states of specified variables which are not located close to each other in the program and thus cannot be displayed simultaneously in a single window, or if you wish to display variables of different POUs, you can select the variables and display them in the variable window. You can view the states of variables of differing POUs simultaneously in the variable window and with the POU Editor in Online mode. This feature of Sucosoft is particularly helpful when testing programs which invoce function modules. This is not relevant with our mini-program. However, try out this function with the “initial_pos” and “motor” variables:

▶ Position the cursor on the “initial_pos” variable and select “Online → Transfer variable” or press the respective button.

“Transfer variable”

▶ Repeat the process with the “motor” variable.

The variable window opens in the background and can be viewed after you changeover to the main window of the T & C.

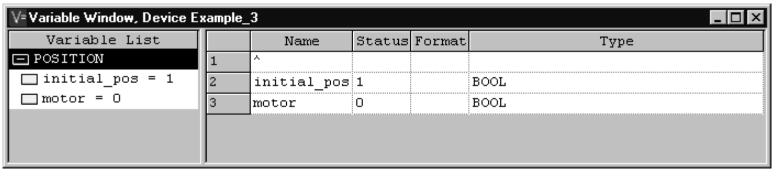
▶ Activate the “Variable List” window and click on the “Variable Window”.

The variable window appears in the foreground.

▶ Double click in the variable list (left hand window) on “POSITION”, in order to display both of the transferred variables in the information section (right hand window).

▶ Switch on the display status for the variables in the window via ‹ Display variable status› or use the respective button to display the current states of the selected variables.

“Display status”



Variable List
- POSITION
  - initial_pos = 1
  - motor = 0

	Name	Status	Format	Type
1	^			
2	initial_pos	1		BOOL
3	motor	0		BOOL

Figure 41: Variable window with status display

If you mark a variable (e.g. “initial_pos”) in the left or right section of the window, on a PS4-300 or PS416, the button for forcing a variable in the RUN state becomes available. More detailed information concerning this topic (Forcing) can be found in the manual “S40 User Interface” (AWB2700-1305-GB).

▶ Exit display of the status by clicking again on the “Display status” button.

▶ Return to the Online Editor window via ALT + TAB.

Online POU modification

If you have modified the POU, the ‹ Online → Activate› menu and the “Activate” button in the toolbar will be available.

“Activate”

▶ Select ‹ Online → Activate› or click on the button.

The changes made will be updated in the POU, in the program code file and in the controller.

▶ Select ‹ File → Close› , in order to end the POU Editor in Online mode.

You have now seen all Sucosoft S40 tools for creating and testing an example PLC program in the IL language. In the following chapter, the example program will be extended using the programming languages LD and FBD.

10 Program Entry in LD/FBD

You can optionally create and commission programs in the LD or FBD graphic programming languages instead of the text based IL language. With LD and FBD, the instruction section of a POU is represented by graphical symbols. The declaration section in LD and FBD is displayed the same as in IL or ST.

The programming languages IL, LD and FBD are interchangeable. You can thus create a program in LD and display and process it in IL or FBD. Change over between ST and the graphic languages is not possible.

In order to create a POU in the LD programming language, activate the editor via ‹ Extras → Programming Language → LD› or via the icon.

“POU Editor LD”

The language element toolbar will then contain LD specific operators, which enables actions for the POU Editor LD.

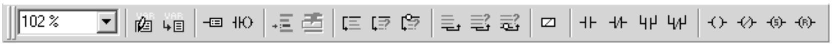


Figure 42: Toolbar LD

In order to create a POU in the FBD programming language, activate the editor via ‹ Extras → Programming Language → FBD› or via the icon.

“POU Editor FBD”

The language element toolbar will then contain FBD specific operators, which enables actions for the POU Editor FBD.

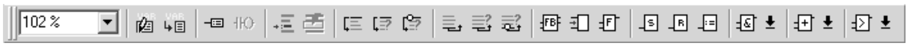


Figure 43: FBD Toolbar

A tool tip (appears when you move the mouse pointer over the button) provides a short description of the operator.

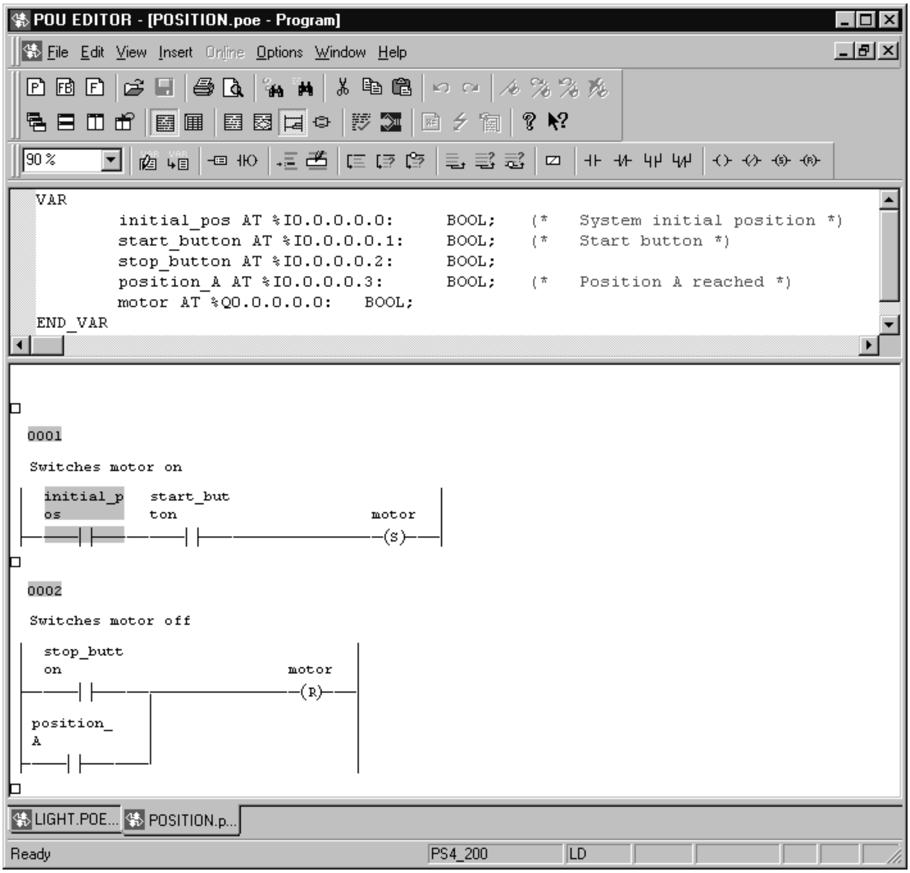
Displaying the example program in LD/FBD

You can display and change the example program which you have created in the Offline mode of the POU Editor in IL, in one of the graphical programming languages.

Display POU in LD

▶ Select the LD programming language as described beforehand.

The POU “position” is displayed as a ladder diagram in the instruction section:



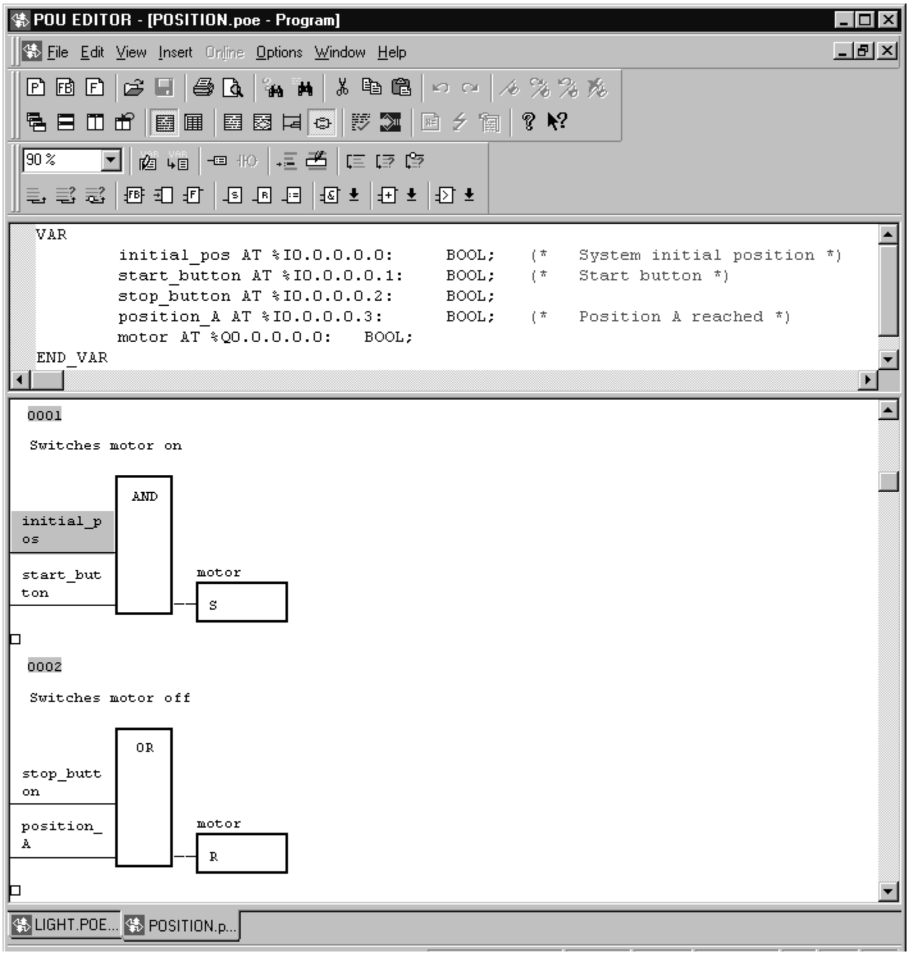
```
VAR
        initial_pos AT %I0.0.0.0.0:     BOOL;   (*  System initial position *)
        start_button AT %I0.0.0.0.1:    BOOL;   (*  Start button *)
        stop_button AT %I0.0.0.0.2:     BOOL;
        position_A AT %I0.0.0.0.3:      BOOL;   (*  Position A reached *)
        motor AT %Q0.0.0.0.0:   BOOL;
END_VAR
```

Figure 44: POU “position” in LD format

Display POU in FBD

▶ Select the FBD programming language as described beforehand.

The POU “position” is displayed in the instruction section using graphical symbols:


POU EDITOR - [POSITION.poe - Program]
File Edit View Insert Online Options Window Help
90 %
LIGHT.POE...
POSITION.p...

```
VAR
        initial_pos AT %I0.0.0.0.0:     BOOL;   (*  System initial position *)
        start_button AT %I0.0.0.0.1:    BOOL;   (*  Start button *)
        stop_button AT %I0.0.0.0.2:     BOOL;
        position_A AT %I0.0.0.0.3:      BOOL;   (*  Position A reached *)
        motor AT %Q0.0.0.0.0:   BOOL;
END_VAR
```

```
0001
Switches motor on
AND: initial_pos, start_button -> motor S

0002
Switches motor off
OR: stop_button, position_A -> motor R
```

Figure 45: POU “position” in FBD format

The logical structure of the program represented in LD and FBD corresponds to the task of the example program.

The instruction comments are not assigned to the individual graphic symbols, but appear in the respective network comment instead.

Entering a program in LD

Preconditions:

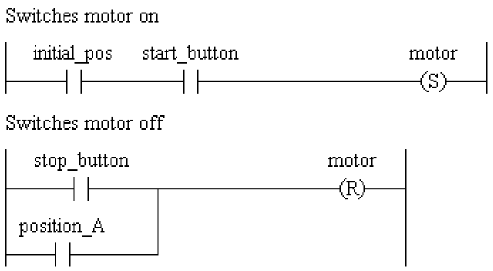
You already know the objective of programming example 1.

The “example” project has been created.

The new POU of type Program has been created.

The required variables have been declared.

Program example in LD:



The program is subdivided into two networks.

The contacts “initial_pos” and “start_button” which are connected in series in network 0001 represent the AND operation required to switch on the motor.

The contacts “stop_button” and “position_A” which are connected in parallel in network 0002 represent the OR operation required to switch off the motor.

A short description of the network program is shown as a comment in each network header.

Now enter the example program shown above in LD yourself. Use the icons on the LD language toolbar.

▶ Select the LD programming language with the menu item ‹ Extras → Programming language → LD› , or click on the appropriate button in the standard toolbar.

▶ Click the Initial LD Network button to create an initial LD network.

This creates a network 0001 with a contact and an output symbol. Both symbols automatically receive the designation “undef_opd” (undefined operand).

“Initial LD network”

▶ Position the cursor between the first and last symbols by clicking with the mouse. This is where you will insert an AND operator.

▶ Click on the AND icon in the language element toolbar.

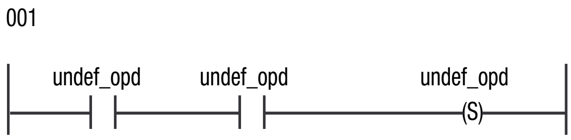
A new contact is inserted in series with the existing contact.

▶ Mark the output symbol in order to insert a network termination coil contact with a set condition.

A set output is created in parallel with the existing output.

▶ Select the “old” output and delete it.

The graphical representation of network 0001 is now complete.



- Click on the Initial LD Network button to create a new network.

This creates a network 0002 with a contact and an output symbol.

- Mark the contact symbol in order to insert an OR operator.
- Click on the “Insert parallel contact” icon.

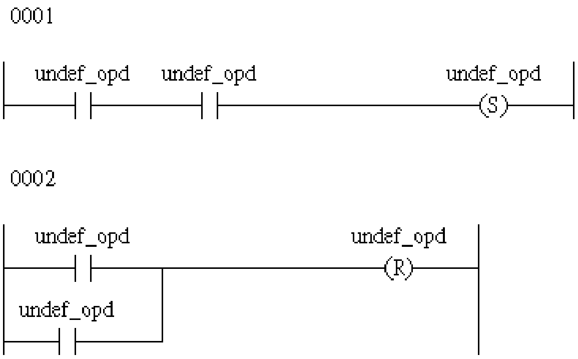
A new contact is created in parallel with the existing contact.

- Mark the output symbol with a mouse click in order to insert a network termination coil contact with a reset condition.

A reset output is created in parallel with the existing output.

- Select the “old” output and delete it.

The graphical representation of both networks is now complete.



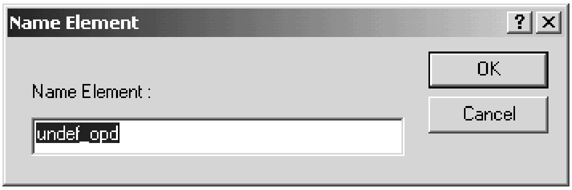
The default variable names “undef_opd” must now be replaced by the declared variable names. There are two different ways of doing this:

Accept a variable name from the list of declared variables (see Section Program entry in the IL Editor on Page 32).

Enter the variable name with the “Name Element“ dialog box.

▶ Double click on the element to be named.

The respective dialog window is started.



Name Element : undef_opd

Figure 46: Name Element dialog box

▶ Enter “initial_pos” and confirm with the OK button.

The variable name appears above the contact symbol.

▶ Enter the rest of the variable names in both networks in the same way.

▶ In order to input the comment, simply position the cursor on the network which requires a comment and call up the ‹ Edit → Network Comment› .

“Network Comment”

The Network Comment window opens:

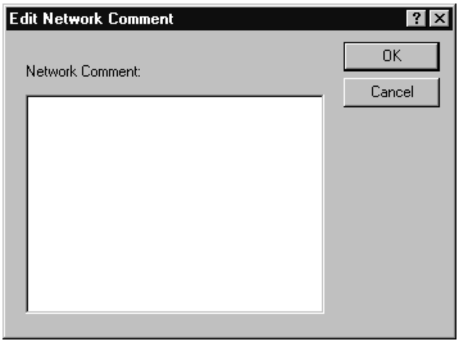


Figure 47: “Network comment”

▶ Enter “motor on” and confirm with the OK button.

The comment you have entered appears in the network header underneath the network number.

▶ Enter the comment in the same way for the network 0002.

The program entry in LD is now complete.

Entering a program in FBD

Preconditions:

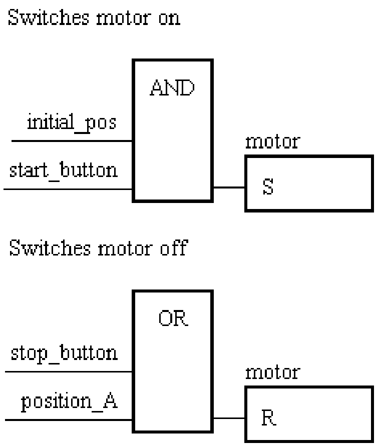
You already know the objective of programming example 1.

The “example” project has been created.

The new POU of type Program has been created.

The required variables have been declared.

Program example in FBD



The program is subdivided into two networks.

In the first network, the variables “initial_pos” and “start_button” are ANDed to switch on the motor. They form the necessary condition for the motor to be switched on.

In the second network, the variables “stop_button” and “position_A” are ORed to switch off the motor. The Stop_button and Position_A are the switch-off conditions.

A short description of the network program is shown as a comment in each network header.

We will now describe how to enter the example program shown above in FBD. Use the icons on the FBD language toolbar.

▶ Select the FBD programming language with the menu item ‹ Extras → Programming language → FBD› , or click on the appropriate button in the standard toolbar.

▶ Click on the "Insert initial FBD network" button to create an initial FBD network.

This creates a network 0001 with an input and an output symbol. The two symbols automatically receive the default name "undef_opd".

"Initial FBD network"

▶ Mark the input connection line and then use the button to insert an AND condition.

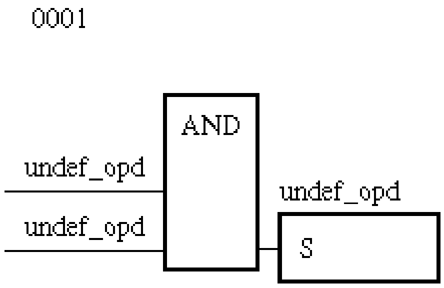
▶ Mark the output symbol with a mouse click and then insert a network termination using the icon and create a set contact "S" as a set function.

The output symbol of the set command is added below the existing output symbol. There is no longer any need for the upper output and this can be deleted.

▶ Select the upper assignment symbol and press the DEL key to delete it.

The upper assignment symbol is erased.

The graphical representation of network 0001 is now complete.



- ▶ Click on the “Initial FBD network” button to create a new network.

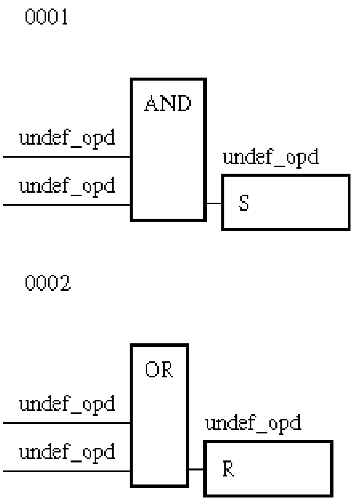
This creates a network 0002 with a contact and an output symbol.

- ▶ Mark the input connection line and then use the button to insert an OR condition.
- ▶ Mark the output symbol with a mouse click and then insert a network termination using the icon and create a “R” reset function.

A reset output is added below the existing output.

- ▶ Select the superfluous assignement symbol and press the DEL key to delete it.

The graphical representation of both networks is now complete.



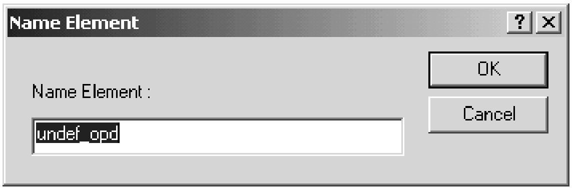
The default variable names “undef_opd” must now be replaced by the declared variable names. There are two different ways of doing this:

Accept a variable name from the list of declared variables (see Section Program entry in the IL Editor on Page 32),

Enter the variable name with the “Name element“ dialog box.

▶ Double click on the element to be named.

The respective dialog window is started.



Name Element :

undef_opd

Figure 48: “Name Element” dialog box

- ▶ Enter "initial_pos" and confirm with the OK button.

The variable name appears above the marked connection line.

- ▶ Enter the rest of the variable names in both networks in the same way.

Next, you enter the network comments, so you select the relevant network first to do this.

- ▶ In order to input the comment, simply position the cursor on the network which requires a comment and call up the ‹ Edit → Network Comment› .

"Network comment"

- ▶ Enter "motor on" and confirm with the OK button.

The comment you have entered appears in the network header underneath the network number.

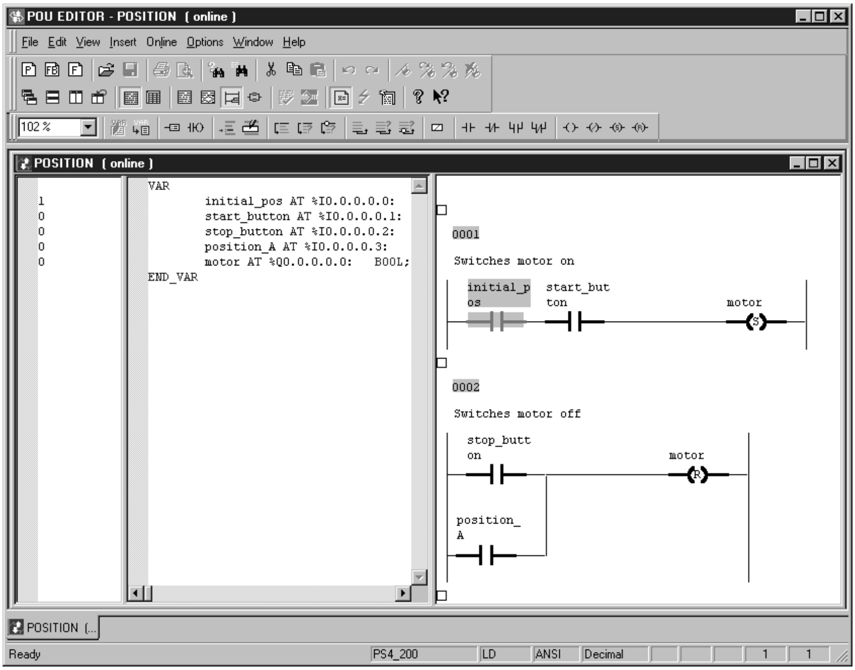
- ▶ Enter the comment in the same way for the network 0002.

The program entry in FBD is now complete.

Online display of the example program in LD or FBD

The program that is executed in the controller can be displayed in the POU Editor in Online mode in either of the programming languages IL, LD or FBD.

If you created the program in the LD/FBD POU Editor, the comments are not assigned to the individual lines in the IL representation, but appear instead at the start of the corresponding program sequence:



```
VAR
        initial_pos AT %I0.0.0.0.0:
        start_button AT %I0.0.0.0.1:
        stop_button AT %I0.0.0.0.2:
        position_A AT %I0.0.0.0.3:
        motor AT %Q0.0.0.0.0:    BOOL;
END_VAR
```

Figure 49: POU “position” in LD format (online)

LD representation

You can switch between the programming languages. This is possible with the status display switched off and with an active status display.

▶ Select the ‹ Extras → Programming language → LD› or click on the appropriate button in the standard toolbar.

“POU Editor LD”

▶ Carry out modifications in the same way as in the IL POU Editor.

You will find detailed information in the AWB 27-1305-GB “Sucosoft S40, Programming Software, User Interface manual”.

If you have modified the POU, the Activate button in the Online menu or the corresponding button in the toolbar will be available.

“Activate”

▶ Click the “Activate” button

to update your modifications in the POU, in the program code file and in the controller.

FBD representation

You can switch between the programming languages. This is possible with the status display switched off and with an active status display.

▶ Select the ‹ Extras → Programming language → FBD› or click on the appropriate button in the standard toolbar.

"FBD Editor"

▶ Carry out modifications in the same way as described in the LD POU Editor.

Entering a program in ST

A further programing language of the Sucosoft is ST structured text. The language is similar to PASCAL.

The declaration section is identical with IL, LD and FBD, the instruction section – is similar to IL – which is text based. You CANNOT however, switch between the graphic based languages LD/FBD and ST.

A toolbar is available to assist you to enter the ST commands.

Figure 50: ST Toolbar

Creating an example program in ST

The example which we created can now be modified for ST.

Preconditions:

You already know the objective of programming example 1.

The “example” project has been created.

The new POU of type Program has been created.

The required variables have been declared.

Let us once again review the task at hand:

If the motor is in the initial position and the “start button” is pressed, the motor is switched on.

The motor is switched off when the “stop button” is pressed or when “Position_A” is reached.

Now the step by step procedure:

▶ Select the ST programming language with the menu item ‹ Extras → Programming language → ST› , or click on the appropriate icon.

“POU Editor ST”

Enter the ST example:

▶ Use the respective icon in the toolbar to enter the IF...THEN framework.

You will see that an empty framework appears:

```
IF ...  THEN...
...;
ELSE
...;
END_IF;
```

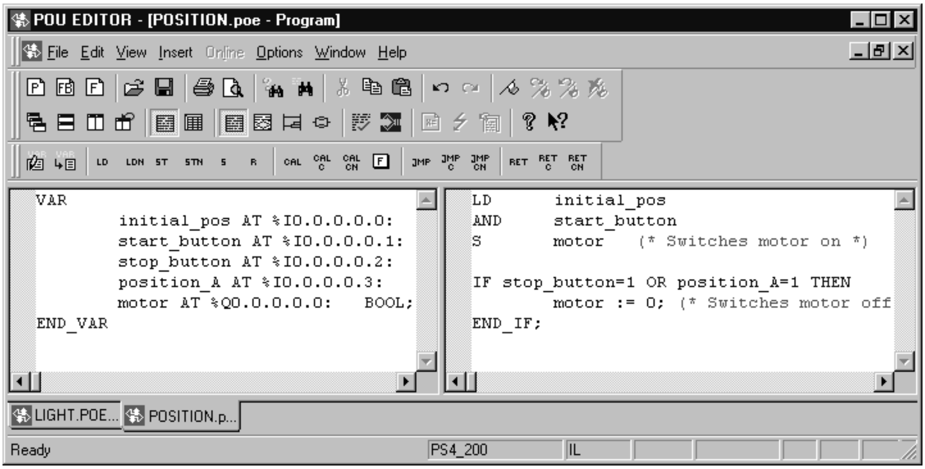
▶ Complete the framework as follows:

```
(* Switches on the motor *)

IF initial_position  = 1 AND start_button=1 THEN
  Motor:=1;
END_IF;

(* Switches off the motor *)

IF stop_button = 1 OR position_A = 1 then
  Motor:=0;
END_IF;
```


POU EDITOR - [POSITION.poe - Program]
File Edit View Insert Online Options Window Help
LD LDN ST STN S R CAL CALC CALCN JMP JMPC JMPCN RET RETC RETCN
LIGHT.POE...
POSITION.p...
Ready
PS4_200
IL

```
VAR
        initial_pos AT %I0.0.0.0.0:
        start_button AT %I0.0.0.0.1:
        stop_button AT %I0.0.0.0.2:
        position_A AT %I0.0.0.0.3:
        motor AT %Q0.0.0.0.0:   BOOL;
END_VAR
```

```
LD      initial_pos
AND     start_button
S       motor    (* Switches motor on *)

IF stop_button=1 OR position_A=1 THEN
        motor := 0;  (* Switches motor off
END_IF;
```

Figure 51: POU “position” in ST format

11 Programming Task 2

A program is to be written for a running light effect with eight LEDs which are connected to the digital outputs such that each of the LEDs flash in succession. The change from one LED to the next should take place at a frequency of 2 Hz.

The running light effect is to start automatically when the controller is switched on and continue until the program is stopped by an external signal. There is no provision for control elements to influence the process. It is thus not necessary to scan inputs.

After commissioning, the program should be modified online: the direction of running should be reversed and the frequency changed to 1 Hz.

Implementation of the task
First read in this chapter how to implement the task as an IL program. It contains all required information on the program structure and on the language elements used. The Section „Entering programming task 2“ from Page 105 then describes how to enter the program in Sucosoft S40.

The two most important variables which must be programmed when implementing the task are the **signal duration** for illuminating each LED and the **switching of the LEDs**.

A manufacturer-defined function block from the timers group can be used to define the **signal duration**. The simplest solution is to use the TimePulse (TP) function block, since no resets or similar steps are necessary.

A “general-purpose” user-defined function block should be written for the **switching of the LEDs** so that it can be used again for further similar tasks. Accordingly, the output byte address and the required value for the signal duration should be transferred to the function block from “outside”, i.e. from the calling POU. This “general-purpose” function block can then be called several times in the same program, for example so that several sets of output bytes with a different signal duration can be controlled by the same program.

Accordingly, a user-defined function block must be created in which

- the LEDs are switched on and off and
- the manufacturer-defined function block TP is called for defining the signal duration.

This function block should be assigned the name LIGHT.

The function block LIGHT is called in the main program (the Program POU). The Program POU should be assigned the name EXP_PS4. The required output byte address and the value for the signal duration are transferred to the function block LIGHT by the Program POU when it is called by the latter.

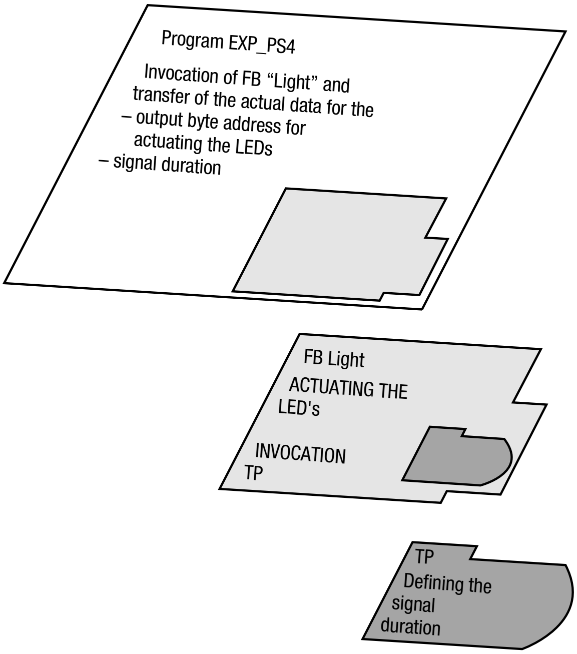

Program EXP_PS4
Invocation of FB "Light" and transfer of the actual data for the
– output byte address for actuating the LEDs
– signal duration
FB Light
ACTUATING THE LED's
INVOCATION TP
TP
Defining the signal duration

Figure 52: Program components for implementing the task

Two POUs must thus be written for the task:

- one POU of type FUNCTION_BLOCK with the name LIGHT
- a second POU of type PROGRAM with the name EXP_PS4.

Draft for function block LIGHT

Switching the LEDs

The running light effect can be achieved by setting a single bit to load a bit pattern into the working register at regular intervals, shifting the bit pattern by one bit and then returning it. The bit pattern is transferred as a variable. Since Sucosoft allows a variable to be read in as an input variable, processed and output as an output variable under the same name, the required variable should be defined as an input/output variable. Declare it with the name Light_strip and with the data type Byte. The keyword for input/output variables is VAR_IN_OUT; the end of a declaration block is defined by the keyword END_VAR for all scopes.

The physical address is not transferred until the variable is called by the higher level program POU, and thus does not need to be considered in the function block. The following declaration block should be used:

```
VAR_IN_OUT
  Light_strip : BYTE ;
END_VAR
```

Either a shift instruction or a rotation instruction can be used for shifting the bit pattern, either to the right or left in either case. The use of a shift instruction, however, requires a 1 to be loaded after every eight shift steps. A rotation command to the right will thus be used here. The instruction lines are the following:

```
LD      Light_strip
ROR     1
ST      Light_strip
```

Defining the signal duration

The manufacturer-defined function block TP (TimePulse) is used for defining the signal duration.

The TP function block (TimePulse)

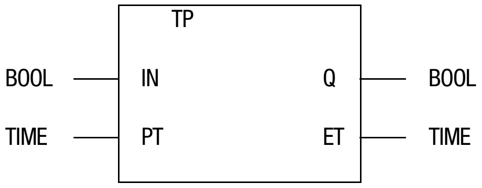


Figure 53: Prototype of the TP function block

IN Start condition
PT Time value setting
Q Binary status of the timer
ET Current time value

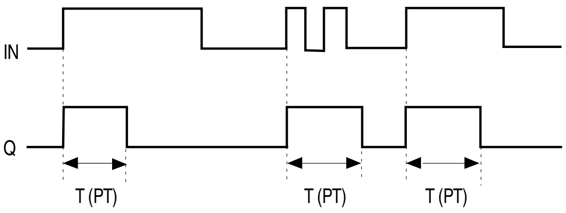


Figure 54: Timing diagram for TP

The timer starts with a rising edge on the IN input and maintains the status 1 on the Boolean output Q for the duration of the predefined time value PT. The ET output indicates the current time value. While the time is running, status 1 is maintained at output Q irrespective of the IN input status.

The IN and Q operands are Boolean operands. The PT and ET operands are of the standard data type TIME (default value T#0s).

Using function blocks

An instance, i.e. a copy, of a function block must be created in the **declaration section** of the invoking POU for each application of the function block. To achieve this, the function block is assigned a freely selectable instance name which must be declared as a **local** variable.

The TP function block will be called in the LIGHT function block. TP must thus be declared in the LIGHT declaration section. In this case the instance name Pulse is assigned to the TP function block. The keyword for local variables is VAR:

```
VAR
  Pulse : TP ;
END_VAR
```

When a function block is called, the values to be processed are transferred to its input operands and the results are returned with the output variables. Additional variables are thus necessary in the invoking POU for the calling parameters and for receiving the results. The variable names are freely selectable.

In our example, the following variables are defined for use by the TP function block in the LIGHT function block:

Start	Start condition which is transferred to the IN operand
Pulse_duration	Time value setting for PT operand
Time_running	Boolean status of the timer which is output via the Q operand
Current_time	Current time value which is output via the ET operand

The Pulse_duration variable is declared as an input variable so that any time value can be transferred to the operand PT of the TP function block with the invocation of the LIGHT function block. The other variables can be declared as local variables since they are only valid within the LIGHT function block and not in the invoking Program POU EXP_PS4. The keyword for input variables is VAR_INPUT and for local variables VAR. The following declaration lines are thus required for the example:

```
VAR_INPUT
  Pulse_duration : TIME ;
END_VAR

VAR
  Start : BOOL ;
  Time_running : BOOL ;
  Current_time : TIME ;
END_VAR
```

The function block is **called** with the CAL instruction followed by the assigned instance name to address the function block, CAL Pulse in our example.

There are three methods available for **parameter transfer**. Two of them will be described here.

With the **first method** the parameters are transferred directly with the invocation of the function block. The parameters are entered in parentheses, separated by a comma. The “I” character is written between the input and the output parameters:

```
  CAL  Pulse (IN := Start,
       PT := Pulse_duration |
       Time_running := Q,
       Current_time := ET)
```

With the **second method,** the input parameters are individually loaded with the LD command and transferred to the function block operands using ST before the function block invocation. The output operands are scanned after the invocation. The syntax for specifying the function block operands is as follows:

Instance name.Operand

The invocation with parameter transfer according to the second method thus has the following form:

```
LD  Start
ST  Pulse.IN
LD  Pulse_duration
ST  Pulse.PT
CAL Pulse
LD  Pulse.Q
ST  Time_running
LD  Pulse.ET
ST  Current_time:
```

The choice of method depends on your personal preference. The only advantage of the second method is that it is one of the standard methods of calling FBs to IEC/EN 61131-3, and thus may be reusable by other IEC systems; with the more compact first method, the specification of the output parameters after the “I” character is Moeller specific. The invocation of TP in the example will be carried out according to the second method.

Completion of the program section for signal duration

A pulse generator is created by returning the negated output Q to the input IN:

```
LDN Time_running
ST  Start
```

The output Q has the status 1 as long as the pulse generator time is running. The status 0, which causes the timer to restart, is then active for the duration of one program cycle. The cycle time of the signal is approximately the value PT; the inaccuracy of one program cycle can be ignored.

The instruction lines for programming the signal duration are then:

```
LD  Start
ST  Pulse.IN
LD  Pulse_duration
ST  Pulse.PT
CAL Pulse
LD  Pulse.Q
ST  Time_running
LD  Pulse.ET
ST  Current_time
LDN Time_running
ST  Start
```

The program can now be simplified: The timer can be directly started with the negated status of the output operand Q. The Start and Time_running variables are no longer necessary. The task definition does not require interrogation of the current time value; however, the current_time variable is still included to allow the elapsed time to be displayed during the online test.

The program now has the following form:

```
LDN     Pulse.Q
ST      Pulse.IN
LD      Pulse_duration
ST      Pulse.PT
CAL     Pulse
LD      Pulse.ET
ST      Current_time
```

Structure of the LIGHT function block

The bit pattern is to be shifted by one position after the time PT has elapsed. This will be implemented with a jump label, which allows the rotation to be skipped as long as the pulse generator time is running. The bit pattern of the Light_strip variable should be shifted when the time has just elapsed, i.e. during status 0 of the output Q. A conditional jump to a jump label can be programmed with the JMPC command.

The jump is carried out if there is a 1 in the working register, i.e. if the current result = 1. The name of the jump label is freely selectable, in this case we will use not_shift:

```
LD      Pulse.Q
JMPC    Not_shift

LD      Light_strip
ROR     1
ST      Light_strip

Not_shift:          (*jump label*)
LDN     Pulse.Q
ST      Pulse.IN
LD      Pulse_duration
ST      Pulse.PT
CAL     Pulse
LD      Pulse.ET
ST      Current_time
```

The program can now be optimised, i.e. it can be shortened with a few small modifications.

The LDN Pulse.Q instruction, which provides the start condition for the pulse generator, can be used simultaneously as a condition for the jump label. The JMPC jump command must be changed to JMPCN, i.e. jump when current result = 0.

```
FUNCTION_BLOCK LIGHT

VAR_INPUT
  Pulse_duration : TIME ;(* time value setting *)
END_VAR

VAR_IN_OUT
  Light_strip : BYTE ;(* running light *)
END_VAR

VAR
  Current_time : TIME ;(* current time value*)
  Pulse : TP ;(* instance of the TP function block *)
END_VAR

  LDN      Pulse.Q                        (* current result=1 if timer elapsed*)
  ST       Pulse.IN                       (* start the timer*)
  JMPCN    Not_shift                      (* skip the bit pattern shift as long
                                          as the time is running *)

  LD       Light_strip
  ROR      1                              (* rotation of the bit pattern *)
  ST       Light_strip

not_push
  LD       Pulse_duration                 (* transfer of the time value *)
  ST       Pulse.PT
  CAL      Pulse                          (* invocation of the function block *)
  LD       Pulse.ET
  ST       Current_time                   (* display of the current value for
                                          the online test*)

END_FUNCTION_BLOCK
```

The program section where the variable Light_strip is rotated is only processed if the time has just elapsed in the timer TP. The timer is then started again and the rotation instructions are skipped for as long as the timer is running.

Program POU EXP_PS4

The LIGHT function block will be called in the program POU "EXP_PS4". The following two parameters must be transferred when called:

- the time value for defining the running speed
- the physical address where the running light is to be indicated.

The function block must be declared first. The instance name "Light_sequence" is assigned in the declaration section of the EXP_PS4 POU:

```
VAR
  Light_sequence : LIGHT ;
END_VAR
```

The running speed will be defined with a variable of data type TIME which is initialised with the value 500 ms which corresponds to a pulse frequency of 2 Hz. The variable is declared as a local variable since it is only required within this POU. The variable is assigned the name Running_speed:

```
VAR
  Running_speed : TIME := T#500ms ;
END_VAR
```

The Display variable is used to output the LED control signals to a specified output. It is declared as a local, directly represented variable. The variable declaration specifies the address of the PLC output, in this case output byte0. Following a cold start, the Display variable should be initialised with the value 1 as follows:

```
VAR
  Display AT %QB0.0.0.0 : BYTE := 1 ;
END_VAR
```

The same result could be achieved by initialising with the bit pattern 00000001:

```
Display AT %QB0.0.0.0 : BYTE := 2#00000001 ;
```

It is only necessary to call the LIGHT function block (under the declared instance name Light_sequence) in the instruction section of the EXP_PS4 POU.
We will use method one, as described earlier, for the parameter transfer:

CAL Light sequence (Pulse duration := Running speed,
Light strip := Display)

The EXP_PS4 program POU has the following form:

```
PROGRAM EXP_PS4

VAR
  Light_sequence : LIGHT ;
  Running_speed : TIME := T#500ms ;
END_VAR

VAR
  Display AT %QB0.0.0.0 : BYTE := 1 ;
END_VAR

   CAL Light_sequence(Pulse_duration :=Running_speed,
    Light_strip := Display)

END_PROGRAM
```

The following figure shows an overview of the steps to follow in the programming task, from program entry to the test run of the complete program.

- Entry of FB LIGHT
  - Variable declaration
  - Program entry
- Entry of EXP_PS4 program
  - Variable declaration
  - Program entry
- Topology configuration of the system components
- Program code generation
- Test and commissioning
  - Transfer to program to PLC
  - Start Program
  - Test program
  - Test and modify program online

Previous sections already covered the topology configuration, the program code generation and the Test and Commissioning (T & C) tool when describing programming task 1. The use of the Test and Commissioning tool (T & C) for programming task 2 will now be described in more detail.

In the following section, you will first enter the LIGHT function block and then the EXP_PS4 program.

Entering programming task 2

This section will show you how to enter the two required POUs in the POU Editor. The POUs which are necessary for programming task 2 were described in the previous section.

Create the LIGHT function block first and then the EXP_PS4 program POU. In principle, the order of entry can be changed, but the following order, i.e. starting with the POUs which are in the lowest structure level (in this case the LIGHT function block), is recommended. The LIGHT function block is called by the EXP_PS4 Program POU, and you should thus create it and check the syntax before it is called.

The variable declaration and the program entry in IL will now be described for both POUs.

First use the Navigator to create a new project folder (new project) with the name “learnPS4” in the folder “PROJECTS”:

▶ Use the menu point ‹ Projects → New...› or the respective icon.

🗋 *“Create New Project”*

The “Create New Project” dialog box opens:

▶ First select “C” for the drive and then the PROJECTS folder.

▶ Then enter “learnPS4” in the “New project folder” field as the name of the folder for the new project.

▶ Confirm with OK.

Entering the LIGHT function block

To create the function block, use the Navigator menu:

▶ Select ‹ Tools → POU editor› or the respective button in the toolbar.

"POU Editor"

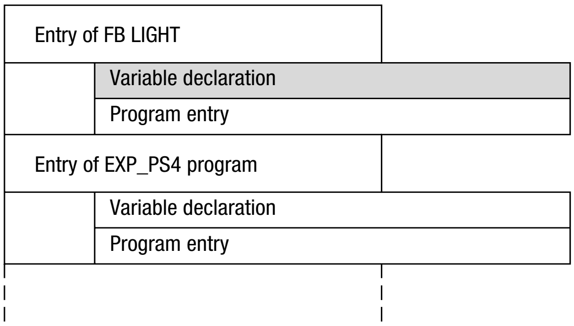
The POU Editor opens.

▶ Click on the FB button in the standard toolbar since the "Program" type of POU required is a function block or,

▶ Select ‹ File → New POU› , then select the "function block" POU type.

The two windows "Declaration and Instruction section" open, arranged on the screen as specified in the basic settings of the POU Editor.

Declaration of the variables



The variables will be declared in the Syntax-Controlled Variable Editor (Syntax mode). In this mode you do not have to worry about the keywords or the syntax.

We already decided on the following variables in the discussion of the program example:

```
VAR_INPUT
  Pulse_duration : TIME ;
    (* time value setting *)
END_VAR

VAR_IN_OUT
  Light_strip : BYTE ;
    (* running light *)
END_VAR

VAR
  Current_time : TIME ;
    (* current time value*)
  Pulse : TP ;
    (* instance of the function block*)
END_VAR
```

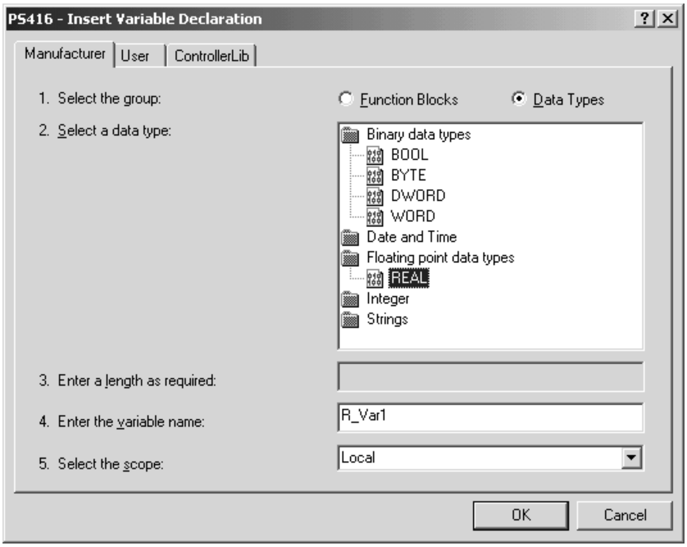
▶ Change to syntax mode – if necessary – with ‹ Options → Variable editor → Syntax mode› or use the corresponding button.

“Syntax mode”

The “local” scope is preset for the creation of function blocks when the variable editor is opened. The tab for the selected variable type is displayed in the foreground.

▶ In order to input the variables, use the menu point “Insert” and “Variable declaration...”.

The following dialog appears:


PS416 - Insert Variable Declaration

Manufacturer | User | ControllerLib

1. Select the group: ○ Function Blocks [✓ ● Data Types]

2. Select a data type:

- Binary data types
  - BOOL
  - BYTE
  - DWORD
  - WORD
- Date and Time
- Floating point data types
  - REAL
- Integer
- Strings

3. Enter a length as required:

4. Enter the variable name: R_Var1

5. Select the scope: Local

OK | Cancel

Figure 55: Variable declaration

Begin with the “Current_time” variable.

▶ The variables must be assigned to a “Data type”. Select the “Data types” group.

In the lower window, all PLC data types are listed and are represented in various groups.

▶ Open the “Date and Time” group and select “TIME”.

You do not need to input a length here, i.e. this is only useful with strings.

▶ Now input the “Current_time” variable names.

▶ Define the scope of the variables, i.e. “LOCAL” in this case.

After OK, the variable is entered into the syntax-controlled variable editor.

Proceed accordingly with the declaration of “Pulse”:

▶ “Pulse” should be assigned to a Manufacturer function block. Select “Function blocks” as the group.

▶ Search for the “timer modules” group under the function blocks and select the “TP” module.

You do not need to define the length here also.

▶ State the instance name of your module: “Pulse”.

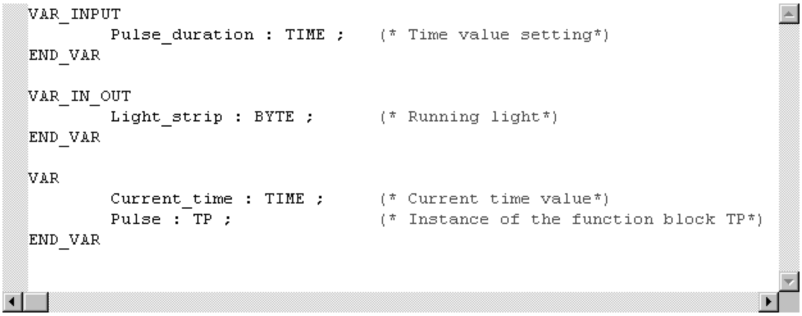
▶ The scope is “Local” here.

You can also input the variables completely via the keyboard, if you do not know the names.

▶ Repeat the steps for “Light_strip” and “Pulse_duration” also.

“Light_strip” should have the “In_Out” scope assignment, “Pulse_duration” should have the “Input” assignment.

▶ Switch over to the free variable editor in order to view the complete range of declared variables.



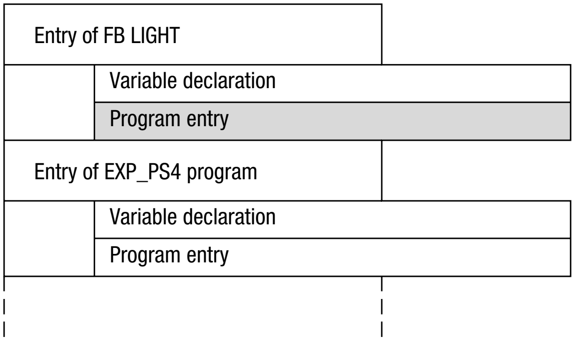
```
VAR_INPUT
        Pulse_duration : TIME ;     (* Time value setting*)
END_VAR

VAR_IN_OUT
        Light_strip : BYTE ;        (* Running light*)
END_VAR

VAR
        Current_time : TIME ;       (* Current time value*)
        Pulse : TP ;                (* Instance of the function block TP*)
END_VAR
```

Figure 56: Declared variables of the LIGHT function block

Entering the instructions in the IL Editor



▶ To use the IL programming language, change to IL if necessary with ‹ Options ➞ Programming Language ➞ IL› or the corresponding button.

“POU Editor IL”

▶ Enter the program in accordance with the figure wihch indicates the “Instruction section of the LIGHT function block“. Observe the following points:

You can use a mixture of upper case and lower case characters.

Write your comments using brackets and asterisks as shown overleaf.

Separate operators, operands and comments with a tab to give a better overview.

Separate instruction sequences which form a self-contained unit from the other instructions using blank lines.

▶ Save the POU with ‹ File → Save As...› or by clicking on the corresponding button.

“Save”

In the “Save As” dialog which opens, the current project is already set.

▶ Enter the name LIGHT and confirm with OK.

▶ Carry out a syntax check with ‹ File → Syntax check› , or click on the corresponding button.

“Syntax Check”

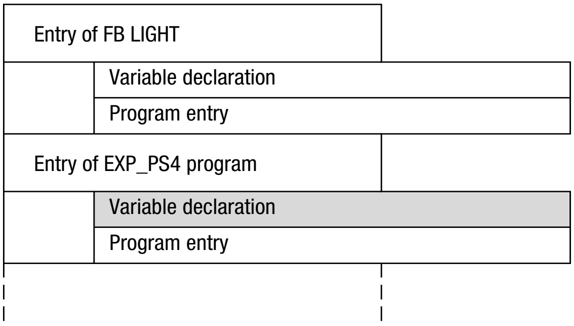
The syntax check should complete successfully if the POU has been entered correctly. If the syntax check reports errors, check everything you have entered, correct any errors you find, and run the syntax check again, the POU will be saved automatically.

Entering the EXP_PS4 program

Stay in the POU Editor in syntax mode with IL selected as the programming language.

▶ Create a new POU of type Program with ‹ File → New POU› or the P button in the standard toolbar.

Declaration of the variables



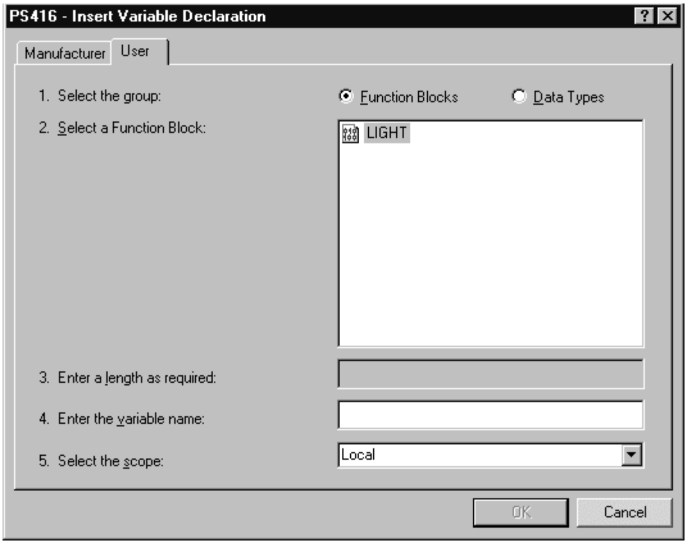
We already decided on the following variables in the discussion of the program example:

```
VAR
  Light_sequence : LIGHT ;
  Running_speed : TIME := T#500ms ;
END_VAR

VAR
  Display AT %QB0.0.0.0 : BYTE := 1 ;
END_VAR
```

Local variables are the default type in Program POUs.

First declare the LIGHT function block which you have just created. Proceed in the same way as with the declaration of the “Pulse” manufacturer function block.


PS416 - Insert Variable Declaration

Manufacturer | User

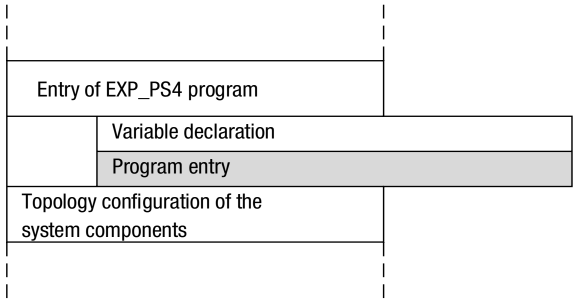
1. Select the group: (● Function Blocks) ○ Data Types
2. Select a Function Block: LIGHT
3. Enter a length as required:
4. Enter the variable name:
5. Select the scope: Local

Figure 57: Variable declaration for the LIGHT function block

- ▶ Select the “User” tab. You will see the “LIGHT” function block listed here.
- ▶ Mark it and enter the name “Light_sequence”.
- ▶ Declare the local variable “Running_speed”. Enter the variable name and the data type and enter the time value t#500ms in the Initial Value field.
- ▶ Declare the local variable “Display”. Enter the variable name and the data type as initial value “1”, and define QB0.0.0.0 as the address.

The declaration section of the POU “program” is now complete.

Program entry in the instruction section



▶ Switch from the declaration section to the instruction section and continue with the programming language IL.

The LIGHT function block must be called with the instance name declared in the POU instruction section. The invocation and the parameter transfer can be entered manually.

With Sucosoft S40, it is also possible to automatically incorporate a predefined function block call.
In order to do this, the function block must be in the current project and the syntax check must already have been executed without errors. Both conditions are fulfilled in our example.

☞ Manufacturer-defined function blocks are project-independent and can be used in any project without any additional steps.

▶ Select ‹ Insert → Insert variable› .

The following dialog appears:

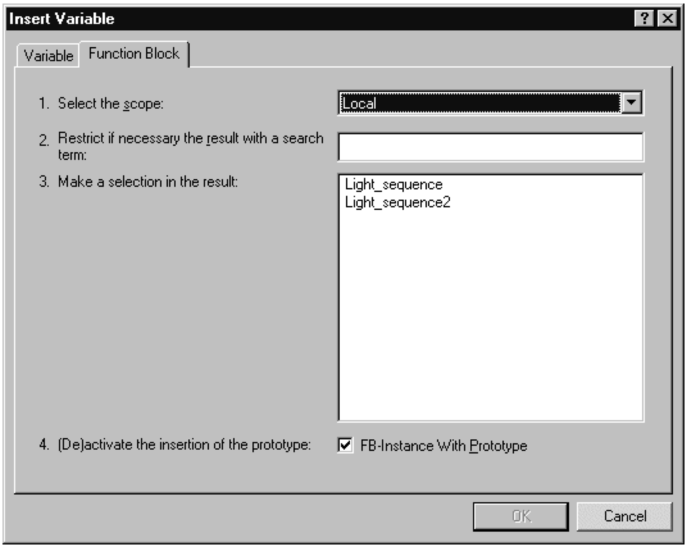
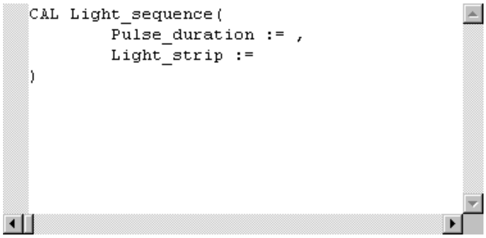

Insert Variable
Variable
Function Block
1. Select the scope:
Local
2. Restrict if necessary the result with a search term:
3. Make a selection in the result:
Light_sequence
Light_sequence2
4. (De)activate the insertion of the prototype:
☑ FB-Instance With Prototype
OK
Cancel

Figure 58: Inserting variables

You can select the scope of the function block above. You now only see the respective function blocks which were assigned to the respective area during declaration. You have created your “Light_sequence” function block as LOCAL.

▶ Select “Light_sequence” and ensure that “FB-Instance With Prototype” is selected.

The following template appears in the instruction section:



```
CAL Light_sequence(
        Pulse_duration := ,
        Light_strip :=
)
```

Figure 59: Template

▶ Fill out this template as follows:

```
CAL Light_sequence(Pulse_duration := Running_speed,

  Light_strip :=Display )
```

The “Pulse_duration” input variable is assigned with the “Running_speed” variable, the IN_OUT variable “Light_strip” is assigned with “Display”.

☞ If you enter the invocation of the function block manually, remember that parameters for IN_OUT variables must always be specified within the brackets, and that IN_OUT parameters must always be assigned a value.

▶ Save the POU under the name EXP_PS4.

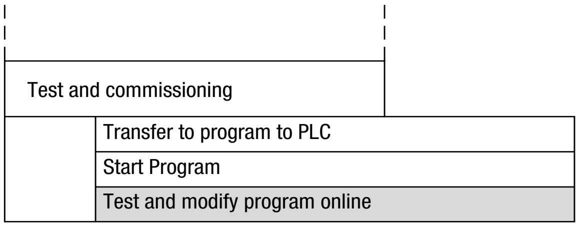
▶ Carry out the syntax check.

The syntax check should complete successfully if you have entered the program correctly. If the syntax check reports errors, check everything you have entered, correct any errors you find, save the function block and run the syntax check again.

Before the programming example can be checked and modified online, the syntax must already be checked and it must be converted into an executable PLC program, transferred to the controller and started. In order to do this, you must define the topology and generate the program code. This was already explained in detail in chapters Section 7 and Section 8 for programming example 1. Follow the same procedure for programming example 2.

Accordingly, the following sections will only describe program test and online modifications for programming example 2.

Testing and modifying the program online



Each of the POUs of the program can be displayed and modified if necessary while the program is running in the controller. We will show you how to change the running direction of the LEDs and the speed. In order to do this, the Program window must be open, the LIGHT POU selected and the corresponding instruction section must be displayed.

▶ Select the “Program” button in the T & C, when the connection list window is open.

“Program”

The “Program” window opens.

Initially, only “RESOURCE” can be seen in the upper level in the left hand section of the window; the name of the POU program appears on the right. The lower levels of the entire program can be displayed by double clicking RESOURCE in the left window. If a program is marked in the left hand window, the program components of the lower levels appear in the right hand window; i.e. function block POUs.

▶ Double click on “Resource”.

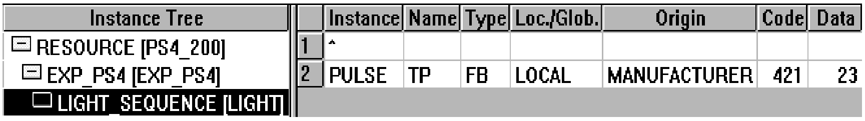
The name of the POU Program “EXP_PS4” appears in the left hand window.

▶ Double click in the left hand window on “EXP_PS4 [EXP_PS4]”.

The subordinate “LIGHT” function block appears.

▶ Double click in the left hand window on “Light_sequence [LIGHT]”.

The “TP” function block is displayed on the right hand window as the lowest level.



Instance Tree
⊟ RESOURCE [PS4_200]
⊟ EXP_PS4 [EXP_PS4]
▭ LIGHT_SEQUENCE [LIGHT]

	Instance	Name	Type	Loc./Glob.	Origin	Code	Data
1	*						
2	PULSE	TP	FB	LOCAL	MANUFACTURER	421	23

▶ Mark the “LIGHT_SEQUENCE [LIGHT]” function block and select the “View/Change POU” button.

View/Change POU

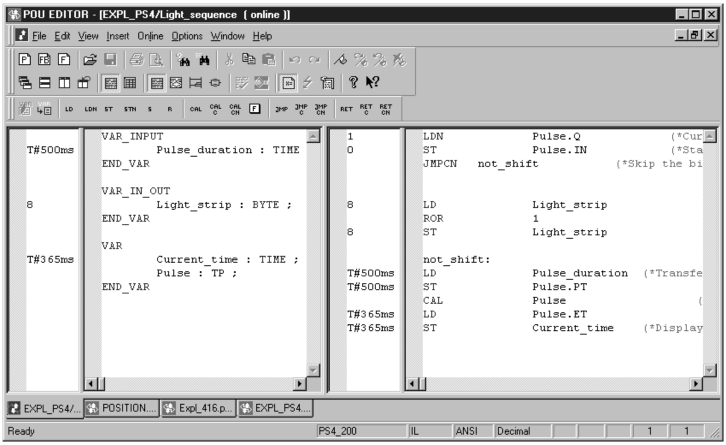
The POU Editor opens in Online mode with the selected POU.

You can display the variable states in this window. This can be helpful when troubleshooting.

▶ Select the “Status display” command in the Online menu.

Status display

In the instruction and declaration section, the states of the individual variables are displayed, and their values are continuously updated:


POU EDITOR - [EXPL_PS4/Light_sequence [online]]
File Edit View Insert Online Options Window Help
LD LDN ST STN S R CAL CAL C CAL CN F JMP JMP C JMP CN RET RET C RET CN
EXPL_PS4/... POSITION.... Expl_416.p... EXPL_PS4....
Ready PS4_200 IL ANSI Decimal 1 1

```
            VAR_INPUT
T#500ms             Pulse_duration : TIME
            END_VAR

            VAR_IN_OUT
8                   Light_strip : BYTE ;
            END_VAR

            VAR
T#365ms             Current_time : TIME ;
                    Pulse : TP ;
            END_VAR
```

```
1          LDN               Pulse.Q              (*Cur
0          ST                Pulse.IN             (*Sta
           JMPCN   not_shift               (*Skip the bi

8          LD                Light_strip
           ROR               1
8          ST                Light_strip

           not_shift:
T#500ms    LD                Pulse_duration  (*Transfe
T#500ms    ST                Pulse.PT
           CAL               Pulse                   (
T#365ms    LD                Pulse.ET
T#365ms    ST                Current_time    (*Display
```

Figure 60: IL Online Editor showing status display of the variables

▶ Cancel the update of the variable states by choosing the Status Display command again or the Status Display button. The previously updated values are now displayed as “frozen” values on the screen.

Modifications in the POU Editor (Online mode)

You can modify your program in online mode with the POU Editor. This saves time because it is not necessary to carry out tasks such as saving the POU, generating program code and transferring the new program to the controller. When you press the “Update” button, every change you have made is automatically updated in the respective POU source file and in the executable program.

Changing the running direction of the LEDs
You can change the running direction of the LEDs by replacing the ROR (rotate right) command by the ROL (rotate left) command.

☞ The arrangement of the digital outputs on PS4 controllers starting with the least significant bit on the far left and the most significant bit on the right means that, with the ROR command, the running light moves to the left and with ROL to the right. This is because the Rotate commands refer to the standard digital representation which places the least significant bit on the right.

The light running speed can be modified to 1 Hz by redefining the time value for the TP function block to 1 second.

▶ Click in the program line which contains the “ROR” command and change it to “ROL”.

The command and the “Activate” button are now available.

▶ Click the Activate button.

“Activate”

The modification is automatically saved in the function block and a new program code is generated. The modified program code is then transferred to the controller automatically and has an immediate effect on the running direction of the executing program, i.e. the running direction of the LEDs connected to the output module is reversed.

Changing the running speed of the LEDs

The running speed was defined by assigning the input variable “Pulse_duration” to the PT input operand in the TP function block. The actual time value is only transferred when the LIGHT function block is called, whereby the variable “Pulse_duration” is given the value of the variable “Running_speed” declared in the program.
This value was initialised in the declaration section of the program POU with 500 ms.

Since only the instruction section and not the declaration section of the program POU can be modified in the Online Editor, the new time value must be transferred in the form of a time constant when one of the two function blocks is called.
This time constant could have already been defined in the LIGHT function block and transferred to the TP standard function block, but the LIGHT function block would then no longer be “general-purpose”.
The new time value should thus be specified in the program POU EXP_PS4 and transferred to the LIGHT function block when it is called. The time constant of 1 second should now be transferred with the “Pulse_duration” variable and not with the “Running_speed” variable which was initialised with 500 ms:

```
CAL Light_sequence(
  Pulse_duration := T#1s,
  Light_strip :=Display
)
```

▶ Close the POU which is opened online.

▶ Select “POE” “EXP_PS4” from the structure tree: Mark the POU in the structure tree on the left side of the program window, and confirm with the “Display/change POU” button.

▶ Click with the mouse on the position to be modified.

- ▶ Change the time value setting into the constant T#1s.
- ▶ Click the Activate button.

“Activate”

The modification is saved in the programming device and in the controller.

The required modification is complete: The direction of the running light has reversed and the running speed has changed.

- ▶ Close the POU which is opened online.
- ▶ Close the T & C tool.

Multiple instances of the Light FB

In order to demonstrate the advantage of address-independent function block programming with local variables, we will now show how to make a simple extension to programming task 2.

The function block LIGHT which you created once and can now be considered as a completed and tested module, will be instanced again to control a further running light. If you are using a PS4-200 compact PLC for the programming example, you can connect a further running light to the output byte of an EM4-101-DD1 expansion module. With the PS416 modular PLC, the output of the second running light should occur on output byte 1 of the PS416-OUT-400 digital output group. A modification of the topology configuration for the PS416 is not necessary.

With the PS4, you must add the expansion module to the topology configuration.

- ▶ Open the topology configuration “DEVICE” in the topology configurator.
- ▶ Mark the PS4 and select ‹ Edit ➞ Decentral expansion› .
- ▶ Select “EM4-101-DD1/88” from the list and confirm with OK.

This configures the expansion module for 8 inputs and 8 outputs.

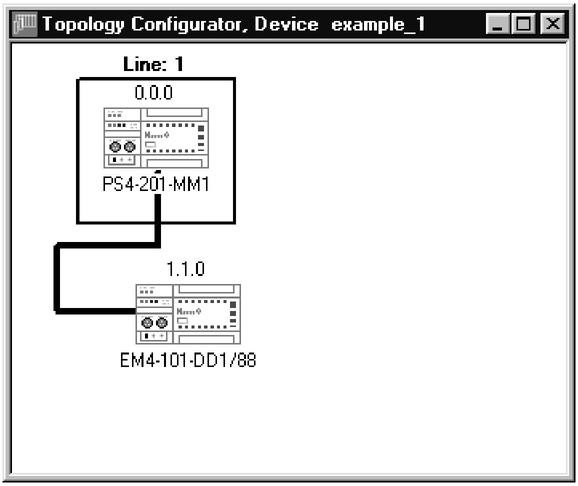


Figure 61: Topology configurator

- ▶ Save the configuration.
- ▶ Open the EXP_PS4 program in the POU Editor and extend the variable declaration in the variable editor to add the variables needed for a second running light.

The LIGHT function block is declared again under the instance name “Light_sequence2” to create a second instance of this function block.

To demonstrate that the second instance runs independently of the first instance called “Light_sequence” without further programming effort, declare two further variables for the parameters of “Light_sequence2”.

The variable “Running_speed2” is initialised with the value T#250ms which results in a pulse frequency of 4 Hz – twice as high as for the first running light. In the case of the PS4, the second running light is output to the output byte of the EM4-101-DD1. The first digit in the address specifies the network line to which the station is connected, in this case line 1. In our example, the EM4-101-DD1 is the first station on this string, the second digit of the address is thus also 1. The expansion module is module 0 since it is directly connected to the network line as a station. The module number is third digit of the address. The fourth digit of the address specifies the byte to be addressed, i.e. byte 0. The complete address for the output of the second running light is thus %QB1.1.0.0 for the (sole) output byte of the EM4-101-DD1.

The extended variable declaration for the compact PLC is then as follows:

```
VAR

  Light_sequence: LIGHT ;
  Running_light: TIME := T#500ms ;
  Display AT %QB0.0.0.0 : BYTE := 1 ;
  Light_sequence2 : LIGHT ;(* Second instance of the
                             LIGHT function block   *)

  Running_light2 : TIME := T#250ms ; (* Pulse duration for
second
                                light sequence => 4 Hz *)

  Display2 AT %QB1.1.0.0 : BYTE := 1;(* Output to the
                                     EM4-101-DD1       *)

END_VAR
```

The corresponding extended variable declaration for the PS 416 PLC is as follows:

```
VAR

  Light_sequence: LIGHT ;
  Running_light: TIME := T#500ms ;
  Display AT %QB0.0.0.0 : BYTE := 1 ;
  Light_sequence2: LIGHT ;(* Second instance of the
                           LIGHT function block *)

  Running_speed 2: TIME := T#250ms ; (*Pulse duration for
second
                                light sequence => 4 Hz *)

  Display2 AT %QB0.0.0.1 : BYTE := 1; (*Output to the
                                      HIGH Byte of the OUT-
400
                                      *)

END_VAR
```

The call for the second instance of the function block now has to be entered in the instruction section.

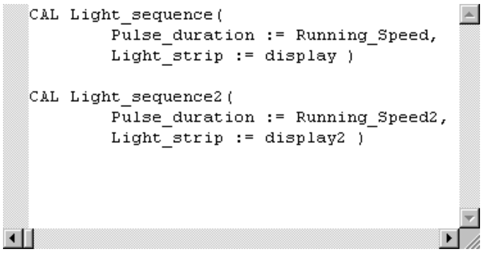
▶ Position the cursor at the end of the program.

▶ Select ‹ Insert → Insert variable...› or use the respective button.

"Insert variable"

Now select the “LOCAL” scope in the dialog if necessary, and double click on “Light_sequence2”.

The new module appears as a call template with all operands, which you have assigned with the variables “Running_speed2” and “Display2”.



```
CAL Light_sequence(
        Pulse_duration := Running_Speed,
        Light_strip := display )

CAL Light_sequence2(
        Pulse_duration := Running_Speed2,
        Light_strip := display2 )
```

Figure 62: Call of Light_sequence and Light_sequence2

- ▶ Save the POU and create a new make file using the Code Generation tool.
- ▶ Then generate the new program with ‹ Generate → Generate program code› or the corresponding button and transfer it to the controller.

After starting the program, you will see that the running light connected to the EM4-101-DD1 runs twice as fast as the light connected to the PS4. On the PS416-OUT-400, the running light on the HIGH byte runs twice as fast as the light connected to the LOW byte. With this programming exercise, you have instanced and used a custom-written function block twice with the minimum of effort without concerning yourself with where the local variables used internally in the LIGHT FB are stored in the controller. The manufacturer-defined function block TP used in the LIGHT function block has also been instanced several times automatically without needing to worry about the assignment of data ranges or marker addresses.

The extended program EXP_PS4 now appears in LD and FBD as follows:

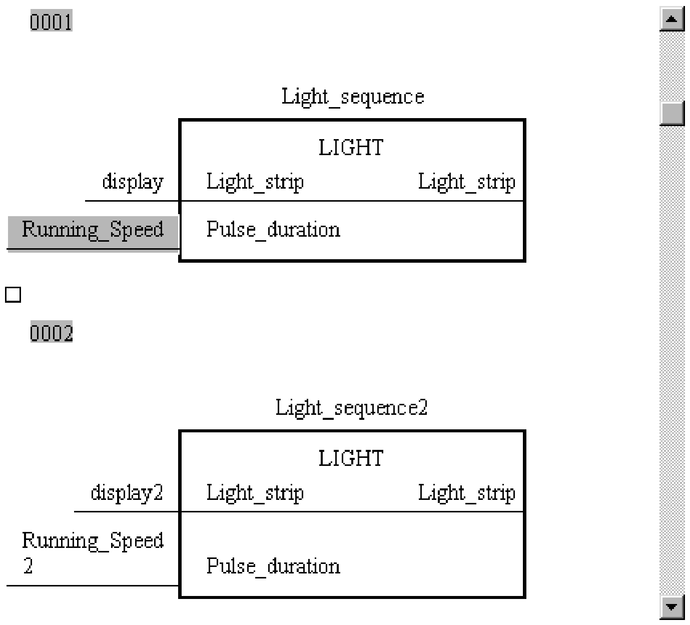


Figure 63: Extended program in LD/FBD

Displaying/forcing inputs/outputs on the PS4

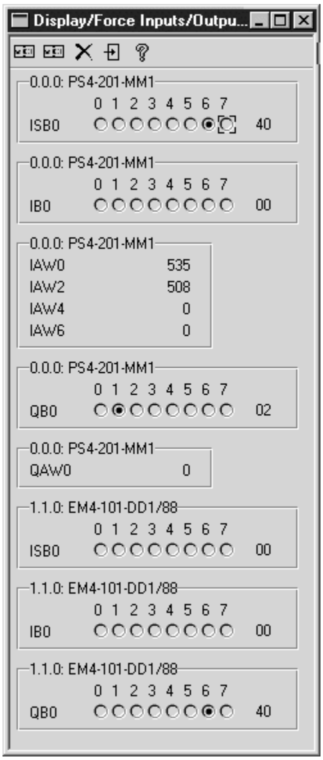
We will now show how to display the states of the inputs/outputs and the diagnostic data for the controller and the expansion module.

▶ Start the T & C.

▶ Click the Topology button. The current topology configuration is read from the controller and displayed.

▶ Mark both the PS4-141-MM1 and EM4-101-DD1/88 devices with the left mouse button while keeping the “Ctrl” button pressed, and click on the “Display/Force Inputs/Outputs...” button.

The inputs and outputs of the selected devices are displayed in a single window as follows. With a CPU of the PS4 type, the inputs are displayed in RUN as well as in STOP; with a CPU of the PS416 type, the outputs are only displayed in the STOP state.



```
Display/Force Inputs/Outpu...

0.0.0: PS4-201-MM1
         0 1 2 3 4 5 6 7
ISB0     ○ ○ ○ ○ ○ ○ ◉ ○    40

0.0.0: PS4-201-MM1
         0 1 2 3 4 5 6 7
IB0      ○ ○ ○ ○ ○ ○ ○ ○    00

0.0.0: PS4-201-MM1
IAW0                535
IAW2                508
IAW4                  0
IAW6                  0

0.0.0: PS4-201-MM1
         0 1 2 3 4 5 6 7
QB0      ○ ◉ ○ ○ ○ ○ ○ ○    02

0.0.0: PS4-201-MM1
QAW0                  0

1.1.0: EM4-101-DD1/88
         0 1 2 3 4 5 6 7
ISB0     ○ ○ ○ ○ ○ ○ ○ ○    00

1.1.0: EM4-101-DD1/88
         0 1 2 3 4 5 6 7
IB0      ○ ○ ○ ○ ○ ○ ○ ○    00

1.1.0: EM4-101-DD1/88
         0 1 2 3 4 5 6 7
QB0      ○ ○ ○ ○ ○ ○ ◉ ○    40
```

Figure 64: Status indication of the inputs/outputs

A monitoring function can be activated by clicking the “Network Diagnostics” button in the Test & Commissioning tool to check the status of each of the devices in a networked system. Faulty devices or cards are highlighted with hatching. The master of a Suconet K string where a station is faulty is additionally marked with a lightning flash. You can demonstrate this feature if you remove the “connection cable” between the controller and the EM4-101-DD1.

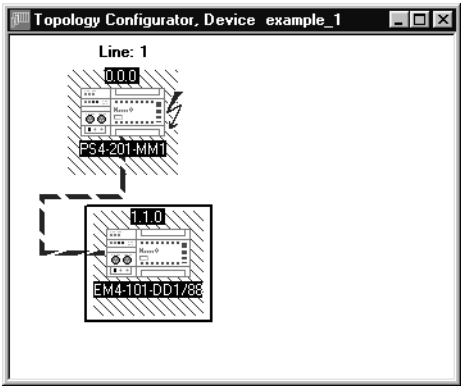
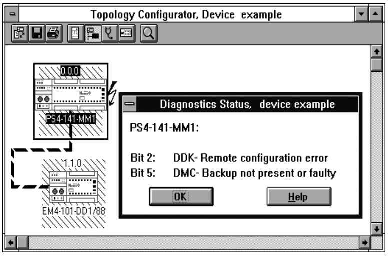


Figure 65: Network disgnostics with a faulty device

You can get more detailed information on the fault if you click the Diagnostic Status... button and select a device or card with the mouse.

"Diagnostic Status" button

The following dialog appears:


Topology Configurator, Device example
0.0.0
PS4-141-MM1
1.1.0
EM4-101-DD1/88
Diagnostics Status, device example
OK
Help

PS4-141-MM1:

Bit 2: DDK- Remote configuration error
Bit 5: DMC- Backup not present or faulty

Figure 66: Diagnostic status of the faulty device

You have now got to know the main tools of the Sucosoft S40 software. Of course, this Training Guide is not designed to demonstrate all of the numerous features provided by Sucosoft S40, but we hope at least that you got a good overview and are looking forward to trying out further possibilities.

You can find more details on each of the steps in the Sucosoft S40 Reference Manuals “User interface” (AWB2700-1305-GB) and “Language Elements for PS4-200, PS4-300 and PS416” (AWB2700-1306-GB). These additional manuals describe in detail all available features of the Sucosoft software and the Sucosoft language elements.

LD/FBD representation of programming task 2

To round off, the following pages show the instruction sections of programming task 2 for the EXP_PS4 program in the graphical programming languages LD and FBD.

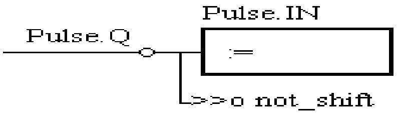
The procedures to follow for generating an executable program, for transferring it to the controller and starting it are the same as for the programming language IL.

LIGHT function block

FBD representation:

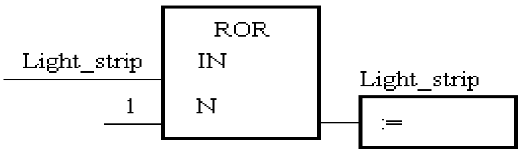
0001

```
Current result =1 if timer has elapsed
Starting the timer
Skip the bit pattern shift as long as
the time is running
```



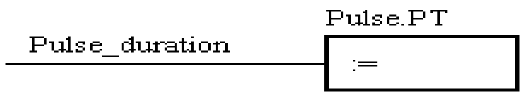
0002

```
Rotation of the bit pattern
```



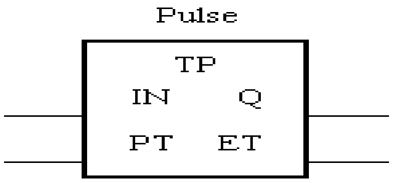
0003 not_shift

Transfer of the time value



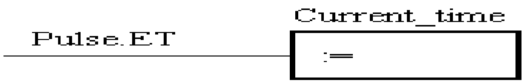
0004

Invocation of the function block



0005

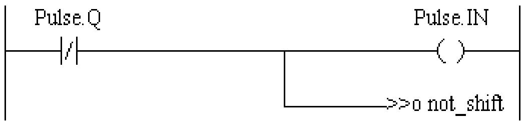
Display of the current value for
the online test



LD representation:

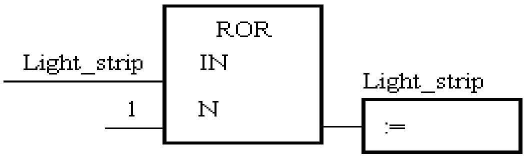
0001

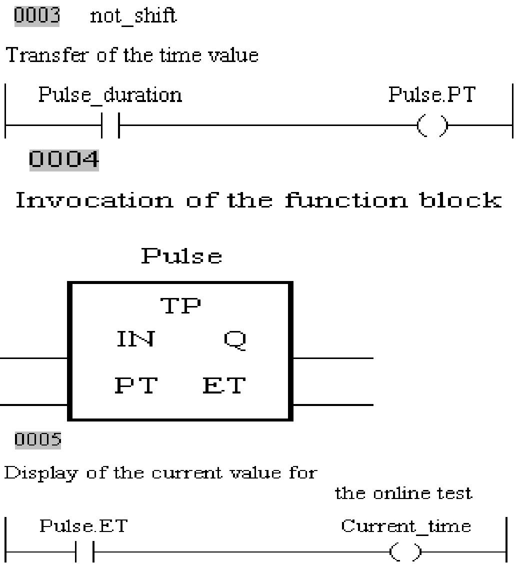
Current result =1 if timer has elapsed. Starting the timer
Skip the bit pattern shift as long as the time is running



0002

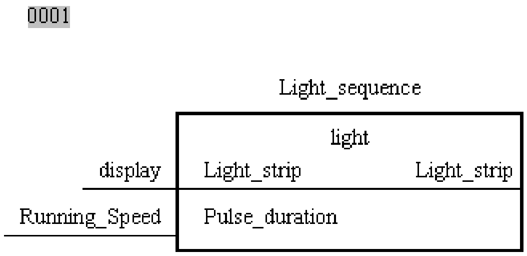
Rotation of the bit pattern




0003 not_shift
Transfer of the time value
Pulse_duration
Pulse.PT
0004
Invocation of the function block
Pulse
TP
IN
Q
PT
ET
0005
Display of the current value for the online test
Pulse.ET
Current_time

EXP_PS4 program

LD and FBD representation:



Index

List of revisions for manual AWB 27-1307 GB

Edition	Page	Topic	New	Changes	Invalid
04/99	General	Sucosoft S 30-S4			×
		Sucosoft S 4 → S 40		×	
		AWB 27-1185/1186			×
		AWB 27-1280-GB → AWB 2700-1305 GB		×	
		AWB 27-1281-GB → AWB 2700-1306 GB		×	
	14	Legend ③	×		
	41	Slave address		×	
	52	Note	×		
	52/53	Graphics/Table		×	
	83	EMC: RFI, Surge		×	
06/99	Entire manual	Rework for Version 4.0, Main areas: Section 5 – 7, 9 – 10		×	