

中国工控网
www.chinakong.com
资料中心

Contents

1	A Brief Introduction to 907 AC 1131	1-1
1.1	What is 907 AC 1131	1-1
1.2	Overview of 907 AC 1131 Functions.....	1-1
2	What is What in 907 AC 1131	2-1
2.1	Project Components.....	2-1
2.2	Languages	2-11
2.2.1	Instruction List (IL).....	2-11
2.2.2	Structured Text (ST).....	2-13
2.2.3	Sequential Function Chart (SFC)	2-20
2.2.4	Function Block Diagram (FBD).....	2-25
2.2.5	Ladder Diagram (LD).....	2-26
2.3	Debugging, Online Functions	2-28
2.4	The Standard	2-29
3	We Write a Little Program	3-1
3.1	Controlling a Traffic Signal Unit.....	3-1
3.2	Visualizing a Traffic Signal Unit.....	3-14
4	The Individual Components	4-1
4.1	The Main Window	4-1
4.2	Options.....	4-4
4.3	Managing Projects	4-14
4.4	Creating and Deleting Objects, etc.....	4-27
4.5	General Editing Functions	4-35
4.6	General Online Functions.....	4-41
4.7	Window set up.....	4-51
4.8	Help when you need it.....	4-52

5	Editors in 907 AC 1131	5-1
5.1	The Declaration Editor	5-1
5.2	The Text Editors	5-8
5.2.1	The Instruction List Editor	5-13
5.2.2	The Editor for Structured Text.....	5-13
5.3	The Graphic Editors.....	5-14
5.3.1	The Function Block Diagram Editor	5-16
5.3.2	The Ladder Editor	5-23
5.3.3	The Sequential Function Chart Editor	5-29
6	The Resources	6-1
6.1	Overview of the Resources.....	6-1
6.2	Global Variables	6-1
6.2.1	Access Variables	6-2
6.2.2	Global Variables	6-3
6.2.3	Variable Configuration	6-3
6.2.4	Document Frame	6-5
6.3	PLC Configuration	6-6
6.3.1	Working in the PLC Configuration.....	6-7
6.3.2	Doing the PROFIBUS-DP Configuration.....	6-7
6.4	Task Configuration	6-22
6.5	Sampling Trace	6-25
6.6	Watch and Receipt Manager	6-32
7	Library Manager.....	7-1
8	Visualization.....	8-1
8.1	Create Visualization.....	8-1
8.2	Visualization Elements, Insert.....	8-2
8.3	Working with Visualization Elements	8-4
8.4	Visualization Elements, Configure	8-5
8.5	Additional Visualization Element Functions	8-15
9	DDE Interface	9-1
10	Appendix	10-1

Appendix A: Use of Keyboard	10-1
Key Combinations	10-1
Appendix B: Data types	10-5
Standard Data types	10-5
Defined Data Types	10-6
Appendix C: IEC Operators	10-11
Bitstring Operators	10-13
Bit-Shift Operators.....	10-15
Selection Operators	10-19
Comparison Operators.....	10-21
Address Operators.....	10-23
Calling Operator	10-24
Type Conversion Functions	10-24
Numeric Functions	10-30
Appendix D: Standard Library Elements	10-35
String Functions	10-35
Bistable Function Blocks	10-39
Trigger	10-41
Counter	10-42
Timer	10-45
Appendix E: Operands in 907 AC 1131	10-49
Operands	10-49
Constants.....	10-49
Variables	10-51
Addresses	10-52
Functions	10-53

Appendix F: Command Line/Command File Commands	10-55
Command Line Commands.....	10-55
Command File (cmdfile) Commands.....	10-55
Appendix G: Error messages	10-59
11 Index	i

1.1 What is 907 AC 1131

907 AC 1131 is a complete development environment for your PLC.

907 AC 1131 puts a simple approach to the powerful IEC language at the disposal of the PLC programmer. Use of the editors and debugging functions is based upon the proven development program environments of advanced programming languages (such as Visual C++).

1.2 Overview of 907 AC 1131 Functions

How is a project structured?

A project is put into a file named after the project. The first POU (Program Organization Unit) created in a new project will automatically be named **PLC_PRG**. The process begins here (in compliance with the main function in a C program), and other POUs can be accessed from the same point (programs, function blocks and functions).

Once you have defined a Task Configuration, it is no longer necessary to create a program named PLC_PRG. You will find more about this in the Task Configuration chapter.

There are different kinds of objects in a project: POUs, data types, display elements (visualizations) and resources. The Object Organizer contains a list of all the objects in your project.

How do I set up my project?

First you should configure your PLC in order to check the accuracy of the addresses used in the project.

Then you can create the POUs needed to solve your problem.

Now you can program the POUs you need in the desired languages.

Once the programming is complete, you can compile the project and remove errors should there be any.

How can I test my project?

Once all errors have been removed, activate the simulation, log in to the simulated PLC and "load" your project in the PLC. Now you are in Online mode.

Now open the window with your PLC Configuration and test your project for correct sequence. To do this, enter input variables manually and observe whether outputs are as expected. You can also observe the value sequence of the local variables in the POUs. In the Watch and Receipt Manager you can configure data records whose values you wish to examine.

Debugging

In case of a programming error you can set breakpoints. If the process stops at such a breakpoint, you can examine the values of all project variables at this point in time. By working through sequentially (single step) you can check the logical correctness of your program.

Additional Online Functions

An additional debugging function: You can set program variables and inputs and outputs at certain values. You can use the flow control to check which program lines have been run. The Sampling Trace allows you to trace and display the actual course of variables over an extended period of time.

Once the project has been set up and tested, the hardware can be loaded in the hardware and tested as well. The same online functions you used with the simulation are available.

Additional **907 AC 1131** Features

The entire project can be documented or exported to a text file at any time.

Summary

907 AC 1131 is a complete development tool used to program your PLC which will save you a measurable amount of time setting up your applications.

This chapter contains a list of the most important concepts to make starting easier.

2.1 Project Components

Project

A project contains all of the objects in a PLC program. A project is saved in a file named after the project. The following objects are included in a project:

POUs (Program Organization Units), data types, visualizations, resources, and libraries.

POU (Program Organization Unit)

Functions, function blocks, and programs are POUs.

Each POU consists of a declaration part and a body. The body is written in one of the IEC programming languages which include IL, ST, SFC, FBD and LD.

907 AC 1131 supports all IEC standard POUs. If you want to use these POUs in your project, you must include the library `standard.lib` in your project.

POUs can call up other POUs. However, recursions are not allowed.

Function

A function is a POU, which yields exactly one data element (which can consist of several elements, such as fields or structures) when it is processed, and whose call in textual languages can occur as an operator in expressions.

When declaring a function do not forget that the function must receive a type. This means, after the function name, you must enter a colon followed by a type.

A correct function declaration can look like this example:

```
FUNCTION Fct:    INT
```

In addition, a result must be assigned to the function. That means that function name is used as an output variable.

A function declaration begins with the keyword `FUNCTION` and ends with `END_FUNCTION`.

Example in IL of a function that takes three input variables and returns the product of the first two divided by the third:

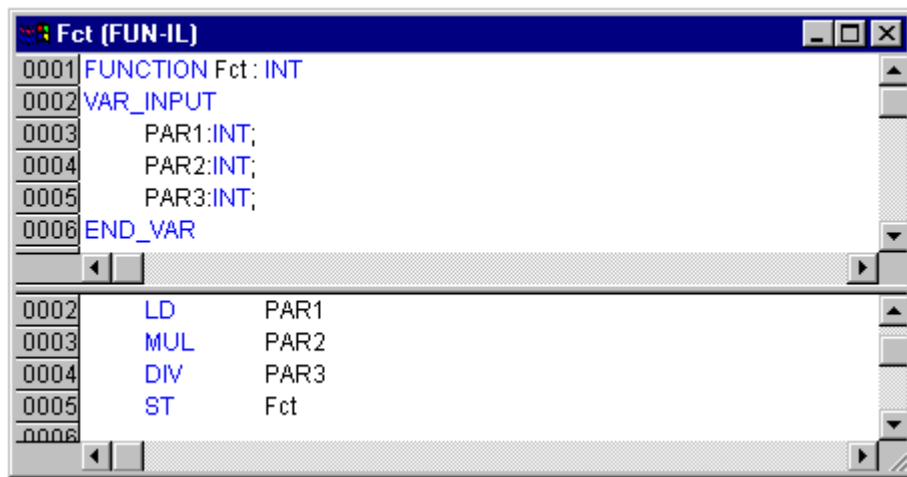


Image 2.1: Function

The call of a function in ST can appear as an operand in expressions.

Functions do not have any internal conditions. That means that calling up a function with the same argument (input parameters) always produces the same value (output).

Examples for calling up the function described above:

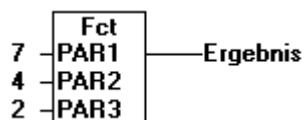
in IL:

```
LD 7
Fct 2,4
ST Result
```

in ST:

```
Result := Fct(7, 2, 4);
```

in FBD:



In SFC a function call can only take place within a step or a transition.



Note: If you define a function in your project with the name **CheckBounds**, you can use it to check for range overflows in your project! The name of the function is defined and may have only this identifier. An example of how this function is implemented is shown below:

```

CheckBounds (FUN-ST)
0001 FUNCTION CheckBounds : INT
0002 VAR_INPUT
0003   index,lower,upper:INT;
0004 END_VAR
0005
0001 IF index < lower THEN
0002   CheckBounds := lower;
0003 ELSIF index > upper THEN
0004   CheckBounds := upper;
0005 ELSE
0006   CheckBounds := index;
0007 END_IF

```

Image 2.2: Example of the Implementation of the Function CheckBounds

The following typical program for testing the CheckBounds function goes beyond the boundaries of a defined array. The CheckBounds functions makes sure that the value TRUE is not assigned to the position A[10], but rather to the upper area boundary A[7] which is still valid. Therefore, the CheckBounds function can be used to correct extensions beyond array boundaries.

```

PLC_PRG (PRG-ST)
0001 PROGRAM PLC_PRG
0002 VAR
0003   A:ARRAY[0..7] OF BOOL;
0004   B:INT:=10;
0004
0005 A[B]:=TRUE;
0006

```

Image 2.3: Test Program for the CheckBounds Function



Note: If you define functions in your project with the name **CheckDivByte**, **CheckDivWord**, **CheckDivDWord** and **CheckDivReal**, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0. The names of the functions are fixed, you must use exactly these terms. See the following example for the implementation of the function CheckDivReal:

```

CheckDivReal (FUN-ST)
0001 FUNCTION CheckDivReal : REAL
0002 VAR_INPUT
0003     divisor:REAL;
0004 END_VAR
0001 IF divisor = 0 THEN
0002     CheckDivReal:=1;
0003 ELSE
0004     CheckDivReal:=divisor;
0005 END_IF;

```

Image 2.4: Example for the implementation of the function CheckDivReal

Operator DIV uses the output of function CheckDivReal as Divisor. In a program like shown in the following example this avoids an division by 0, the divisor (d) is set from 0 to 1. So the result of the division is 799.

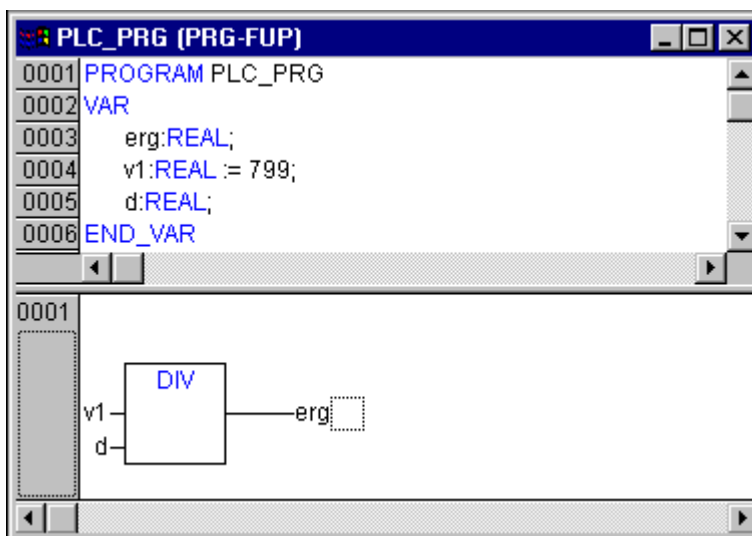


Image 2.5: Example program of the function CheckDivReal

Function Block

A function block is a POU which provides one or more values during the procedure. As opposed to a function, a function block provides no return value.

A function block declaration begins with the keyword `FUNCTION_BLOCK` and ends with `END_FUNCTION_BLOCK`.

Example in IL of a function block with two input variables and two output variables. One output is the product of the two inputs, the other a comparison for equality:

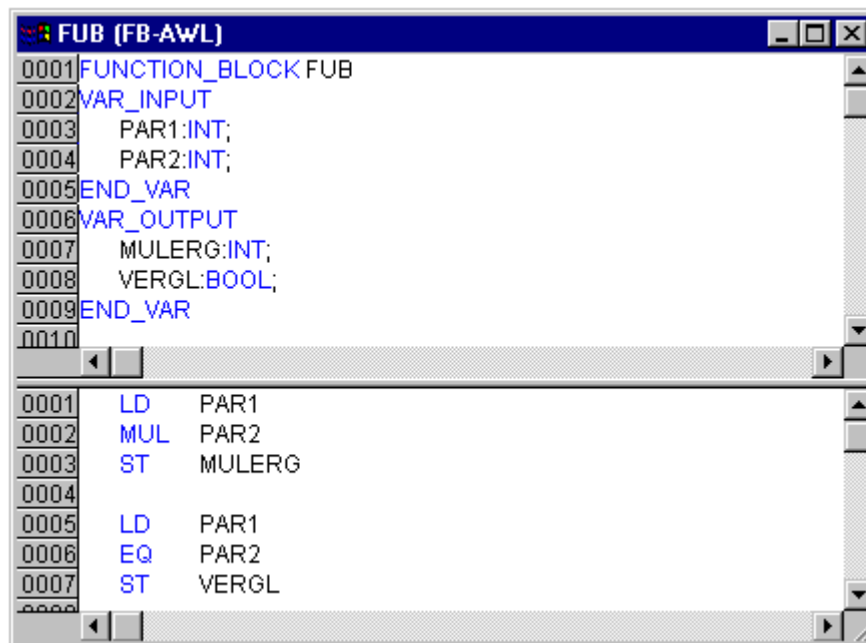


Image 2.6: Function Block

Function Block Instances

Reproductions or instances (copies) of a function block can be created.

Each instance possesses its own identifier (the Instance name), and a data structure which contains its inputs, outputs, and internal variables. Instances are declared locally or globally as variables, whereas the name of the function block is indicated as the type of an identifier.

Example of an instance with the name INSTANCE of the FUB function block:

```
INSTANCE: FUB;
```

Function blocks are always called through the instances described above.

Only the input and output parameters can be accessed from outside of an function block instance. This means the internal variables of the function blocks remain invisible to the user of the function block.

Example for accessing an input variable:

The function block FB has an input variable in1 of the type INT.

```
PROGRAM prog
VAR
  inst1:fb;
END_VAR
LD 17
ST inst1.in1
CAL inst1
END_PROGRAM
```

The declaration parts of function blocks and programs can contain instance declarations. Instance declarations are not permitted in functions.

Access to a function block instance is limited to the POU in which it was declared unless it was declared globally.

The instance name of a function block instance can be used as the input for a function or a function block.



Note: All values are retained after processing a function block until the next it is processed. Therefore, function block calls with the same arguments do not always return the same output values!

Calling a function block

You can use the variables of the function block by entering the instance name, a point, and then the variable name.

If you would like to set the input parameters when you open the function block, you can do this in the text languages IL and ST by assigning values to the parameters after the instance name of the function block in parentheses (this assignment takes place using ":= " just as with the initialization of variables at the declaration position).

Examples for calling function block FUB described above:

The multiplication result is saved in the variable ERG, and the result of the comparison is saved in QUAD. An instance of FUB with the name INSTANCE is declared.

In IL the function block is called as shown in the following image:

```
0001 PROGRAM AWLaufruf
0002 VAR
0003   QUAD:   BOOL;
0004   INSTANZ: FUB;
0005   ERG:    INT:=0;
0006 END_VAR
0007
0001 CAL INSTANZ(PAR1:=5,PAR2:=5)
0002
0003 LD INSTANZ.VERGL
0004 ST QUAD
0005
0006 LD INSTANZ.MULERG
0007 ST ERG
0008
```

Image 2.7: Function Block Call in IL

In the example below the call is shown in ST. The declaration part is the same as with IL:

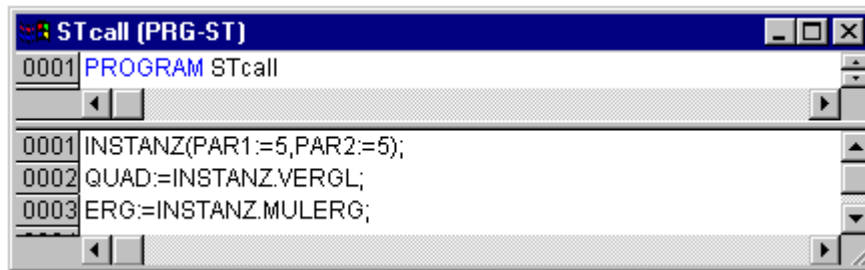


Image 2.8: Function Block Call in ST

In FBD the instance of a function block is called as shown in the following image (declaration part the same as with IL):

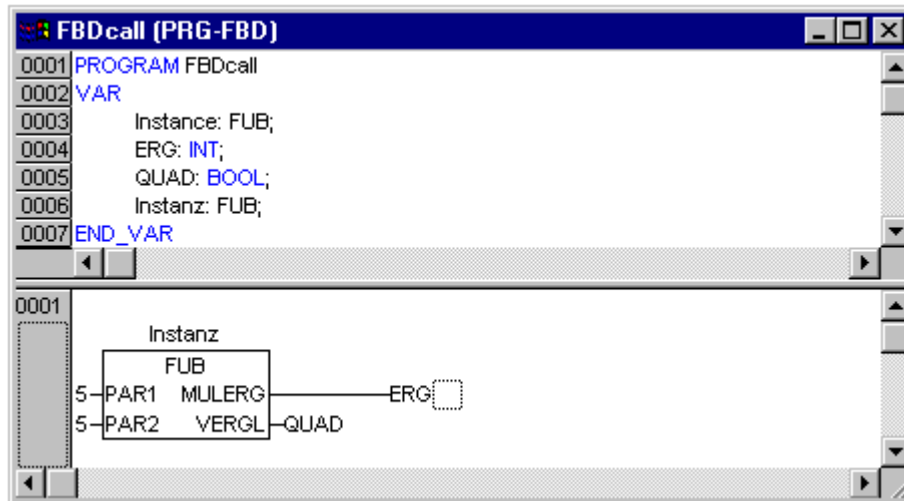


Image 2.9: Function Block Call in FBD

In SFC function block calls can only take place in steps.

Program

A program is a POU which returns several values during operation. Programs are recognized globally throughout the project. All values are retained from the last time the program was run until the next.

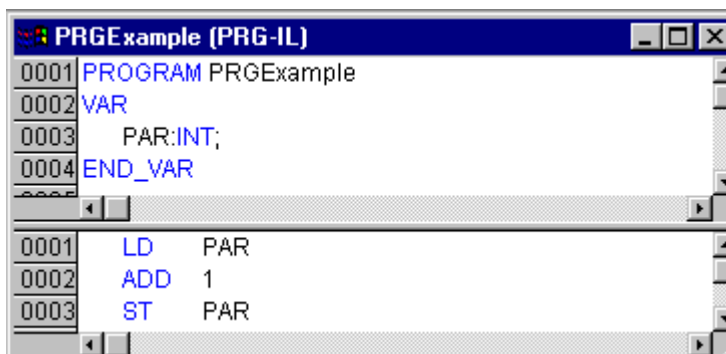


Image 2.10: Example of a program

Programs can be called. A program call in a function is not allowed. There are also no instances of programs.

If a POU calls a program, and if thereby values of the program are changed, then these changes are retained the next time the program is called, even if the program has been called from within another POU.

This is different from calling a function block. There only the values in the given instance of a function block are changed.

These changes therefore play a role only when the same instance is called.

A program declaration begins with the keyword PROGRAM and ends with END_PROGRAM.

Examples of calls of the program described above:

In IL:

```
CAL PRGExample
LD PRGexample.PAR
ST ERG
```

in ST:

```
PRGExample;
Erg := PRGexample.PAR;
```

In FBD:



Example for a possible call sequence for PLC_PRG:

```
LD 0
ST PRGexample.PAR (*Default setting for PAR is 0*)
CAL IL call (*ERG in IL call results in 1*)
CAL ST call (*ERG in ST call results in 2*)
CAL FBD call (*ERG in FBD call results in 3*)
```

If the variable PAR from the program PRGexample is initialized by a main program with 0, and then one after the other programs are called with above named program calls, then the ERG result in the programs will have the values 1, 2, and 3. If one exchanges the sequence of the calls, then the values of the given result parameters also change in a corresponding fashion.

PLC_PRG

The PLC_PRG is a special predefined POU. Each project must contain this special program. This POU is called exactly once per control cycle.

The first time the "**Project**" "**Object Add**" command is used after a new project has been created, the default entry in the POU dialog box will be a POU named PLC_PRG of the program type. You should not change this default setting!

If tasks have been defined, then the project may not contain any PLC_PRG, since in this case the procedure sequence depends upon the task assignment.



Attention: Do not delete or rename the POU PLC_PRG (assuming you are not using a Task Configuration, see chapter Task Configuration!) PLC_PRG is generally the main program in a single task program.

Action

Actions can be defined to function blocks and programmes. The action represents a further implementation which can be entirely created in another language as the "normal" implementation. Each action is given a name.

An action works with the data from the function block or programme which it belongs to. The action uses the same input/output variables and local variables as the "normal" implementation uses.

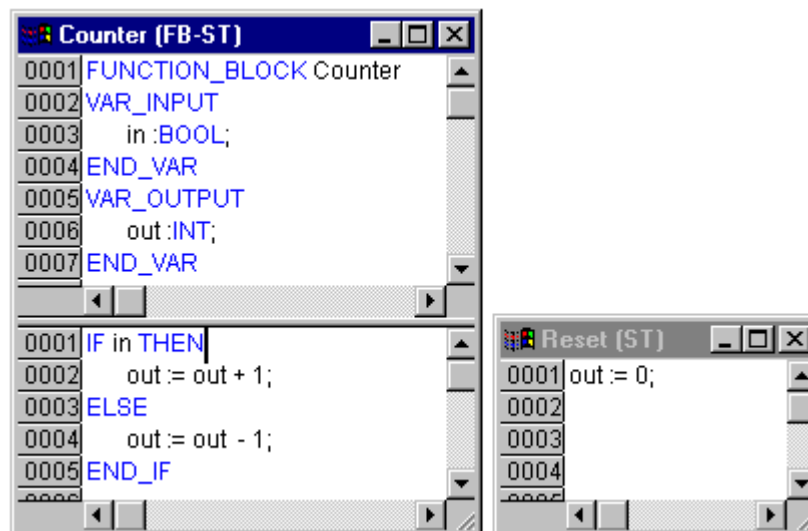


Image 2.11: Example for an action of a function block

In the example given, calling up the function block Counter increases or decreases the output variable "out", depending on the value of the input variable "in". Calling up the action Reset of the function block sets the output variable to zero. The same variable "out" is written in both cases.

An action is called up with <Program_name>.<Action_name> or <Instance_name>.<Action_name>. If it is required to call up the action within its own block, one just uses the name of the action in the text editors and in the graphic form the function block call up without instance information.

Examples for the calling-up of the above action:

Declaration for all Examples:

```
PROGRAM PLC_PRG
VAR
  Inst : Counter;
END_VAR
```

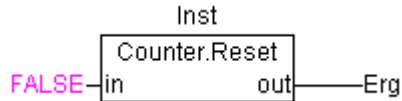
In AWL:

```
CAL    Inst.Reset(In := FALSE)
LD     Inst.out
ST     ERG
```

In ST:

```
Inst.Reset(In := FALSE);
Erg := Inst.out;
```

In FUP:



Note: Actions play an important role in blocks in sequential function charts, see the chapter 2.2.3 Sequential Function Chart - SFC.

The IEC standard does not recognise actions other than actions of the sequential function chart.

Resources

You need the resources for configuring and organizing your project and for tracing variable values:

- Global Variables which can be used throughout the project
- PLC Configuration for configuring your hardware
- Task Configuration for guiding your program through tasks
- Sampling Trace for graphic display of variable values
- Watch and Receipt Manager for displaying variable values and setting default variable values

See the chapter called "The Resources".

Libraries

You can include in your project a series of libraries whose POU's, data types, and global variables you can use just like user-defined variables. The library "standard.lib" is a standard part of the program and is always at your disposal.

See the chapter called "Library Manager".

Data types

Along with the standard data types the user can define his own data types. Structures, enumeration types and references can be created.

See "Standard" and "Defined data types" in the appendix.

907 AC 1131 provides visualizations so that you can display your project variables. You can plot geometric elements off-line with the help of the visualization. They can then change their form online, depending upon certain variable values.

See the chapter called "Visualization".

2.2 Languages

2.2.1 Instruction List (IL)

An instruction list (IL) consists of a series of instructions. Each instruction begins in a new line and contains an operator and, depending on the type of operation, one or more operands separated by commas.

In front of an instruction there can be an identification mark (label) followed by a colon (:).

A comment must be the last element in a line. Empty lines can be inserted between instructions.

Example:

```
LD    17
ST    lint          (* comment *)
GE    5
JMPC  next
LD    idword
EQ    istruct.sdword
STN   test
next:
```

Modifiers and operators in IL

In the IL language the following operators and modifiers can be used.

Modifiers:

- C with JMP, CAL, RET: The instruction is only then executed if the result of the preceding expression is TRUE.
- N with JMPC, CALC, RETC: The instruction is only then executed if the result of the preceding expression is FALSE.
- N otherwise: Negation of the operand (not of the accumulator)

Below you find a table of all operators in IL with their possible modifiers and the relevant meaning:

Operator	Modifiers	Meaning
LD	N	Make current result equal to the operand
ST	N	Save current result at the position of the operand
S		Then put the Boolean operand exactly at TRUE if the current result is TRUE

R		Then put the Boolean operand exactly at FALSE if the current result is TRUE
AND	N,(Bitwise AND
OR	N,(Bitwise OR
XOR	N,(Bitwise exclusive OR
ADD	(Addition
SUB	(Subtraction
MUL	(Multiplication
DIV	(Division
GT	(>
GE	(>=
EQ	(=
NE	(<>
LE	(<=
LT	(<
JMP	CN	Jump to the label
CAL	CN	Call programor function block or
RET	CN	Leave POU and return to caller.
)		Evaluate deferred operation

You find a list of all IEC operators in the appendix.

Example of an IL program while using some modifiers:

```

LD    TRUE    (*load TRUE in the accumulator*)
ANDN  BOOL1   (*execute AND with the negated value of the BOOL1
               variable*)
JMPC  mark    (*if the result was TRUE, then jump to the label
               "mark"*)

LDN   BOOL2   (*save the negated value of *)
ST    ERG     (*BOOL2 in ERG*)
label:
LD    BOOL2   (*save the value of *)
ST    ERG     (*BOOL2 in ERG*)

```

It is also possible in IL to put parentheses after an operation. The value of the parenthesis is then considered as an operand.

For example:

```

LD    2
MUL  2
ADD  3
ST    ERG

```

Here is the value of Erg 7. However, if one puts parentheses:

```

LD    2
MUL  2
(

```

```

ADD 3
)
ST   ERG

```

Here the resulting value for Erg is 10, the operation MUL is only then evaluated if you come to ")"; as operand for MUL 5 is then calculated.

2.2.2 Structured Text (ST)

The Structured Text consists of a series of instructions which, as determined in high level languages, ("IF..THEN..ELSE") or in loops (WHILE..DO) can be executed.

Example:

```

IF value < 7 THEN
  WHILE value < 8 DO
    value:=value+1;
  END_WHILE;
END_IF;

```

Expressions

An expression is a construction which returns a value after its evaluation.

Expressions are composed of operators and operands. An operand can be a constant, a variable, a function call, or another expression.

Valuation of expressions

The evaluation of expression takes place by means of processing the operators according to certain binding rules. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed.

Operators with equal binding strength are processed from left to right.

Below you find a table of the ST operators in the order of their binding strength:

Operation	Symbol	Binding strength
Put in parentheses	(expression)	Strongest binding
Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negate	-	
Building of complements	NOT	
Multiply	*	
Divide	/	
Modulo	MOD	
Add	+	
Subtract	-	
Compare	<,>,<=,>=	

Equal to	=	
Not equal to	<>	
Boolean AND	AND	
Boolean XOR	XOR	
Boolean OR	OR	Weakest binding

There are the following instructions in ST, arranged in a table together with example:

Instruction type	Example
Assignment	A:=B; CV := CV + 1; C:=SIN(X);
Calling a function block and use of the FB output	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
RETURN	RETURN;
IF	D:=B*B; IF D<0.0 THEN C:=A; ELSIF D=0.0 THEN C:=B; ELSE C:=D; END_IF;
CASE	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;
FOR	J:=101; FOR I:=1 TO 100 BY 2 DO IF ARR[I] = 70 THEN J:=I; EXIT; END_IF; END_FOR;
WHILE	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;

REPEAT	J:=-1;
	REPEAT
	J:=J+2;
	UNTIL J= 101 OR ARR[J] = 70
	END_REPEAT;
EXIT	EXIT;
Empty instruction	;

The name already indicates, the Structured Text is designed for structure programming, i.e. ST offers predetermined structures for certain often used constructs such as loops for programming.

This offers the advantages of low error probability and increased clarity of the program.

For example, let us compare two equally significant program sequences in IL and ST:

A loop for calculating powers of two in **IL**:

```

Loop:
LD      Counter

JMPC   end
LD      Var1
MUL    2
ST      Var1

LD      Counter
SUB    1
ST      Counter
JMP    Loop

End:
LD      Var1
ST      ERG

```

The same loop programmed in ST would produce:

```

WHILE counter<>0 DO
  Var1:=Var1*2;
  Counter:=counter-1;
END_WHILE

```

```
Erg:=Var1;
```

You can see, the loop in ST is not only faster to program, but is also significantly easier to read, especially in view of the convoluted loops in larger constructs.

The different structures in ST have the following significance:

Assignment operator

On the left side of an assignment there is an operand (variable, address) to which is assigned the value of the expression on the right side with the assignment operator :=

Example:

```
Var1 := Var2 * 10;
```

After completion of this line Var1 has the tenfold value of Var2.

Calling function blocks in ST

A function block is called in ST by writing the name of the instance of the function block and then assigning the values of the parameters in parentheses. In the following example a timer is called with assignments for the parameters IN and PT. Then the result variable Q is assigned to the variable A.

The result variable, as in IL, is addressed with the name of the function block, a following point, and the name of the variable:

```
CMD_TMR(IN := %IX5, PT := 300);  
A:=CMD_TMR.Q
```

RETURN instruction

The RETURN instruction can be used to leave a POU, for example depending on a condition

IF instruction

With the IF instruction you can check a condition and, depending upon this condition, execute instructions.

Syntax:

```
IF <Boolean_expression1> THEN  
  <IF_instructions>  
{ELSIF <Boolean_expression2> THEN  
  <ELSIF_instructions1>  
  .  
  .  
ELSIF <Boolean_expression n> THEN  
  <ELSIF_instructions n-1>  
ELSE  
  <ELSE_instructions>}  
END_IF;
```

The part in braces {} is optional.

If the <Boolean_expression1> returns TRUE, then only the <IF_Instructions> are executed and none of the other instructions.

Otherwise the Boolean expressions, beginning with <Boolean_expression2>, are evaluated one after the other until one of the expressions returns TRUE. Then only those instructions after this Boolean expression and before the next ELSE or ELSIF are evaluated.

If none of the Boolean expressions produce TRUE, then only the <ELSE_instructions> are evaluated.

Example:

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

Here the heating is turned on when the temperature sinks below 17 degrees. Otherwise it remains off.

CASE instruction

With the CASE instructions one can combine several conditioned instructions with the same condition variable in one construct.

Syntax:

```
CASE <Var1> OF
<Value1>:    <Instruction 1>
<Value2>:    <Instruction 2>
<Value3, Value4, Value5>: <Instruction 3>
<Value6 .. Value10>: <Instruction 4>
...
<Value n>:    <Instruction n>
ELSE <ELSE instruction>
END_CASE;
```

A CASE instruction is processed according to the following model:

- If the variable in <Var1> has the value <Value i>, then the instruction <Instruction i> is executed.
- If <Var 1> has none of the indicated values, then the <ELSE Instruction> is executed.
- If the same instruction is to be executed for several values of the variables, then one can write these values one after the other separated by commas, and thus condition the common execution.
- If the same instruction is to be executed for a value range of a variable, one can write the initial value and the end value separated by two dots one after the other. So you can condition the common condition.

Example:

```
CASE INT1 OF
1, 5:    BOOL1 := TRUE;
        BOOL3 := FALSE;
2: BOOL2 := FALSE;
        BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
        BOOL3:= TRUE;
ELSE
        BOOL1 := NOT BOOL1;
        BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR loop

With the FOR loop one can program repeated processes.

Syntax:

```
INT_Var :INT;
```

```
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>}  
DO  
  <Instructions>  
END_FOR;
```

The part in braces {} is optional.

The <Instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before executing the <Instructions> so that the <instructions> are never executed if <INIT_VALUE> is greater than <END_VALUE>.

When <Instructions> are executed, <INT_Var> is always increased by <Step size>. The step size can have any integer value. If it is missing, then it is set to 1. The loop must also end since <INT_Var> only becomes greater.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO  
  Var1:=Var1*2;  
END_FOR;  
Erg:=Var1;
```

Let us assume that the default setting for Var1 is the value 1. Then it will have the value 32 after the FOR loop.



Note: <END_VALUE> must not be equal to the limit value of the counter <INT_VAR>. For example: If the variable Counter is of type SINT and if <END_VALUE> is 127, you will get an endless loop.

WHILE loop

The WHILE loop can be used like the FOR loop with the difference that the break-off condition can be any Boolean expression. This means you indicate a condition which, when it is fulfilled, the loop will be executed.

Syntax:

```
WHILE <Boolean expression>  
  <Instructions>  
END_WHILE;
```

The <Instructions> are repeatedly executed as long as the <Boolean_expression> returns TRUE. If the <Boolean_expression> is already FALSE at the first evaluation, then the <Instructions> are never executed. If

<Boolean_expression> never assumes the value FALSE, then the <Instructions> are repeated endlessly which causes a relative time delay.



Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example, by counting up or down one counter.

Example:

```
WHILE counter<>0 DO
  Var1 := Var1*2;
  Counter := Counter-1;
END_WHILE
```

The WHILE and REPEAT loops are, in a certain sense, more powerful than the FOR loop since one doesn't need to know the number of cycles before executing the loop. In some cases one will, therefore, only be able to work with these two loop types. If, however, the number of the loop cycles is clear, then a FOR loop is preferable since it allows no endless loops.

REPEAT loop

The REPEAT loop is different from the WHILE loop because the break-off condition is checked only after the loop has been executed. This means that the loop will run through at least once, regardless of the wording of the break-off condition.

Syntax:

```
REPEAT
  <Instructions>
UNTIL <Boolean expression>
END_REPEAT;
```

The <Instructions> are carried out until the <Boolean expression> returns TRUE.

If <Boolean expression> is produced already at the first TRUE evaluation, then <Instructions> are executed only once. If <Boolean_expression> never assumes the value TRUE, then the <Instructions> are repeated endlessly which causes a relative time delay.



Note: The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example by counting up or down one counter.

Example:

```
REPEAT
  Var1 := Var1*2;
  Counter := Counter-1;
UNTIL
  Counter=0
```

END_REPEAT;

EXIT instruction

If the EXIT instruction appears in a FOR, WHILE, or REPEAT loop, then the innermost loop is ended, regardless of the break-off condition.

2.2.3 Sequential Function Chart (SFC)

The Sequential Function Chart is a graphically oriented language which makes it possible to describe the chronological order of different actions within a program.

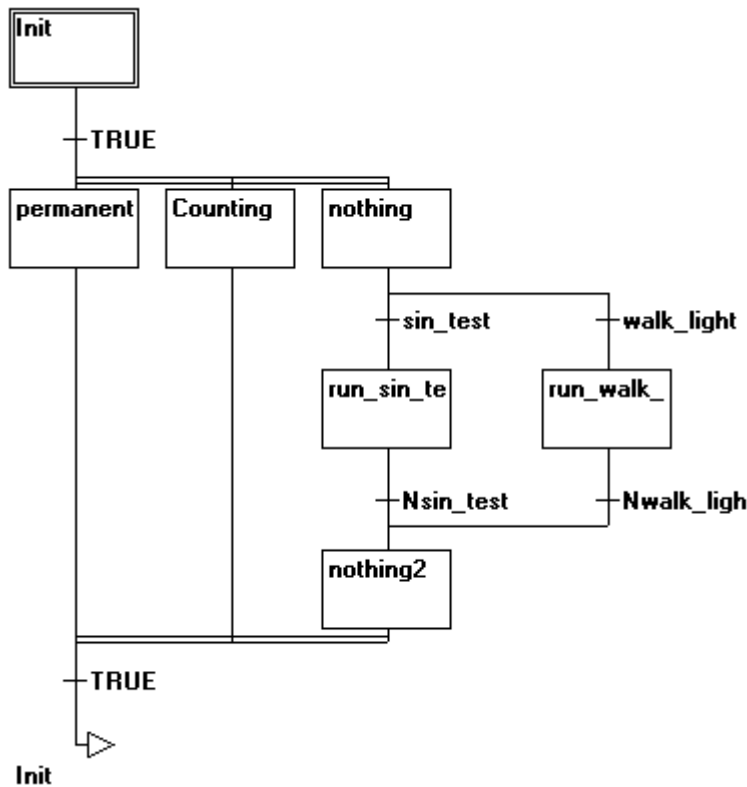


Image 2.12: Network in SFC

Step

A POU written in a Sequential Function Chart consists of a series of steps which are connected with each other through directed connections (transitions).

There are two types of steps.

- The simplified type consists of an action and a flag which shows if the step is active. If the action of a step is implemented, then a small triangle appears in upper right corner of the step.
- An IEC step consists of a flag and one or more assigned actions or boolean variables. The associated actions appear to the right of the step. For detailed information see Chapter 'The Sequential Function Chart Editor'.

Action

An action can contain a series of instructions in IL or in ST, a lot of networks in FBD or in LD, or again in Sequential Function Chart (SFC).

With the simplified steps an action is always connected to a step. In order to edit an action, click twice with the mouse on the step to which the action belongs. Or select the step and select the menu command **"Extras" "Zoom Action/Transition"**.

Actions of IEC steps hang in the Object Organizer directly under their SFC-POU and are loaded with a doubleclick or by pressing <Enter> in their editor. New actions can be created with **"Project" "Add Action"**.

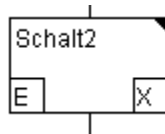
Entry or exit actionf

You can add an entry action and an exit action to a step. An entry action is executed only once, right after the step has become active. An exit action is executed only once before the step is deactivated.

A step with entry action is indicated by an "E" in the lower left corner, the exit action by an "X" in the lower right corner.

The entry and exit action can be implemented in any language. In order to edit an entry or exit action, doubleclick in the corresponding corner in the step with the mouse.

Example of a step with entry and exit action:



Transition / Transition condition

Between the steps there are so-called transitions.

A transition condition can be a Boolean variable, a boolean address, a boolean constant, or a series of instructions with a Boolean result in ST syntax (i.e. (i <= 100) AND b) or fully programmed in any language.

Active step

After calling the SFC POU, the action (surrounded by a double border) belonging to the initial step is executed first. A step, whose action is being executed, is called active. If the step is active, then the appropriate action is executed once per cycle. In Online mode active steps are shown in blue.

In a control cycle all actions are executed which belong to active steps. Thereafter the respective following steps of the active steps become active if the transition conditions of the following steps are TRUE. The currently active steps will be executed in the next cycle.

Along with the simplified steps the standard IEC steps in SFC are available.

Any number of actions can be assigned to an IEC step. The actions of IEC steps lie separated from the steps and can be used repeatedly within their POU. For this they must be associated to the single steps with the command "**Extras**" "**Associate action**".

Along with actions, Boolean variables can be assigned to steps. Through the so-called qualifiers the actions and Boolean variables are activated and deactivated, partially with time delays. Since an action can still be active, even if the next step has been processed, for example through the qualifier S (Set), one can achieve concurrent processes.

An associated boolean variable is set or reset with each call of the AS. That means, that with each call it gets the value TRUE or FALSE.

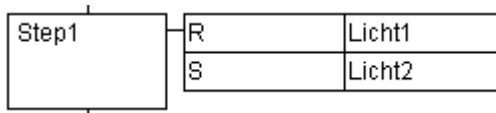
The actions associated with an IEC step are shown at the right of the step in a two-part box. The left field contains the qualifier, possibly with time constant, and the right field contains the action name respectively boolean variable name.



Note: If an action has been inactivated, it will be executed **once more**. That means, that each action is executed at least twice (also an action with qualifier P).

In case of a call first the deactivated actions, then the active actions are executed, in alphabetical order each time.

An example for an IEC step with two actions:



In order to make it easier to follow the processes, all active actions in online mode are shown in blue like the active steps. After each cycle a check is made to see which actions are active.

Whether a newly inserted step is an IEC step depends upon whether the menu command "**Extras**" "**Use IEC-Steps**" has been chosen.

In the Object Organizer the actions hang directly underneath their respective SFC POUs. New actions can be created with "**Project**" "**Add Action**".

In order to use IEC steps you must include in your project the special SFC library lecsfc.lib .

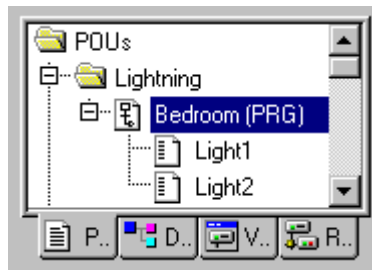


Image 2.13: SFC POU with actions in the Object Organizer

Qualifier

In order to associate the actions with IEC steps the following qualifiers are available:

N	Non-stored	The action is active as long as the step
R	overriding	The action is deactivated
	Reset	
S	Set (Stored)	The action is activated and remains active until a Reset
L	time Limited	The action is activated for a certain time, maximum as long as the step is active
D	time Delayed	The action becomes active after a certain time if the step is still active and then it remains active as long as the step is active.
P	Pulse	The action is executed just one time if the step is active
SD	Stored and time Delayed	The action is activated after a certain time and remains active until a Reset
DS	Delayed and Stored	The action is activated after a certain time as long as the step is still active and remains active up to a Reset
SL	Stored and time limited	The action is activated for a certain time

The qualifiers L, D, SD, DS and SL need a time value in the TIME constant format.

Implicit variables in SFC

There are implicitly declared variables in the SFC which can be used.

A flag belongs to each step which stores the state of the step. The step flag (active or inactive state of the step) is called **<StepName>.x** for IEC steps or just **<StepName>** for the simplified steps. This Boolean variable has the value TRUE when the associated step is active and FALSE when it is inactive. It can be used in every action and transition of the SFC block.

One can make an enquiry with the variable **<ActionName>.x**. as to whether an IEC action is active or not.



Note: During the deactivation of the IEC action the variable already has the value FALSE.

For IEC steps the implicit variables **<StepName>.t** can be used to enquire about the active time of the steps.

SFC Flags

If a step is active in SFC for longer than its attributes state, some special flags are set. There are also variables which can be set in order to control the program flow in the sequential function chart. To use the flags it is necessary, somewhere, globally or locally, to declare them as output or input variables.

SFCEnableLimit: This variable is of the type BOOL. When it has the value TRUE, the timeouts of the steps will be registered in SFCError. Other timeouts will be ignored.

SFCInit: This variable is also of the type BOOL. When the variable has the value TRUE the sequential function chart is set back to the Init step and the other SFC flags are reset. The Init step remains active, but is not executed, for as long as the variable has the value TRUE. It is only when SFCInit is again set to FALSE that the block can be processed normally again.

SFCQuitError: A variable of the type BOOL. Execution of the SFC diagram is stopped for as long as the variable has the value TRUE whereby a possible timeout in the variable SFCError is reset. All previous times in the active steps are reset when the variable again assumes the value FALSE.

SFCPause: A variable of the type BOOL. Execution of the SFC diagram is stopped for as long as the variable has the value TRUE.

SFCError: This Boolean variable is set when a timeout has occurred in a SFC diagram.

SFCTrans: This variable is of the type BOOL. The variable takes on the value TRUE when a transition is actuated.

SFCErrorStep: This variable is of the type STRING. In this variable the name of the step is stored which has caused a timeout to occur.

SFCErrorPOU: This variable of the type STRING contains the name of the block in which a timeout has occurred.

SFCCurrentStep: This variable is of the type STRING. The name of the step is stored in this variable which is active, independently of the time monitoring. In the case of simultaneous sequences the step is stored in the branch on the outer right.

No further timeout will be registered if a timeout occurs and the variable SFCError is not reset again.

Alternative branch

Two or more branches in SFC can be defined as alternative branches. Each alternative branch must begin and end with a transition. Alternative branches can contain parallel branches and other alternative branches. An alternative

branch begins at a horizontal line (alternative beginning) and ends at a horizontal line (alternative end) or with a jump.

If the step which precedes the alternative beginning line is active, then the first transition of each alternative branch is evaluated from left to right. The first transition from the left whose transition condition has the value TRUE is opened and the following steps are activated (see active step).

Parallel branch

Two or more branches in SFC can be defined as parallel branches. Each parallel branch must begin and end with a step. Parallel branches can contain alternative branches or other parallel branches. A parallel branch begins with a double line (parallel beginning) and ends with a double line (parallel end) or with a jump.

If the parallel beginning line of the previous step is active and the transition condition after this step has the value TRUE, then the first steps of all parallel branches become active (see active step). These branches are now processed parallel to one another. The step after the parallel end line becomes active when all previous steps are active and the transition condition before this step produces the value TRUE.

Jump

A jump is a connection to the step whose name is indicated under the jump symbol. Jumps are required because it is not allowed to create connections which lead upward or cross each other.

2.2.4 Function Block Diagram (FBD)

The Function Block Diagram is a graphically oriented programming language. It works with a list of networks whereby each network contains a structure which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction.

An example of a typical network in the Function Block Diagram as it could appear in 907 AC 1131 :

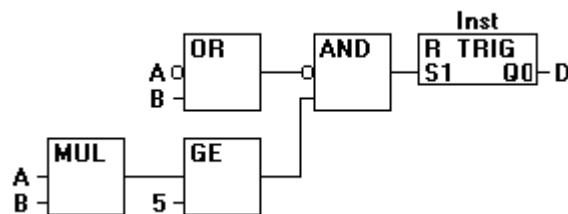


Image 2.14: Network in Function Block Diagram

See also chapter 'The Continuous Function Chart Editor' at 'The Editors in 907 AC 1131 '.

2.2.5 Ladder Diagram (LD)

The Ladder Diagram is also a graphics oriented programming language which approaches the structure of an electric circuit.

On the one hand, the Ladder Diagram is suitable for constructing logical switches, on the other hand one can also create networks as in FBD. Therefore the LD is very useful for controlling the call of other POU's. For more information see Chapter 'The Editors in 907 AC 1131, The Ladder Editor'.

The Ladder Diagram consists of a series of networks. A network is limited on the left and right sides by a left and right vertical current line. In the middle is a circuit diagram made up of contacts, coils, and connecting lines.

Each network consists on the left side of a series of contacts which pass on from left to right the condition "ON" or "OFF" which correspond to the Boolean values TRUE and FALSE. To each contact belongs a Boolean variable. If this variable is TRUE, then the condition is passed from left to right along the connecting line. Otherwise the right connection receives the value OFF.

Example of a typical network in the Ladder Diagram as it could appear in 907 AC 1131 :

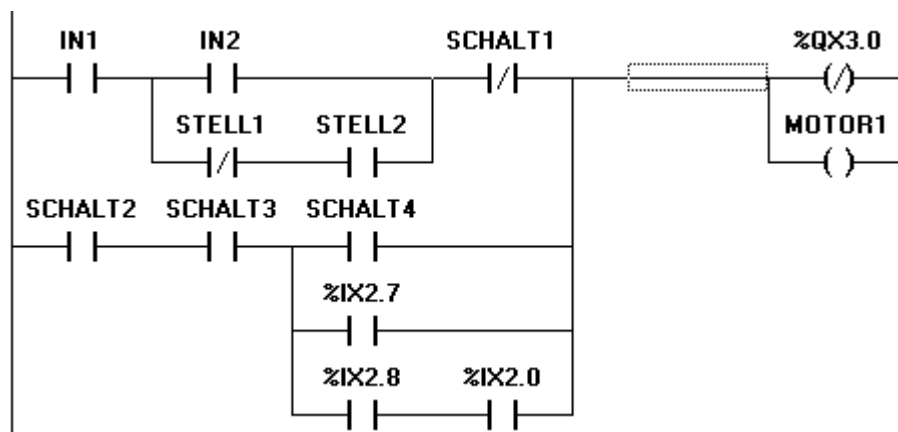


Image 2.15: Network in a Ladder Diagram made up of Contacts and Coils

Contact

Each network in LD consists on the left side of a network of contacts (contacts are represented by two parallel lines: | |) which from left to right show the condition "On" or "Off".

These conditions correspond to the Boolean values TRUE and FALSE. A Boolean variable belongs to each contact. If this variable is TRUE, then the condition is passed on by the connecting line from left to right, otherwise the right connection receives the value "Out".

Contacts can be connected in parallel, then one of the parallel branches must transmit the value "On" so that the parallel branch transmits the value "On"; or the contacts are connected in series, then contacts must transmit the condition "On" so that the last contact transmits the "On" condition. This therefore corresponds to an electric parallel or series circuit.

A contact can also be negated, recognizable by the slash in the contact symbol: $/$. Then the value of the line is transmitted if the variable is FALSE.

Coil

On the right side of a network in LD there can be any number of so-called coils which are represented by parentheses: (). They can only be in parallel. A coil transmits the value of the connections from left to right and copies it in an appropriate Boolean variable. At the entry line the value ON (corresponds to the Boolean variable TRUE) or the value OFF (corresponding to FALSE) can be present.

Contacts and coils can also be negated (in the example the contact SWITCH1 and the coil %QX3.0 is negated). If a coil is negated (recognizable by the slash in the coil symbol: $/$), then it copies the negated value in the appropriate Boolean variable. If a contact is negated, then it connects through only if the appropriate Boolean value is FALSE.

Function blocks in the Ladder Diagram

Along with contacts and coils you can also enter function blocks and programs. In the network they must have an input and an output with Boolean values and can be used at the same places as contacts, that is on the left side of the LD network

Set/Reset coils

Coils can also be defined as set or reset coils. One can recognize a set coil by the "S" in the coil symbol: **(S)** It never writes over the value TRUE in the appropriate Boolean variable. That is, if the variable was once set at TRUE, then it remains so.

One can recognize a reset coil by the "R" in the coil symbol: **(R)** It never writes over the value FALSE in the appropriate Boolean variable: If the variable has been once set on FALSE, then it remains so.

LD as FBD

When working with LD it is very possible that you will want to use the result of the contact switch for controlling other POU's. On the one hand you can use the coils to put the result in a global variable which can then be used in another place. You can, however, also insert the possible call directly into your LD network. For this you introduce a POU with EN input.

Such POU's are completely normal operands, functions, programs, or function blocks which have an additional input which is labeled with EN. The EN input is always of the BOOL type and its meaning is: The POU with EN input is evaluated when EN has the value TRUE.

An EN POU is wired parallel to the coils, whereby the EN input is connected to the connecting line between the contacts and the coils. If the ON information is transmitted through this line, this POU will be evaluated completely normally.

Starting from such an EN POU, you can create networks similar to FBD.

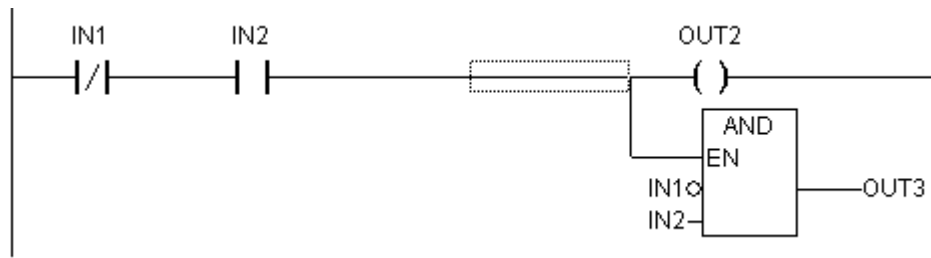


Image 2.16: Part of a LD Network with an EN POU

2.3 Debugging, Online Functions

Sampling Trace

The Sampling Trace allows you to trace the value sequence of variables, depending upon the so-called trigger event. This is the rising edge or falling edge of a previously defined Boolean variable (trigger variable). **907 AC 1131** permits the tracing of up to 20 variables. 500 values can be traced for each variable. The trace buffer has a size of 4990 Bytes.

Debugging

The debugging functions of **907 AC 1131** make it easier for you to find errors.

In order to debug, run the command "**Project**" "**Options**" and in the dialog box that pops up under Build options select the item **Debugging**.

Breakpoint

A breakpoint is a place in the program at which the processing is stopped. Thus it is possible to look at the values of variables at specific places within the program.

Breakpoints can be set in all editors. In the text editors breakpoints are set at line numbers, in FBD and LD at network numbers, and in SFC at steps.

Single step

Single step means:

- In IL: Execute the program until the next CAL, LD or JMP command.
- In ST: Execute the next instruction.
- In FBD, LD: Execute the next network.
- In SFC: Continue the action until the next step.

By proceeding step by step you can check the logical correctness of your program.

Single Cycle

If Single cycle has been chosen, then the execution is stopped after each cycle.

Change values online

During operations variables can be set once at a certain value (write value) or also described again with a certain value after each cycle (forcing). In online mode one also can change the variable value by double click on the value. By that boolean variables change from TRUE to FALSE or the other way round, for each other types of variables one gets the dialog **Write Variable xy**, where the actual value of the variable can be edited.

Monitoring

In addition to the variable declarations visible on the screen, the current values from the PLC are continuously read and displayed.

In Online mode **907 AC 1131** monitors all variables visible on the screen. At the same time variables can be put together in the Watch and Receipt Manager whose data you would like to see all in one place.

Simulation

During the simulation the created PLC program is not processed in the PLC, but rather in the calculator on which **907 AC 1131** is running. All online functions are available. That allows you to test the logical correctness of your program without PLC hardware.



Note: POU's of external libraries are not running in simulation mode !

2.4 The Standard

The standard IEC 1131-3 is an international standard for programming languages of Programmable Logic Controllers.

The programming languages offered in **907 AC 1131** conform to the requirements of the standard.

According to this standard, a program consists of the following elements:

- Structures
- POU's
- Global Variables

The processing of a 907 AC 1131 program begins with the special POU PLC_PRG. The POU PLC_PRG can call other POU's.

3.1 Controlling a Traffic Signal Unit

Let us now start to write a small example program. It is for a simple traffic signal unit which is supposed to control two traffic signals at an intersection. The red/green phases of both traffic signals alternate and, in order to avoid accidents, we will insert yellow or yellow/red transitional phases. The latter will be longer than the former. We further imagine the use of a Profibus system and will do the corresponding configuration.

In this example you will see how time dependent programs can be shown with the language resources of the IEC1131-3 standard, how one can edit the different languages of the standard with the help of **907 AC 1131** , and how one can easily connect them while becoming familiar with the simulation of **907 AC 1131** .

Create POU

Starting always is easy: Start **907 AC 1131** and choose **"File" "New"**.

In the dialog box which appears, the first POU has already been given the default name PLC_PRG. Keep this name, and the type of POU should definitely be a program. Each project needs a program with this name. In this case we choose as the language of this POU the Structured Text (ST).

Now create three more objects with the command **"Project" "Object Add"** with the menu bar or with the context menu (press right mouse button in the Object Organizer). A program in the language Sequential Function Chart (SFC) named SEQUENCE, a function block in the language Function Block Diagram (FBD) named TRAFFICSIGNAL, along with a POU WAIT, also of the type function block, which we want to program as an Instruction List (IL).

What does TRAFFICSIGNAL do?

In the POU TRAFFICSIGNAL we will assign the individual trafficsignal phases to the lights, i.e. we will make sure that the red light is lit red in the red phase and in the yellow/red phase, the yellow light in the yellow and yellow/red phases, etc.

What does WAIT do?

In WAIT we will program a simple timer which as input will receive the length of the phase in milliseconds, and as output will produce TRUE as soon as the time period is finished.

What does SEQUENCE do?

In SEQUENCE all is combined so that the right light lights up at the right time for the desired time period.

What does PLC_PRG do?

In PLC_PRG the input start signal is connected to the traffic lights' sequence and the "color instructions" for each lamp are provided as outputs.

"TRAFFICSIGNAL" declaration

Let us now turn to the POU TRAFFICSIGNAL. In the declaration editor you declare as input variable (between the keywords VAR_INPUT and END_VAR) a variable named STATUS of the type INT. STATUS will have four possible conditions, that is one for the TRAFFICSIGNAL phases green, yellow, yellow/red and red.

Correspondingly our TRAFFICSIGNAL has three outputs, that is RED, YELLOW and GREEN. You should declare these three variables. Then the declaration part of our function block TRAFFICSIGNAL will look like this:

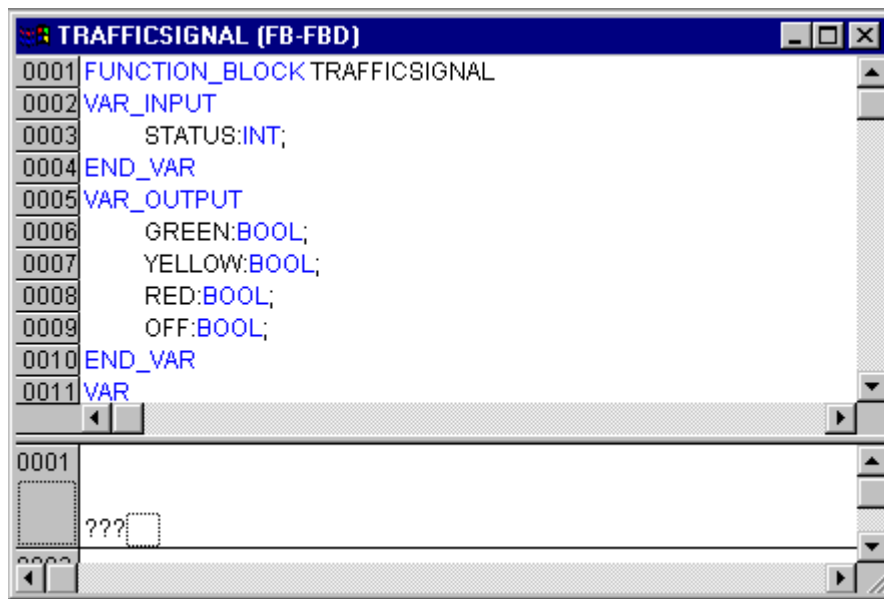


Image 3.1: Function block TRAFFICSIGNAL, declaration part

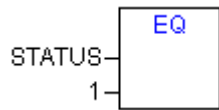
"TRAFFICSIGNAL" body

Now we determine the values of the output variables depending on the input STATUS of the POU. To do this go into the body of the POU. Click on the field to the left beside the first network (the gray field with the number 1). You have now selected the first network. Choose the menu item "**Insert**" "**Operator**".

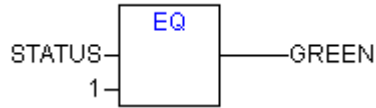
In the first network a box is inserted with the operator AND and two inputs:



Click on the text AND with the mouse pointer and change the text into EQ. Select the three question marks from the upper of the two inputs and enter the variable STATUS. Then select the lower of the three question marks and put a 1 underneath it. You get the following network:



Click now on a place behind the EQ Box. Now the output of the EQ operation is selected. Choose **"Insert" "Assignment"**. Change the three question marks ??? to GREEN. You now have created a network with the following structure:



STATUS is compared with 1, the result is assigned to GREEN. This network thus switches to GREEN if the preset state value is 1.

For the other TRAFFICSIGNAL colors we need two more networks. You create them with the command **"Insert" "Network (after)"**. These networks should be set up as in the example. The finished POU now looks like follows:

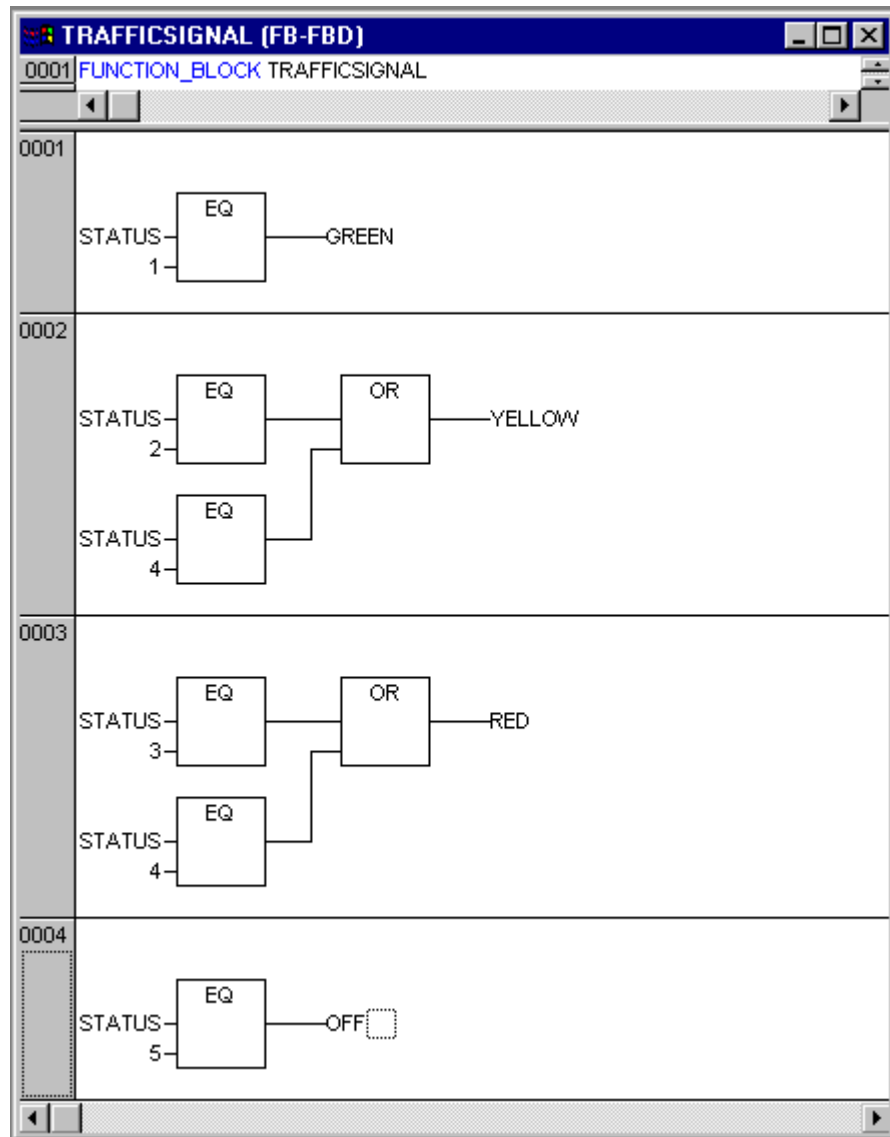


Image 3.2: Function block TRAFFICSIGNAL, instruction part

In order to insert an operator in front of another operator, you must select the place where the input to which you want to attach the operator feeds into the box.

Then use the command "**Insert**" "**Operator**". Otherwise you can set up these networks in the same way as the first network.

Now our first POU has been finished. TRAFFICSIGNAL, according to the input of the value STATUS, controls whichever light color we wish.

Connecting the standard.lib

For the timer in the POU WAIT we need a POU from the standard library. Therefore, open the library manager with "**Window**" "**Library Manager**". Choose "**Insert**" "**Additional library**". The dialog box appears for opening files. From the list of the libraries choose standard.lib.

"WAIT" declaration

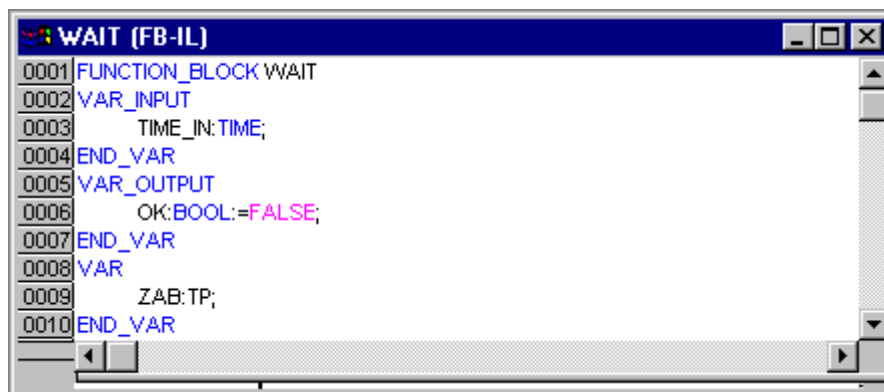
Now let us turn to the POU WAIT. This POU is supposed to become a timer with which we can determine the length of the time period of each TRAFFICSIGNAL phase. Our POU receives as input variable a variable TIME of the type TIME, and as output it produces a Boolean value which we want to call OK and which should be TRUE when the desired time period is finished. We set this value with FALSE by inserting at the end of the declaration (before the semicolon, however) " := FALSE ".

For our purposes we need the POU TP, a clock generator. This has two inputs (IN, PT) and two outputs (Q, ET). TP does the following:

As long as IN is FALSE, ET is 0 and Q is FALSE. As soon as IN provides the value TRUE, the time is calculated at the output ET in milliseconds. When ET reaches the value PT, then ET is no longer counted. Meanwhile Q produces TRUE as long as ET is smaller than PT. As soon as the value PT has been reached, then Q produces FALSE again. In addition you will find a short description of all POUs from the standard library in the appendix.

In order to use the POU TP in the POU WAIT we must create a local instance from TP. For this we declare a local variable ZAB (for elapsed time) of the type TP (between the keywords VAR, END_VAR).

The declaration part of WAIT thus looks like this:



```
0001 FUNCTION_BLOCK WAIT
0002 VAR_INPUT
0003     TIME_IN: TIME;
0004 END_VAR
0005 VAR_OUTPUT
0006     OK: BOOL := FALSE;
0007 END_VAR
0008 VAR
0009     ZAB: TP;
0010 END_VAR
```

Image 3.3: Function Block WAIT, Declaration Part

"WAIT" body

In order to create the desired timer, the body of the POU must be programmed as follows:

```
0001 FUNCTION_BLOCK WAIT
0002 LD ZAB.Q
0003 JMPC mark
0004
0005 CAL ZAB(IN:=FALSE)
0006 LD TIME_IN
0007 ST ZAB.PT
0008 CAL ZAB(IN:=TRUE)
0009 JMP end
0010
0011 mark:
0012 CAL ZAB
0013 end:
0014 LDN ZAB.Q
0015 ST OK
0016 RET
```

Image 3.4: Function Block WAIT, Instruction Part

At first it is checked whether Q has already been set at TRUE (as though the counting had already been executed), in this case we change nothing with the occupation of ZAB, but we call the function block ZAB without input (in order to check whether the time period is already over).

Otherwise we set the variable IN in ZAB at FALSE, and therefore at the same time ET at 0 and Q at FALSE. In this way all variables are set at the desired initial condition. Now we assign the necessary time from the variable TIME into the variable PT, and call ZAB with IN:=TRUE. In the function block ZAB the variable ET is now calculated until it reaches the value TIME, then Q is set at FALSE.

The negated value of Q is saved in OK after each execution of WAIT. As soon as Q is FALSE, then OK produces TRUE.

The timer is finished at this point. Now it is time to combine our two function blocks WAIT and TRAFFIC SIGNAL in the main program PLC_PRG.

"SEQUENCE" first expansion level

First we declare the variables we need. They are: an input variable START of the type BOOL, two output variables TRAFFIC SIGNAL1 and TRAFFIC SIGNAL2 of the type INT and one of the type WAIT (DELAY as delay). The program SEQUENCE now looks like shown here:

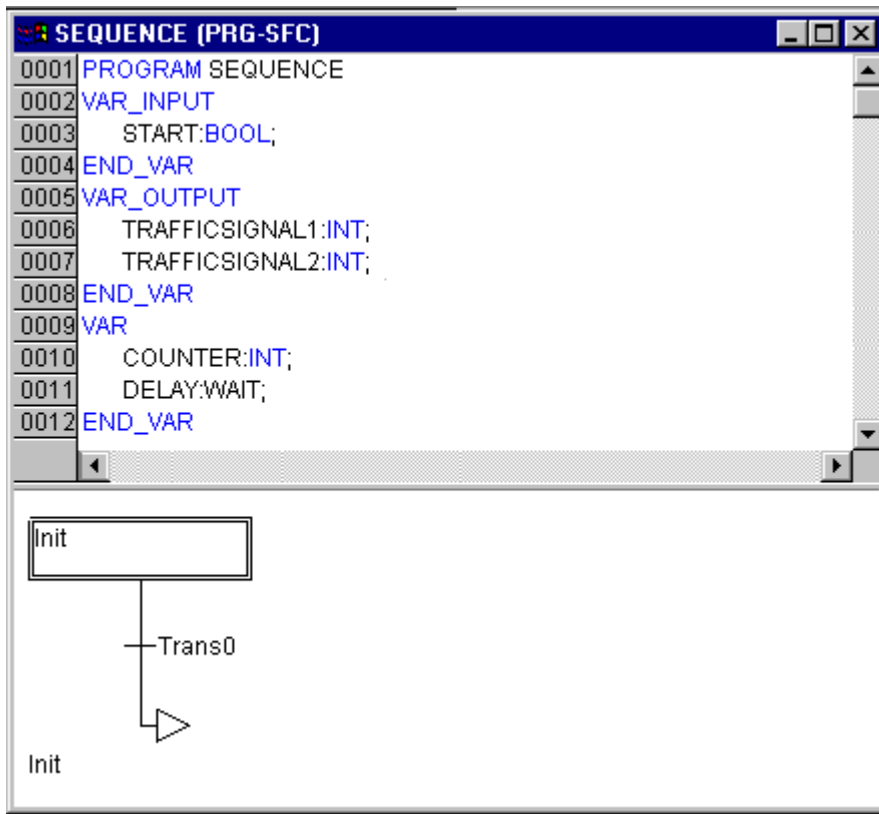


Image 3.5: Program Sequence, First Expansion Level, Declaration Part

Create a SFC diagram

The beginning diagram of a POU in SFC always consists of an action "Init" of a following transition "Trans0" and a jump back to Init. We have to expand that.

Before we program the individual action and transitions let us first determine the structure of the diagrams. We need one step for each TRAFFICSIGNAL phase. Insert it by marking Trans0 and choosing **"Insert" "Step transition (after)"**. Repeat this procedure three more times.

If you click directly on the name of a transition or a step, then this is marked and you can change it. Name the first transition after Init "START", and all other transitions "DELAY.OK".

The first transition switches through when START is TRUE and all others switch through when DELAY in OK produces TRUE, i.e. when the set time period is finished.

The steps (from top to bottom) receive the names Switch1, Green2, Switch2, Green1, whereby Init of course keeps its Name. "Switch" should include a yellow phase, at Green1 TRAFFICSIGNAL1 will be green, at Green2 TRAFFICSIGNAL2 will be green. Finally change the return address of Init after Switch1. If you have done everything right, then the diagram should look like in the following image:

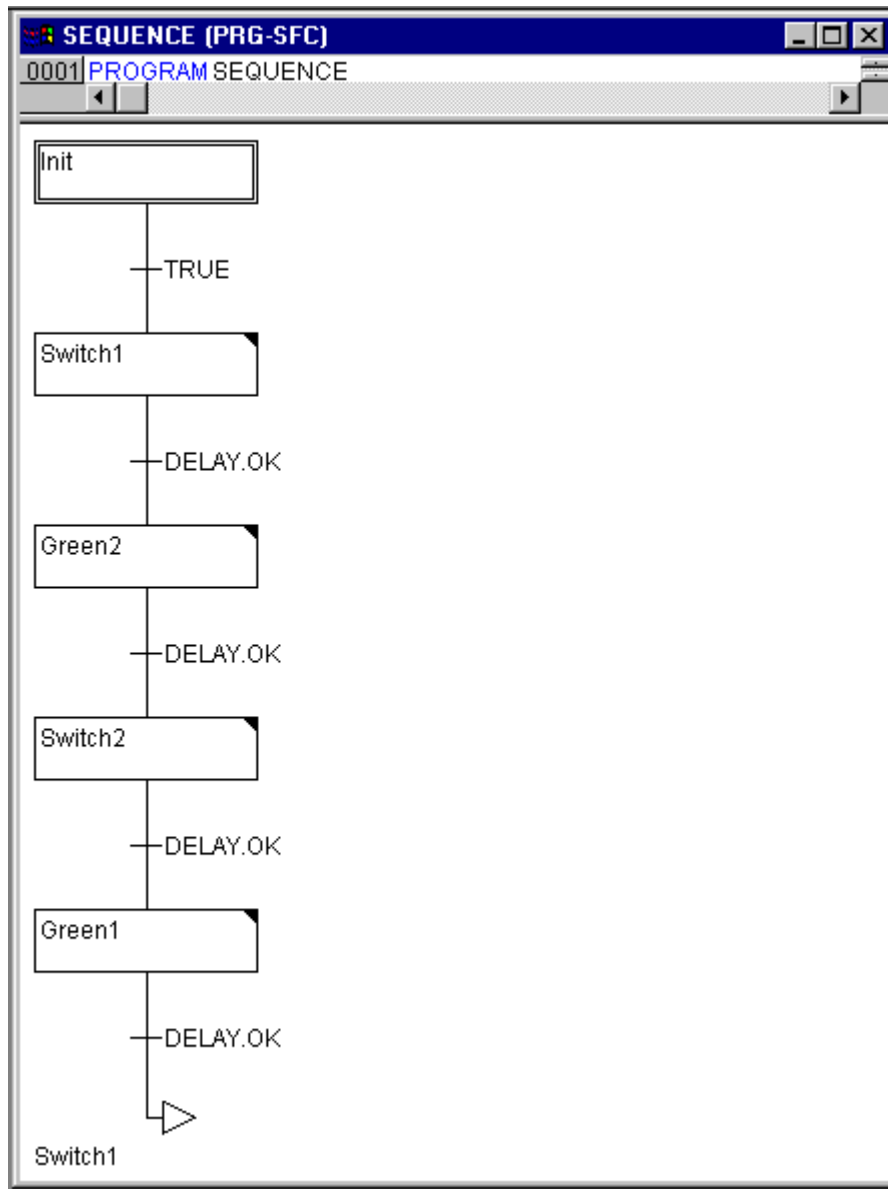


Image 3.6: Program SEQUENCE, First Expansion Level, Instruction Part

Now we have to finish the programming of the individual steps. If you doubleclick on the field of a step, then you get a dialog for opening a new action. In our case we will use IL (Instruction List).

Actions and transition conditions

In the action of the step **Init** the variables are initialized, the STATUS of TRAFFICSIGNAL1 should be 1 (green). The state of TRAFFICSIGNAL2 should be 3 (red). The action Init then looks like in the following image:

```

Action Init (IL)
0001 LD 1
0002 ST TRAFFICSIGNAL1
0003 LD 3
0004 ST TRAFFICSIGNAL2
  
```

Image 3.7: Action Init

Switch1 changes the state of TRAFFICSIGNAL1 to 2 (yellow), and that of TRAFFICSIGNAL2 to 4 (yellow-red). In addition, a time delay of 2000 milliseconds is set. The action is now as follows:

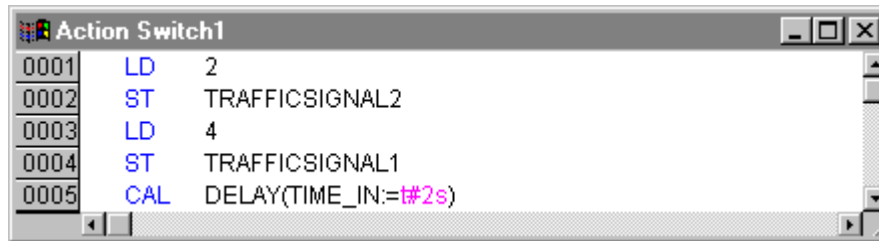


Image 3.8: Action Switch1

With **Green2** TRAFFICSIGNAL1 is red (STATUS:=3), TRAFFICSIGNAL2 green (STATUS:=1), and the delay time is 5000 milliseconds.

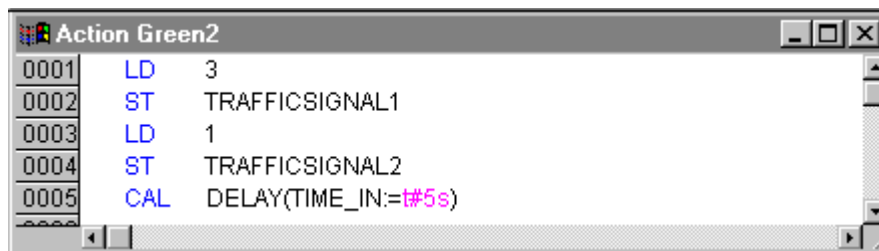


Image 3.9: Action Green2

At **Switch2** the STATUS of TRAFFICSIGNAL1 changes to 4 (yellow-red), that of TRAFFICSIGNAL2 to 2 (yellow). A time delay of 2000 milliseconds is now set.

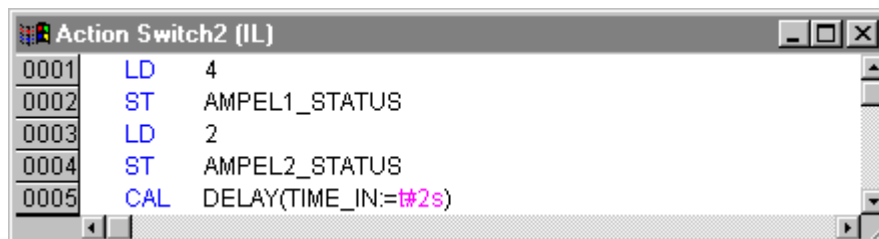


Image 3.10: Action Switch2

With **Green1** TRAFFICSIGNAL1 is green (STATUS:=1), TRAFFICSIGNAL2 is red (STATUS:=3), and the time delay is set to 5000 milliseconds.

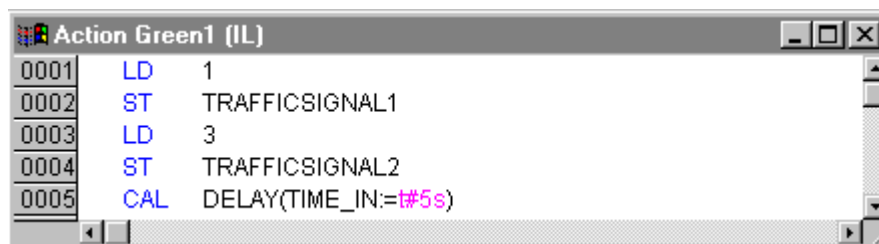


Image 3.11: Action Green1

The first expansion phase of our program is completed. Now you can compile it and also test the simulation.

"SEQUENCE" second expansion level

In order to ensure that our diagram has at least one alternative branch, and so that we can turn off our traffic light unit at night, we now include in our program a counter which, after a certain number of TRAFFICSIGNAL cycles, turns the unit off.

At first we need a new variable COUNTER of the type INT. Declare this as usual in the declaration part of PLC_PRG, and initialize it in Init with 0.

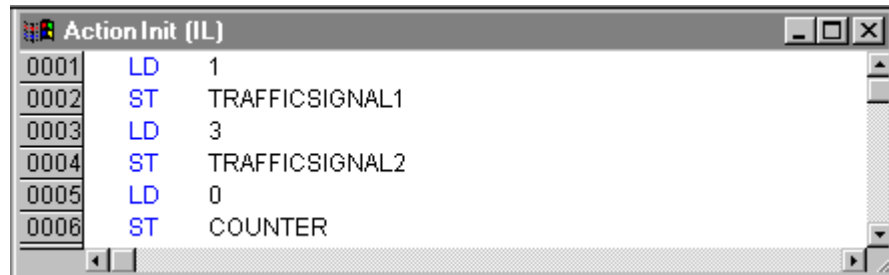


Image 3.12: Action Init, Second Version

Now select the transition after Switch1 and insert a step and then a transition. Select the resulting transition and insert an alternative branch to its left. After the left transition insert a step and a transition. After the resulting new transition insert a jump after Switch1.

Name the new parts as follows: the upper of the two new steps should be called "Count" and the lower "Off". The transitions are called (from top to bottom and from left to right) EXIT, TRUE and DELAY.OK. The new part should look like the part marked with the black border in the following image:

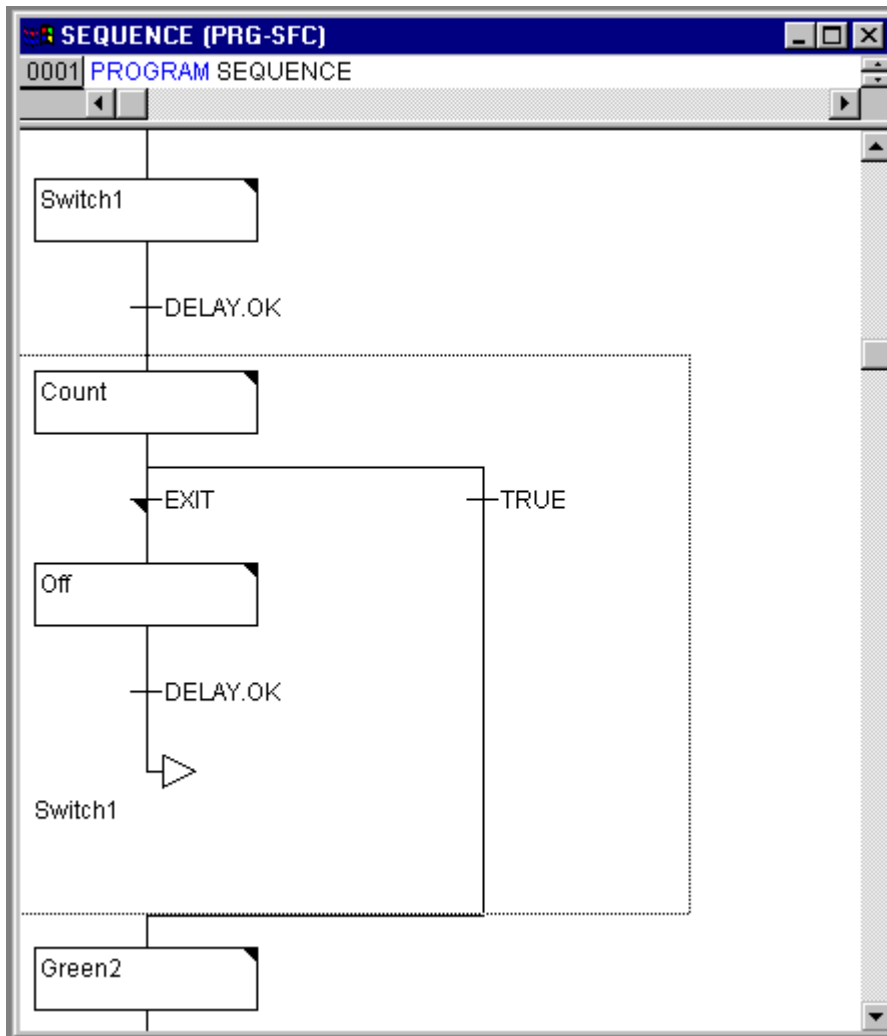


Image 3.13: Program SEQUENCE, Second Expansion Level, Instruction Part

Now two new actions and a new transition condition are to be implemented. At the step Count the variable COUNTER is increased by one:

Address	Instruction	Operand
0001	LD	COUNTER
0002	ADD	1
0003	ST	COUNTER

Image 3.14: Action Count

The EXIT transition checks whether the counter is greater than a certain value, for example 7:

Address	Instruction	Operand
0001	LD	COUNTER
0002	GT	7

Image 3.15: Transition EXIT

At Off the state of both lights is set at 5(OFF), (or each other number not equal 1,2,3 or 4) the COUNTER is reset to 0, and a time delay of 10 seconds is set:

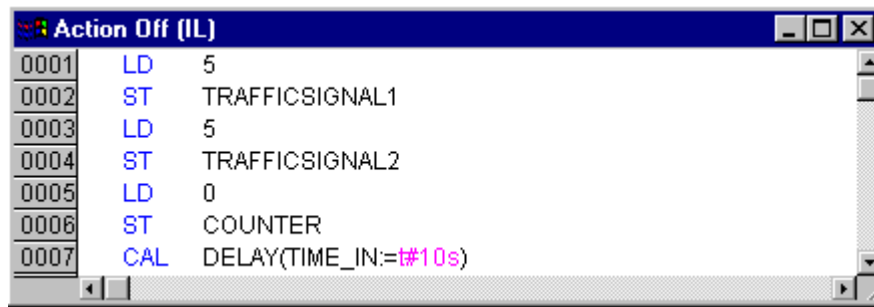


Image 3.16: Action Off

The result

In our hypothetical situation, night falls after seven TRAFFICSIGNAL cycles, for ten seconds the TRAFFICSIGNAL turns itself off, then we have daylight again, the traffic light unit turns itself on again, and the whole process starts again from the beginning.

PLC_PRG

We have defined and correlated the time sequencing of the phases for both sets of traffic lights in the block SEQUENCE. Since, however, we see the traffic lights system as a PROFIBUS-DP system and wish to create the controller configuration over a PROFIBUS, it is necessary for us to make input and output variables available in the block PLC_PRG. We want to start-up the traffic lights system over an ON switch (DP slave) and we want to send each of the six (each traffic light red, green, yellow) lamps (DP slave) the corresponding "signal command" for each step of the SEQUENCE. We are now declaring appropriate Boolean variables for these six outputs and one input in the central processing unit (DP master), before we create the programme in the editor, and are allocating them, at the same time, to the corresponding IEC addresses.

The next step is declare the variables Light1 and Light2 of the type Phases in the declaration editor.



Figure 3.17: Declaration LIGHT1 and LIGHT2

These deliver the Boolean value of each of the six lights to the above mentioned six outputs for each step of the block SEQUENCE. We are not, however, declaring the output variables which are foreseen within the PLC_PRG block but under Resources for Global Variables instead. The Boolean input variable IN, which is used to set the variable START in the block

SEQUENCE to TRUE, can be set in the same way. ON is also allocated to an IEC address.

Select the tab Resources and open the list Global Variables.

Make the declaration as follows:

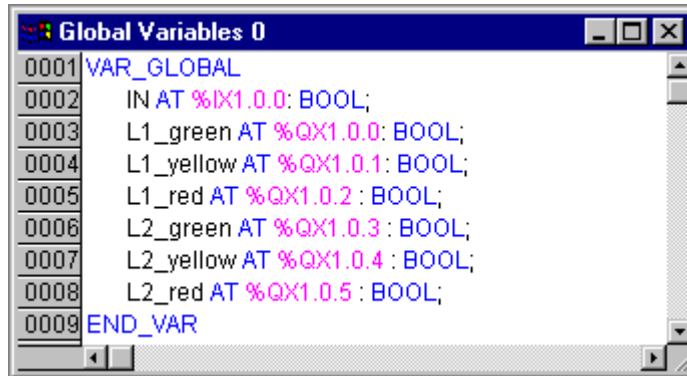


Figure 3.18: Declaration of the Input-/Output Variables for the PROFIBUS-DP Configuration

The name of the variable (e.g. IN) is followed, after AT, by a percent sign which begins the IEC address. I stands for input, Q for output, X for bit, the subsequent '1' in this case references the slot for the PROFIBUS coupler and the last digit indicates the byte offset (see Chapter 'PLC Configuration'). We will be handling the controller configuration afterwards but we want, first of all, to finish off the block PLC_PRG.

For this we go into the editor window and write a program in the language Structured Text.

We call program SEQUENCE and consign the status of variable IN to the input variable START.

Then we allocate the color status (variables TRAFFICSIGNAL1 and TRAFFICSIGNAL2), which is demanded in the currently active step in SEQUENCE, to the function block TRAFFICSIGNAL (input variable STATUS). The function block instances LIGHT1 and LIGHT2 give out the correlating color values for each trafficsignal (LIGHT1.GREEN, LIGHT1.YELLOW, LIGHT1.RED, correspondingly LIGHT2.GREEN etc.). Those are consigned to the output variables L1_green, L2_yellow etc.

Your program should finally look like the example shown here.

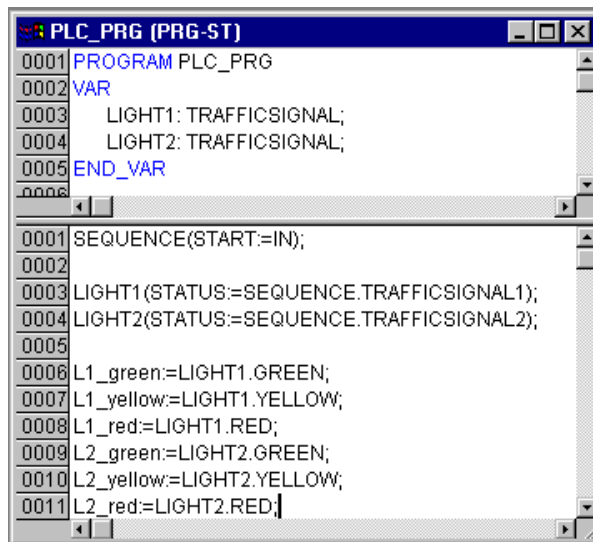


Figure 3.19: PLC_PRG, Declaration and presentation with the Structured Text

Controller configuration

Before we actually test the program, to see whether and how it runs, and finally, as our climax, produce a small visualization we first of all want to connect the traffic lights system over the PROFIBUS-DP to the I/O level.

To achieve this go to the tab Resources and select "PLC Configuration". Here the hardware must be described which the actual project is being created for. Click with the mouse on the line "Hardware configuration". A dotted box appears within which you can **Append a Subelement**. Select 'DP-Master' and the dialog **Select a PROFIBUSDP-Master** will open. Here you can choose from a selection list (**Device Name**) device 07 KT 97. After leaving the dialog with **OK** the master will be shown in the configuration tree.

Additionally we need a slave device which will process the input and output data for our traffic signs. To add a slave to the configuration, select the master in the configuration tree and choose (again by "Insert" "Append DP-Slave" or by using the right mouse button) a DP Slave (for example 07 KT 97-DPS) and confirm with **OK**. The properties of the slave device and its selection of input/output modules you can check and adapt in the following way: Select the slave device in the configuration tree and use the right mouse button or the command "Extras" "Properties" to get to a dialog titled with the device name. For our example you can leave the first tab "Standard Parameters" as it is. The second tab "Input/Output" offers on its left side a list of the devices' input and output modules. Select one output module with a width of 1 byte by mouse click and copy it to the configuration list on the right side by pressing >>. Do the same for a input module (switch).

Close the dialog with **OK** and see for the new inserted modules in the configuration tree. Doubleclick on the plus sign at the slave element to open the sub-modules down to the level of the input and output addresses.

This is the way in which the program gets connected to the IEC addresses %IX1.0.x and %QX1.0.x, assigned in the Global Variables list.

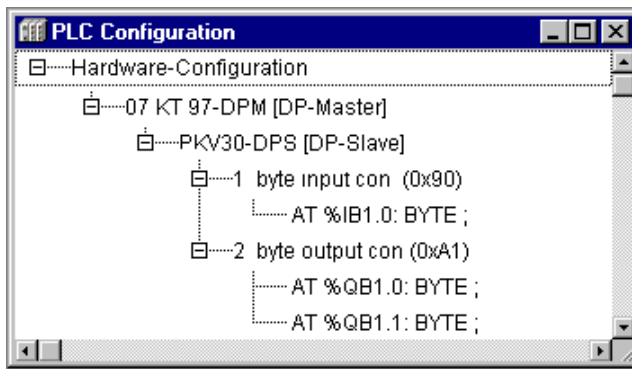


Figure 3.20: PROFIBUS-DP Controller configuration of the traffic lights system


TRAFFICSIGNAL simulation

Now test your program. For this you must compile it ("**Project**" "**Rebuild all**") login ("**Online**" "**Login**" and then load it "**Online**" "**Download**"). If you now select "**Online**" "**Run**", the chronological order of the individual steps of your main program can be followed. The window of the POU PLC_PRG has now changed to the monitor window. Click twice on the plus sign in the declaration editor, the variable display drops down, and you can see the values of the individual variables.

3.2 Visualizing a Traffic Signal Unit

With the visualization of **907 AC 1131** you can quickly and easily bring project variables to life. You find an complete description of the visualization in Chapter 8. We will now plot two traffic signals and an ON-Switch for our traffic light unit which will illustrate the switching process.

Creating a new visualization

In order to create a visualization you must first select the range of **Visualization** in the Object Organizer. First click on the lower edge of the window on the left side with the **POU** on the register card with this symbol  and the name **Visualization**. If you now choose the command "**Project**" "**Object Add**", then a dialog box opens.

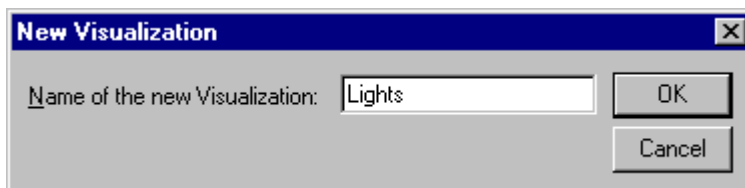


Image 3.21: Dialog Box for Opening a New Visualization

Enter here any name. When you confirm the dialog with **OK**, then a window opens in which you can set up your new visualization.

Insert element in Visualization

For our TRAFFICSIGNAL visualization you should proceed as follows:

- Give the command **"Insert" "Ellipse"** and try to draw a medium sized circle (Ø2cm). For this click in the editor field and draw with pressed left mouse button the circle in its length.
- Now doubleclick the circle. The dialog box for editing visualization elements opens
- Choose the category **Variables** and enter in the field **Change color** the variable name **.L1_red** or choose this variable using the input assistance (button <F2>). This addresses the variable **RED** of the function block instance **TRAFFICSIGNAL1** of the POU **PLC_PRG**.

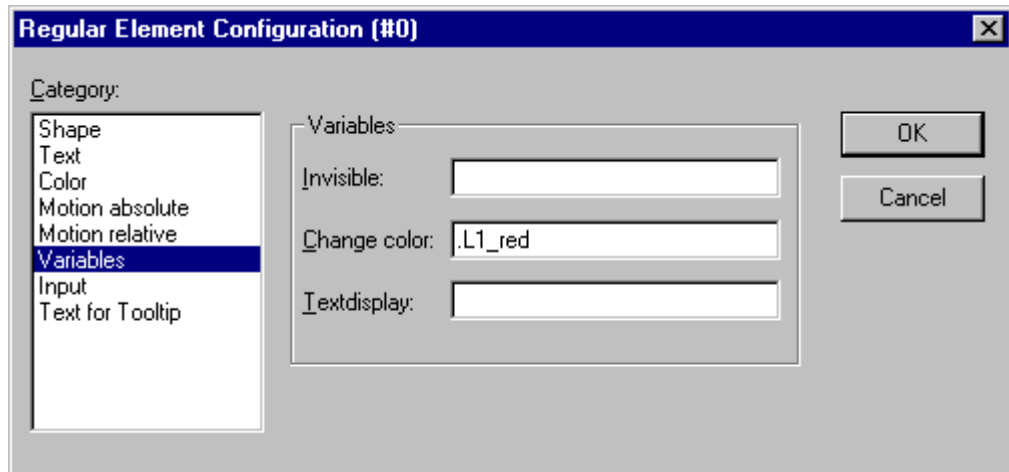


Image 3.22: Visualization Dialog Box Variables

- Then choose the category **Color** and click on the button **Inside** in the area **Color**. Choose as neutral a color as possible, such as black.
- Now click on the button **within** in the area **Alarm color** and choose the red which comes closest to that of a red light.

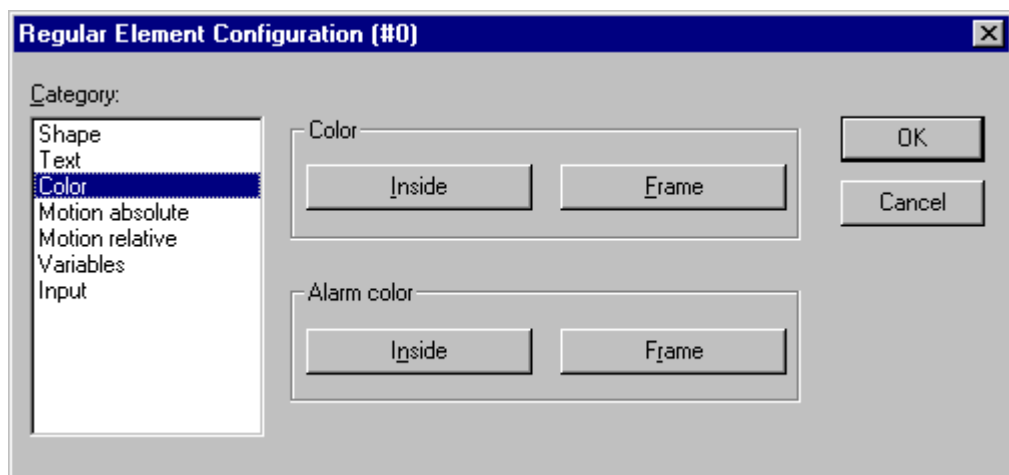


Image 3.23: Visualization Configuration Dialog Box (Color category)

The resulting circle will normally be black, and when the variable **RED** from **TRAFFICSIGNAL1** is **TRUE**, then its color will change to red. We have therefore created the first light of the first **TRAFFICSIGNAL1**!

The other traffic lights

Now enter the commands **"Edit" "Copy"** (<Ctrl>+<C>) and then twice **"Edit" "Paste"** (<Ctrl>+<V>). That gives you two more circles of the exact same size lying on top of the first one. You can move the circles by clicking on the circle and dragging it with pressed left mouse button. The desired position should, in our case, be in a vertical row in the left half of the editor window. Doubleclick on one of the other two circles in order to open the configuration dialog box again. Enter in the field **Change Color** of the corresponding circle the following variables:

for the middle circle: L1_yellow

for the lowest circle: L1-green

Now choose for the circles in the category **Color** and in the area **Alarm color** the corresponding color (yellow or green).

The TRAFFICSIGNAL case

Now enter the command **"Insert" "Rectangle"**, and insert in the same way as the circle a rectangle which encloses the three circles. Once again choose as neutral a color as possible for the rectangle and give the command **"Extras" "Send to back"** so that the circles are visible again.

If simulation mode¹ is not yet turned on, you can activate it with the command **"Online" "Simulation"**.

If you now start the simulation with the commands **"Online" "Login"** and **"Online" "Run"**, then you can observe the color change of the first traffic signal.

The second traffic signal

The simplest way to create the second traffic signal is to copy all of the elements of the first traffic signal. For this you select all elements of the first traffic signal and copy them (as before with the lights of the first traffic signal) with the commands **"Edit" "Copy"** and **"Edit" "Paste"**. You then only have to change the text "TRAFFICSIGNAL1" in the respective dialog boxes into "TRAFFICSIGNAL2", and the visualization of the second traffic signal is completed.

The ON switch

Insert a rectangle and award it, as described above, a colour for a traffic light of your choice and enter .ON at **Variables** for the **Change color**. Enter "ON" in the input field for **Content** in the category Text.

¹ The simulation mode is active if a check mark (✓) appears in front of the menu item "Simulation" in the "Online" menu..

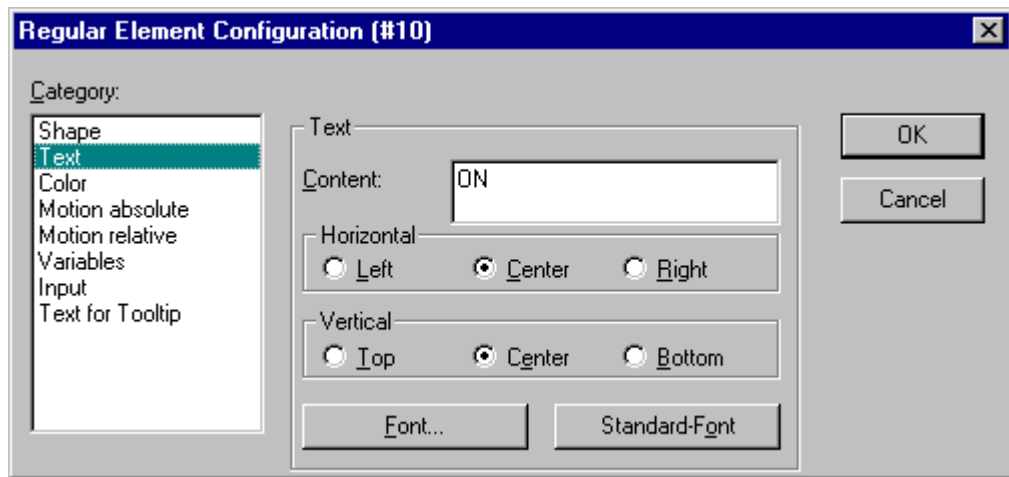


Image 3.24: Dialog to configure the visualization elements (Category Text)

In order to set the variable ON to TRUE with a mouse click on the switch, the variable .ON must be entered into the category **Variables**. Also select the Option Variable keying and enter the variable .ON at this point. Variable keying means that when a mouse click is made on the visualization element the variable .ON is set to the value TRUE but is reset to the value FALSE when the mousekey is released again (we have created hereby a simple switch-on device for our traffic lights program).

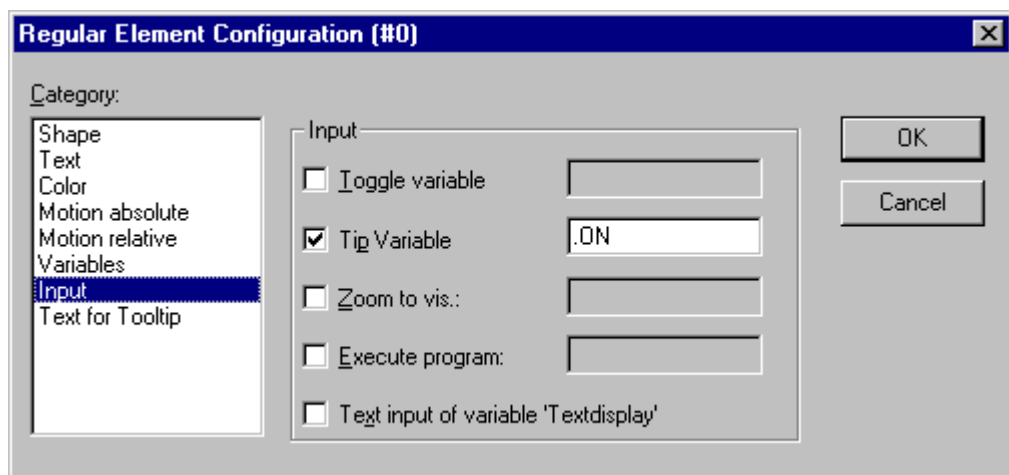


Image 3.25: Dialog to configure the visualization elements (Category Input)

Font in the visualization

In order to complete the visualization you should first insert two more rectangles which you place underneath the traffic signals.

In the visualizations dialog box set white in the category **Color** for **Frame** and write in the category **Text** in the field **Contents** "Light1" or "Light2". Now your visualization looks like this:

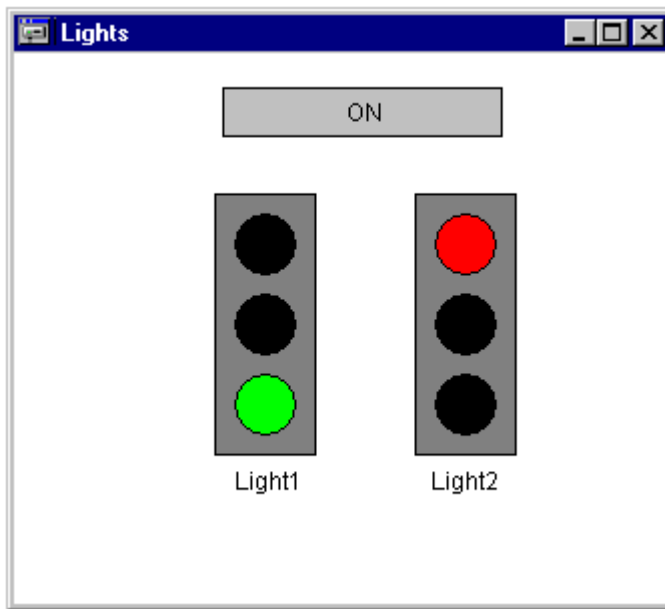


Image 3.26: Visualization for the Sample Project Trafficsignal

4 The Individual Components

4.1 The Main Window

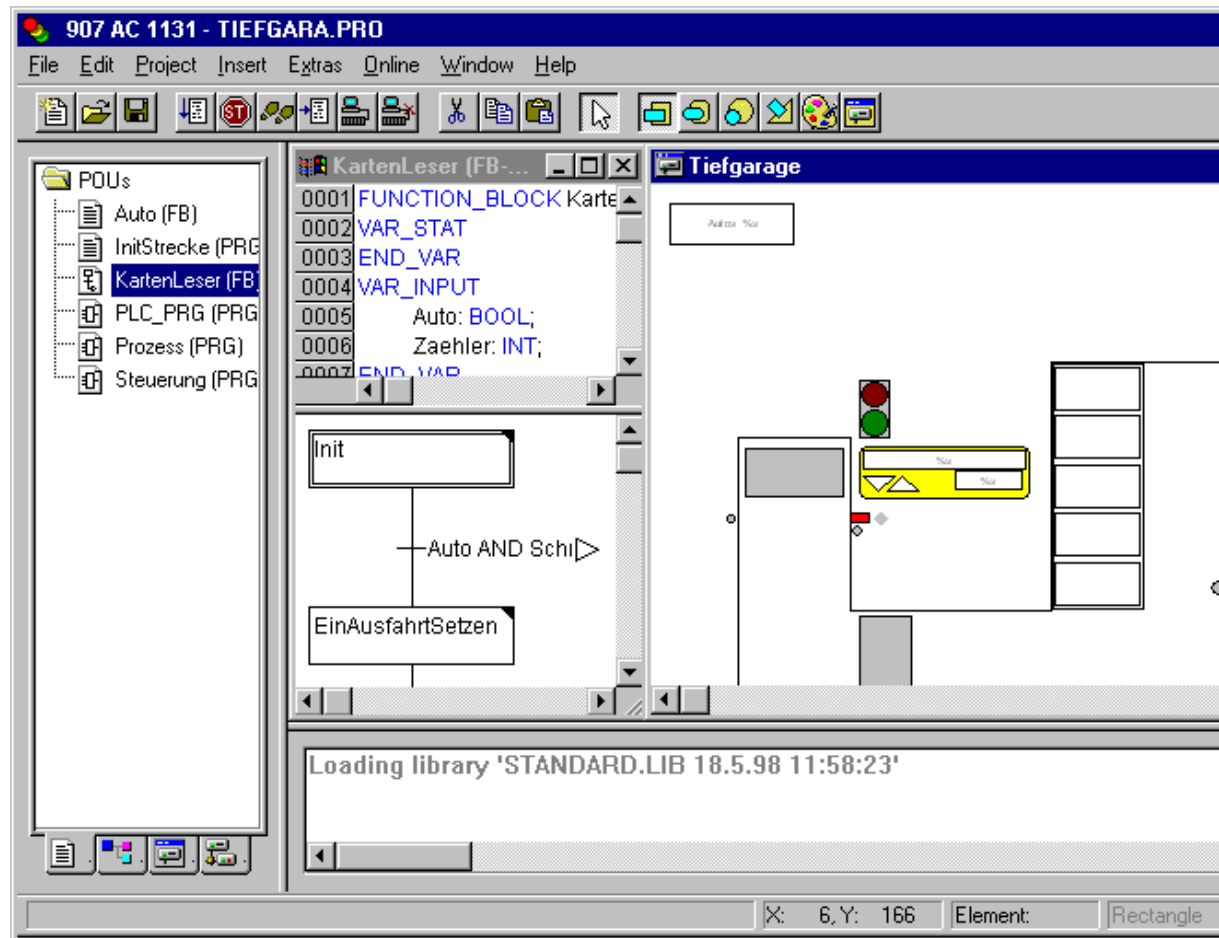


Image 4.1: The Main Window

The following elements are found in the main window of **907 AC 1131** (from top to bottom):

- The **menu bar**
- The **Tool bar** (optional); with buttons for faster selection of menu commands.
- The **Object Organizer** with register cards for **POUs**, **Data types**, **Visualizations**, and **Resources**
- A vertical **screen divider** between the Object Organizer and the Work space of **907 AC 1131**
- The **Work space** in which the editor windows are located
- The **message window** (optional)
- The **Status bar** (optional); with information about the current status of the project

Menu bar

The menu bar is located at the upper edge of the main window. It contains all menu commands.

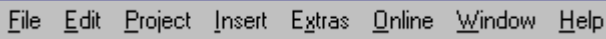


Image 4.2: Menu Bar

Tool bar

By clicking with the mouse on a symbol you can select a menu command more quickly. The choice of the available symbols automatically adapts itself to the active window.

The command is only carried out when the mouse button is pressed on the symbol and then released.

If you hold the mouse pointer for a short time on a symbol in the tool bar, then the name of the symbol is shown in a Tooltip.

In order to see a description of each symbol on the tool bar, select in Help the editor about which you want information and click on the tool bar symbol in which you are interested.

The display of the tool bar is optional (see "**Project**" "**Options**" category **Desktop**).



Image 4.3: Tool bar with symbols

Object Organizer

The Object Organizer is always located on the left side of 907 AC 1131 . At the bottom there are four register cards with symbols for the four types of objects **POUs**, **Data types**, **Visualizations** and **Resources**. In order to change between the respective object types click with the mouse on the corresponding register card or use the left or right arrow key.

You will learn in chapter Creating and Deleting Objects, etc.how to work with the objects in the Object Organizer.

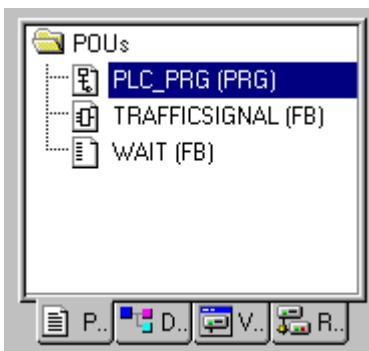


Image 4.4: Object Organizer

Screen divider

The screen divider is the border between two non-overlapping windows. In **907 AC 1131** there are screen dividers between the Object Organizer and the Work space of the main window, between the interface (declaration part) and the implementation (instruction part) of POU's and between the Work space and the message window.

You can move the screen divider with the mouse pointer. You do this by moving the mouse with the left mouse button pressed.

Make sure the screen divider always remains at its absolute position, even when the window size has been changed. If it seems that the screen divider is no longer present, then simply enlarge your window.

Work space

The Work space is located on the right side of the main window in **907 AC 1131**. All editors for objects and the library manager are opened in this area.

You find the description of the editors in Chapter 5.

Under the menu item "**Window**" you find all commands for window management.

Message window

The message window is separated by a screen divider underneath the work space in the main window.

It contains all messages from the previous compilations, checks, or comparisons.

If you doubleclick with the mouse in the message window on a message or press <Enter>, the editor opens with the object. The relevant line of the object is selected. With the commands "Edit" "Next error" and "Edit" "**Previous error**" you can quickly jump between the error messages.

The display of the message window is optional (see "**Window**" "**Messages**").

Status bar

The status bar at the bottom of the window frame of the main window in **907 AC 1131** gives you information about the current project and about menu commands.

If an item is relevant, then the concept appears on the right side of the status bar in black script, otherwise in gray script.

When you are working in online mode, the concept **Online** appears in black script. If you are working in the offline mode it appears in gray script.

In Online mode you can see from the status bar whether you are in the simulation (**SIM**), the program is being processed (**RUNS**), a breakpoint is set (**BP**), or variables are being forced (**FORCE**).

With text editor the line and column number of the current cursor position is indicated (e.g. **Line:5, Col.:11**). In online mode 'OV' is indicated black in the status bar. Pressing the <Ins> key switches between Overwrite and Insert mode.

If the mouse point is in a visualization, the current **X** and **Y position** of the cursor in pixels relative to the upper left corner of the screen is given. If the mouse pointer is on an **Element**, or if an element is being processed, then its number is indicated. If you have an element to insert, then it also appears (e.g. **Rectangle**).

If you have chosen a menu command but haven't yet confirmed it, then a short description appears in the status bar.

The display of the statusbar is optional (see "**Project**" "**Options**" category **Desktop**).

Context Menu

Shortcut: <Shift>+<F10>

Instead of using the menu bar for executing a command, you can use the right mouse button. The menu which then appears contains the most frequently used commands for a selected object or for the active editor. The choice of the available commands adapts itself automatically to the active window. The choice of the available commands adapts itself automatically to the active window.

4.2 Options

In **907 AC 1131** you can configure the view of the main window (and have more than one viewpoint). In addition you can make other settings. For this you have the command "**Project**" **Options**" at your disposal. The settings you make thereby are, unless determined otherwise, saved in the file "907 AC 1131 .ini" and restored at the next **907 AC 1131** startup.

"Project" "Options"

With this command the dialog box for setting options is opened. The options are divided into different categories. Choose the desired category on the left side of the dialog box by means of a mouse click or using the arrow keys and change the options on the right side.

You have at your disposal the following categories:

- Load & Save
- User information
- Editor
- Desktop
- Color
- Directories
- Build

- Passwords

Load & Save

If you choose this category , then you get the following dialog box:

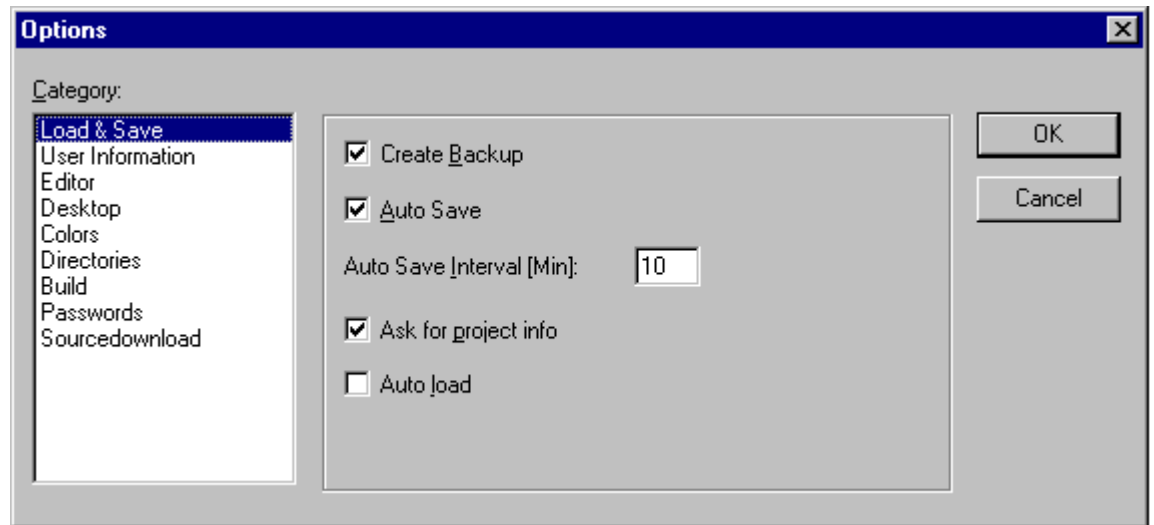


Image 4.5: Option dialog box of the category Load & Save

When activating an option, a check (✓) appears before the option.

If you choose the option **Create Backup**, then **907 AC 1131** creates a backup file at every save with the extension ".bak". In this way you can always restore the versions before the last save.

If you choose the option **Auto Save** , then while you work your project is constantly saved to a temporary file with the extension ".asd" according to a set time interval (**Auto Save Interval**). This file is erased at a normal exit from the program. If for any reason **907 AC 1131** is not shut down "normally" (e.g. due to a power failure), then the file is not erased. When you open the file again the following message appears:

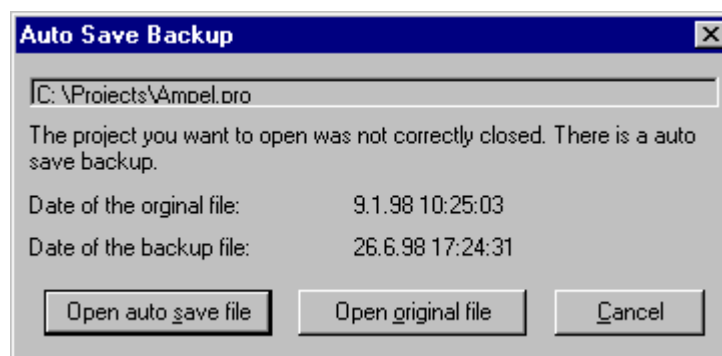


Image 4.6: There is an auto save backup.

You can now decide whether you want to open the original file or the auto save file.

If you request the option **Ask for project info**, then when saving a new project, or saving a project under a new name, the project info is automatically called.

You can visualize the project info with the command "**Project**" "**Project info**" and also process it.

If you choose the option **Auto Load**, then at the next start of **907 AC 1131** the last open project is automatically loaded. The loading of a project at the start of **907 AC 1131** can also take place by entering the project in the command line.

User information

If you choose this category , then you get the following dialog box:



Image 4.7: Options dialog box of the category User information

To User information belong the **Name** of the user, his **Initials** and the **Company** for which he works. Each of the entries can be modified.

Editor

If you choose this category , then you get the following dialog box:

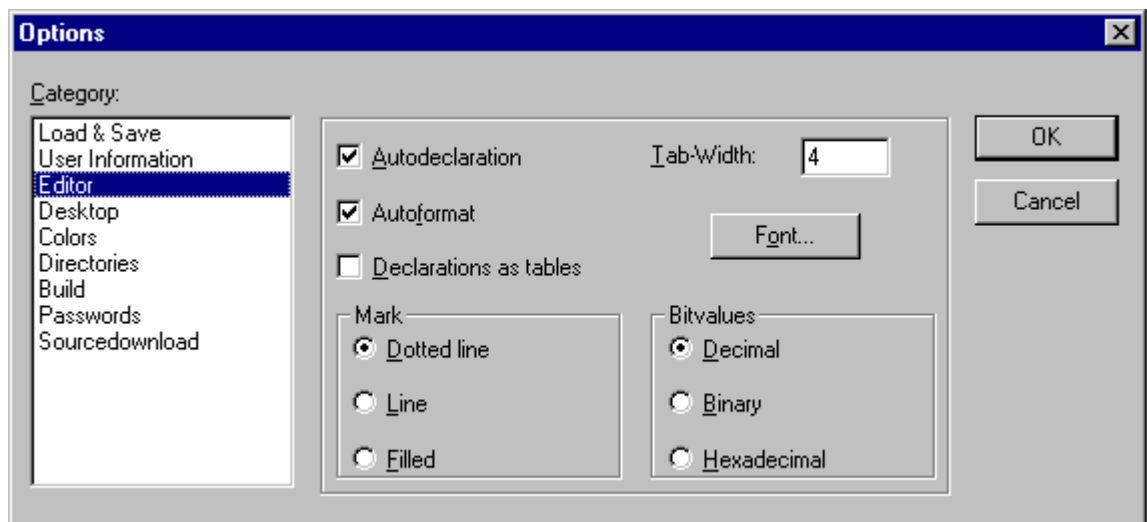


Image 4.8: Options dialog box of the category Editor

When activating an option, a check (✓) appears before the option.

You can make the following settings for the Editors:

- Autodeclaration
- Autoformat
- Declaration as table
- Tab width
- Font
- Display of the text selection
- Display of the Bitvalues

Autodeclaration

If you have chosen the **Autodeclaration**, then (following the input of a not-yet-declared variable) a dialog box will appear in all editors with which this variable can be declared.

Autoformat

If the option **Autoformat** in the category **Editor** of the options dialog box has been chosen, then **907 AC 1131** executes automatic formatting in the IL editor and in the declaration editor. When you finish with a line, the following formatting is made:

- Operators written in small letters are shown in capitals;
- Tabs are inserted so that the columns are uniformly divided.

Declarations as tables

If the option **Declarations as tables** in the **Editor** category in the Options dialog box is selected, then you can edit variables in a table instead of using the usual declaration editor (see chapter 'The Declaration Editor'). This table is arranged as a card-index box in which there are register cards for input, output, local, and input/output variables. For each variable you have available the fields **Name**, **Address**, **Type**, **Initial**, and **Comment**.

Tab-Width

In the field **Tab-Width** in the category **Editor** of the Options dialog box you can determine the width of a tab as shown in the editors. The default setting is four characters, whereby the character width depends upon the font which is chosen.

Font

By clicking on the button **Font** in the category **Editor** of the Options dialog box you can choose the font in all **907 AC 1131** editors. The font size is the basic unit for all drawing operations. The choice of a larger font size thus enlarges the printout, even with each editor of **907 AC 1131**.

After you have entered the command, the font dialog box opens for choosing the font, style and font size.

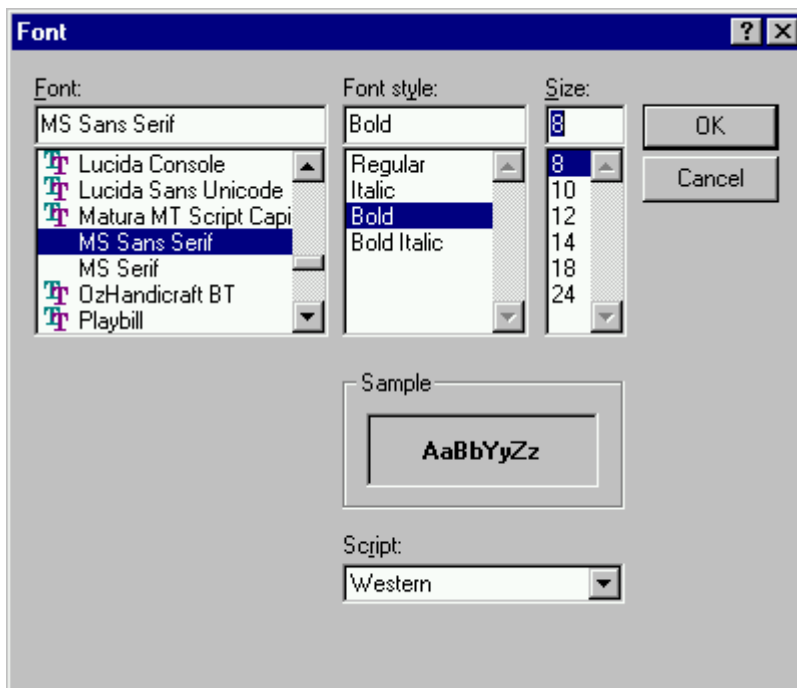


Image 4.9: Dialog box for setting the font

Mark

When choosing **Mark** in the **Editor** category in the Options dialog box you can choose whether the current selection in your graphic editors should be represented by a dotted rectangle (**Dotted**), a rectangle with continuous lines (**Line**) or by a filled-in rectangle (**Filled**). In the last case the selection is shown inverted.

The selection is activated in front of which a (●) point appears.

Bitvalues

When choosing **Bitvalues** in the category **Editor** of the Options dialog box you can choose whether binary data (type BYTE, WORD, DWORD) during monitoring should be shown **Decimal**, **Hexadecimal**, or **Binary**.

The selection is activated in front of which a (●) point appears.

Options for the Desktop

If you choose this category, then you get the following dialog box:

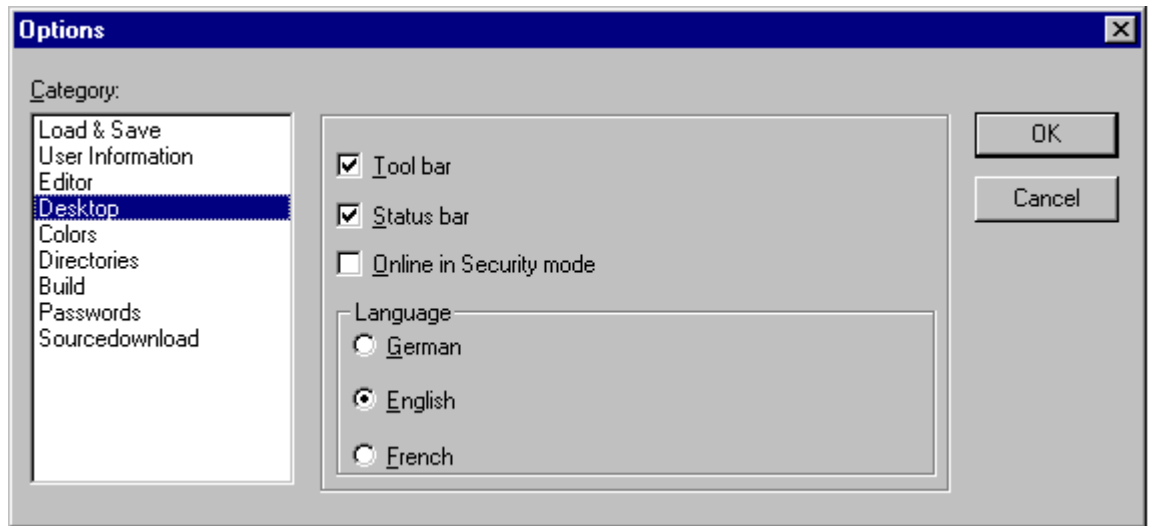


Image 4.10: Options dialog box of the category Desktop

If the option **Tool bar** has been chosen, then the tool bar with the buttons for faster selection of menu commands becomes visible underneath the menu bar.

If the option **Status bar** has been chosen, then the status bar at the lower edge of the **907 AC 1131** main window becomes visible.

If the option **Online in Security mode** has been chosen, then in Online mode with the commands "**Run**", "**Stop**", "**Reset**", "**Toggle Breakpoint**", "**Single cycle**", "**Write values**", "**Force values**" and "**Release force**", a dialog box appears with the confirmation request whether the command should really be executed. This option is saved with the project.

In **language** you can define, in which language the menu and dialog texts are displayed.



Note: Please note, that the language choice is only possible under Windows NT !

Colors

If you choose this category, then you get the following dialog box:

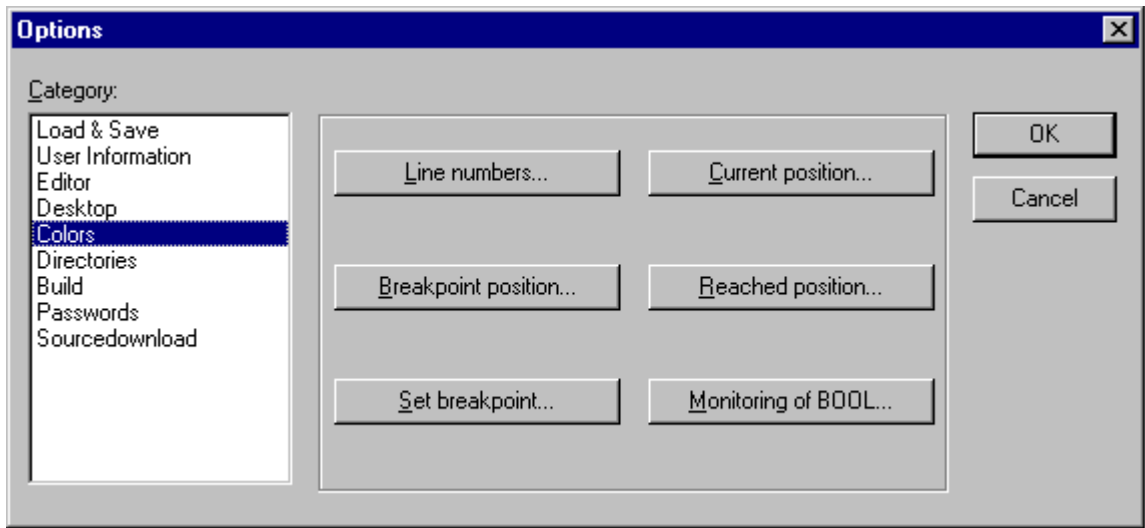


Image 4.11: Options dialog box of the category Color

You can edit the default color setting of **907 AC 1131**. You can choose whether you want to change the color settings for **Line numbers** (default setting: light gray), for **Breakpoint positions** (dark gray), for a **Set breakpoint** (light blue), for the **Current position** (red), for the **Reached Positions** (green) or for the **Monitoring of Boolean values** (blue).

If you have chosen one of the indicated buttons, the dialog box for the input of colors opens.



Image 4.12: Dialog box for setting colors

Directories

If you choose this category, then you get the following dialog box:

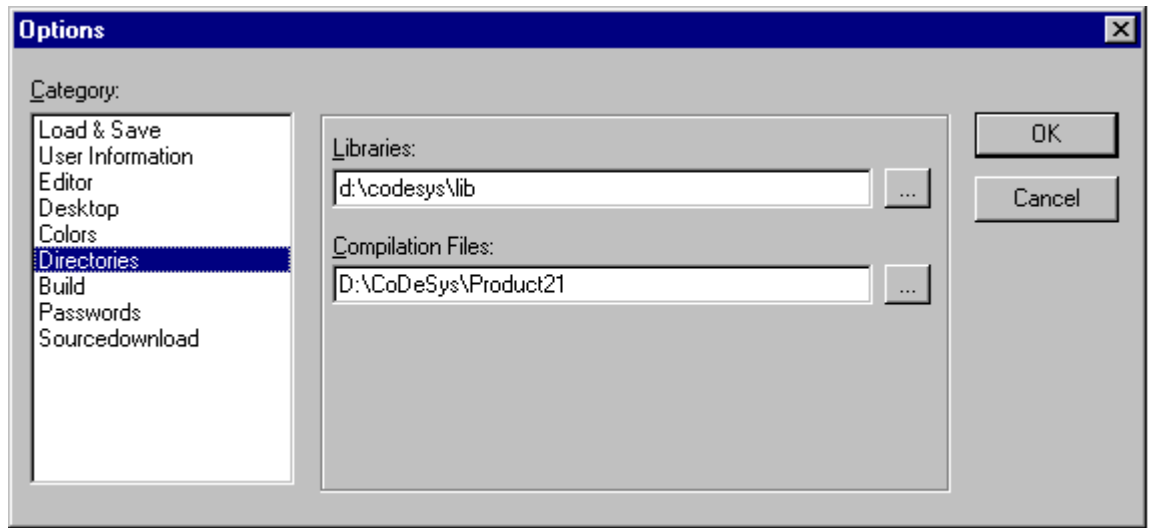


Image 4.13: Options dialog box of the category Directories

In the input fields **Libraries** and **Compilation Files** you can indicate directories from which **907 AC 1131** should extract the libraries or compilation files. If you activate the button (...) behind a field, then the dialog box for selecting a directory opens.

Build

If you choose this category, then you get the following dialog box:

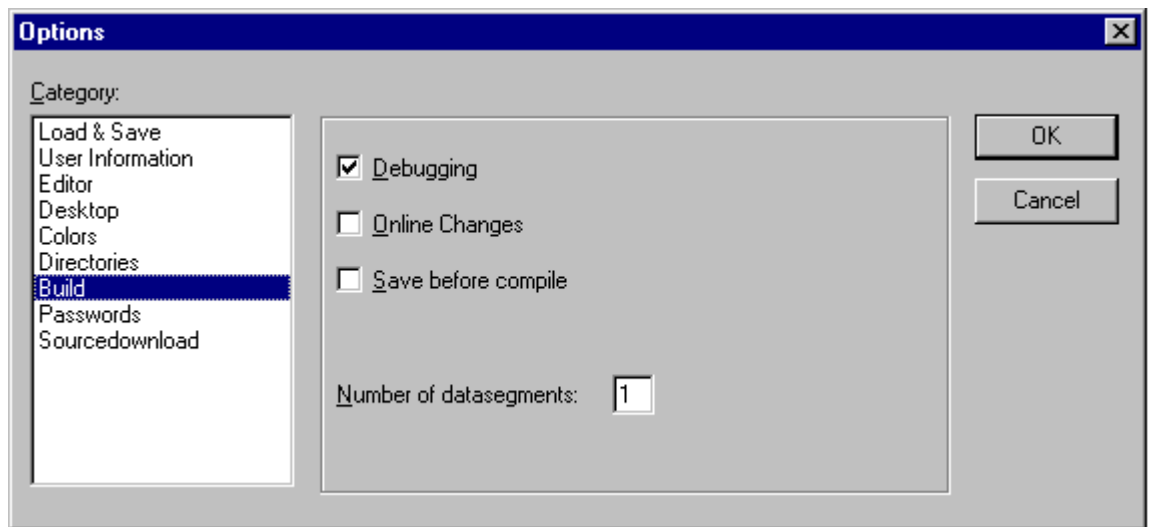


Image 4.14: Options dialog box of the category Build

If the option **Debugging** has been chosen, then the code can significantly increase in size. Choosing this option makes it possible to generate additional debugging codes. This is necessary in order to use the **907 AC 1131** debugging functions. If you deactivate this option, then you make possible faster processing and a smaller code. This option is saved with the project.

If the option **Online Changes** has been chosen, then your project can be changed in the Online mode. With this new compilation only the changed POU's are loaded into the PLC. (See "**Project**" "**Build**")

If the option **Save before compile**, then your project will be saved every time before compilation.

With the indication of the **Number of data segments** you can determine how much space is reserved in the PLC for the data of your project. If during Build you get the message: "The global variables need too much memory", then increase the number of segments in "Project" "Build" ", which increases the number of the data segments.

These options are saved with the project.

Passwords

If you choose this category, then you get the following dialog box:

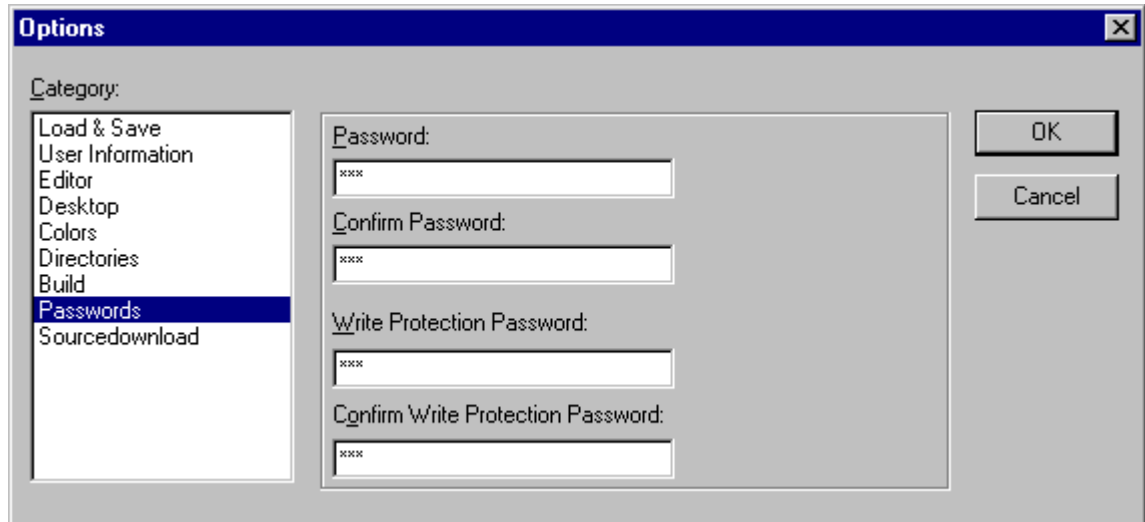


Image 4.15: Options dialog box of the category Passwords

To protect your files from unauthorized access **907 AC 1131** offers the option of using a password to protect against your files being opened or changed.

Enter the desired password in the field **Password**. For each typed character an asterisk (*) appears in the field. You must repeat the same word in the field **Confirm Password**. Close the dialog box with **OK**. If you get the message:

"The password does not agree with the confirmation",

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

If you now save the file and then reopen it, then you get a dialog box in which you are requested to enter the password. The project can then only be opened if you enter the correct password. Otherwise **907 AC 1131** reports:

"The password is not correct."

Along with the opening of the file, you can also use a password to protect against the file being changed. For this you must enter a password in the field **Write Protection Password** and confirm this entry in the field underneath.

A write-protected project can be opened without a password. For this simply press the button **Cancel**, if **907 AC 1131** tells you to enter the write-protection password when opening a file. Now you can compile the project, load it into the PLC, simulate, etc., but you cannot change it.

Of course it is important that you memorize both passwords. However, if you should ever forget a password, then contact the manufacturer of your PLC.

The passwords are saved with the project.

In order to create differentiated access rights you can define user groups and "**Passwords for user groups**").

'Sourcedownload'

The following dialog will be opened when you select this category:

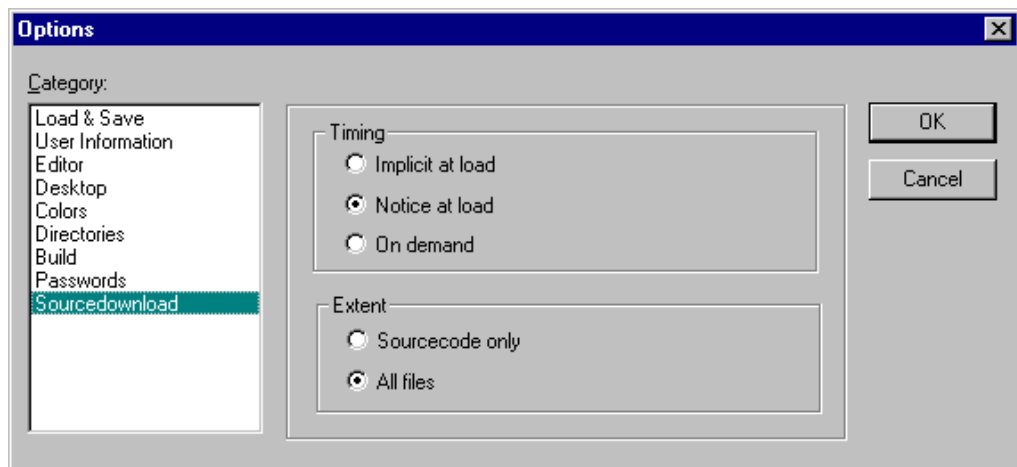


Image 4.16: Option dialog for the category Sourcedownload

You can choose to which **Timing** and what **Extent** the project is loaded into the controller system. The option **Sourcecode only** exclusively involves just the 907 AC 1131 file (file extension .pro). The option **All files** also includes files such as the associated library files, visualization bitmaps, configuration files, etc.

Using the option **Implicit at load** allows the selected file range to be automatically loaded into the controller system on the command '**Online**' '**Load**'.

Using the option **Notice at load** offers a dialog, when the command '**Online**' '**Load**' is given, with the question "Do you want to write the source code into the controller system?". Pressing **Yes** will automatically load the selected range of files into the controller system, or you can alternatively finish with **No**.

When using the option **On demand** the selected range of files must be expressly loaded into the controller system by giving the command '**Online**' '**Sourcecode download**'.

The project which is stored in the controller system can be retrieved by using 'File' 'Open' with **Open project from PLC**. The files will be unpacked in the process.

See Chapter 4.3, 'File' 'Open' for details !

4.3 Managing Projects

The commands which refer to entire project are found under the menu items "File" and "Project". Some of the commands under "Project" deal with objects and are therefore described in the chapter Creating and Deleting Objects, etc..

"File" "New"

Symbol: 

With this command you create an empty project with the name "Untitled". This name must be changed when saving.

"File" "Open"

Symbol: 

With this command you open an already existing project. If a project has already been opened and changed, then **907 AC 1131** asks whether this project should be saved or not.

The dialog box for opening a file appears, and a project file with the extension "*.pro" or a library file with the extension "*.lib" must be chosen. This file must already exist. It is not possible to create a project with the command "Open".

To upload a project from the PLC, press PLC at **Open project from PLC**. If there is no current connection to the PLC, the dialog Communication parameters appears to define the transmission parameters. If an online connection is made, there will be checked whether there are already project files with equal names in the local PC directories. In this case the **dialog Load project from PLC** is opened where you can decide to replace or not to replace the local files by that used in the PLC.



Note: Please note, that you in any case have to give a new name to a project, when you load it from the PLC to your local directory, otherwise it is unnamed.

If there has not yet been loaded a project to the PLC, you get an error message.

(See also 'Project' 'Options' category 'Sourcedownload').

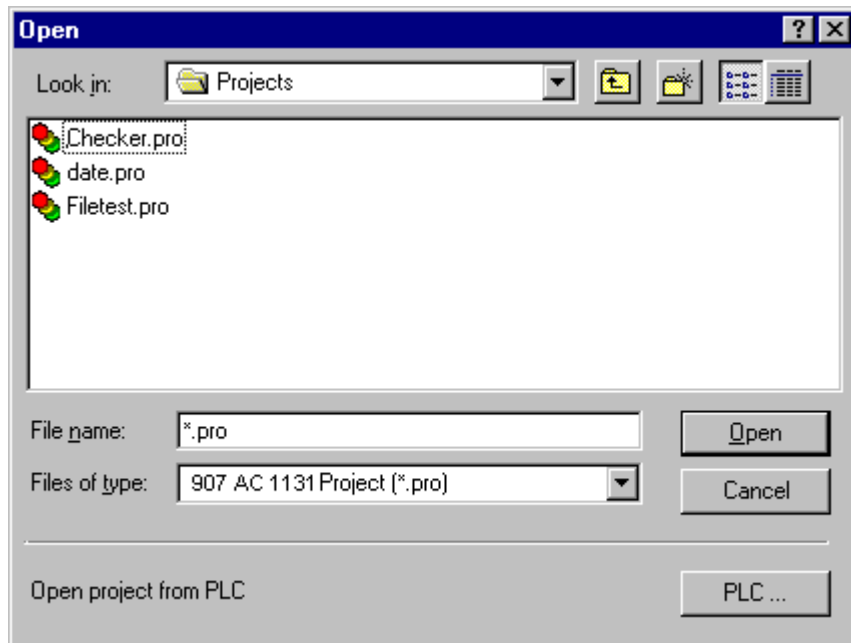


Image 4.17: Standard dialog box for opening a file in 907 AC 1131

The most recently opened files are listed under the command **"File" "Exit"**. If you choose one of them, then this project is opened.

If **Passwords** or **User groups** have been defined for the project, then a dialog box appears for entering the password.

"File" "Close"

With this command you close the currently-open project. If the project has been changed, then **907 AC 1131** asks if these changes are to be saved or not.

If the project to be saved carries the name "Untitled", then a name must be given to it (see **"File" "Save as"**).

"File" "Save"

Symbol:  **Shortcut:** <Ctrl>+<S>

With this command you save any changes in the project.

If the project to be saved is called "Untitled", then you must give it a name (see **"File" "Save as"**).

"File" "Save as"

With this command the current project can be saved in another file or as a library. This does not change the original project file.

After the command has been chosen the Save dialog box appears. Choose either an existing **File name** or enter a new file name and choose the desired **file type**.

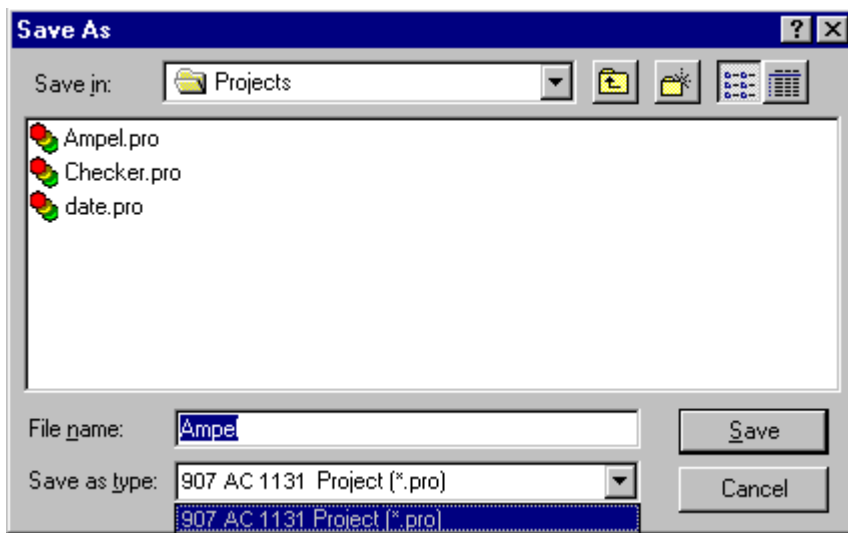


Image 4.18: Dialog box for Save as

If the project is to be saved under a new name, then choose the file type **907 AC 1131 Project (*.pro)**.

If you choose the file type **Project Version 1.5 (*.pro)** or **2.0**, then the current project is saved as if it were created with the version 1.5 or 2.0. Specific data of the version 2.1 can thereby be lost! However, the project can be executed with the version 1.5 or 2.0.

You can also save the current project as a library in order to use it in other projects. Choose the file type **Internal library (*.lib)** if you have programmed your POU's in **907 AC 1131** .

Choose the file type **External library (*.lib)** if you want to implement and integrate POU's in other languages (e.g. C). This means that another file is also saved which receives the file name of the library, but with the extension "*.h". This file is constructed as a C header file with the declarations of all POU's, data types, and global variables. If external libraries are used, in the simulation mode the implementation, written for the POU's in 907 AC 1131 , will be executed. Working with the real hardware the implementation written in C will be executed.

Then click **OK**. The current project is saved in the indicated file. If the new file name already exists, then you are asked if you want to overwrite this file.

When saving as a library, the entire project is compiled. If an error occurs thereby, then you are told that a correct project is necessary in order to create a library. The project is then not saved as a library.

"File" "Print"

Shortcut: <Ctrl>+<P>

With this command the content of the active window is printed.

After the command has been chosen, then the Print dialog box appears. Choose the desired option or configure the printer and then click **OK**. The active window is printed.

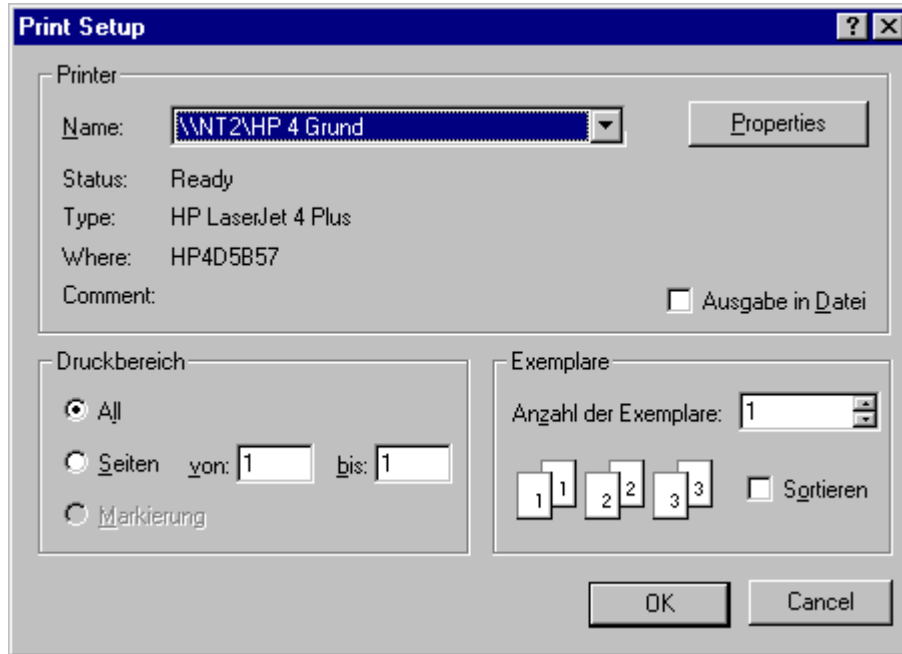


Image 4.19: Print dialog box

You can determine the **number of the copies** and print the version to a file.

With the button **Properties** you open the dialog box to set up the printer.

You can determine the layout of your printout with the command **"File" "Printer Setup"**.

During printing the dialog box shows you the number of pages already printed. When you close this dialog box, then the printing stops after the next page.

In order to document your entire project, use the command **"Project" "Document"**.

If you want to create a document frame for your project, then open a global variables list and use the command **"Extras" "Make Docuframe file"**.

"File" "Printer setup"

With this command you can determine the layout of the printed pages. The following dialog box is now opened:

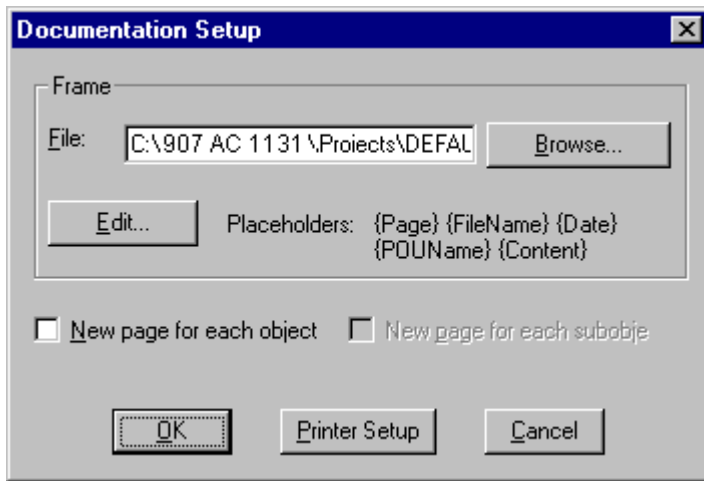


Image 4.20: Page Layout Dialog Box

In the field **File** you can enter the name of the file with the extension ".dfr" in which the page layout should be saved. The default destination for the settings is the file DEFAULT.DFR.

If you would like to change an existing layout, then browse through the directory tree to find the desired file with the button **Browse**

You can also choose whether to begin a **new page for each object** and **for each subobject**. Use the **Printer Setup** button to open the printer configuration.

If you click on the **Edit** button, then the frame for setting up the page layout appears. Here you can determine the page numbers, date, filename and POU name, and also place graphics on the page and the text area in which the documentation should be printed.

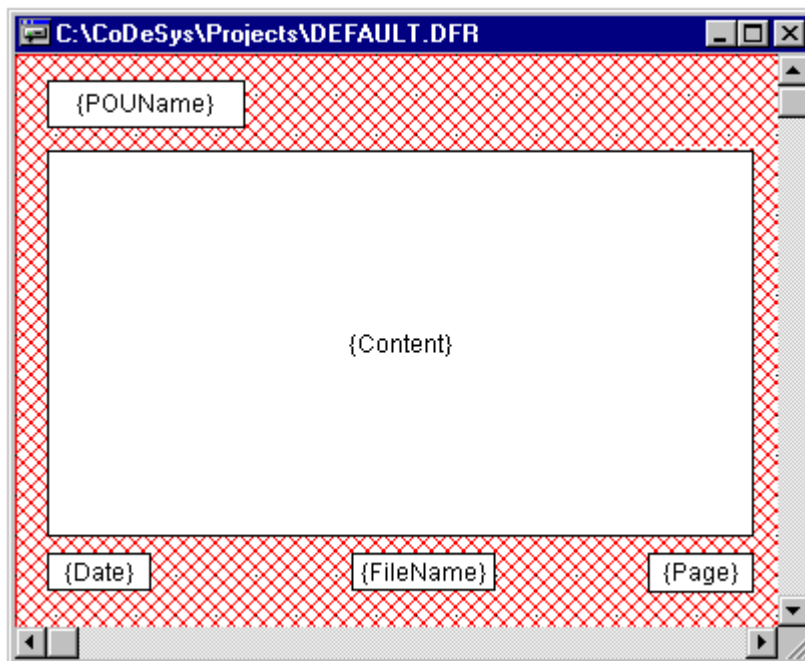


Image 4.21: Window for pasting the placeholders on the page layout

With the menu item "**Insert**" "**Placeholder**" and subsequent selection among the five placeholders (**Page**, **POU name**, **File name**, **Date**, and **Content**), insert into the layout a so-called placeholder by dragging a rectangle² on the layout while pressing the left mouse button. In the printout they are replaced as follows:

Command	Placeholder	Effect
Page	{Page}	Here the current page number appears in the printout.
POU name	{POU Name}	Here the current name of the POU appears.
File name	{File Name}	Here the name of the project appears.
Date	{Date}	Here the current date appears.
Contents	{Contents}	Here the contents of the POU appear.

In addition, with "**Insert**" "**Bitmap**" you can insert a bitmap graphic (e.g. a company logo) in the page. After selecting the graphic, a rectangle should also be drawn here on the layout using the mouse. Other visualization elements can be inserted (see chapter 'Visualizations').

If the template was changed, then **907 AC 1131** asks when the window is closed if these changes should be saved or not.

"File" "Exit"

Shortcut: <Alt>+<F4>

With this command you exit from **907 AC 1131** .

If a project is opened, then it is closed as described in "**File**" "**Save**".

"Project" "Check"

With this command you can check the static correctness of your program. If an error crops up, then it is announced in the message window, as with the building of the program.

In contrast to the command "**Rebuild all**", no code is created

"Project" "Build"

With this command all changed POU's are built. When loading the program, only the modified POU's are sent to the PLC. The rest of the program remains unchanged in the PLC.



Note: The command "**Build**" is only supported if **907 AC 1131** is equipped with the Online Change function. Otherwise the command "**Build**" acts like "**Rebuild all**".

With larger changes use the function "**Project**" "**Register changes**".

² Drawing a rectangle on the layout by dragging the mouse diagonally while pressing the left mouse button.

Online Change function means that parts of a program can be exchanged (sent to the PLC) without interrupting the PLC. All data is retained as far as possible.



Warning: If you select "**Build**" two times in a row without sending the program to the PLC in the interim, then the error message appears: "Modifications not compatible with each other".

Following this the program must be rebuilt ("**Rebuild all**") and sent in full to the PLC!

Online Change

Online Change function means that parts of a program can be exchanged (sent to the PLC) without interrupting the PLC. All data is retained as far as possible.

"Project" "Rebuild all"

With this command you rebuild all POU's. The message window is opened which shows the progress of the building process and any errors which may be discovered.

A list of all error messages is to be found in the appendix.

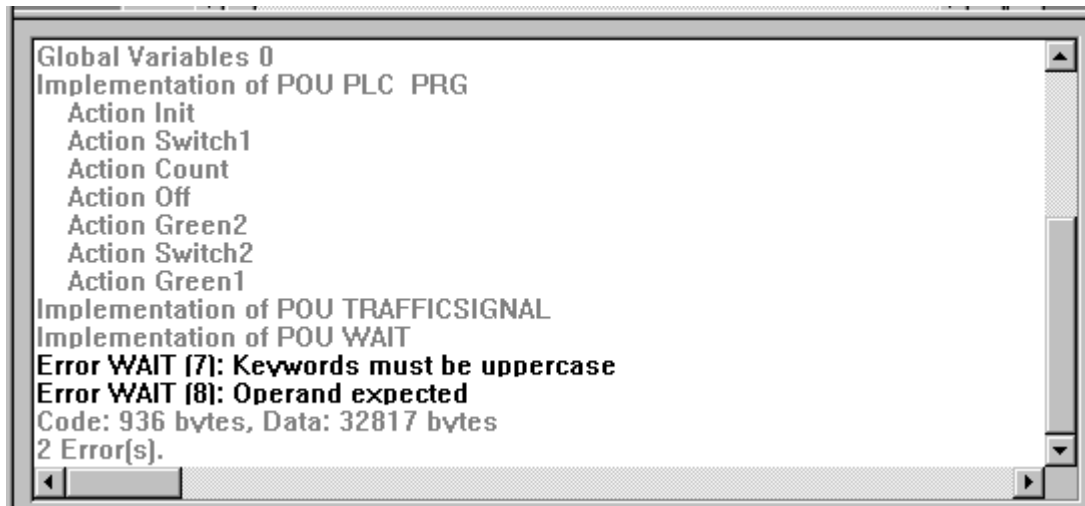


Image 4.22: Message window of a project with three POU's and two error messages

With the command "**Online**" "**Login**" the command "**Rebuild all**" is automatically executed if the project has been modified since the last compilation.

If the option **Save before compile** in the Options dialog box in the category Build has been chosen, then the project is backed up before compilation.



Note: The cross references are created during compilation and are not saved in the project! In order to use the commands "**Show call tree**", "**Show**

cross reference list, and "**Show unused variables**" the project must be rebuilt after loading and after a modification.

"Project" "Document"

This command lets you print the documentation of your entire project. The elements of a complete documentation are:

- The POU's,
- the contents of the documentation,
- the data types,
- the visualizations
- the resources (Access variables, global variables, variables configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, the Watch and Receipt Manager)
- the call trees of POU's and data types, as well as
- the cross reference list.

For the last two items the project must have been built without errors.

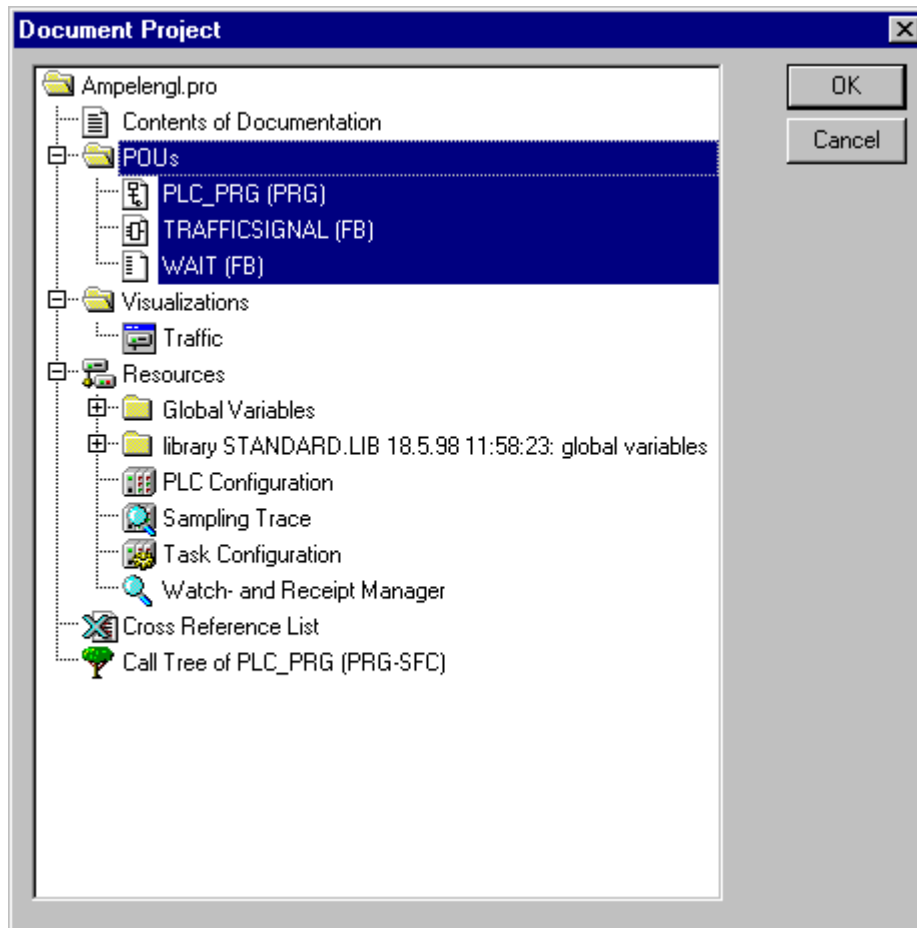


Image 4.23: Dialog box for project documentation

Only those areas in the dialog box are printed which are highlighted in blue.

If you want to select the entire project, then select the name of your project in the first line.

If, on the other hand, you only want to select a single object, then click on the corresponding object or move the dotted rectangle onto the desired object with the arrow key. Objects which have a plus sign in front of their symbols are organization objects which contain other objects. With a click on a plus sign organization object is expanded, and with a click on the resulting minus sign it can be closed up again. When you select an organization object, then all relevant objects are also selected. By pressing the <Shift> key you can select a group of objects, and by pressing the <Ctrl> key you can select several individual objects.

Once you have made your selection, then click on **OK**. The Print dialog box appears. You can determine the layout of the pages to be printed with "**File**" "**Printer setup**".

"Project" "Export"

With **907 AC 1131** projects can be exported or imported. That allows you to exchange programs between different IEC programming systems.

There is a standardized exchange format for POU's in IL, ST, and SFC (the Common Elements format of IEC 1131-3). For the POU's in LD and FBD and the other objects **907 AC 1131** has its own filing format since there is no text format for this in IEC 1131-3. The selected objects are written to an ASCII file.

POU's, data types, visualizations, and the resources can be exported.

Once you have made your selection in the dialog box window (the same way as with "**Project**" "**Document**"), you can decide, whether you want to export the selected parts to one file or to export in separate files, one for each object. Switch on or off the option **One file for each object** then click on **OK**. The dialog box for saving files appears. Enter a file name with the expansion ".exp" respectively a directory for the object export files, which then will be saved there with the file name <objectname.exp>.

"Project" "Import"

In the resulting dialog box for opening files select the desired export file.

The data is imported into the current project. If an object with the same name already exists in the same project, then a dialog box appears with the question "Do you want to replace it?": If you answer **Yes**, then the object in the project is replaced by the object from the import file. If you answer **No**, then the name of the new objects receives as a supplement an underline and a digit ("_0", "_1", ..). With **Yes, all** or **No, all** this is carried out for all objects.

In the message window the import is registered.

With this command you can compare the open project with another. For example, if you prepare to save and want to know where you have made changes in the current project, then you can compare the open project with the last saved version of it.

After you have given this command, then the dialog box for opening files appears. Choose the project with which you want to compare the current project. If you press **OK**, then you will see the result of the comparison in the message window. All of the objects of the chosen project are listed and the changes in the object are shown afterward in parentheses. There are five possible messages:

- "Unchanged": The object was not changed.
- "Deleted": The object is no longer present in the current project.
- "Implementation changed": The instruction part of the POU has been changed.
- "Interface changed": The declaration part of the object has been changed.
- "Interface and implementation changed": Both the instruction part and the declaration part of the POU have been changed.

A doubleclick on a message selects the first change in this object.

"Project" "Merge"

With this command you can merge objects (POUs, data types, visualizations, and resources) from other projects into your project.

When the command has been given, first the standard dialog box for opening files appears. When you have chosen a file there, a dialog box appears in which you can choose the desired object. The selection takes place as described with **"Project" "Document"** .

If an object with the same name already exists in the project, then the name of the new object receives the addition of an underline and a digit ("_1", "_2" ...).

"Project" "Project info"

Under this menu item the information about your project can be saved. When the command has been given, then the following dialog box opens:

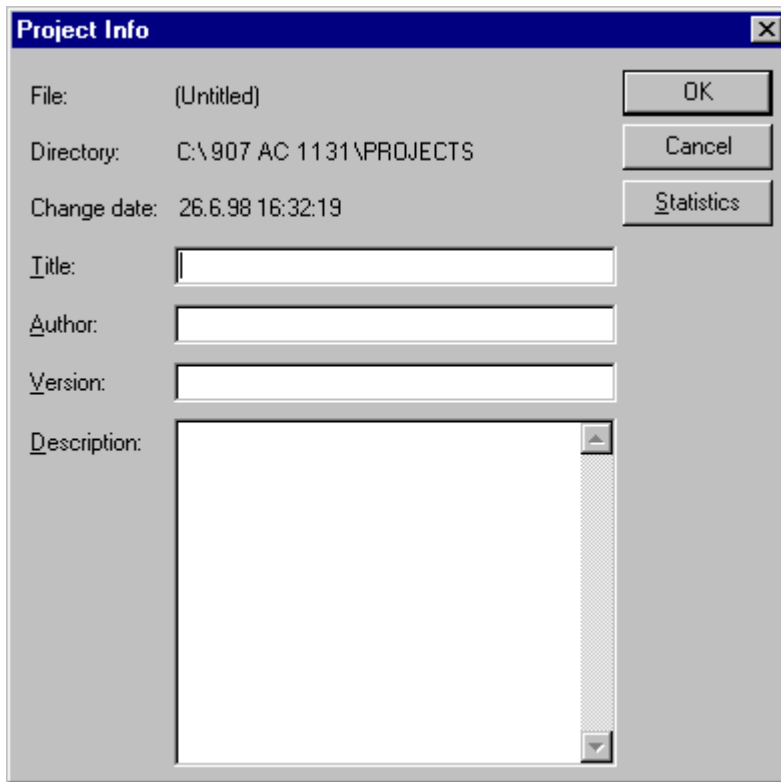


Image 4.24: Dialog box for entering project information

The following project information is displayed:

- **File** name
- **Directory** path
- The time of the most recent change (**Change date**)

This information can not be changed.

In addition, you can add the following information:

- A **Title** of the project,
- the name of the **Author**,
- the **Version** number, and
- a **Description** of the project.

This information is optional. When you press the button **Statistics** you receive statistical information about the project.

It contains information such as the number of the POU's, data types, and the local and global variables as they were traced at the last compilation.

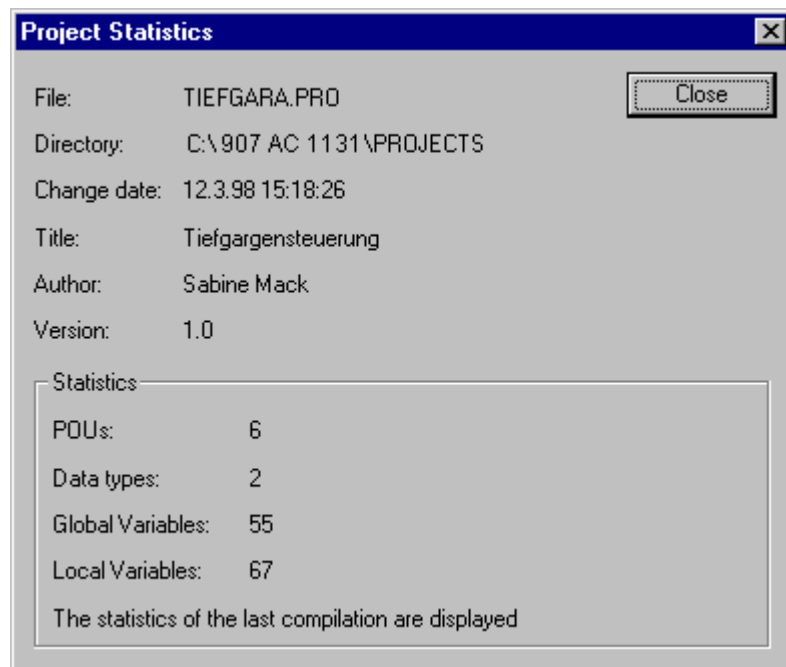


Image 4.25: Example of project statistics

If you choose the option **Ask for project info** in the category **Load & Save** in the Options dialog box, then while saving a new project, or while saving a project under a new name, the project info is called automatically.

"Project" "Global Search"

With this command you can search for the location of a text in POUs, data types, or in the objects of the global variables.

When the command is entered, a dialog box opens in which you can choose the desired object. The selection is made as in the **"Project" "Document"** description.

When you have confirmed the selection with **OK**, then the search dialog box appears. If a text in an object has been found, then the object is loaded into the appropriate editor, and its location is shown.

"Project" "Global replace"

With this command you can search for the location of a text in POUs, data types, or the objects of the global variables and replace this text by another. This is executed in the same way as with **"Project" "Global Search"** or **"Edit" "Replace"**.

"Project" "Register changes"

This command is necessary when significant changes have to be made to a project without interrupting the PLC. (Online Change).

Copy your project, make your changes, and test your changes. Select the command **"Project" "Compare"** in order to compare the two projects. With the command **"Register changes"** all differences between the current project and the comparison are traced. Then, with the command **"Build"** the modified

POUs can be compiled. When downloading the program only the modified POU's are sent to the PLC. The rest of the program remains unchanged in the PLC.

User groups

In **907 AC 1131** up to eight user groups with different access rights to the POU's, data types, visualizations, and resources can be set up. Access rights for single objects or all of them can be established. Only a member of a certain user group can open a project. A member of such a user group must identify himself by means of a password.

The user groups are numbered from 0 to 7, whereby the Group 0 has the administrator rights, i.e. only members of group 0 may determine passwords and access rights for all groups and/or objects.

When a new project is launched, then all passwords are initially empty. Until a password has been set for the 0 group, one enters the project automatically as a member of the 0 group.

If a password for the user group 0 is existing while the project is loaded, then a password will be demanded for all groups when the project is opened. For this the following dialog box appears:

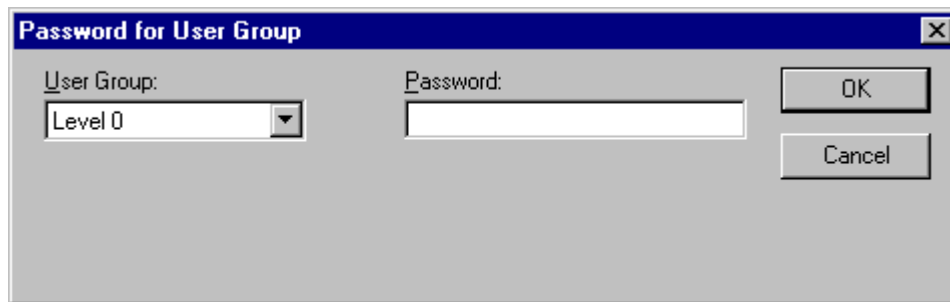


Image 4.26: Dialog box for password entry

In the combobox **User group** on the left side of the dialog box, enter the group to which you belong and enter on the right side the relevant **password**. Press **OK**. If the password does not agree with the saved password, then the message appears:

"The password is not correct."

Only when you have entered the correct password can the project be opened.

With the command "**Passwords for user group**" you can assign the passwords, and with "**Object**" "**Access rights**" you can define the rights for single objects or for all of them.

"Project" "Passwords for user groups"

With this command you open the dialog box for password assignment for user groups. This command can only be executed by members of group 0. When the command has been given, then the following dialog box appears:

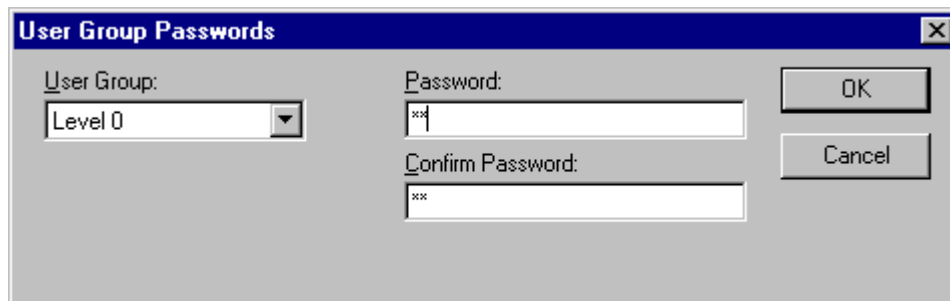


Image 4.27: Dialog box for password assignment

In the left combobox **User group** you can select the group. Enter the desired password for the group in the field **Password**. For each typed character an asterisk (*) appears in the field. You must repeat the same password in the field **Confirm password**. Close the dialog box after each password entry with **OK**. If you get the message:

"The password does not agree with the confirmation",

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

Then, if necessary, assign a password for the next group by calling the command again.

With the command "**Object**" "**Access rights**" you can assign the rights for single objects or all of them.

4.4 Creating and Deleting Objects, etc.

Now we shall explain how to work with objects and what help is available to keep track of a project (Folders, Call tree, Cross reference list,..).


Object

POUs, data types, visualizations and the resources (Access variables, global variables, the variable configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, and the Watch and Receipt Manager are all defined as "objects". The folders inserted for structuring the project are partially involved. All objects of a project are in the Object Organizer.

If you hold the mouse pointer for a short time on a POU in the Object Organizer, then the type of the POU (Program, Function or Function block) is shown in a Tooltip. For the global variables the tooltip shows the keyword (VAR_ACCESS, VAR_GLOBAL, VAR_CONFIG).

Folder

In order to keep track of larger projects you should group your POUs, data types, visualizations, and global variables systematically in folders.

You can set up as many levels of folders as you want. If a plus sign is in front of a closed folder symbol,  then this folder contains objects and/or additional

folders. With a click on the plus sign the folder is opened and the subordinated objects appear. With a click on the minus (which has replaced the plus sign) the folder can be closed again. In the context menu you find the commands "**Expand nodes**" and "**Collapse nodes**" with the same functions.

With Drag&Drop you can move the objects as well as the folders within their object type. For this select the object and drag it with pressed left mouse button to the desired position.



Note: Folders have no influence on the program, but rather serve only to structure your project clearly.

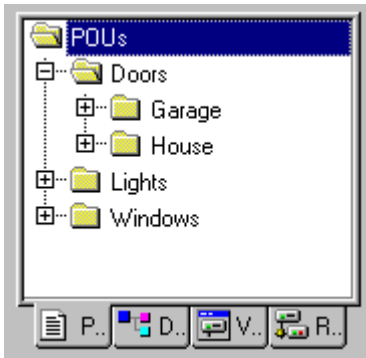


Image 4.28: Example of folders in the Object Organizer

"New Folder"

With this command a new folder is inserted as a structural object. If a folder has been selected, then the new one is created underneath it. Otherwise it is created on the same level.

The context menu of the Object Organizer which contains this command appears when an object or the object type has been selected and you have pressed the right mouse button or <Shift>+<F10>.

"Expand nodes" "Collapse nodes"

With the command expand the objects are visibly unfolded which are located in the selected object. With Collapse the subordinated objects are no longer shown.

With folders you can open or close them with a double mouse click or by pressing <Enter>.

The context menu of the Object Organizer which contains this command appears when an object or the object type has been selected and you have pressed the right mouse button or <Shift>+<F10>.

"Project" "Object Delete"

Shortcut: <Delete>

With this command the currently selected object (a POU, a data type, a visualization, or global variables), or a folder with the subordinated objects is removed from the Object Organizer and is thus deleted from the project.

For safety you are asked once more for confirmation.

If the editor window of the object was open, then it is automatically closed.

If you delete with the command "**Edit**" "**Cut**", then the object is parked on the clipboard.

"Project" "Object Add"

Shortcut: <Insert>

With this command you create a new object. The type of the object (POU, data type, visualization, or global variables) depends upon the selected register card in the Object Organizer. Enter the name of the new object in the dialog box which appears. Remember that the name of the object may not have already been used.

If you are dealing with a POU, then you must also choose the type of the POU (Program, Function, or Function block) and the language in which it is to be programmed.

After confirmation of the entry, then the suitable entry window for the object appears.

Image 4.29: Dialog box for creating a new POU

If, on the other hand, you use the command "**Edit**" "**Paste**", then the object is pasted from the clipboard, and no dialog box appears.

"Project" "Object Rename"

Shortcut: <Spacebar>

With this command you give a new name to the currently-selected object or folder. Remember that the name of the object may not have already been used.

If the editing window of the object is open, then its title is changed automatically when the name is changed.

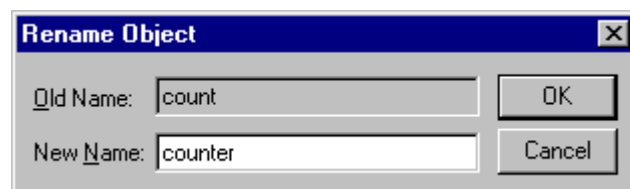


Image 4.30: Dialog box for renaming a POU

"Project" "Object Convert"

This command can only be used with POUs. You can convert POUs from the languages SFC, ST, FBD, LD, and IL into one of the three languages IL, FBD, and LD.

For this the project must be compiled. Choose the language into which you want to convert and give the POU a new name. Remember that the name of the POU may not have already been used. Then press **OK**, and the new POU is added to your POU list.

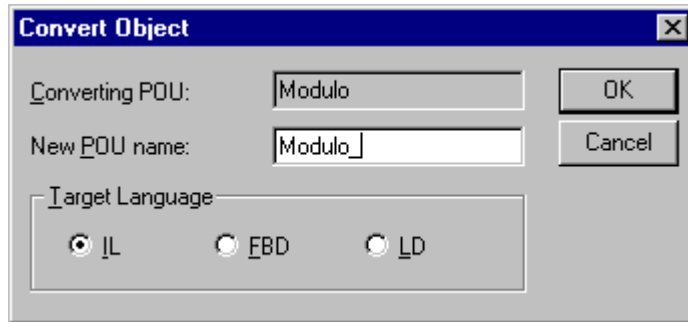


Image 4.31: Dialog box for converting a POU

"Project" "Object Copy"

With this command a selected object is copied and saved under a new name. Enter the name of the new object in the resulting dialog box. Remember that the name of the object may not have already been used.

If, on the other hand, you used the command "**Edit**" "**Copy**", then the object is parked on the clipboard, and no dialog box appears.

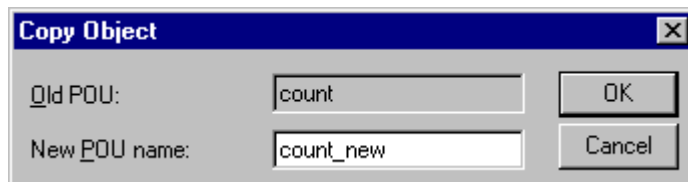


Image 4.32: Dialog box for copying a POU

"Project" "Object Open"

Shortcut: <Enter>

With the command you load a selected object within the Object Organizer into the respective editor. If a window with this object is already open, then it gets a focus, is moved into the foreground and can now be edited.

There are two other ways of opening an object:

- Doubleclick with the mouse on the desired object
- type in the Object Organizer the first letter of the object name. Then a dialog box opens in which all objects of the available object types with this initial letter are shown. Select the desired object and click on the button

Open in order to load the object in its edit window. This option is supported with the object type Resources only for global variables. This last possibility is especially useful in projects with many objects.

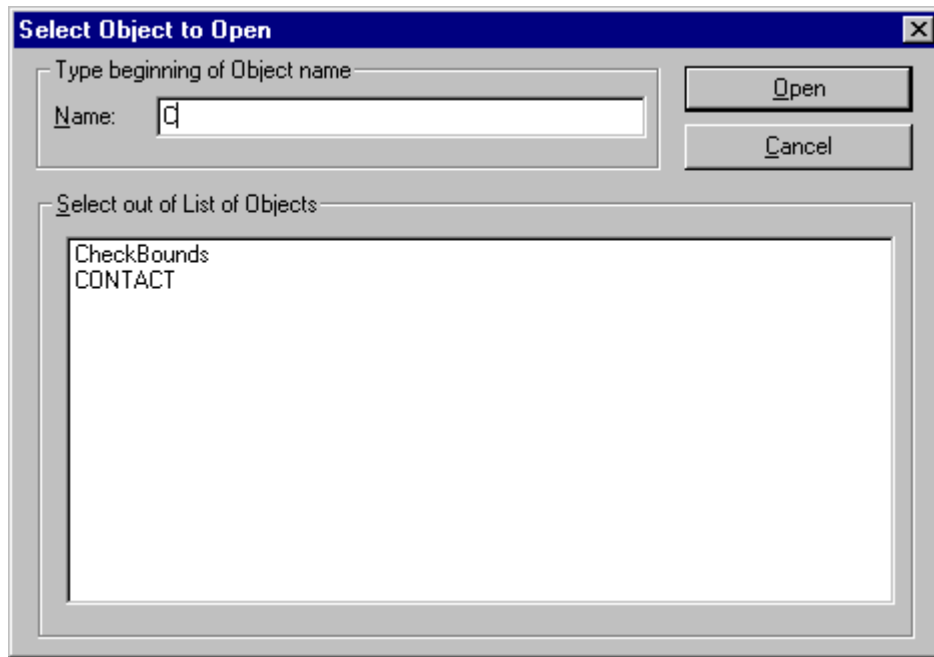


Image 4.33: Dialog box for choosing the object to be opened

"Project" "Object Access rights"

With this command you open the dialog box for assigning access rights to the different user groups. The following dialog box appears:

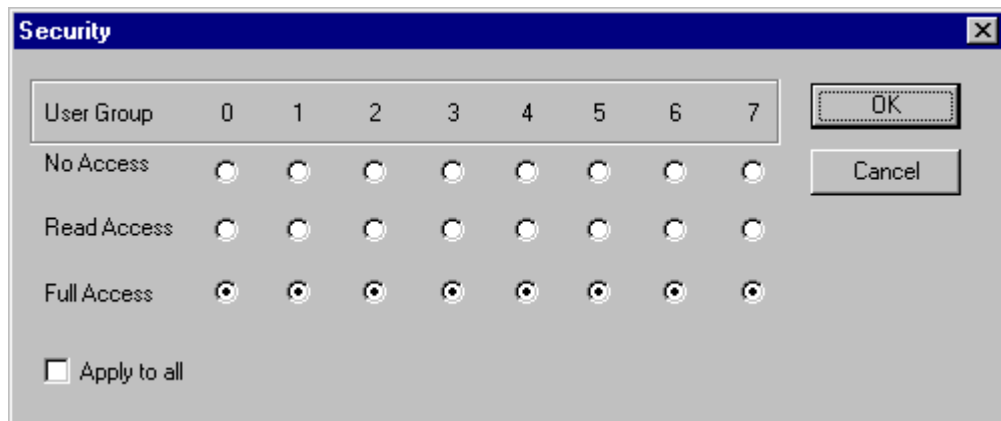


Image 4.34: Dialog box for assigning access rights

Members of the user group 0 can now assign individual access rights for each user group. There are three possible settings:

- **No Access:** the object may not be opened by a member of the user group.
- **Read Access:** the object can be opened for reading by a member of the user group but not changed.
- **Full Access:** the object may be opened and changed by a member of the user group.

The settings refer either to the currently-selected object in the Object Organizer or, if the option **Apply to all** is chosen, to all POUs, data types, visualizations, and resources of the project.

The assignment to a user group takes place when opening the project through a password request if a password was assigned to the user group 0.

'Project' 'Add Action'

This command is used to generate an action allocated to a selected block in the Object Organizer. One selects the name of the action in the dialog which appears and also the language in which the action should be implemented.

The new action is placed under your block in the Object Organizer. A plus sign appears in front of the block. A simple mouse click on the plus sign causes the action objects to appear and a minus sign appears in front of the block. Renewed clicking on the minus sign causes the actions to disappear and the plus sign appears again. This can also be achieved over the context menu commands **'Expand Node'** and **'Collapse Node'**.

"Project" "View instance"

With this command you can open and show single instances of function blocks. The function block whose instance should be open must first be selected in the Object Organizer before you can execute this command. In the resulting dialog box you can choose the desired instance of this function block.



Note: Instances can be opened only after logging in! (Project was correctly compiled and sent with **"Online"** **"Login"** to the PLC).

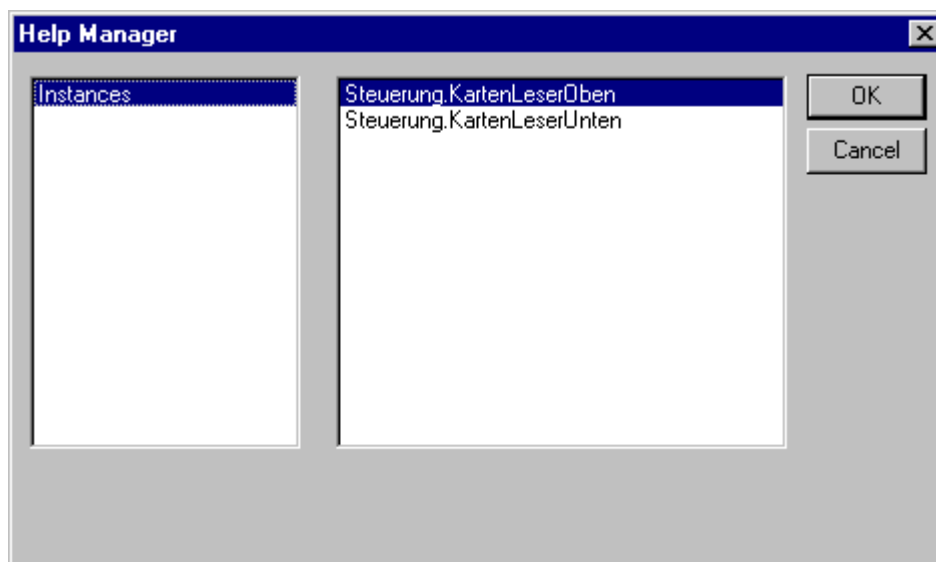


Image 4.35: Dialog box for opening an instance

"Project" "Show call tree"

With this command you open a window which shows the call tree of the object chosen in the Object Organizer. For this the project must be compiled (see

"**Rebuild all**"). The call tree contains both calls for POU's and references to data types.

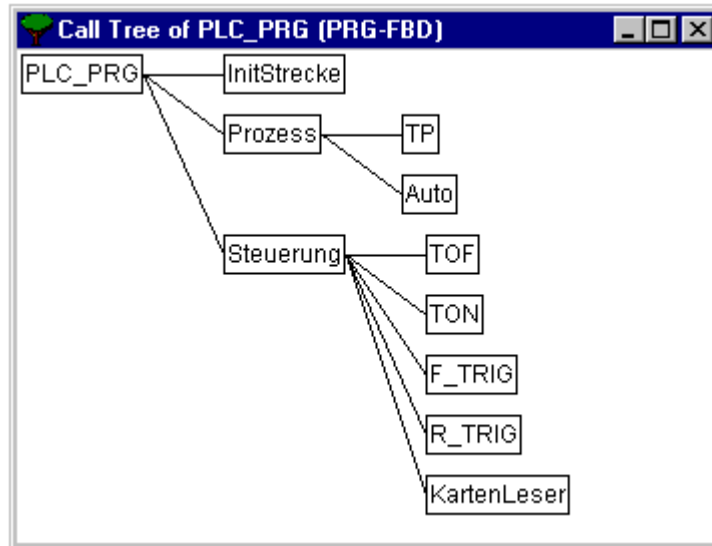


Image 4.36: Example of a call tree

"Project" "Show cross reference list"

With this command you open a dialog box which makes possible the output of all application points for a variable, address, or a POU. For this the project must be compiled (see "**Rebuild all**").

Choose first the category **Variable**, **Address**, or **POU** and then enter the name of the desired element. By clicking on the button **Get References** you get the list of all application points. Along with the POU and the line or network number, it is shown whether this point has read only access or full access, whether it is a local or global variable and whether the variable is connected to an address.

When you select a line of the cross reference list and press the button **Go To** or doubleclick on the line, then the POU is shown in its editor at the corresponding point. In this way you can jump to all application points without a time-consuming search.

In order to make processing easier, you can use the **Send to message window** button to bring the current cross reference list into the message window and from there change to the respective POU.

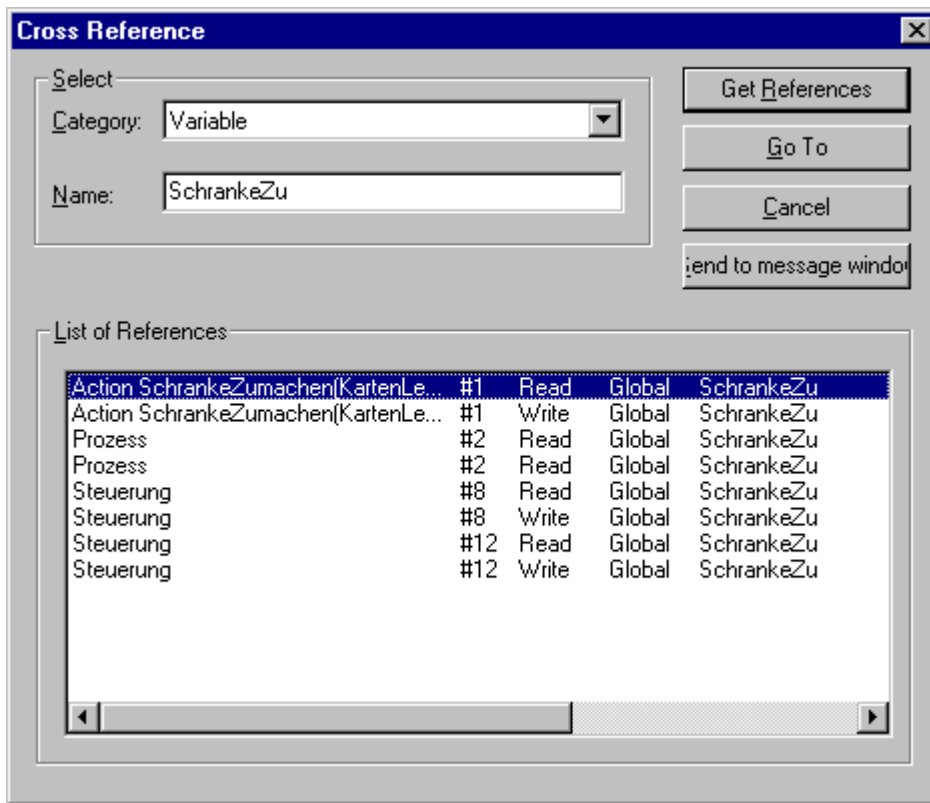


Image 4.37: Dialog box and example of a cross reference list

"Project" "Show unused variables"

With this command a list of variables is shown which, to be sure, have been declared in the project but are not being used anywhere. For this the project must be compiled (see "**Rebuild all**").

If there are no unused variables in your project, then that is announced. Otherwise the following window appears:

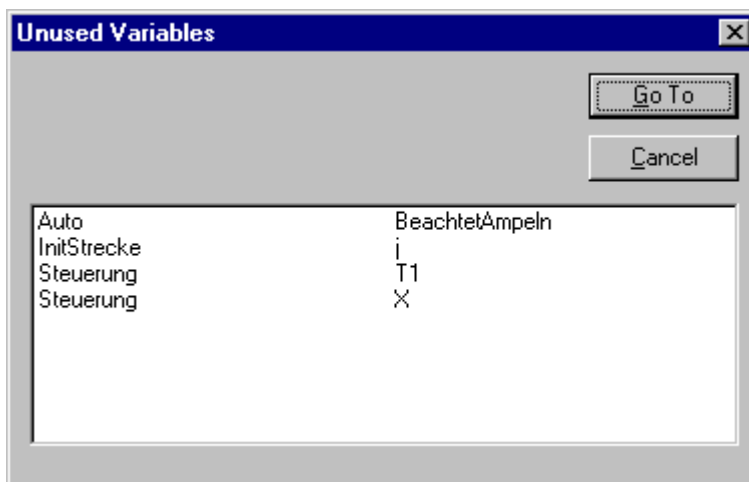


Image 4.38: Unused variables of a project

If you select a variable and press the button **Go To** or doubleclick on the variable, then you change to the respective object in which the variable has been declared.

"Extras"

Previous version

With this command you can restore the current object to the last saved state. The restored state is either that of the most recent manual save ("**File**" "**Save**") or that which was kept after the automatic save, depending upon which version is most recent.

4.5 General Editing Functions

You can use the following commands in all editors and some of them in the Object Organizer. All of the commands are located under the menu item "**Edit**".

"Edit" "Undo"

Shortcut: <Ctrl>+<Z>

This command undoes the action which was most recently executed in the currently-open editor window or in the Object Organizer.

By repeatedly selecting this command, all actions can be undone back to the point at which the window was opened. This applies to all actions in the editors for POU's, data types, visualizations, and global variables and in the Object Organizer.

With "**Edit**" "**Redo**" you can restore an action which you have undone.



Note: The commands "**Undo**" and "**Redo**" apply to the current window. Each window carries its own action list. If you want to undo actions in several windows, then you must activate the corresponding window. When undoing or redoing in the Object Organizer the focus must lie here.

"Edit" "Redo"

Shortcut: <Ctrl>+<Y>

With the command in the currently-open editor window or in the Object Organizer you can restore an action you have undone ("**Edit**" "**Undo**").

As often as you have previously executed the command "**Undo**", you can also carry out the command "**Redo**".



Note: The commands "**Undo**" and "**Redo**" apply to the current window. Each window carries its own action list. If you want to undo actions in several

windows, then you must activate the corresponding window. When undoing or redoing in the Object Manager must lie there.

"Edit" "Cut"

Symbol: 

Shortcut: <Ctrl>+<X> or <Shift>+<Delete>

This command transfers the current selection from the editor to the clipboard. The selection is removed from the editor.

In the Object Organizer this similarly applies to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration.

Remember that not all editors support the cut command, and that its use can be limited in some editors.

The form of the selection depends upon the respective editor:

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle.

In order to paste the content of the clipboard you use the command **"Edit" "Paste"**. In the SFC editor you can also use the commands **"Extras" "Insert parallel branch (right)"** or **"Extras" "Paste after"**.

In order to copy a selection onto the clipboard without deleting it, use the command **"Edit" "Copy"**.

In order to remove a selected area without changing the clipboard, use the command **"Edit" "Delete"**.

"Edit" "Copy"

Symbol:  **Shortcut:** <Ctrl>+<C>

This command copies the current selection from the editor to the clipboard. This does not change the contents of the editor window.

With the Object Organizer this similarly applies to the selected object, whereby not all objects can be copied, e.g. the PLC Configuration.

Remember that not all editors support copying and that it can be limited with some editors.

For the type of selection the same rules apply as with **"Edit" "Cut"**.

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor you can also use the commands **"Extras" "Insert parallel branch (right)"** or **"Extras" "Paste after"**.

"Edit" "Paste"

Symbol:  **Shortcut: <Ctrl>+<V>**

Pastes the content of the clipboard onto the current position in the editor window. In the graphic editors the command can only be executed when a correct structure results from the insertion.

With the Object Organizer the object is pasted from the clipboard.

Remember that pasting is not supported by all editors and that its use can be limited in some editors.

The current position can be defined differently according to the type of editor:

With the text editors (IL, ST, Declarations) the current position is that of the blinking cursor (a vertical line) which you place by clicking with the mouse).

In the FBD and LD editors the current position is the first network with a dotted rectangle in the network number area. The contents of the clipboard are inserted in front of this network. If a partial structure has been copied, then it is inserted in front of the selected element.

In the SFC editor the current position is determined the selection which is surrounded by a dotted rectangle. Depending upon the selection and the contents of the clipboard, these contents are inserted either in front of the selection or into a new branch (parallel or alternative) to the left of the selection.

In SFC the commands **"Extras" "Insert parallel branch (right)"** or **"Extras" "Paste after"** can be used in order to insert the contents of the clipboard.

"Edit" "Delete"

Shortcut: <Delete>

Deletes the selected area from the editor window. This does not change the contents of the clipboard.

In the Object Organizer this applies likewise to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration.

For the type of selection the same rules apply as with **"Edit" "Cut"**.

In the library manager the selection is the currently selected library name.

"Edit" "Find"

Symbol: 

With this command you search for a certain text passage in the current editor window. The Find dialog box opens. It remains open until the button **Cancel** is pressed.

In the field **Find what** you can enter the series of characters you are looking for.

In addition, you can decide whether the text you are looking for **Match whole word only** or not, or also whether **Match case** is to be considered, and whether the search should proceed **Up** or **Down** starting from the current cursor position.

The button **Find next** starts the search which begins at the selected position and continues in the chosen search direction. If the text passage is found, then it is highlighted. If the passage is not found, then a message announces this. The search can be repeated several times in succession until the beginning or the end of the contents of the editor window has been reached.

Remember that the found text can be covered up by the Find dialog box.

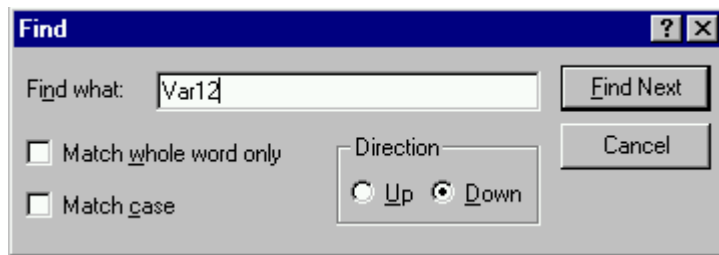


Image 4.39: Find dialog box

"Edit" "Find next"

Symbol:  **Shortcut:** <F3>

With this command you execute a search with the same parameters as with the most recent action "**Edit**" "**Find**".

"Edit" "Replace"

With this command you search for a certain passage just as with the command "**Edit**" "**Find**", and replace it with another. After you have chosen the command the dialog box for find and replace appears. This dialog box remains open until the button **Cancel** or **Close** is pressed.

The button **Replace** replaces the current selection with the text in the field **Replace with**.

The button **Replace all** replaces every occurrence of the text in the field **Find next** after the current position with the text in the field **Replace with**. At the end of the procedure a message announces how many replacements were made.

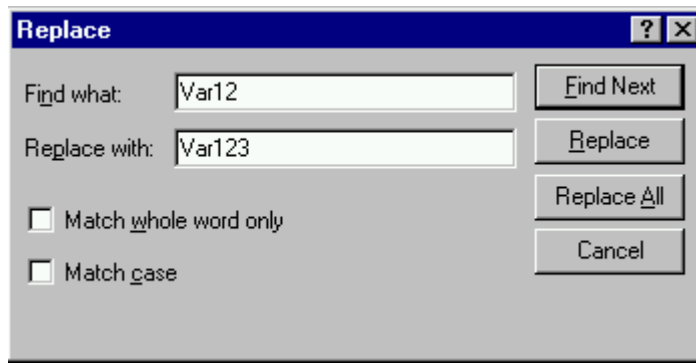


Image 4.40: Dialog box for find and replace

"Edit" "Input Assistant"

Shortcut: <F2>

This command provides a dialog box for choosing possible inputs at the current cursor position in the editor window. In the left column choose the desired input category, select the desired entry in the right column, and confirm your choice with **OK**. This inserts your choice at this position.

The categories offered depend upon the current cursor position in the editor window, i.e. upon that which can be entered at this point (e.g. variables, operators, POUs, conversions, ...).

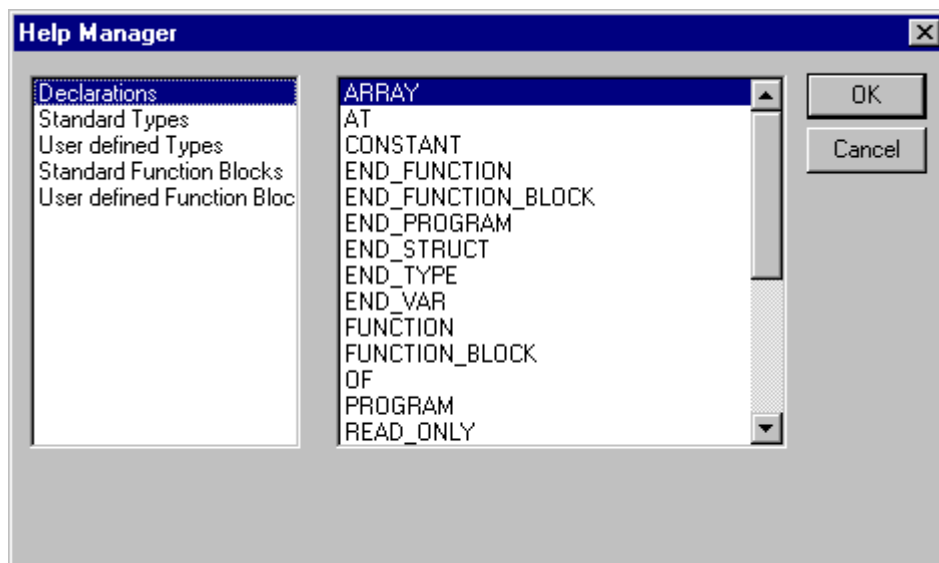


Image 4.41: Input Assistant Dialog Box

In some positions (e.g. in the watch list) multilevel variable names are necessary. At first the Input Assistant dialog box contains a list of all POUs along with a single point for the global variables. After each POU name there is a point. Doubleclicking with the mouse or pressing <Enter> opens a list of the variables for a selected POU. Instances and data types can, when appropriate, be opened again. By pressing **OK** you accept the selected variable.

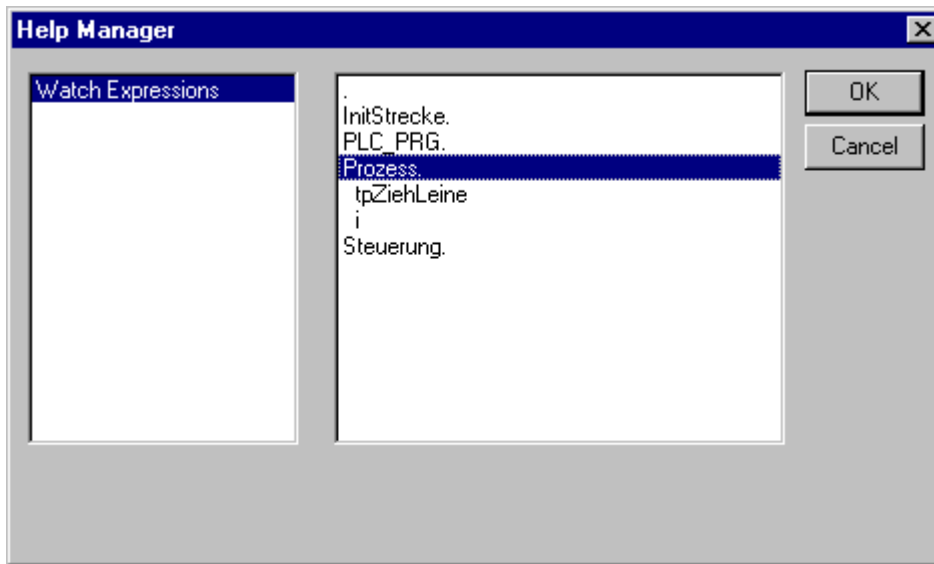


Image 4.42: Input Assistant dialog box with multilevel variable names



Note: Some entries (e.g. global variables) are only brought up to date in the Input Assistant after a compilation run.

'Edit"Declare Variable'

Kurzform: <Shift>+<F2>

This command opens the dialog for the declaration of a variable. This dialog also opens automatically when the option 'Project' 'Options' 'Editor' 'Autodeclaration' is switched on and when a new undefined variable is used the declaration editor.

"Edit" "Next error"

Shortcut: <F4>

After the incorrect compilation of a project this command can show the next error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown.

"Edit" "Previous error"

Shortcut: <Shift>+<F4>

After the incorrect compilation of a project this command shows the previous error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown.

4.6 General Online Functions

The available online commands are assembled under the menu item **"Online"**. The execution of some of the commands depends upon the active editor.

The online commands become available only after logging in.

"Online" "Login"

Symbol:  **Kurzform:** <Alt>+<F8>

This command combines the programming system with the PLC (or starts the simulation program) and changes into the online mode.

If the current project has not been compiled since opening or since the last modification, then it is compiled now (as with **"Project" "Rebuild all"**). If errors occur during compilation, then **907 AC 1131** does not change into Online mode.

After a successful login all online functions are available (if the corresponding settings in "Options" category Build have been entered). The current values are monitored for all visible variable declarations.

Use the **"Online" "Logout"** command to change from online back to offline mode.

If the system reports

Error:

"A connection to the PLC could not be established"

Verify whether the parameters selected in the **"Online" "Communication Parameters"** agree with those of your PLC.

You should especially verify that the interface number is correct. (If you have set it to COM1, the cable should also be physically plugged into COM1.) Furthermore, verify that the baud rates in the PLC and the Program System agree with each other. (Default setting in 907 AC 1131 : 9600 Bd).

Error:

"The program has been modified! Should the new program be loaded?"

The project which is open in the editor is incompatible with the program currently found in the PLC (or with the Simulation Mode program being run). Monitoring and debugging is therefore not possible. You can either choose "No," logout, and open the right project, or use "Yes" to load the current project in the PLC.

"Online" "Logout"

Symbol:  **Kurzform** <Strg>+<F8>

The connection to the PLC is broken, or, the Simulation Mode program is ended and is shifted to the offline mode.

Use the "**Online**" "**Login**" command to change to the online mode.

"Online" "Download"

This command loads the compiled project in the PLC.

If you use C-Code generation, then prior to the download, the C-Compiler is called up, which creates the download file. If this is not the case, then the download file is created during the compiling.

"Online" "Run"

Symbol:  **Shortcut:** <F5>

This command starts the program in the PLC or in Simulation Mode.

This command can be executed immediately after the "**Online**" "**Download**" command, or after the user program in the PLC has been ended with the "**Online**" "**Stop**" command, or when the user program is at a break point, or when a Single Cycle has been performed.

"Online" "Stop"

Symbol:  **Kurzform** <Umschalt>+<F8>

Stops the execution of the program in the PLC or in Simulation Mode between two cycles.

Use the "**Online**" "**Run**" command to restart the program.

"Online" "Reset"

If you have initialized the variables with a specific value, then this command will reset the variables to the initialized value. All other variables are set at a standard initialization (for example, integers at 0). As a precautionary measure, **907 AC 1131** asks you to confirm your decision before all of the variables are overwritten.

Use the "**Online**" "**Run**" command to restart the program.

"Online" "Toggle Breakpoint"

Symbol:  **Shortcut:** <F9>

This command sets a breakpoint in the present position in the active window. If a breakpoint has already been set in the present position, that breakpoint will be removed.

The position at which a breakpoint can be set depends on the language in which the POU in the active window is written.

In the Text Editors (IL, ST), the breakpoint is set at the line where the cursor is located, if this line is a breakpoint position (recognizable by the dark-gray color

of the line number field). You can also click on the line number field to set or remove a breakpoint in the text editors.

In FBD and LD, the breakpoint is set at the currently selected network. In order to set or remove a breakpoint in the FBD or LD Editor, you can also click on the network number field.

In SFC, the breakpoint is set at the currently selected step. In SFC you can also use <Shift> with a doubleclick to set or remove a breakpoint.

If a breakpoint has been set, then the line number field or the network number field or the step will be displayed with a light-blue background color.

If a breakpoint is reached while the program is running, the program will stop, and the corresponding field will be displayed in a red background color. In order to continue the program, use the "Online" "Run", "Online" "Step in", or "Online" "Step Over" commands.

Online "Breakpoint Dialog Box"

This command opens a dialog box to edit breakpoints throughout the entire project. The dialog box also displays all breakpoints presently set.

In order to set a breakpoint, choose a POU in the **POU** combobox and the line or the network in the **Location** combobox where you would like to set the breakpoint; then press the **Add** button. The breakpoint will be added to the list.

In order to delete a breakpoint, highlight the breakpoint to be deleted from the list of the set breakpoints and press the **Delete** button.

The **Delete All** button can be used to delete all the breakpoints.

In order to go to the location in the editor where a certain breakpoint was set, highlight the respective breakpoint from the list of set breakpoints and press the **Go to** button.

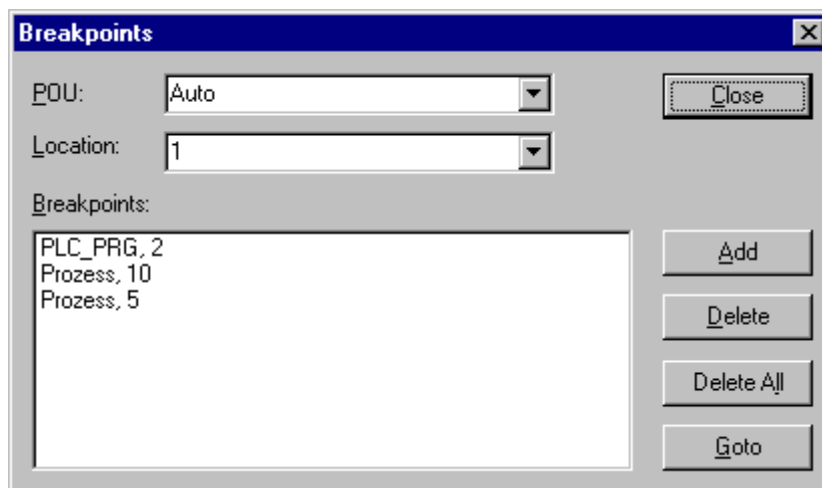


Image 4.43: Breakpoint Editing Dialog Box

"Online" "Step over"

Symbol:  **Shortcut: <F10>**

This command causes a single step to execute. If a POU is called, the program stops after its execution. In SFC a complete action is executed.

If the present instruction is the call-up of a function or of a function block, then the function or function block will be executed completely. Use the "**Online**" "**Step In**" command, in order to move to the first instruction of a called function or function block.

If the last instruction has been reached, then the program will go on to the next instruction in the POU.

"Online" "Step in"

Shortcut: <F8>

A single step is executed. The program is stopped before the first instruction of a called POU.

If necessary, there will be a changeover to an open POU.

If the present position is a call-up of a function or of a function block, then the command will proceed on to the first instruction in the called POU.

In all other situations, the command will function exactly as "**Online**" "**Step Over**".

"Online" "Single Cycle"

Shortcut: <Ctrl>+<F5>

This command executes a single PLC Cycle and stops after this cycle.

This command can be repeated continuously in order to proceed in single cycles.

The Single Cycle ends when the "**Online**" "**Run**" command is executed.

"Online" "Write Values" or "Force Values"

Shortcut: <Ctrl>+<F7> (Write values)

Shortcut: <F7> (Force values)

In order to change the value of a variable containing a single element, you must first use the mouse to doubleclick on the line in which the variable is declared, or alternatively you can use <Enter>. Next you can enter the new value of the variable in the dialog box that pops up. In the case of Boolean variables, the value is toggled without the dialog box appearing. The new value is displayed in red.

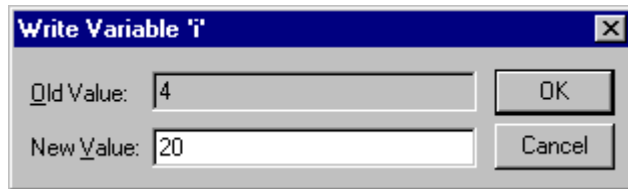


Image 4.44: Dialog Box for Writing a New Variable Value.

The new value is not yet written in the PLC.

Several variables can be set to specific values and afterwards written all at once (cycle consistently) in the PLC.

With "**Write Values**", the values are written just once and can immediately be written over again.

With "**Force Values**", the values continue to be written after each cycle until this procedure is stopped with "**Release Force**".

"Online" "Release Force"

Shortcut: <Shift>+<F7>

This command ends the forcing of variables in the PLC. All forced variables once again change their values normally.

If no forced values are available, the command will have no effect.

"Online""Show Call Stack"

You can run this command when the Simulation Mode stops at a breakpoint. You will be given a dialog box with a list of the POU's currently in the Call Stack.

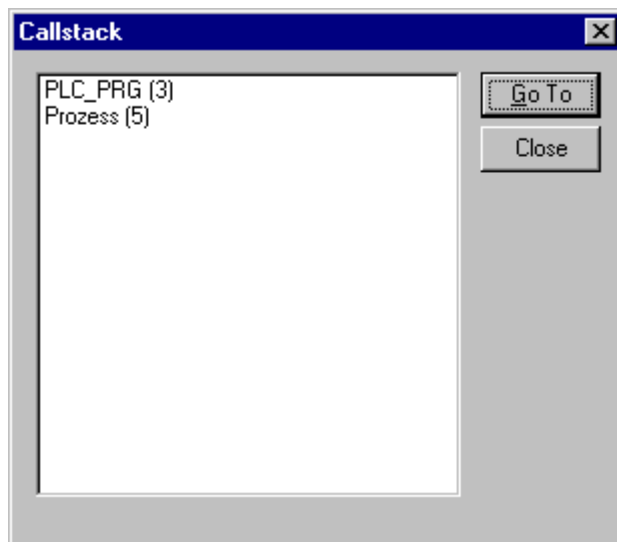


Image 4.45: Example of a Call Stack

The first POU is always PLC_PRG, because this is where the executing begins.

The last POU is always the POU being executed.

After you have selected a POU and have pressed the **Go to** button, the selected POU is loaded in its editor, and it will display the line or network being processed.

"Online" "Flow Control"

If you have selected the **flow control**, then a check(✓) will appear in front of the menu item. Following this, every line or every network will be marked which was executed in the last PLC Cycle.

The line number field or the network number field of the lines or networks which just run will be displayed in green. An additional field is added in the IL-Editor in which the present contents of the accumulator are displayed. In the graphic editors for the Function Block Diagram and Ladder Diagram, an additional field will be inserted in all connecting lines not transporting any Boolean values. When these Out- and Inputs are verified, then the value that is transported over the connecting line will be shown in this field. Connecting lines that transport only Boolean values will be shaded blue when they transport TRUE. This enables constant monitoring of the information flow.

"Online" "Simulation"

If **Simulation Mode** is chosen, then a check(✓) will appear in front of the menu item.

In the simulation mode, the user program runs on the same PC under Windows. This mode is used to test the project. The communication between the PC and Simulation Mode uses the Windows Message mechanism.

If the program is not in simulation mode, then the program will run on the PLC. The communication between the PC and the PLC typically runs over the serial interface.

The status of this flag is stored with the project.

"Online" "Communication Parameters"

The parameters for transferring through the serial interface can be entered in a dialog box. It is important that these parameters agree with those entered in the PLC .

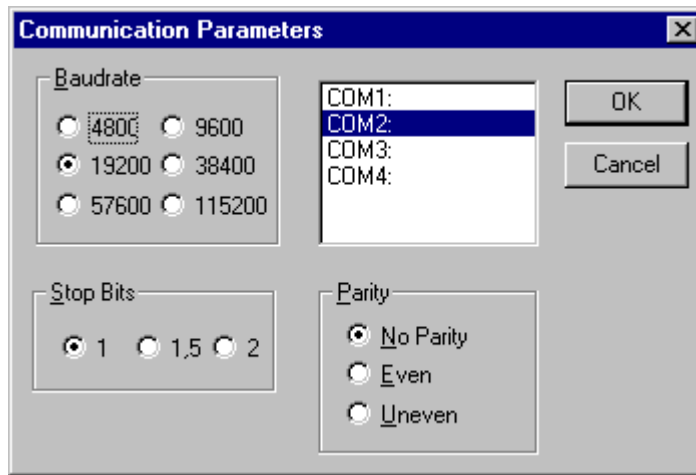


Image 4.46: Dialog box for Entering Communication Parameters

Possible adjustments include: the **Baudrate**; whether the transfer should be made with **Even**, **Uneven**, or **No Parity**; the number of **Stop Bits**; and also the interface (COM1, COM2, etc.) via which the transfer is to occur. The selected parameters are stored with the project.

'Online' 'Communications Parameters' for the use of Gateway

You are offered a special dialog for setting communications parameters when the communication between the local PC and the run-time system is running over a gateway server in your system.

Let us examine the principle of the gateway system before explaining the operation of the dialog:

A gateway server can be used to allow your local PC to communicate with one or more run-time systems. The setting concerning which run-time systems can be addressed, which is specifically configured for each gateway server, and the connection to the desired gateway server, is made on the local PC. Here it is possible that both the gateway server and the run-time system(s) can run together on the local PC. If we are dealing with a gateway server which is running on another PC we must ensure that it has been started there. If you are selecting a locally installed gateway server, it automatically starts when you log onto the target run-time system. You can recognise this through the appearance of a 907 AC 1131 symbol on the bottom right in the task bar. This symbol lights up as long as you are connected to the run-time system over the gateway. The menu points **Info** and **Finish** are obtained by clicking with the right mousekey on the symbol. **Finish** is used to switch off the gateway.

See the following scheme for presenting a gateway system:

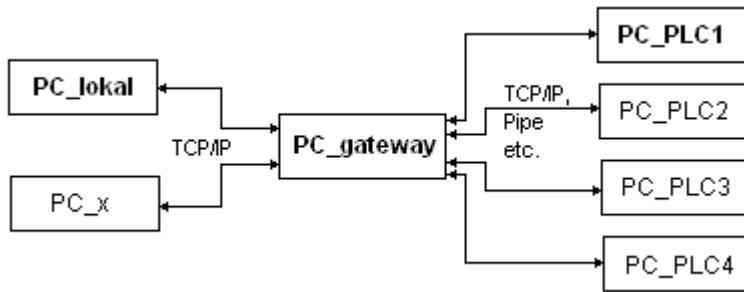


Image 4.47: Example of a Gateway server system

PC_lokal is your local PC, **PC_x** is another PC, which gateway addresses. **PC_gateway** is the PC on which the gateway server is installed, **PC_PLC1** through to **PC_PLC4** are PCs on which the run-time systems are running. The diagram shows the modules as separated but it is fully possible for the Gateway server and / or run-time systems to be installed together on the local PC.



Important: Please note that a connection to gateway is only possible over TCP/IP so make sure that your PC is configured appropriately!

The connections from gateway to the various run-time computers can, on the other hand, run over different protocols (TCP/IP, Pipe, etc.).

Let us now return to the communications parameters dialog on the local PC: It shows the current situation on the selected gateway server, which can be called up at any time using the button **Update**.

The dialog will appear as follows if the communications parameters have already been configured according to the example shown above:

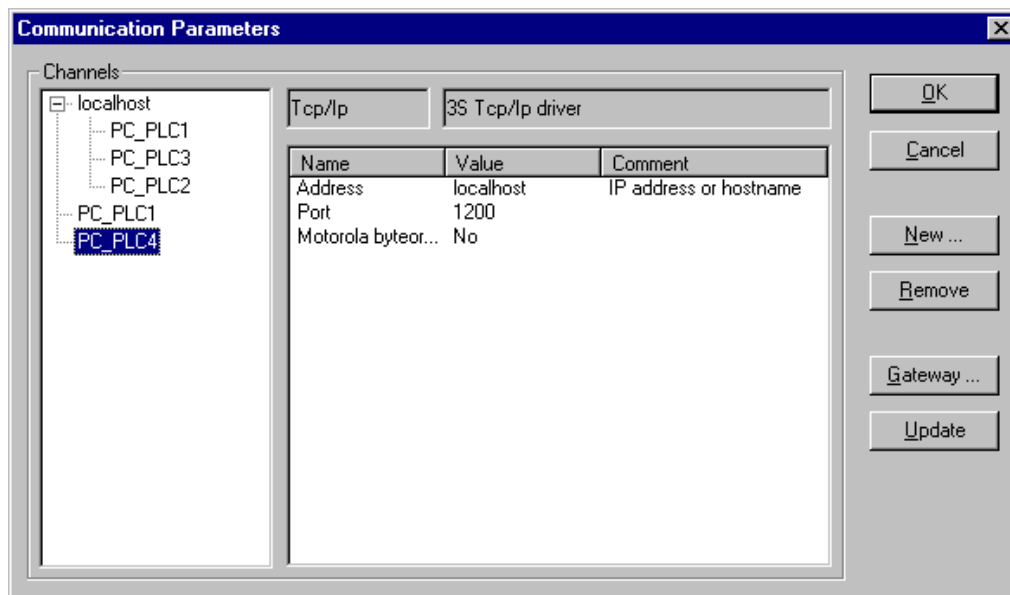


Image 4.48: Dialog for setting the gateway communications parameters, example

The heading **Channels** lists two categories of connections:

On the one hand all of the connections are shown which are installed on the currently connected gateway server called 'localhost'. Here the address or the name of this gateway is located on the upper position behind the minus sign, which in our example is running on the local computer. The appropriate address 'localhost' corresponds in the normal case to the IP address 127.0.0.1 of the local computer (PC_local). Below, indented to the right, are three addresses of run-time computers which the gateway channels are set-up to (PC_PLC1 to 3). They could have been configured both from the local PC or from the other PCs (PC_x) which are or were connected to the gateway server.

The second category of the channels describes includes all connections to the gateway which can be set up from your local PC, over this configuration dialog for example. They create the "branch" which leads from the minus sign directly below to PC_PLC1 and PC_PLC4. These channel addresses do not necessarily have to be known yet at the gateway. For PC_PLC4 in the example described above, the configuration parameters are stored locally in the project but they will first be known to the gateway the next time log-in to the run-time system occurs. This has already occurred for PC_PLC1 since the associated gateway address has appeared as an additional "sub-branch" to the "channel tree".

In the central part of the dialog one finds the designation, in each case, of the left selected channel and the associated parameter under **Name**, **Value** and **Comment**.

Let us now turn to the way the communications parameters themselves can be set in dialog:

To define the connection to the desired gateway we open the dialog Communication Parameters Gateway by pressing the button **Gateway**.

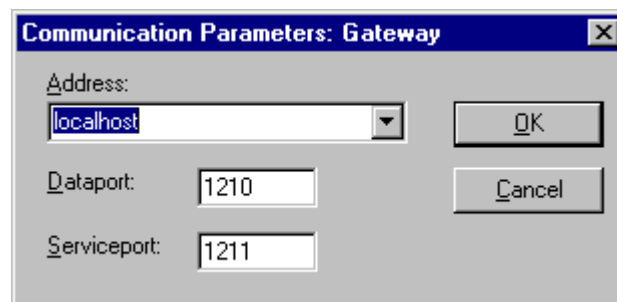


Image 4.49: Example dialog, definition of the local connection to the gateway

Here it is possible to enter or edit the name of the computer on which the gateway server is running as well as its **Dataport** and **Serviceport**. By first activation 'localhost' is offered by default as the name of the computer. You can also enter an IP address instead of the computer name. The name 'localhost' is usually identical to the local IP address 127.0.0.1 in most cases but you may have to enter this directly into the field Address. Suitable values for the selected Gateway are usually already present in the fields Dataport and Serviceport.

By ending the dialog with **OK** the appropriate input appears in the title **Channels** of the dialog Communications Parameters, in the upper position. A sign in brackets showing "not connected" appears behind the address when it is not possible to establish a connection to the selected gateway address, either because it is not running or because the address is incorrect.

Press the button **New** if you wish to install a new channel on the gateway server. You are offered the following dialog.

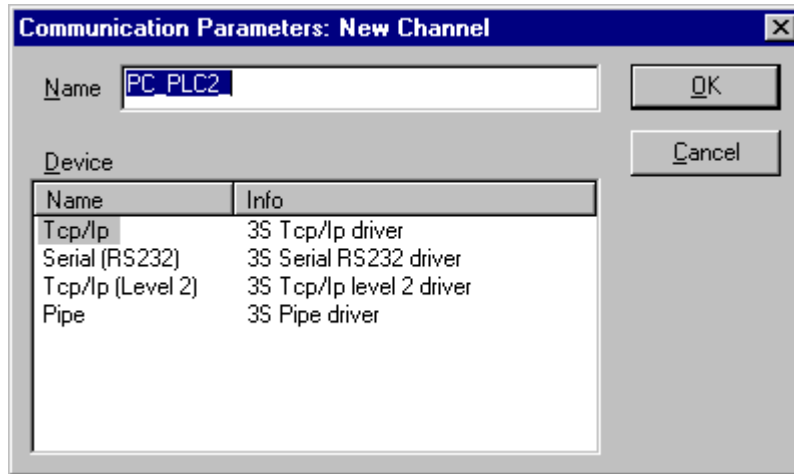


Image 4.50: Example dialog, installing a new channel

The input field **Name** always contains the name used for the last inputted channel. The current gateway name e.g. 'localhost_' appears if a connection has not been entered yet. You can edit the channel name at this point. The channel name is purely informative, it does not have to be a unique name but it is recommended to use one. Select a driver which is offered in the column **Name** with a mouse click (the offering depends on your individual installation conditions) and the commentary to it may possibly be shown in the column **Info** if one exists.

If you now close the dialog with **OK**, the newly defined channel appears in the communications parameter dialog as a further entry for **Channels** at the lowest position under the minus sign. It is only stored locally in the project at first (see above). It is still possible to edit the column **Value** while it is in this condition (see tips below); confirm the parameters with **OK** and to leave the dialog.

To make the newly established connection known in the gateway, and thus to make it generally available, it necessary for you to log-on into the run-time system. When you open the communications parameter dialog again the new channel, apart from being in its former position, also appears indented under the address/name of the gateway computer. If a communications error occurs when logging in, the interface cannot be opened (e.g. COM1 for a serial connection) possibly because it is used by another device.

The parameters for a channel already known by the gateway server can no longer be edited in the configuration dialog. The parameter fields appear grey. You can, however, delete the connection as long as it is not active.



Important: Please note that the deletion of a channel is not reversible. It occurs at the moment that you press on the button **Remove!**

Tips for editing the parameters in the communications parameters dialog:

You can only edit the text fields in the column **Value**.

Select a text field with the mouse, and get into the editing mode by double clicking or by pressing the space bar. The text input is finished by pressing the <Enter> key.

You can use <Tabulator> or <Shift> + <Tabulator> to jump to the next or the previous switching or editing possibility.

To edit numerical values it is possible with the arrow keys or the Page Up/Down keys to change the value by one or ten units respectively. A double click with the mouse also changes the value by increasing by one unit. A typing check is installed for numerical values: <Ctrl> + <Home> or <Ctrl> + <End> deliver the lowest or the highest value respectively for the possible input values for the type of parameter in question.

'Online' 'Sourcecode download'

This command loads the source code for the project into the controller system. See also 'Project' 'Options' 'Sourcedownload'.

4.7 Window set up

Under the "**Window**" menu item you will find all commands for managing the windows. There are commands both for the automatic set up of your window as well as for opening the library manager and for changing between open windows. At the end of the menu you will find a list of all open windows in the sequence they were opened. You can switch to the desired window by clicking the mouse on the relevant entry. A check will appear in front of the active window.

"Window" "Tile Horizontal"

With this command you can arrange all the windows horizontally in the work area so that they do not overlap and will fill the entire work area.

"Window" "Tile Vertical"

With this command you can arrange all the windows vertically in the work area so that they do not overlap and will fill the entire work area.

"Window" "Cascade"

With this command you can arrange all the windows in the work area in a cascading fashion, one behind another.

"Window" "Arrange Symbols"

With this command you can arrange all of the minimized windows in the work area in a row at the lower end of the work area.

"Window" "Close All"

With this command you can close all open windows in the work area.

"Window" "Messages"

Shortcut: <Shift>+<Esc>

With this command you can open or close the message window with the messages from the last compiling, checking, or comparing procedure.

If the messages window is open, then a check (✓) will appear in front of the command.

4.8 Help when you need it

Should you encounter any problems with **907 AC 1131** during your work, online help is available to help to solve them. There you will find all the information that is also contained in this handbook.

"Help" "Contents and Index"

With this command you can open the help topics window.

Under the **Contents** register card you will find the contents. The books can be opened and closed using a doubleclick or the corresponding button. Doubleclicking or activating the **Show** button on a highlighted topic will display the topic in the main window of help or in the index window.

Click on the **Index** register card to look for a specific word, and click on the **Search** register card to select a full-text search. Follow the instructions in the register cards.

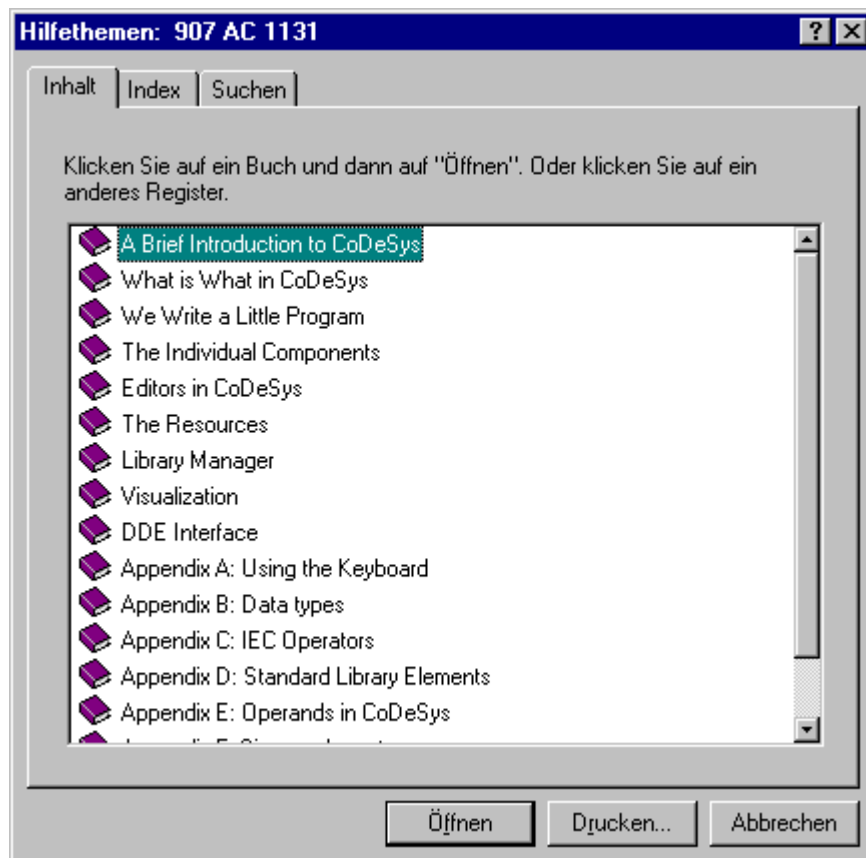


Image 4.51: Help Topics Window

Main Help Window

In the main help window topics are displayed with index entries listed below them.

The following buttons are available:

- **Help topics** opens the help topics window
- **Back** shows the help entry that was previously displayed
- **Print** opens the dialog box for printing
- << shows the help entry that comes prior in sequence to the present entry
- >> shows the help entry that is next in sequence

In addition you can use the following menu commands:

- With "**File**" "**Print Topics**" you can print out the present help entry.
- If you use the "**Edit**" "**Copy**" command, the selected text will be copied into the clipboard. From here you can insert the text into other applications and use it there.
- If you use the "**Edit**" "**Annotate**" command, a dialog box will be opened. There is an editing field on the left side of the dialog box in which you can enter an annotation to the help page. On the right side there are buttons for **storing** the text, for canceling the program, for **deleting** the notation, for **copying** a highlighted text on the clipboard, and for **past**ing a text from the clipboard. If you have made an annotation to a help entry, a small green paper clip will appear in the upper left-hand corner. By clicking the mouse on the

paper clip, you can open the dialog box with the annotation that has been made.

- If you would like to mark a page from help, then you can set a bookmark. To do so, choose the "**Define**" "**Bookmark**" command. A dialog box will appear in which you can enter a new name (The name of the page can serve as a starter) or can delete an old bookmark. If bookmarks were defined, then these will be displayed in the "**Bookmark**" menu. By choosing these menu items, you can access the desired page.
- Under "**Options**", you can define whether the help window always appears in the foreground or in the background or in the standard setting.
- With "**Display previous topics**" under "**Options**", you are furnished with a selection window with the previously displayed help topics. Doubleclick the entry you wish to view.
- Under "**Options**", you can select the desired "**Font**" in **small**, **normal**, or **large**.
- If "**Options**" "**Use System Color**" has been chosen, help will not be displayed in the colors that were set, but in the system colors instead.

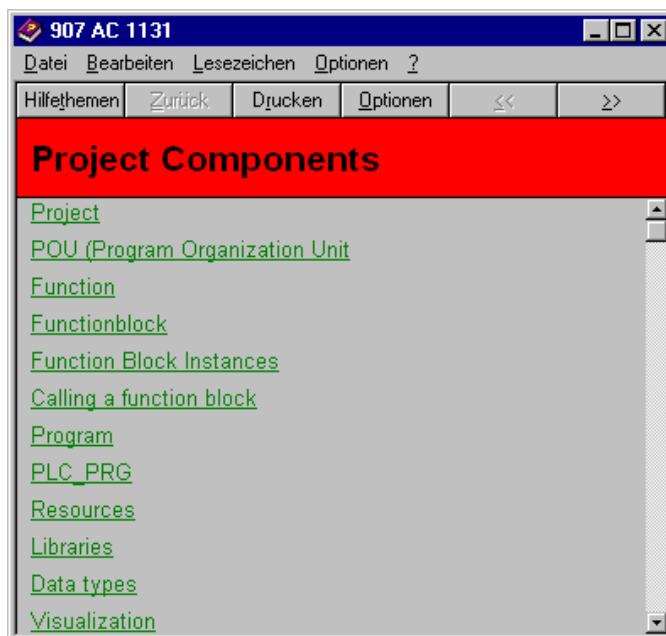


Image 4.52: Main Help Window

Index Window

The index window contains explanations about the menu commands, terms, or sequences.

The index window will always remain on the surface by default, unless the help option is placed in the background in the main window of help.

The following buttons are available:

- **Help topics** opens the help topics window
- **Back** shows the help entry that was previously displayed
- **Print** opens the dialog box for printing
- **<<** shows the help entry directly prior to the present entry
- **>>** shows the help entry that is next in sequence

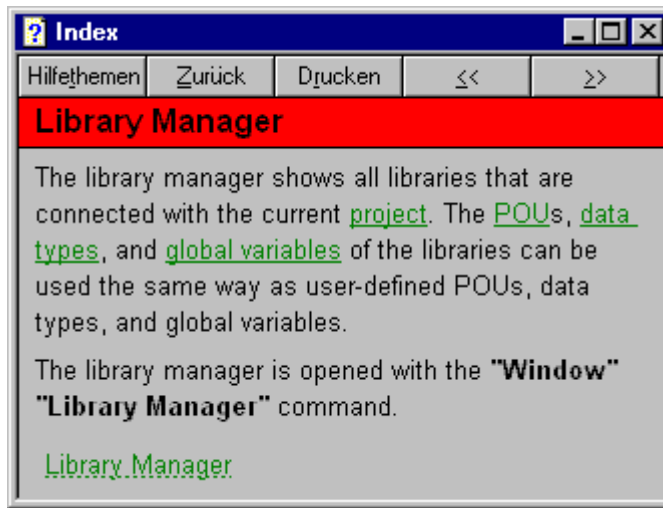


Image 4.53: Index Window

Context Sensitive Help

Shortcut: <F1>

You can use the <F1> key in an active window, in a dialog box, or above a menu command. When you perform a command from the menu, the help for the command called up at that time is displayed.

You can also highlight a text (for example, a key word or a standard function) and have the help displayed for that item..

5.1 The Declaration Editor

Declaration editors are used to declare variables of POU (Program Organization Units) and global variables, for data type declarations, and in the Watch and Receipt Manager.

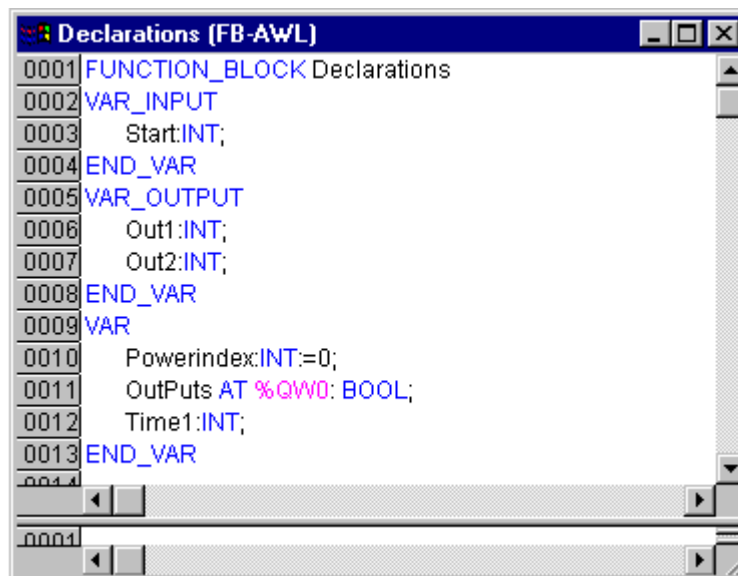
The declaration of variables is supported by syntax coloring.

All editors for POU (Program Organization Units) consist of a declaration part and a body. These are separated by a screen divider that can be dragged, as required, by clicking it with the mouse and moving it up or down.

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Declaration Part

All variables to be used only in this POU are declared in the declaration part of the POU. These can include: input variables, output variables, input/output variables, local variables, retain variables, and constants. The declaration syntax is based on the IEC1131-3 standard. An example of a correct declaration of variables in **907 AC 1131** -Editor:



```

0001 FUNCTION_BLOCK Declarations
0002 VAR_INPUT
0003     Start:INT;
0004 END_VAR
0005 VAR_OUTPUT
0006     Out1:INT;
0007     Out2:INT;
0008 END_VAR
0009 VAR
0010     Powerindex:INT:=0;
0011     OutPuts AT %QW0: BOOL;
0012     Time1:INT;
0013 END_VAR
0014
0001

```

Image 5.1: Declaration Editor

Input Variable

Between the key words **VAR_INPUT** and **END_VAR**, all variables are declared that serve as input variables for a POU. That means that at the call position, the value of the variables can be given along with a call.

Example:

```
VAR_INPUT  
  in1:INT (* 1. Inputvariable*)  
END_VAR
```

Output Variable

Between the key words **VAR_OUTPUT** and **END_VAR**, all variables are declared that serve as output variables of a POU. That means that these values are carried back to the POU making the call. There they can be answered and used further.

Example:

```
VAR_OUTPUT  
  out1:INT; (* 1. Outputvariable*)  
END_VAR
```

Input and Output Variables

Between the key words **VAR_IN_OUT** and **END_VAR**, all variables are declared that serve as input and output variables for a POU.

Attention: With this variable, the value of the transferred variable is changed ("transferred as a pointer"). That means that the input value for such variables cannot be a constant.

Example:

```
VAR_IN_OUT  
  inout1:INT; (* 1. Inputoutputvariable *)  
END_VAR
```

Local Variables

Between the keywords **VAR** and **END_VAR**, all of the local variables of a POU are declared. These have no external connection; in other words, they can not be manipulated from the outside.

Example:

```
VAR  
  loc1:INT; (* 1. Local Variable*)  
END_VAR
```

Retain Variables

Retain variables are identified by the key word **RETAIN**. These variables maintain their value, even after a power failure. When the program is run again, the stored values will be processed further. A practical example would be an operations timer that recommences timing after a power failure.

All other variables are newly initialized, either with their initialized values or with the standard initializations.

Example:

```
VAR RETAIN  
    rem1:INT; (* 1. Retain variable*)  
END_VAR
```

Constants

Constants are identified by the key word **CONSTANT**. They can be declared locally or globally.

Syntax:

```
VAR CONSTANT  
    <Identifier>:<Type> := <initialization>;  
END_VAR
```

Example:

```
VAR CONSTANT  
    con1:INT:=12; (* 1. Constant*)  
END_VAR
```

You will find a listing of possible constants here in the appendix.

Keywords

In all editors, all keywords are written in capital letters. Keywords may not be used as variables.

Variables declaration

A variables declaration has the following syntax:

```
<Identifier> {AT <Address>}:<Type> {:=<initialization>;}
```

The parts in the braces {} are optional.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The first 32 characters are significant.

All declarations of variables and data type elements can include initialization. They are brought about by the "!=" operator. For variables of elementary types, these initializations are constants. The default-initialization is 0 for all declarations.

Example:

```
var1:INT:=12; (* Integer variable with initial value of 12*)
```

If you wish to link a variable directly to a definite address, then you must declare the variable with the keyword **AT**.

For faster input of the declarations, use the shortcut mode.

In function blocks you can also specify variables with incomplete address statements. In order for such a variable to be used in a local instance, there must be an entry for it in the variable configuration.

Pay attention to the possibility of an automatic declaration

AT Declaration

If you wish to link a variable directly to a definite address, then you must declare the variable with the keyword **AT**. The advantage of such a procedure is that you can assign a meaningful name to an address, and that any necessary changes of an incoming or outgoing signal will only have to be made in one place (e.g., in the declaration).

Notice that variables requiring an input cannot be accessed by writing. A further restriction is that AT declarations can only be made for local and global variables, and not for input- and output variables from POU's.

Examples:

```
counter_heat7 AT %QX0.0: BOOL;  
lightcabinetimpulse AT %IX7.2: BOOL;  
download AT %MX2.2: BOOL;
```

"Insert" "Declarations keywords"

You can use this command to open a list of all the keywords that can be used in the declaration part of a POU. After a keyword has been chosen and the choice has been confirmed, the word will be inserted at the present cursor position.

You also receive the list, when you open the Input Assistant and choose the **Declarations** category.

"Insert" "Type"

With this command you will receive a selection of the possible types for a declaration of variables. You also receive the list when you access the Input Assistant.

The types are divided into these categories:

- Standard types BOOL, BYTE, etc.
- Defined types Structures, enumeration types, etc.
- Standard function blocks for instance declarations
- Defined function blocks for instance declarations

907 AC 1131 supports all standard types of IEC1131-3:

Examples for the use of the various types are found in the appendix.

Syntax Coloring

In all editors you receive visual support in the implementation and declaration of variables. Errors are avoided, or discovered more quickly, because the text is displayed in color.

A comment left unclosed, thus annotating instructions, will be noticed immediately; keywords will not be accidentally misspelled, etc.

The following color highlighting will be used:

- Blue Keywords
- Green Comments in the text editors
- Pink Special constants (e.g. TRUE/FALSE, T#3s, %IX0.0)
- Red Input error (for example, invalid time constant, keyword, written in lower case,...)
- Black Variables, constants, assignment operators, ...

Shortcut Mode

The declaration editor for **907 AC 1131** allows you to use the shortcut mode. This mode is activated when you end a line with <Ctrl><Enter>

The following shortcuts are supported:

- All identifiers up to the last identifier of a line will become declaration variable identifiers
- The type of declaration is determined by the last identifier of the line. In this context, the following will apply:
 - B or BOOL gives the result BOOL
 - I or INT gives the result INT
 - R or REAL gives the result REAL
 - S or string gives the result STRING
- If no type has been established through these rules, then the type is BOOL and the last identifier will not be used as a type (Example 1.).
- Every constant, depending on the type of declaration, will turn into an initialization or a string (Examples 2. and 3.).
- An address (as in %MD12) is extended around the AT... attribute(Example 4.).
- A text after a semicolon (;) becomes a comment (Example 4.).
- All other characters in the line are ignored (e.g., the exclamation point in Example 5.).

Examples:

Shortcut	Declaration
A	A: BOOL;
A B I 2	A, B: INT := 2;
ST S 2; A string	ST:STRING(2); (* A string *)
X %MD12 R 5 Real Number	X AT %MD12: REAL := 5.0;(* Real Number *)
B !	B: BOOL;

Autodeclaration

If the **Autodeclaration** of the Options dialog box is enabled, then a dialog box will appear in all editors after the input of a variable that has not yet been declared. With the help of this dialog box, the variable can now be declared.

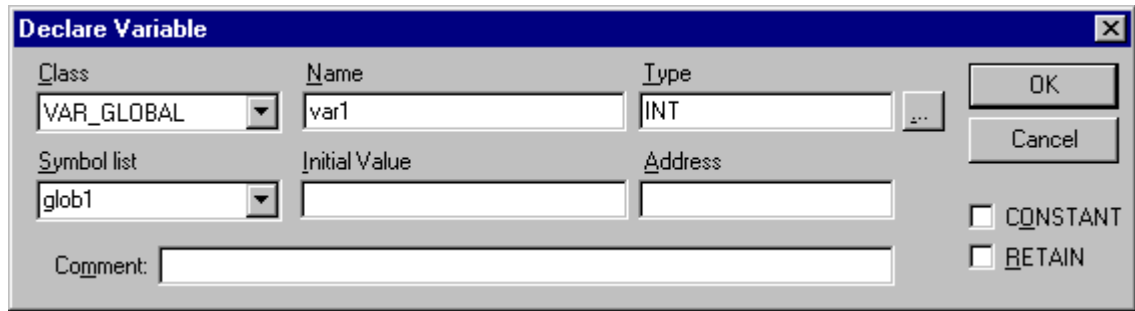


Image 5.2: Dialog Box for Declaration of Variables

With the help of the **Class** combobox, select whether you are dealing with a local variable (**VAR**), input variable (**VAR_INPUT**), output variable (**VAR_OUTPUT**), input/output variable (**VAR_INOUT**), or a global variable (**VAR_GLOBAL**).

With the **CONSTANT** and **RETAIN** options, you can define whether you are dealing with a constant or a retain variable

The variable name you entered in the editor has been entered in the **Name** field, **BOOL** has been placed in the **Type** field. The ... button opens the Input Assistant dialog which allows you to select from all possible types.

In the **Initial Value** field you can assign a value to the variable; otherwise the standard initial value will be used.

In the **Address** field you can link a variable to the address (AT declaration)

If necessary, insert a **Comment**.

By pressing **OK** you will enter the variable in the corresponding declaration editor.



Note: The dialog box for variable declaration you also get by the command 'Edit' 'Declare Variable'.

Line Numbers in the Declaration Editor

In offline mode, a simple click on a special line number will mark the entire text line.

In the online mode, a single click on a specific line number will open up or close the variable in this line, in case a structural variable is involved.

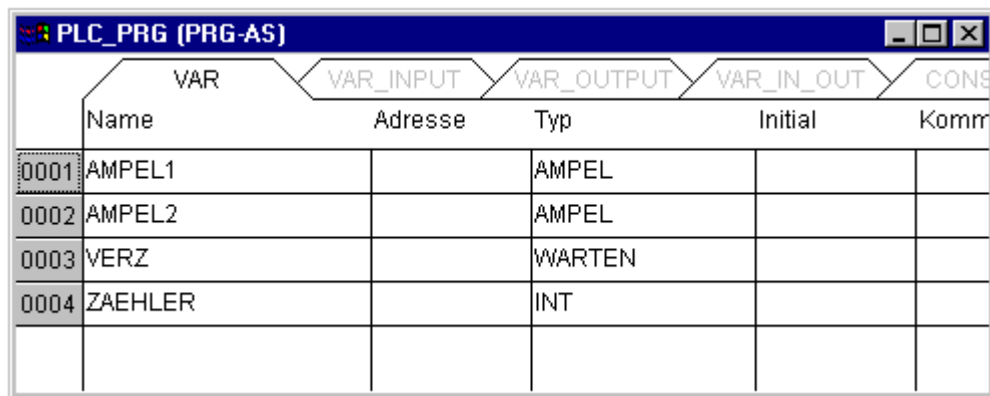
Declarations as tables

If the **Declarations as tables** option is set in the Options dialog box in the **Editor** category, the declaration editor looks like a table. As in a card-index box, you can select the register cards of the respective variable types and edit the variables.

For each variable you are given the following entry fields.

- Name:** Input the identifier of the variable.
Address: If necessary, input the address of the variable (AT declaration)
Type: Input the type of the variable. (Input the function block when instantiating a function block)
Initial: Enter a possible initialization of the variable (corresponding to the " := " assignment operator).
Comment: Enter a comment here.

Both of the display types of the declaration editor can be changed without causing any problems. In the online mode, there are no differences for the display.



	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CONST
	Name	Adresse	Typ	Initial	Kommentar
0001	AMPEL1		AMPEL		
0002	AMPEL2		AMPEL		
0003	VERZ		WARTEN		
0004	ZAEHLER		INT		

Image 5.3: Declaration Editor as a Table

"Insert" "New Declaration"

With this command you bring a new variable into the declaration table of the declaration editor. If the present cursor position is located in an field of the table, then the new variable will be pasted in the preceding line; otherwise, the new variable is pasted at the end of the table. Moreover, you can paste a new declaration at the end of the table by using the right arrow key or the tab key in the last field of the table.

You will receive a variable that has "Name" located in the **Name** field, and "Bool" located in the **Type** field, as its default setting. You should change these values to the desired values. Name and type are all that is necessary for a complete declaration of variables.

Declaration Editors in Online Mode

In online mode , the declaration editor changes into a monitor window. In each line there is a variable followed by the equal sign (=) and the value of the

variable. If the variable at this point is undefined, three question marks (???) will appear.

In front of every multi-element variable there is a plus sign. By pressing <Enter> or after doubleclicking on such a variable, the variable is opened up. In the example, the traffic signal structure would be opened up.

```
[-] AMPEL1
  .....STATUS = 3
  .....GRUEN = FALSE
  .....GELB = FALSE
  .....ROT = TRUE
  .....AUS = FALSE
```

When a variable is open, all of its components are listed after it. A minus sign appears in front of the variable. If you doubleclick again or press <Enter>, the variable will be closed, and the plus sign will reappear.

Pressing <Enter> or doubleclicking on a single-element variable will open the dialog box to write a variable. Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

The new value will turn red and will remain unchanged. If the "**Online**" "**Write values**" command is given, then all variables are placed in the selected list and are once again displayed in black.

If the "**Online**" "**Force values**" command is given, then all variables will be set to the selected values, until the "**Release force**" command is given.

Comment

User comments must be enclosed in the special character string "*" and "*".

Comments are allowed in all text editors at any preferred location. These include all declarations, the IL and ST languages, and user-defined data types.

In FBD and LD comments can be entered into every network. In order to do so, select the network that you wish to comment, and enable "**Insert**" "**Comment**".

In SFC you can enter comments for the step in the dialog box for editing step attributes.

Interlocking comments are not allowed.

In Online mode, if you place the mouse pointer briefly above a variable, then the type and, if necessary, address and comments about the variable will be displayed in a Tooltip.

5.2 The Text Editors

The text editors (the Instruction List editor and the editor for) of **907 AC 1131** offer the usual capabilities of Windows text editors.

The implementation in the text editors is supported by syntax coloring.

In Overwrite mode the status bar shows a black **OV**. You can switch between Overwrite mode and Insert mode by key <Ins>

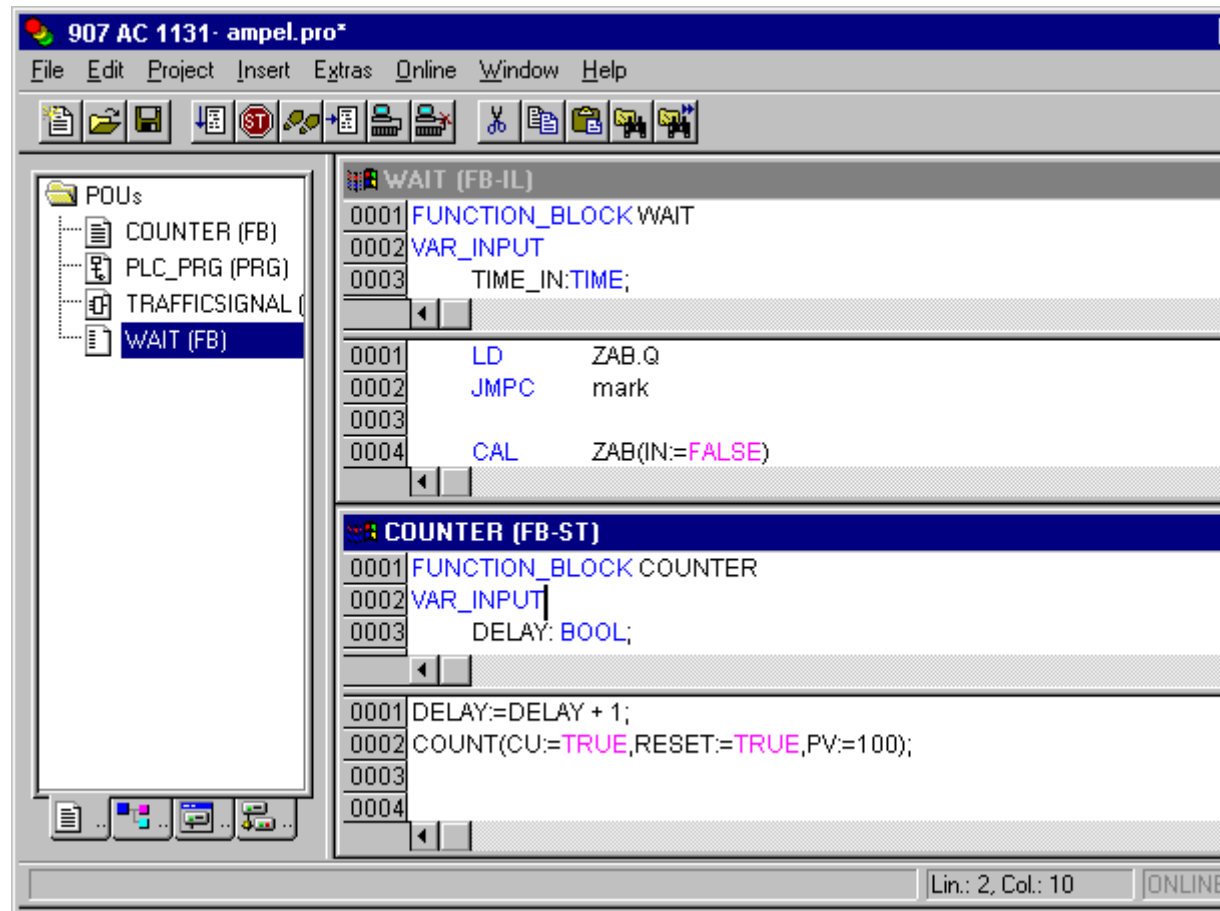


Image 5.4: Text Editors for the Instruction List and Structured Text

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

The text editors use the following menu commands in special ways:

"Insert" "Operator"

With this command all of the operators available in the current language are displayed in a dialog box.

If one of the operators is selected and the list is closed with **OK**, then the highlighted operator will be inserted at the present cursor position.

"Insert" "Operand"

With this command all variables in a dialog box are displayed. You can select whether you would like to display a list of the global, the local, or the system variables.

If one of the operands is chosen, and the dialog box is closed with **OK**, then the highlighted operand will be inserted at the present cursor position.

"Insert" "Function"

With this command all functions will be displayed in a dialog box. You can choose whether to have a list displaying user-defined or standard functions.

If one of the functions is selected and the dialog box is closed with **OK**, then the highlighted function will be inserted at the current cursor position.

If the **With arguments** option was selected in the dialog box, then the necessary input and output variables will also be inserted.

"Insert" "Function Block"

With this command all function blocks are displayed in a dialog box. You can choose whether to have a list displaying user-defined or standard function blocks.

If one of the function blocks is selected and the dialog box is closed with **OK**, then the highlighted function block will be inserted at the current cursor position.

If the **With arguments** option was selected in the dialog box, then the necessary input variables of the function block will also be inserted.

The text editors in Online mode

The online functions in the editors are set breakpoint and single step processing (steps). Together with the monitoring, the user thus has the debugging capability of a modern Windows standard language debugger.

In Online mode, the text editor window is vertically divided in halves. On the left side of the window you will then find the normal program text; on the right side you will see a display of the variables whose values were changed in the respective lines.

The display is the same as in the declaration part. That means that when the PLC is running, the present values of the respective variables will be displayed.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

"Extras" "Monitoring Options"

With this command you can configure your monitoring window. In the text editors, the window is divided into two halves during monitoring. The program is located in the left half. In the right half, all variables that are located in the corresponding program line are monitored.

You can specify the Monitor Window **Width** and which **Distance** two variables should have in a line. An distance declaration of 1 corresponds, in this case, to a line height in the selected font.

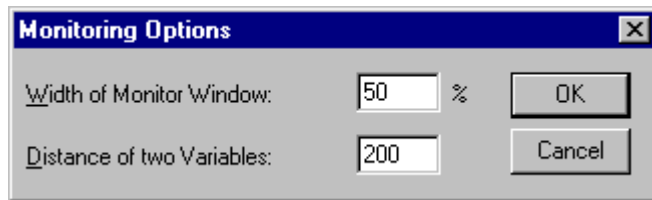


Image 5.5: Monitoring Options Dialog Box

Breakpoint Positions

Since in **907 AC 1131** several IL lines are internally combined into a single C-code line, breakpoints can not be set in every line. Breakpoint positions include all positions in a program at which values of variables can change or where the program flow branches off. (Exception: function calls. If necessary, a breakpoint in the function must be set here.) At the positions lying inbetween, a breakpoint would not even make sense, since nothing has been able to change in the data since the preceding breakpoint position.

This results in the following breakpoint positions in the IL:

- At the start of the POU
- At every LD, LDN (or, in case a LD is located at a label, then at the label)
- At every JMP, JMPC, JMPCN
- At every label
- At every CAL, CALC, CALCN
- At every RET, RETC, RETCN
- At the end of the POU

Structured Text accommodates the following breakpoint positions:

- At every assignment
- At every RETURN and EXIT instruction
- in lines where conditions are being evaluated (WHILE, IF, REPEAT)
- At the end of the POU

Breakpoint positions are marked by the display of the line number field in a darker gray.

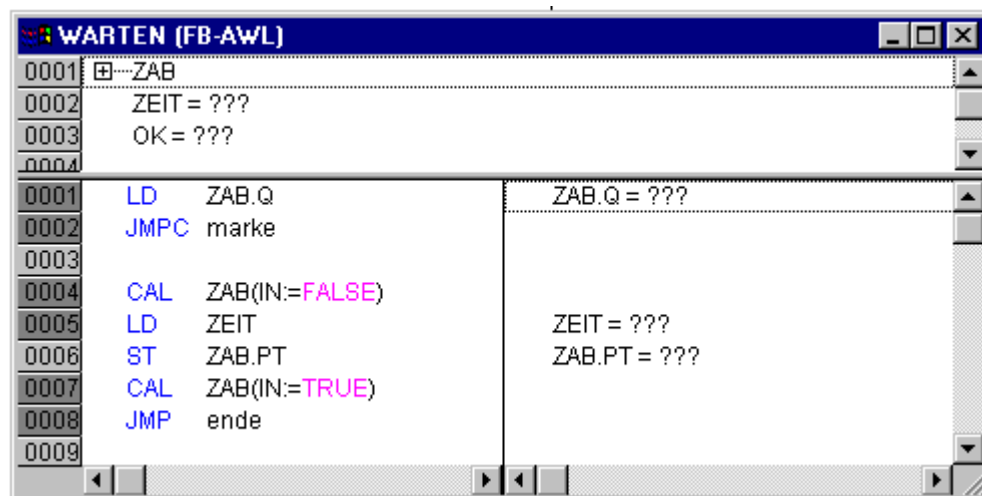


Image 5.6: IL Editor with Possible Breakpoint Positions (darker number fields)

How do you set a breakpoint?

In order to set a breakpoint, click the line number field of the line where you want to set a breakpoint. If the selected field is a breakpoint position, then the color of the line numbers field will change from dark gray to light blue, and the breakpoint will be activated in the PLC.

Deleting Breakpoints

Correspondingly, in order to delete a breakpoint, click on the line number field of the line with the breakpoint to be deleted.

Setting and deleting of breakpoints can also be selected via the menu ("**Online**" "**Toggle Breakpoint**"), via the function key <F9>, or via the symbol in the tool bar.

What happens at a breakpoint?

If a breakpoint is reached in the PLC, then the screen will display the break with the corresponding line. The line number field of the line where the PLC is positioned will appear in red. The user program is stopped in the PLC.

If the program is at a breakpoint, then the processing can be resumed with "**Online**" "**Run**".

In addition, with "**Online**" "**Step over**" or "**Step in**" you can cause the program to run to the next breakpoint position. If the instruction where you are located is a CAL command, or, if there is a function call in the lines up to the next breakpoint position, then you can use "**Step over**" to bypass the function call. With "**Step in**", you will branch to the open POU.

Line Number of the Text Editor

The line numbers of the text editor give the number of each text line of an implementation of a POU.

In Off-line mode, a simple click on a special line number will mark the entire text line.

In Online mode, the background color of the line number indicates the breakpoint status of every line:

- dark gray: This line is a possible position for a breakpoint.
- light blue: a breakpoint has been set in this line.
- red: The program has reached this point.

In Online mode, simply clicking the mouse will change the breakpoint status of this line.

5.2.1 The Instruction List Editor

This is how a POU written in the IL looks under the corresponding **907 AC 1131** editor:

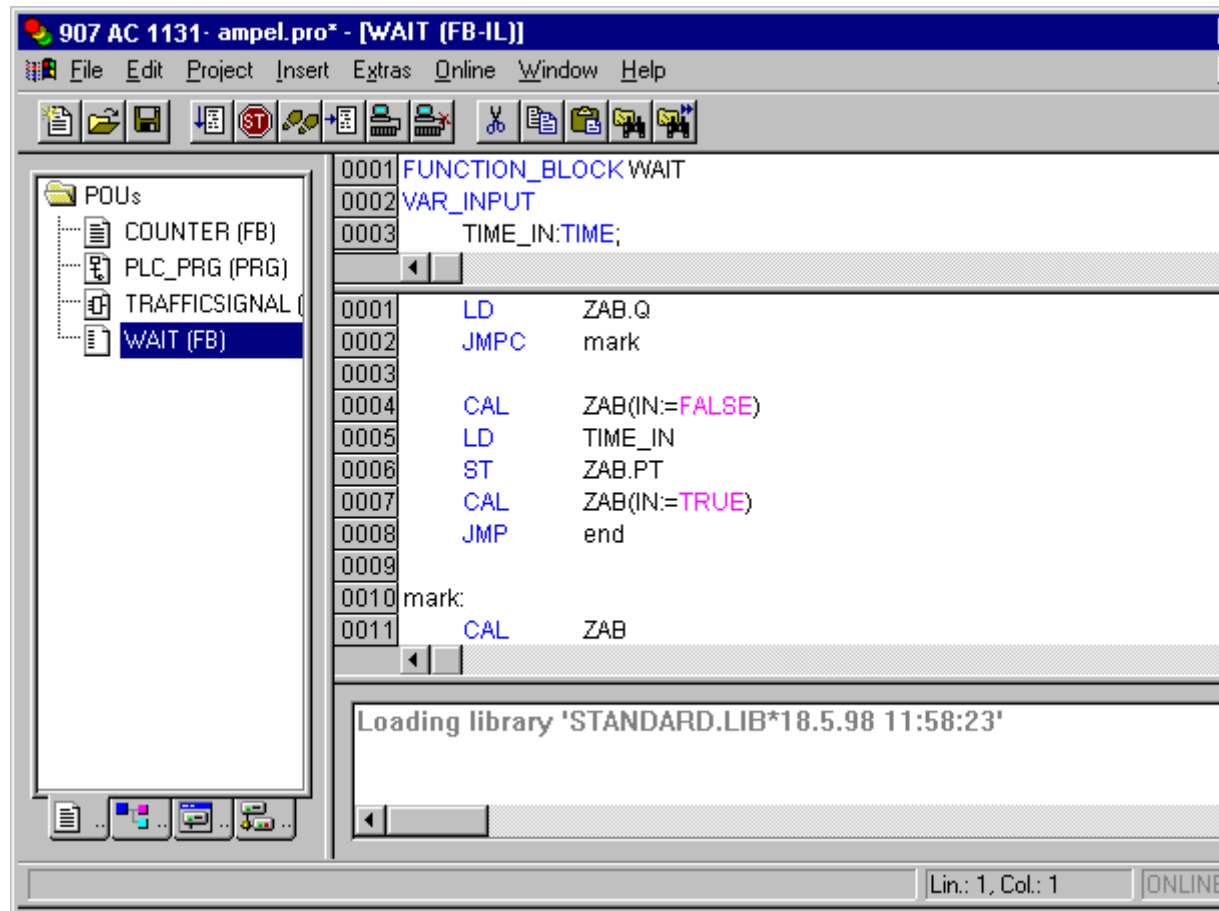


Image 5.7: IL Editor

All editors for POU's consist of a declaration part and a body. These are separated by a screen divider.

The Instruction List editor is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the IL editor in Online mode, see Text Editors in Online Mode.

For information about the language, see the Instruction Lists.

Flow Control

With the **"Online" "Flow control"** command, an additional field in which the accumulator contents is displayed is inserted in the IL editor on the left side of every line.

5.2.2 The Editor for Structured Text

This is how a POU written in ST appears under the corresponding **907 AC 1131** editor:

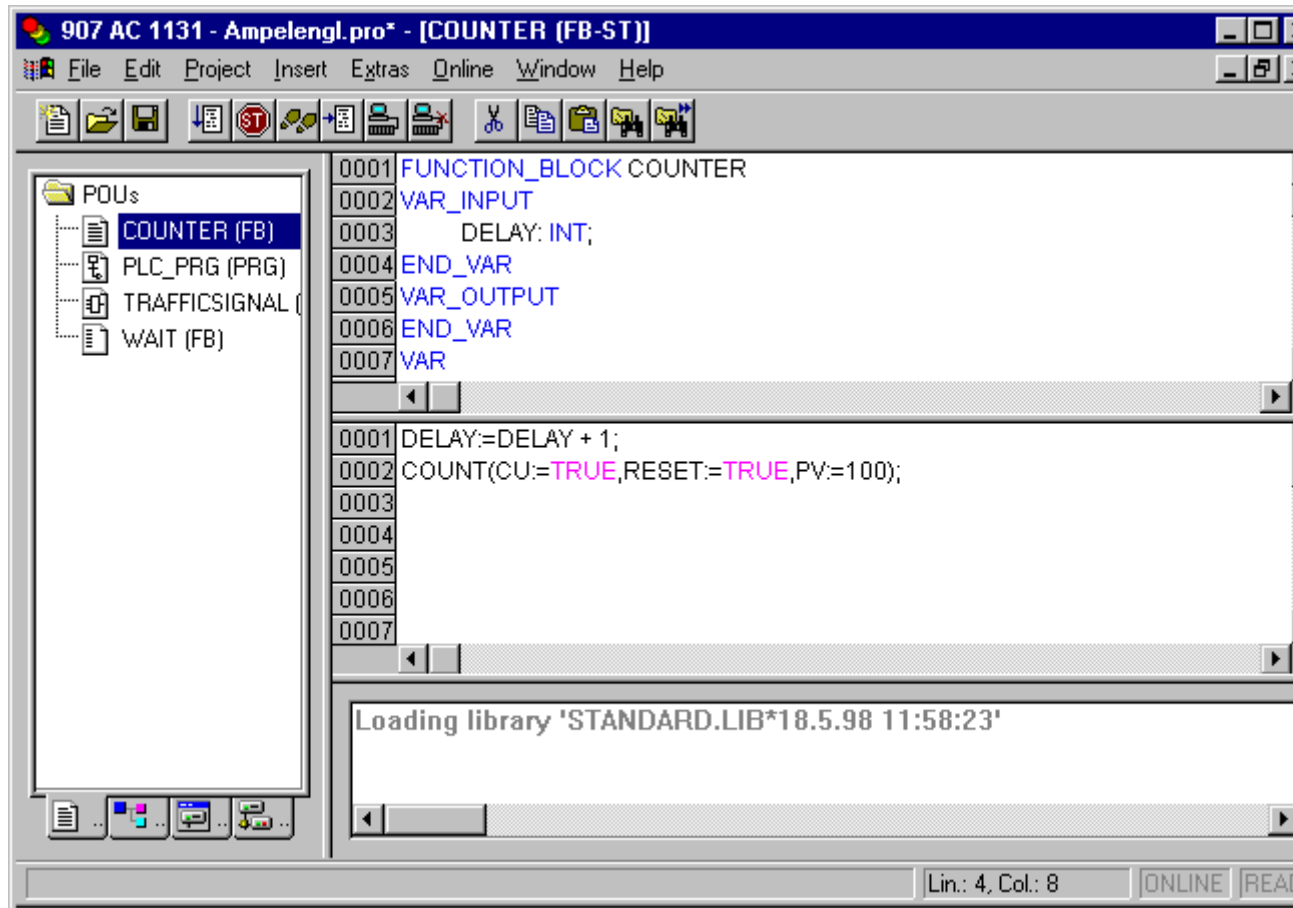


Image 5.8: Editor for Structured Text

All editors for POU's consist of a declaration part and a body. These are separated by a screen divider.

The editor for Structured Text is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the ST editor in Online mode, read Text Editors in Online Mode.

For information about the language, read the chapter Structured Text(ST).

5.3 The Graphic Editors

The editors of both of the graphically oriented languages, LD and FBD, have many points in common. These points are summarized in the following chapters. The editor of the Sequential Function Chart digresses from this point and is, consequently, not described until Chapter The Sequential Function Chart Editor.

The implementation in the graphic editors is supported by syntax coloring.

Label

Each network has a label that can optionally be left empty. This label is edited by clicking the first line of the network, directly next to the network number. Now you can enter a label, followed by a colon.

Network Comments

Every network can be supplied with a multi-lined comment. In "**Extras**" "**Options**", you can enter the maximum number of lines to be made available for a network comment. This entry is made in the **maximum comment size** field. (The default value here is 4.) You can also enter the number of lines that generally should be reserved for comments (**minimum comment size**). If, for example, the number 2 is entered, then, at the start of each network there will be two empty lines after the label line. The default value here is 0, which has the advantage of allowing more networks to fit in the screen area.

If the minimal comment size is greater than 0, then in order to enter a comment you simply click in the comment line and then enter the comment. Otherwise you must next select the network to which a comment is to be entered, and use "**Insert**" "**Comment**" to insert a comment line. In contrast to the program text, comments are displayed in gray.

"Insert" "Network (after)" or
"Insert" "Network (before)"

Shortcut: <Shift>+<T> (Network after)

In order to insert a new network in the FBD or the LD editor, select the "**Insert**" "**Network (after)**" or the "**Insert**" "**Network (before)**" command, depending on whether you want to insert the new network before or after the present network. The present network can be changed by clicking the network number. You will recognize it in the dotted rectangle under the number. With the <Shift key> and a mouse click you can select from the entire area of networks, from the present one to the one clicked.

The network editors in the
online mode

In the FBD and the LD editors you can only set breakpoints for networks. The network number field of a network for which a breakpoint has been set, is displayed in blue. The processing then stops in front of the network, where the breakpoint is located. In this case, the network number field is displayed in red. With single step processing (steps), you can jump from network to network.

All values are monitored upon entering and exiting network POUs (Program Organization Units).

The flow control is run with the "**Online**" "**Flow control**" command. Using the flow control, you can view the present values that are being carried in the networks over the connecting lines. If the connecting lines do not carry Boolean values, then the value will be displayed in a specially inserted field. If the lines

carry Boolean values, then they will be shaded blue, in the event that they carry TRUE. Therefore, you can accompany the flow of information while the PLC is running.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.1 The Function Block Diagram Editor

This is how a POU written in the FBD under the corresponding **907 AC 1131** editor looks:

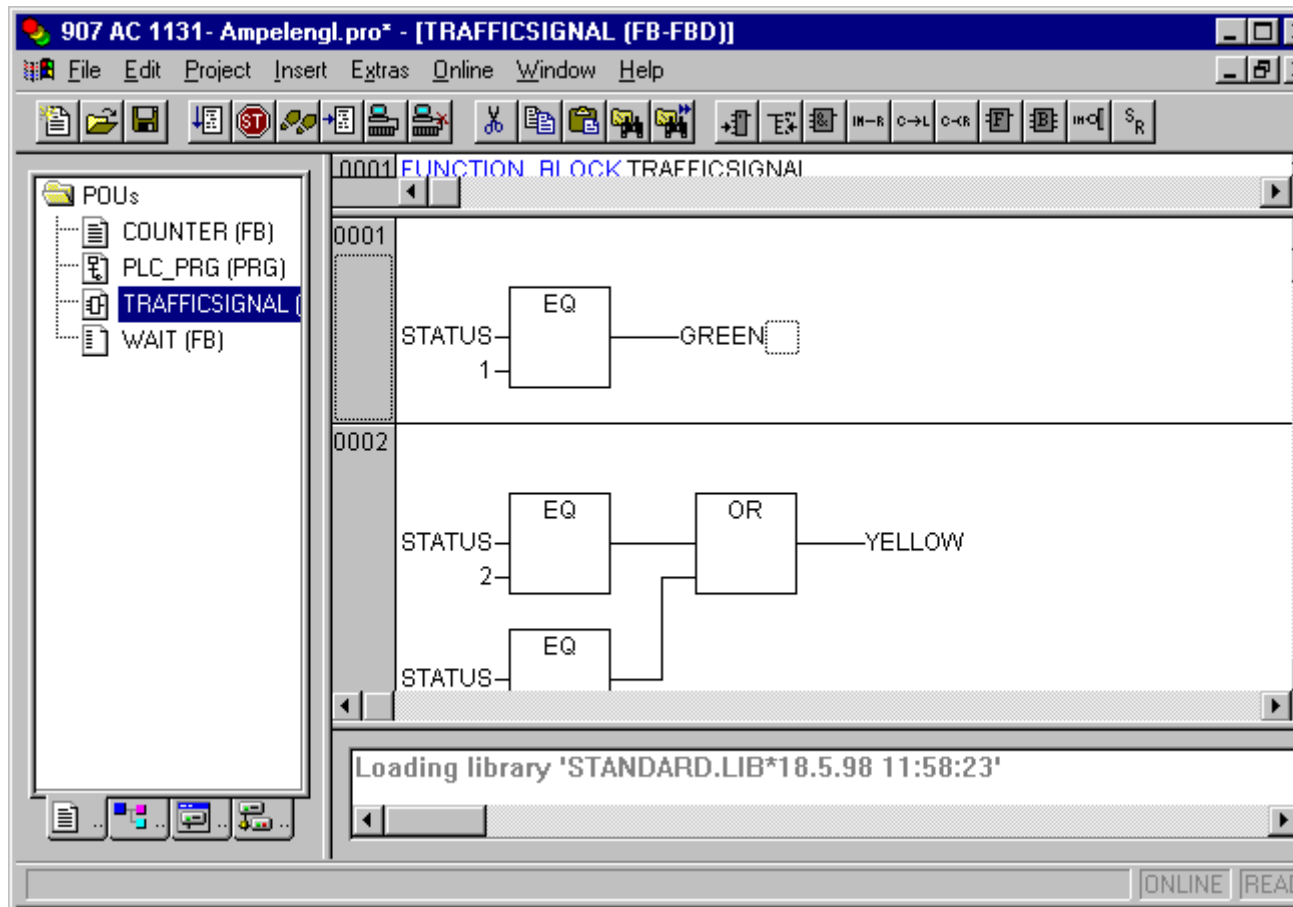


Image 5.9: Editor for the Function Block Diagram

The Function Block Diagram editor is a graphic editor. It works with a list of networks, in which every network contains a structure that displays, respectively, a logical or an arithmetical expression, the calling up of a function block, a jump, or a return instruction.

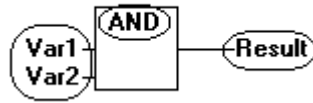
The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Cursor positions in FBD

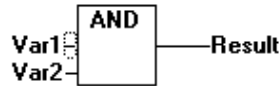
Every text is a possible cursor position. The selected text is on a blue background and can now be changed.

You can also recognize the present cursor position by a dotted rectangle. The following is a list of all possible cursor positions with an example:

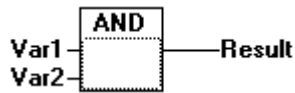
- 1) Every text field (possible cursor positions framed in black):



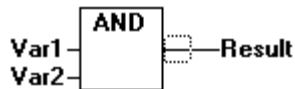
- 2) Every input:



- 3) Every operator, function, or function block:



- 4) Outputs, if an assignment or a jump comes afterward:



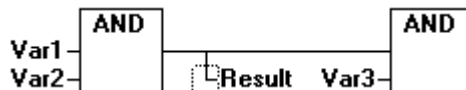
- 5) The lined cross above an assignment, a jump, or a return instruction:



- 6) Behind the outermost object on the right of every network ("last cursor position," the same cursor position that was used to select a network):



- 7) The lined cross directly in front of an assignment:



How to set the cursor

The cursor can be set at a certain position by clicking the mouse, or with the help of the keyboard.

Using the arrow keys, you can jump to the nearest cursor position in the selected direction at any time. All cursor positions, including the text fields, can be accessed this way. If the last cursor position is selected, then the <up> or

<down> arrow keys can be used to select the last cursor position of the previous or subsequent network.

An empty network contains only three question marks "???". By clicking behind these, the last cursor position is selected.

"Insert" "Assignment"

Symbol:  **Shortcut:** <Ctrl>+<A>

This command inserts an assignment.

Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4), directly before the selected line cross (Cursor Position 5), or at the end of the network (Cursor Position 6). For an inserted assignment, a selection can be made accompanying the entered text "???", and the assignment can be replaced by the variable that is to be assigned. For this you can also use the Input Assistant.

In order to insert an additional assignment to an existing assignment, use the **"Insert" "Output"** command.

"Insert" "Jump"

Symbol:  **Shortcut:** <Ctrl>+<L>

This command inserts a jump.

Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4), directly before the selected line cross (Cursor Position 5), or at the end of the network (Cursor Position 6).

For an inserted jump, a selection can be made accompanying the entered text "???", and the jump can be replaced by the label to which it is to be assigned.

"Insert" "Return"

Symbol:  **Shortcut:** <Ctrl>+<R>

This command inserts a RETURN instruction.

Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4), directly before the selected line cross (Cursor Position 5), or at the end of the network (Cursor Position 6)

"Insert" "Operator"

Symbol:  **Shortcut:** <Ctrl>+<O>

This command inserts an operator. Insertion takes place according to the selected position.

If an input is selected (Cursor Position 2), then the operator is inserted in front of this input. The first input of this operator is linked to the branch on the left of the selected input. The output of the new operator is linked to the selected input.

If an output is selected (Cursor Position 4), then the operator is inserted after this output. The first input of the operator is connected with the selected output. The output of the new operator is linked to the branch with which the selected output was linked.

If an operator, a function, or a function block is selected (Cursor Position 3), then the old element will be replaced by the new operator. As far as possible, the branches will be connected the same way as they were before the replacement. If the old element had more inputs than the new one, then the unattachable branches will be deleted. The same holds true for the outputs.

If a jump or a return is selected, then the operator will be inserted before this jump or return. The first input of the operator is connected with the branch to the left of the selected element. The output of the operator is linked to the branch to the right of the selected element.

If the last cursor position of a network is selected (Cursor Position 6), then the operator will be inserted following the last element. The first input of the operator is linked to the branch to the left of the selected position.

The inserted operator is always an AND. By selecting and overwriting the text, you can convert this operator into any other operator. With the Input Assistant, you can choose the desired operator from the list of the supported operators. If the new operator has a different lowest number of inputs, then these will be attached. If the new operator has a lesser highest number of inputs, then the last inputs, including those on the branches situated in front of them will be deleted.

All operator inputs that could not be linked will receive the text "???". This text must be clicked and changed into the desired constant or variable.

"Insert" "Function" or "Insert"
"Function Block"

Symbol:  **Shortcut:** <Ctrl>+<F> **(Function)**

Symbol:  **Shortcut:** <Ctrl>+ **(Function block)**

This command inserts a function or a function block. Insertion takes place according to the selected position. First the Input Assistant dialog box, which contains all functions and function blocks, is opened.

The insertion then takes place analogous to the "**Insert**" "**Operator**" command. The assignment of inputs and outputs is also analogous. If there is a branch to

the right of an inserted function block, then the branch will be assigned to the first output. Otherwise the outputs will remain unallocated.

"Insert" "Input"

Symbol:  **Shortcut:** <Ctrl>+<U>

This command inserts an operator input. With many operators, the number of inputs may vary. (For example, ADD can have 2 or more inputs.)

In order to extend such an operator by an input, you need to select the input in front of which you wish to insert an additional input (Cursor Position 1); or you must select the operator itself (Cursor Position 3), if a lowest input is to be inserted.

The inserted input is allocated with the text "???". This text must be clicked and changed into the desired constant or variable. For this you can also use the Input Assistant.

"Insert" "Output"

Symbol: 

This command inserts an additional assignment into an existing assignment. This capability serves the placement of so-called assignment combs; i.e., the assignment of the value presently located at the line to several variables.

If you select the lined cross above an assignment (Cursor Position 5) or the output directly in front of it (Cursor Position 4), then there will be another assignment inserted after the ones already there.

If the line cross directly in front of an assignment is selected (Cursor Position 4), then another assignment will be inserted in front of this one.

The inserted output is allocated with the text "???". This text must be clicked and changed into the desired variable. For this you can also use the Input Assistant.

"Extras" "Negation"

Symbol:  **Shortcut:** <Ctrl>+<N>

With this command you can negate the inputs, outputs, jumps, or RETURN instructions. The symbol for the negation is a small circle at a connection.

If an input is selected (Cursor Position 2), then this input will be negated.

If an output is selected (Cursor Position 4), then this output will be negated.

If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

"Extras" "Set/Reset"

Symbol: 

With this command you can define outputs as Set or Reset Outputs. A grid with Set Output is displayed with [S], and a grid with Reset Output is displayed with [R].

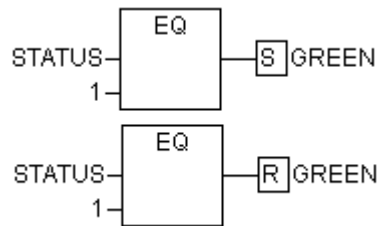


Image 5.10: Set/Reset Outputs in FBD

An Output Set is set to TRUE, if the grid belonging to it returns TRUE. The output now maintains this value, even if the grid jumps back to FALSE.

An Output Reset is set to FALSE, if the grid belonging to it returns FALSE. The output maintains its value, even if the grid jumps back to FALSE.

With multiple executions of the command, the output will alternate between set, reset, and normal output.

"Extras" "Zoom"

Shortcut: <Alt>+<Enter>

With this command a selected POU is loaded into its editor (Cursor Position 3).

If you are dealing with a POU from a library, then the library manager is called up, and the corresponding POU is displayed.

Cutting, Copying, Pasting, and Deleting in FBD

The commands used to "**Cut**", "**Copy**", "**Paste**", and "**Delete**" are found under the "**Edit**" menu item.

If a line cross is selected (Cursor Position 5), then the assignments, jumps, or RETURNS located below the crossed line will be cut, deleted, or copied.

If an operator, a function, or a function block is selected (Cursor Position 3), then the selected object itself, will be cut, deleted, or copied, along with all of the branches dependent on the inputs, with the exception of the first branch.

Otherwise, the entire branch located in front of the cursor position will be cut, deleted, or copied.

After copying or cutting, the deleted or copied part is located on the clipboard and can now be pasted, as desired.

In order to do so, you must first select the pasting point. Valid pasting points include inputs and outputs.

If an operator, a function, or a function block has been loaded onto the clipboard (As a reminder: in this case all connected branches except the first are located together on the clipboard), the first input is connected with the branch before the pasting point.

Otherwise, the entire branch located in front of the pasting point will be replaced by the contents of the clipboard.

In each case, the last element pasted is connected to the branch located in front of the pasting point.



Note: The following problem is solved by cutting and pasting: A new operator is pasted in the middle of a network. The branch located on the right of the operator is now connected with the first input, but must be connected with the second input. You can now select the first input and perform the command "**Edit**" "**Cut**". Following this, you can select the second input and perform the command "**Edit**" "**Paste**". This way, the branch is dependent on the second input.

The Function Block Diagram in the Online Mode

In the Function Block Diagram, breakpoints can only be set to networks. If a breakpoint has been set to a network, then the network numbers field will be displayed in blue. The processing then stops in front of the network where the breakpoint is located. In this case, the network numbers field will become red. Using stepping (single step), you can jump from network to network.

The current value is displayed for each variable. Doubleclicking on a variable opens the dialog box for writing a variable. Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

The new value will turn red and will remain unchanged. If the "**Online**" "**Write values**" command is given, then all variables are placed in the selected list and are once again displayed in black.

The flow control is run with the "**Online**" "**Flow control**" command. Using the flow control, you can view the present values that are being carried in the networks over the connecting lines. If the connecting lines do not carry Boolean values, then the value will be displayed in a specially inserted field. If the lines carry Boolean values, then they will be shaded blue in the event that they carry TRUE. By this means, you can accompany the flow of information while the PLC is running.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.2 The Ladder Editor

This is how a POU written in the LD appears in the **907 AC 1131** editor:

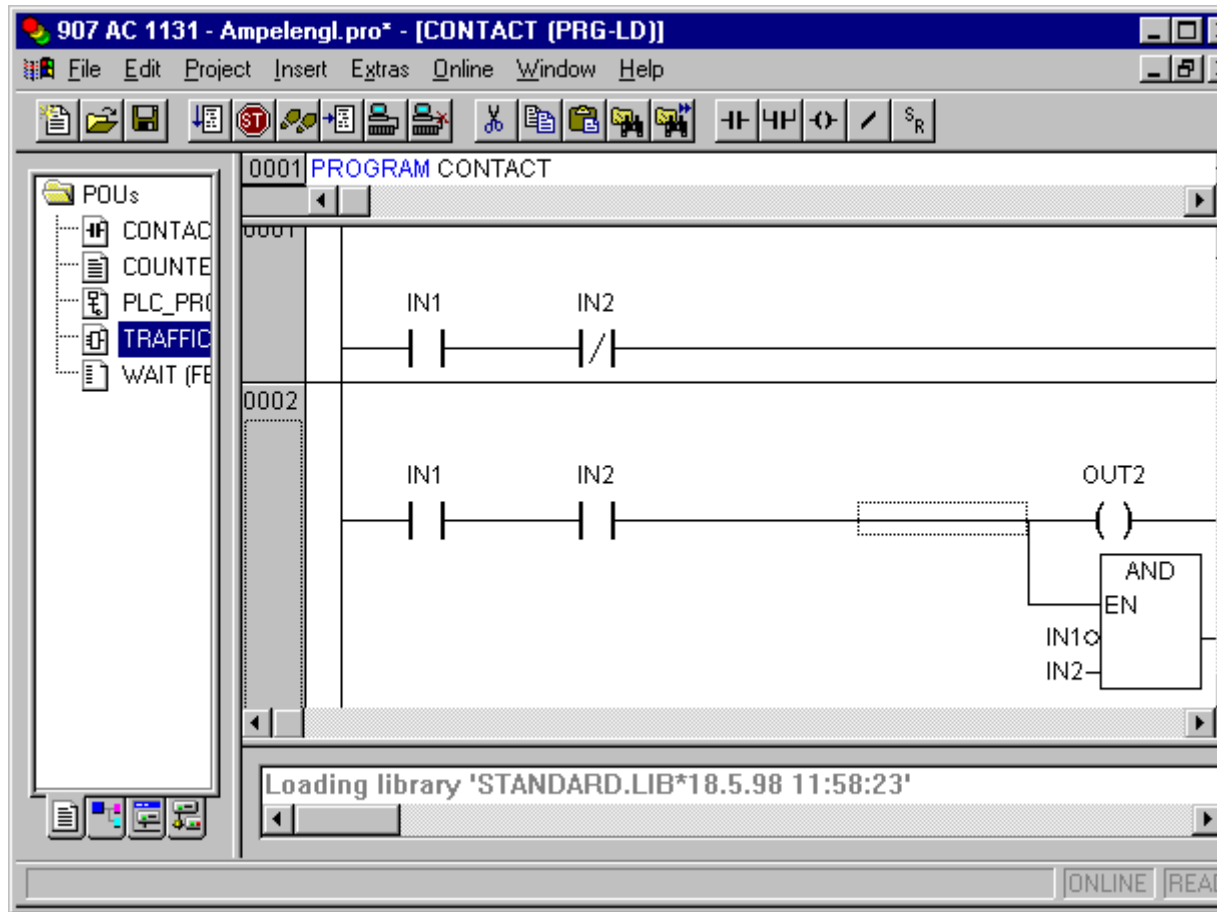


Image 5.11: POU in the Ladder Diagram

All editors for POU's consist of a declaration part and a body. These are separated by a screen divider.

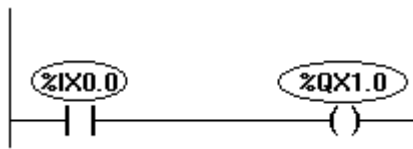
The LD editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the elements, see Ladder Diagram (LD).

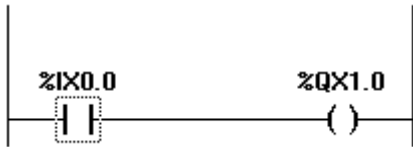
Cursor Positions in the LD Editors

The following locations can be cursor positions, in which the function block and program accessing can be handled as contacts. POU's with EN inputs and other POU's connected to them are treated the same way as in the Function Block Diagram. Information about editing this network part can be found in Chapter on the FBD Editor.

1. Every text field (possible cursor positions framed in black)



2. Every Contact or Function Block



3. Every Coil



4. The Connecting Line between the Contacts and the Coils.



The Ladder Diagram uses the following menu commands in a special way:

"Insert" "Contact"

Symbol:  **Shortcut:** <Ctrl>+<O>

Use this command in the LD editor in order to insert a contact in front of the marked location in the network.

If the marked position is a coil (Cursor Position 3) or the connecting line between the contacts and the coils (Cursor Position 4), then the new contact will be connected serially to the previous contact connection.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the Input Assistant.

"Insert" "Parallel Contact"

Symbol:  **Shortcut:** <Ctrl>+<R>

Use this command in the LD editor to insert a contact parallel to the marked position in the network.

If the marked position is a coil (Cursor Position 3) or the connection between the contacts and the coils (Cursor Position 4), then the new contact will be connected in parallel to the entire previous contact connection.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the Input Assistant.

"Insert" "Function Block"

Shortcut: <Ctrl>+

You can use this command to open a dialog box for selecting a function block or a program. You can select between user-defined or standard (default) POU's.

The selected POU is inserted, according to the same rules used to insert a contact. In both cases, the first input of the POU is set on the input connection, and the first output is set on the output connection. For this reason both of these variables definitely must be of the BOOL type. All other POU inputs and outputs are occupied with the text "???". These default settings can be changed to other constants, variables, or addresses. For this you can also use the Input Assistant.

"Insert" "Coil"

Symbol:  **Shortcut:** <Ctrl>+<L>

You can use this command in the LD editor to insert a coil in parallel to the previous coils.

If the marked position is a connection between the contacts and the coils (Cursor Position 4), then the new coil will be inserted as the last. If the marked position is a coil (Cursor Position 3), then the new coil will be inserted directly above it.

The coil is given the text "???" as a default setting. You can click on this text and change it to the desired variable. For this you can also use the Input Assistant.

POUs with EN Inputs

If you want to use your LD network as a PLC for calling up other POU's, then you must merge a POU with an EN input. Such a POU is connected in parallel to the coils. Beyond such a POU you can develop the network further, as in the Function Block Diagram. You can find the commands for insertion at an EN POU under the menu item **"Insert" "Insert at Blocks"**

An operator, a function block, or a function with EN input performs the same way as the corresponding POU in the Function Block Diagram, except that its execution is controlled on the EN input. This input is annexed at the connecting

line between coils and contacts. If this connection carries the information "On", then the POU will be evaluated.

If a POU has been created once already with EN input, then this POU can be used to create a network. This means that data from usual operators, functions, and function blocks can flow in an EN POU and an EN POU can carry data to such usual POUs.

If, therefore, you want to program a network in the LD editor, as in FBD, you only need first to insert an EN operator in a new network. Subsequently, from this POU, you can continue to construct from your network, as in the FBD editor. A network thus formed will perform like the corresponding network in FBD.

"Insert" "Operator with EN"

Use this command to insert an operator with EN input into a LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new operator is inserted in parallel to the coils and underneath them; it contains initially the designation AND. If you wish, you can change this designation to another one. For this you can also use the Input Assistant.

"Insert" "Function Block with EN"

With this command you can insert a function block with EN input into a LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new function block is inserted in parallel to the coils, below them. From the Input Assistant dialog box that appears, you can select whether to insert a user-defined, or a standard (default) function block.

"Insert" "Function with EN"

With this command you can insert a function with EN input into an LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new function is inserted in parallel to the coils, below them. From the Input Assistant dialog box that appears, you can select whether to insert a user-defined, or a standard function block.

"Insert" "Insert at Blocks"

With this command you can insert additional elements into a POU that has already been inserted (also a POU with EN input). The commands below this menu item can be executed at the same cursor positions as the corresponding commands in the Function Block Diagram (See Chapter 5.7).

With **Input** you can add a new input to the POU.

With **Output** you can add a new output to the POU.

With **Operator** you can add a new operator to the POU, whose output is deposited onto the selected input.

With **Assignment** you can add an assignment to the selected input or output.

With **Function** you can add a function to the selected input.

With **Function Block** you can add a function block to the selected input.

"Insert" "Jump"

With this command you can insert a parallel jump in the selected LD editor, in parallel, at the end of the previous coils. If the incoming line delivers the value "On", then the jump will be executed to the indicated label.

The marked position must be the connection between the contacts and the coils(Cursor Position 4) or a coil (Cursor Position 3).

The jump is present with the text "???". You can click on this text and make a change in the desired label.

"Insert" "Return"

In the LD editor, you can use this command to insert a Return instruction in parallel at the end of the previous coils. If the incoming line delivers the value "On," then the processing of the POU in this network is broken off.

The marked position must be the connection between the contacts and the coils(Cursor Position 4) or a coil (Cursor Position 3).

"Extras" "Paste after"

Use this command in the LD editor to insert the contents of the clipboard as a serial contact after the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

"Extras" "Paste below"

Shortcut: <Ctrl>+<U>

Use this command in the LD editor to insert the contents of the clipboard as parallel contact below the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

"Extras" "Paste above"

Use this command in the LD editor to insert the contents of the clipboard as parallel contact above the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

"Extras" "Negate"

Symbol:  **Shortcut:** <Ctrl>+<N>

Use this command to negate a contact, a coil, a jump or return instruction, or an input or output of EN POUs at the present cursor position (Cursor Position 2 and 3).

Between the parentheses of the coil or between the straight lines of the contact, a slash will appear (**(/)** or **|/|**). If there are jumps, returns, or inputs or outputs of EN POUs, a small circle will appear at the connection, just as in the FBD editor.

The coil now writes the negated value of the input connection in the respective Boolean variable. Right at this moment, a negated contact switches the status of the input to the output, if the respective Boolean variable carries the value FALSE.

If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

"Extras" "Set/Reset"

If you execute this command on a coil, then you will receive a Set Coil. Such a coil never overwrites the value TRUE in the respective Boolean variable. This means that once you have set the value of this variable to TRUE, it will always remain at TRUE. A Set Coil is designated with an "S" in the coil symbol.

If you execute this command once again, then you will be given a Reset Coil. Such a coil never overwrites the value FALSE in the respective Boolean variable. This means that once you have set the value of this variable to FALSE, it will always remain at FALSE. A Reset Coil is designated with an "R" in the coil symbol.

If you execute this command repeatedly, the coil will alternate between set, reset and normal coil.

The Ladder Diagram in the Online Mode

In Online mode, the contacts and coils in the Ladder Diagram that are in the "On" state are colored blue. Likewise, all lines over which the "On" is carried are also colored blue. At the inputs and outputs of function blocks, the values of the corresponding variables are indicated.

Breakpoints can only be set on networks; by using stepping, you can jump from network to network.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

5.3.3 The Sequential Function Chart Editor

This is how a POU written in the SFC appears in the **907 AC 1131** editor:

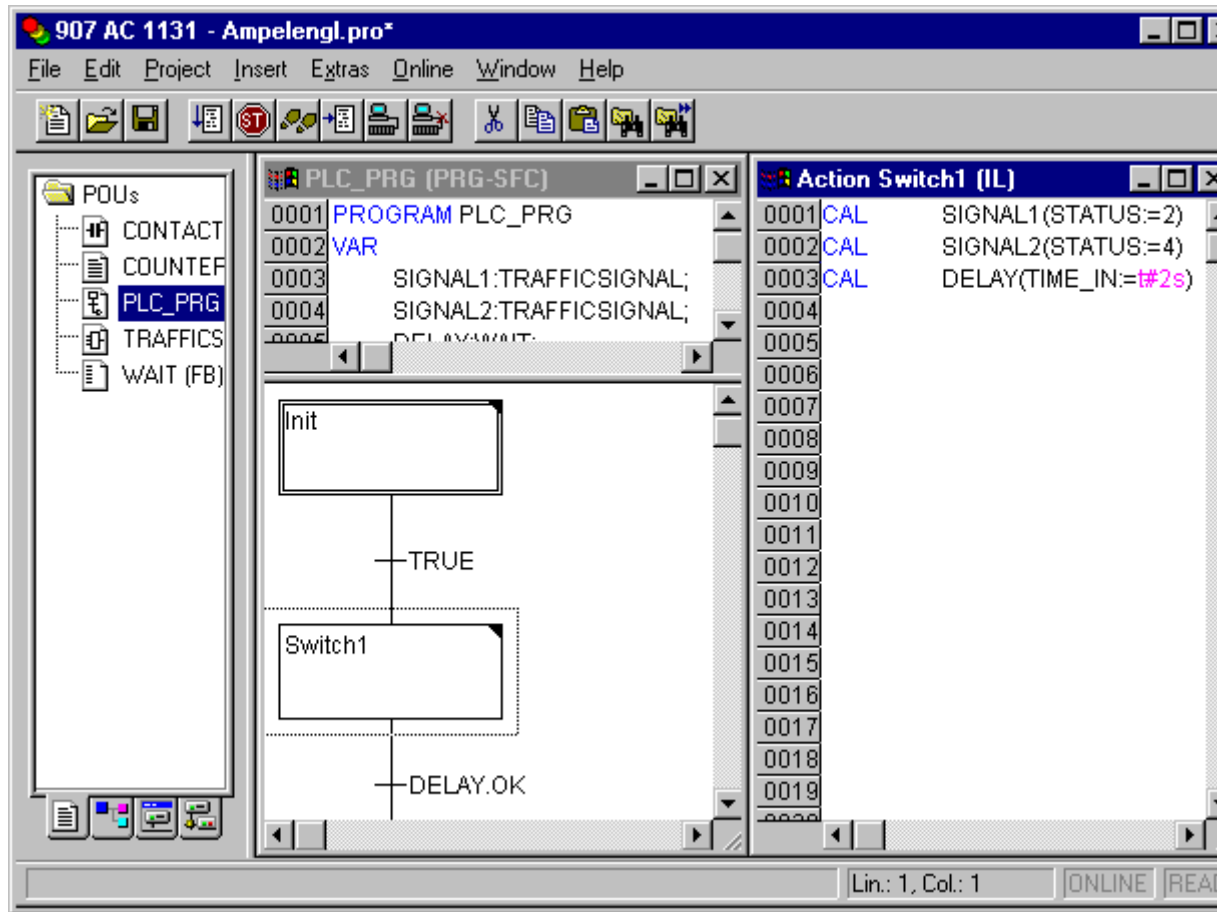


Image 5.12: Sequential Function Chart Editor with an opened Action

All editors for POU's consist of a declaration part and a body. These are separated by a screen divider.

The Sequential Function Chart editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

For information about the Sequential Function Chart, see Sequential Function Chart.

The editor for the Sequential Function Chart must agree with the particulars of the SFC. In reference to these, the following menu items will be of service.

Marking Blocks in the SFC

A marked block is a bunch of SFC elements that are enclosed in a dotted rectangle. (In the example somewhat above, the step is marked Shift1.)

You can select an element (a step, a transition, or a jump) by pointing the mouse on this element and pressing the left mouse button, or you can use the arrow keys. In order to mark a group of several elements, press <Shift> for a block already marked, and select the element in the lower left or right corner of

the group. The resulting selection is the smallest cohesive group of elements that includes both of these elements.

Observe that all commands can only be executed, if they do not contradict the conventions of the language.

"Insert" "Step Transition
(before)"

Symbol:  **Shortcut:** <Ctrl>+<T>

This command inserts a step in the SFC editor followed by a transition in front of the marked block.

"Insert" "Step Transition (after)"

Symbol:  **Shortcut:** <Ctrl>+<E>

This command inserts a step in the SFC editor followed by a transition after the first transition in the marked block.

"Insert" "Alternative Branch
(right)"

Symbol:  **Shortcut:** <Ctrl>+<A>

This command inserts an alternative branch in the SFC editor as a right branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

"Insert" "Alternative Branch
(left)"

Symbol: 

This command inserts an alternative branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

"Insert" "Parallel Branch (right)"

Symbol:  **Shortcut:** <Ctrl>+<L>

This command inserts a parallel branch in the SFC editor as the right branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step.

"Insert" "Parallel Branch (left)"

Symbol: 

This command inserts a parallel branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step.

"Insert" "Jump"

Symbol:  **Shortcut:** <Ctrl>+<U>

This command inserts a jump in the SFC editor at the end of the branch, to which the marked block belongs. For this the branch must be an alternative branch.

For a inserted jump then the text field 'Step' can be selected and be replaced by the label of the step which is target of the jump.

"Insert" "Transition-Jump"

Symbol: 

This command inserts a transition in the SFC editor, followed by a jump at the end of the selected branch. For this the branch must be a parallel branch.

For a inserted jump then the text field 'Step' can be selected and be replaced by the label of the step which is target of the jump.

"Insert" "Add Entry-Action"

With this command you can add an entry-action to a step. An entry-action is only executed once, right after the step has become active. The entry-action can be implemented in a language of your choice.

A step with an entry-action is designated by an "E" in the bottom left corner.

"Insert" "Add Exit-Action"

With this command you can add an exit-action to a step. An exit-action is only executed once, before the step is deactivated. The exit-action can be implemented in a language of your choice.

A step with an exit-action is designated by an "X" in the lower right corner.

"Extras" "Paste Parallel Branch
(right)"

This command pastes the contents of the clipboard as a right parallel branch of the marked block. For this the marked block must both begin and end with a step. The contents of the clipboard must, likewise, be an SFC block that both begins and ends with a step.

"Extras" "Paste after"

This command pastes the SFC block on the clipboard after the first step or the first transition of the marked block. (Normal copying pastes it in front of the marked block.) This will now be executed, if the resulting SFC structure is correct, according to the language norms.

"Extras" "Zoom Action/Transition"

Shortcut: <Alt>+<Enter>

The action of the first step of the marked block or the transition body of the first transition of the market block is loaded into the editor in the respective language, in which it has been written. If the action or the transition body is empty, then the language must be selected, in which it has been written.

"Extras" "Clear Action/Transition"

With this command you can delete the actions of the first step of the marked block or of the transitions body of the first transition.

If, during a step, you implement either only the action, the entry-action, or the exit-action, then the same will be deleted by the command. Otherwise a dialog box appears, and you can select which action or actions are to be deleted.

If the cursor is located in the action of an IEC step, then only this association will be deleted. If an IEC step with an associated action is selected, then this association will be deleted. During an IEC step with several actions, a selection dialog box will appear.

"Extras" "Step Attributes"

With this command you can open a dialog box in which you can edit the attributes for the marked step.

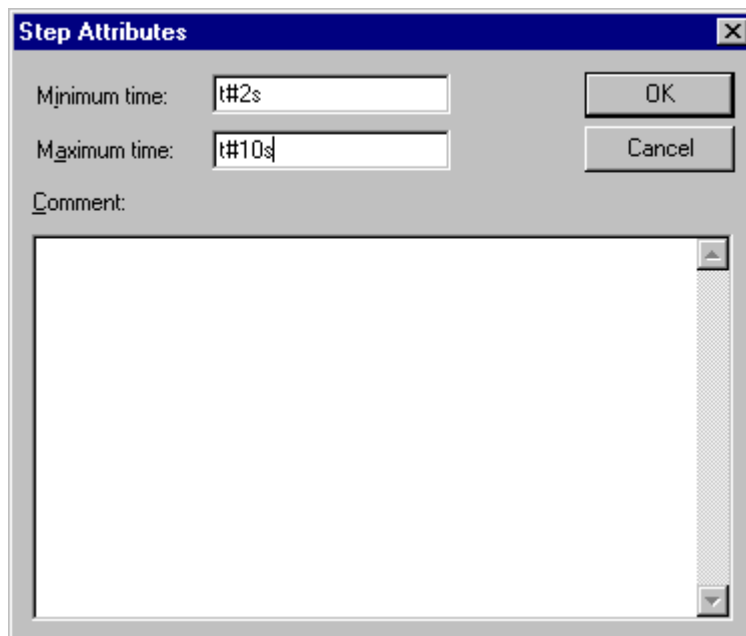


Image 5.13: Dialog Box for Editing Step Attributes

You can take advantage of three different entries in the step attribute dialog box. Under **Minimum Time**, you enter the minimum length of time that the processing of this step should take. Under the **Maximum Time**, you enter the maximum length of time that the processing of this step should take. Note that the entries are of the **TIME** type, so you use a TIME constant (i.e. T#3s) or a variable of the TIME type. Under **Comment** you can insert a comment to the step. With "**Extras**" "**Options**" you can then adjust whether, in the SFC editor, the comments or the time setting for the steps should be displayed. On the right, next to the step, either the comment or the time setting will appear.

If the maximum time is exceeded, SFC flags are set for the user to make inquiries.



The example shows a step whose execution should last at least two, and at the most, ten seconds. In Online mode, there is, in addition to these two times, a display of how long the step has already been active.

"Extras""Time Overview"

With this command you can open a window in which you can edit the time settings of your SFC steps:

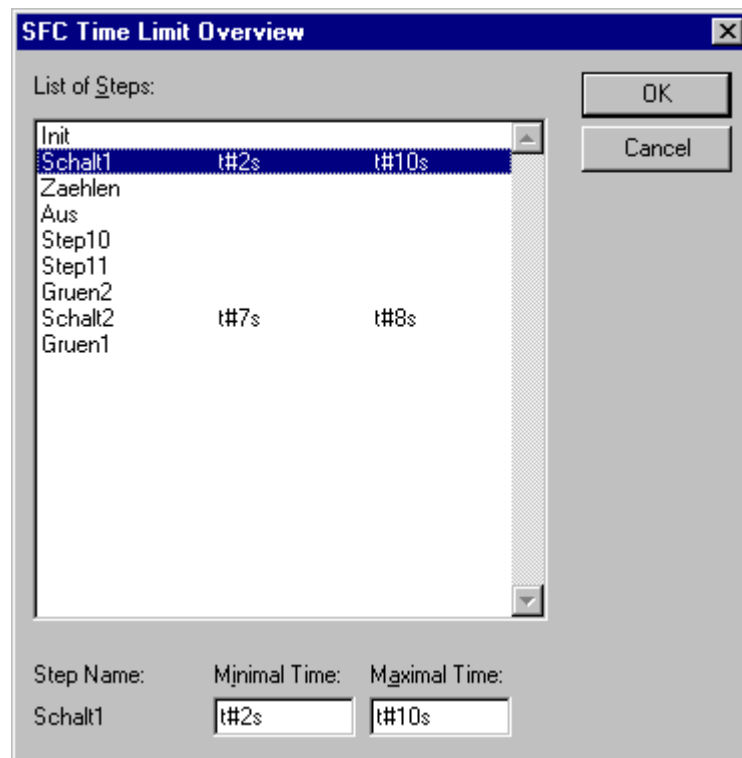


Image 5.14: Time Boundaries Overview for a SFC POU

In the time boundaries overview, all steps of your SFC POU are displayed. If you have entered a time boundary for a step, then the time boundary is displayed to the right of the step (first, the lower limit, then the upper limit). You can also edit the time boundaries. To do so, click on the desired step in the overview. The name **of the step** is then shown below in the window. Go to the **Minimum Time** or **Maximum Time** field, and enter the desired time boundary there. If you close the window with **OK**, then all of the changes will be stored.

In the example, steps 2 and 6 have a time boundary. Shift1 lasts at least two, and at most, ten seconds. Shift2 lasts at least seven, and at most, eight seconds.

"Extras" "SFC-Overview"

With this command you are given a reduced display of the active SFC POU. A check appears before the menu item. For better orientation you can display the names of the steps, transitions, and jumps in tooltips by placing the mouse pointer on an element.

In order to switch back to the normal SFC display, set a marking and then either press <Enter> or select the command once again, and the change will take place.

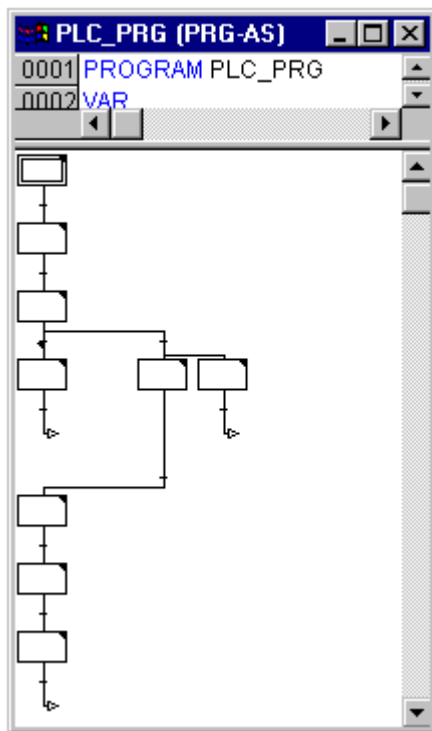


Image 5.15: SFC Overview

"Extras" "Options"

With this command you open a dialog box in which you can set different options for your SFC POU.

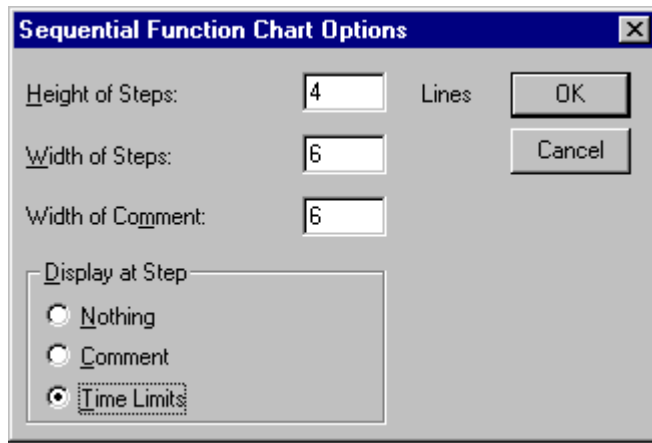


Image 5.16: Dialog Box for Sequential Function Chart Options

In the SFC Options dialog box you can undertake five entries. Under **Step Height**, you can enter how many lines high an SFC step can be in your SFC editor. 4 is the standard setting here. Under **Step Width**, you can enter how many columns wide a step should be. 6 is the standard setting here. You can also preset the **Display at Step**. With this, you have three possibilities: You can either have **Nothing** displayed, or the **Comment**, or the **Time Limits**. The last two are shown the way you entered them in "Extras" "Step Attributes".

"Extras" "Associate Action"

With this command actions and Boolean variables can be associated with IEC steps.

To the right of, and next to the IEC step, an additional divided box is attached, for the association of an action. It is preset in the left field with the qualifier "N" and the name "Action." Both presets can be changed. For this you can use the Input Assistant.

New actions for IEC steps are created in the Object Organizer for an SFC POU with the "**Project**" "**Add Action**" command.

"Extras" "Use IEC-Steps"

Symbol: 

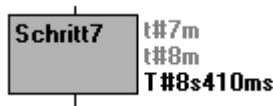
If this command is activated (denoted by a check in front of the menu item and a printed symbol in the Tool bar), then IEC steps will be inserted instead of the simplified steps upon insertion of step transitions and parallel branches.

If this option is switched on, the Init step is set as an IEC step when you create a new SFC POU.

This settings are saved in the file "907 AC 1131 .ini" and are restored when 907 AC 1131 gets started again.

The Sequential Function Chart in the Online Mode

With the Sequential Function Chart editor in Online mode, the currently-active steps will be displayed as blue steps. If you have set it under "**Extras**" "**Options**", then the time management is depicted next to the steps. Under the lower and upper bounds that you have set, a third time indicator will appear from which you can read how long the step has already been active.



In the picture above the step depicted has already been active 8 seconds and 410 milliseconds. The step must, however, be active for at least 7 minutes before the step will be left.

With "**Online**" "**Toggle Breakpoint**" , a breakpoint is set at a step. The processing then stops in front of the execution of this step. The step with the breakpoint is colored light blue.

If several steps are active in a parallel branch, then the active step whose action will be processed next is displayed in red.

If IEC steps have been used, then all active actions in Online mode will be displayed in blue.

SFC also includes support for stepping a step at a time ("**Online**" "**Step Over**"). With this stepping, the program will always proceed to the next step whose action is to be executed.

With "**Online**" "**Step in**" you can step into actions or transitions. Within the transitions or actions, all debugging capabilities of the corresponding editor are at the user's disposal.

If you place the mouse pointer briefly above a variable, then the type and the comment about the variable will be displayed in a Tooltip.

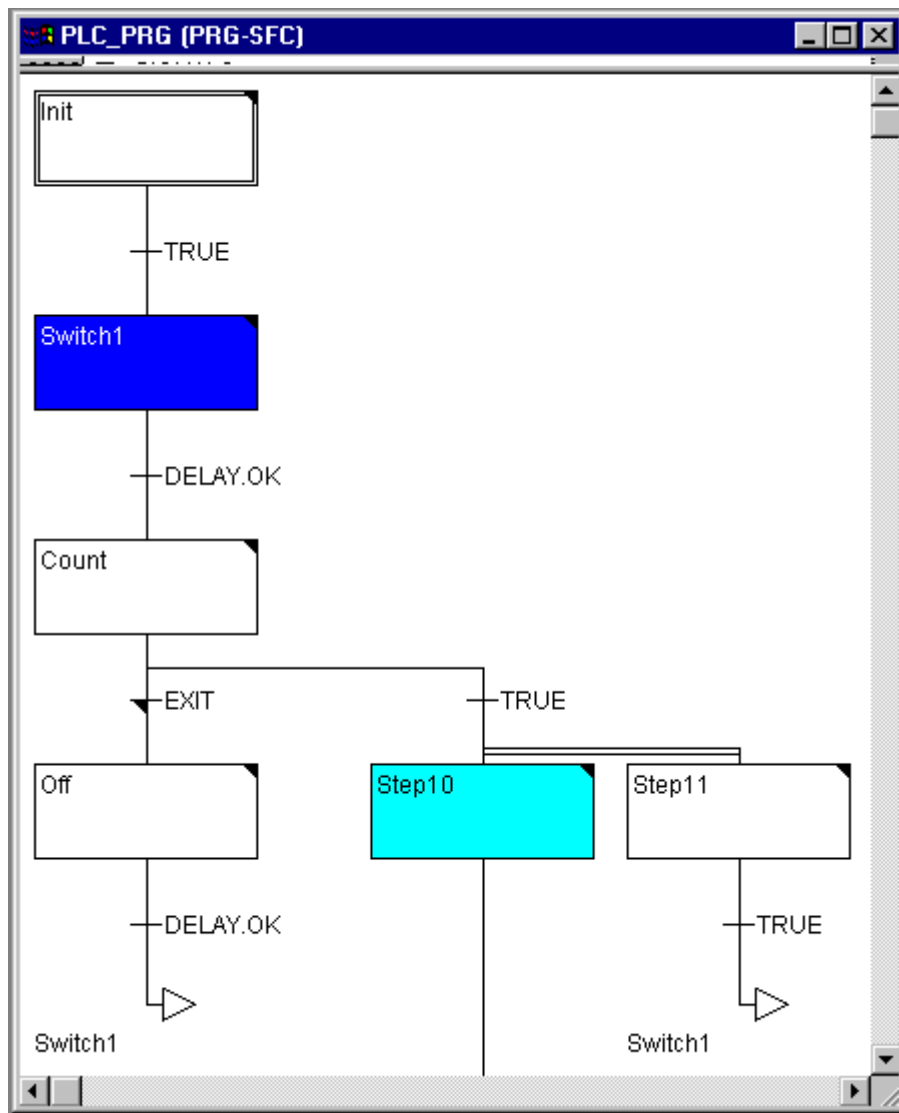


Image 5.17: Sequential Function Chart in the Online Mode with an Active Step (Shift1) and a Breakpoint (Step10)

6.1 Overview of the Resources

In the **Resources** register card of the Object Organizer, there are objects for configuring and organizing your project and for keeping track of the values of the variables:

- **Global Variables** that can be utilized in the entire project; the global variables of the project as well as the libraries.
- **PLC Configuration** for configuring your hardware
- **Task Configuration** for controlling your program control via tasks
- **Sampling Trace** for graphic logging of variable values
- **Watch and Receipt Manager** for indicating and presetting variable values

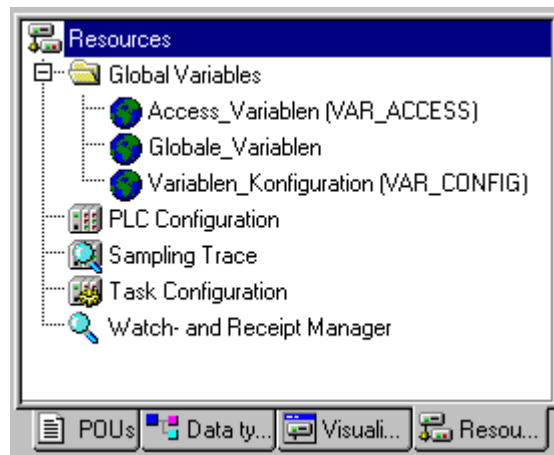


Image 6.1: Resources

6.2 Global Variables

Editing Global Variables

In the Object Organizer, you will find three objects in the **Resources** register card in the **Global Variables** folder (default names of the objects in parentheses).

- Access Variables List (Access Variables)
- Global Variables List (Global Variables)
- Variables Configuration (Variable Configuration)

All variables defined in these objects are recognized throughout the project.


If the global variables folder is not opened up (plus sign in front of the folder), you can open it with a doubleclick <Enter> in the line.

Select the corresponding object. The **"Object Open"** command opens a window with the previously defined global variables. The editor for this works the same way as the declaration editor.

Several Variables Lists


Access variables (Key Word **VAR_ACCESS**), normal global variables (**VAR_GLOBAL**), and variable configurations (**VAR_CONFIG**) must be defined in separate objects.

If you have declared a large number of global variables, and you would like to structure your global variables list better, then you can create further variables lists.

In the Object Organizer, select the **Global Variables** folder or one of the existing  objects with global variables. Then execute the "Project" "Object Add" command. Give the object that appears in the dialog box a corresponding name. With this name an additional object will be created with the key word **VAR_GLOBAL**. If you prefer an object with access variables, or if you want to have a variable configuration, change the corresponding key word to **VAR_ACCESS** or **VAR_CONFIG**.

6.2.1 Access Variables

Access variables assist in communicating with other PLCs (Programmable Logic Controllers).

They are defined in an object between the key words **VAR_ACCESS** and **END_VAR**. In the Object Organizer in the **Resources** register card, the object  **Access_Variables** is generally available. The object can be renamed, and further objects for access variables can be created.

The editor for access variables works the same way as the declaration editor.

Access variables can be used in the entire project.

Syntax:

```
VAR_ACCESS  
  <Identifier> : "<Access Path>" : <Access Mode> <Type>  
END_VAR
```


In the project, an access variable is addressed through its <Identifier>, just like any other variable. The <Access Path> is specified in single quotes. The access path and the implementation are program specific.

The two <Access Modes>, **READ_ONLY** and **READ_WRITE** are available. If no access mode is specified, then **READ_ONLY** will be accepted. This means that the variable in the project can only be read. With **READ_WRITE**, you can also access the variable by writing.

Examples:

```
sensor3 : "control2.sens3": BOOL READ_ONLY;  
counter2 : "control2.count2": UINT;  
displaytext : "control2.text": STRING READ_WRITE;
```


6.2.2 Global Variables

Normal global variables are defined in an object between the key words **VAR_GLOBAL** and **END_VAR**. In the Object Organizer in the **Resources** register card, the object  **Global_Variables** will generally be available. The object can be renamed and further objects for global variables can be created.

To create a new global variables list open the dialog **New global variable list** by 'Project' 'Object' 'Add'. Here you insert a **Name** for the new list. If you want to use an export file (*.esp) or a DCF file (*.dcf), which contains the relevant variable, you can connect it by **Link to file**. Press **Select** to get the standard dialog **Select text file**. DCF files are converted to IEC syntax when called.

If the settings in dialog 'New global variable list' are confirmed by pressing the OK button, the new object is created and the variables' editor opens. The editor for global variables works just the same as the Declaration Editor. Though if an external variable list is used, it cannot be edited in 907 AC 1131 . External variable lists only can be edited external and they will be read each time the project is opened or compiled again.

Global variables can be used in the entire project.

Syntax:

```
VAR_GLOBAL  
  (*Declaration of Variables *)  
END_VAR
```

Global retain variables additionally receive the key word **RETAIN**.

Syntax:

```
VAR_GLOBAL RETAIN  
  (*Declaration of Variables *)  
END_VAR
```

Global constants additionally receive the key word **CONSTANT**.

Syntax:

```
VAR_GLOBAL CONSTANT  
  (*Declaration of Variables *)  
END_VAR
```

6.2.3 Variable Configuration

In function blocks it is possible to specify addresses for inputs and outputs that are not completely defined, if you put the variable definitions between the key words **VAR** and **END_VAR**. Addresses not completely defined are identified with an asterisk.

Here two local I/O-variables are defined, a local-In (%I*) and a local-Out (%Q*).

If you want to configure local I/Os for variables configuration in the Object Organizer in the **Resources** register card, the object **Variable_Configuration**

will generally be available. The object then can be renamed and other objects can be created for the variables configuration.

The editor for variables configuration works like the declaration editor.

Variables for local I/O-configurations must be located between the key words **VAR_CONFIG** and **END_VAR**.

The name of such a variable consists of a complete instance path through which the individual POU's and instance names are separated from one another by periods. The declaration must contain an address whose class (input/output) corresponds to that of the incompletely specified address (%I*, %Q*) in the function block. Also the data type must agree with the declaration in the function block.

Configuration variables, whose instance path is invalid because the instance does not exist, are also denoted as errors. On the other hand, an error is also reported if no configuration exists for an instance variable. In order to receive a list of all necessary configuration variables, the "**All Instance Paths**" menu item in the "**Insert**" menu can be used.

Example:

Assume that the following definition for a function block is given in a program:

```
PROGRAM PLC_PRG
VAR
  Hugo: locio;
  Otto: locio;
END_VAR
```

Then a corrected variable configuration would look this way:

```
VAR_CONFIG
  PLC_PRG. Hugo.loci AT %IX1.0 : BOOL;
  PLC_PRG. Hugo.loco AT %QX0.0 : BOOL;
  PLC_PRG. Otto.loci AT %IX1.0 : BOOL;
  PLC_PRG. Otto.loco AT %QX0.3 : BOOL;
END_VAR
```



Note: Be aware not to describe an output, which is used in the variables configuration, additionally within the project or by a variable (AT declaration). This would not be noticed.

"Insert" "All Instance Paths"

With this command a **VAR_CONFIG** - **END_VAR**-block is generated that contains all of the instance paths available in the project. Declarations already on hand do not need to be reinserted in order to contain addresses already in existence. This menu item can be found in the window for configuration of variables if the project is compiled ("**Project**" "**Rebuild All**").

6.2.4 Document Frame

Document Frame

If a project is to receive multiple documentations, perhaps with German and English comments, or if you want to document several similar projects that use the same variable names, then you can save yourself a lot of work by creating a docuframe with the **"Extras" "Make Docuframe File"** command.

The created file can be loaded into a desired text editor and can be edited. The file begins with the **DOCUFILE** line. Then a listing of the project variables follows in an arrangement that assigns three lines to each variable: a **VAR** line that shows when a new variable comes; next, a line with the name of the variable; and, finally, an empty line. You can now replace this line by using a comment to the variable. You can simply delete any variables that you are unable to document. If you want, you can create several document frames for your project.

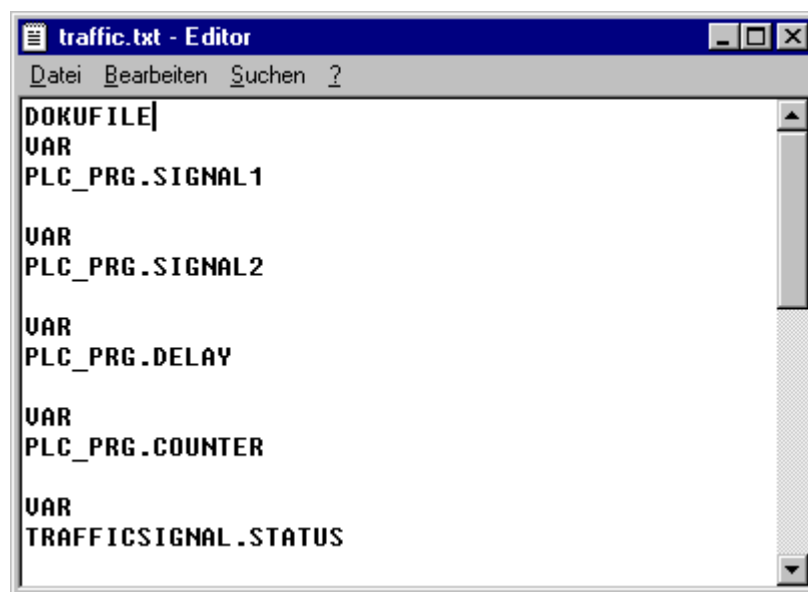


Image 6.2: Windows Editor with Document Frame

In order to use a document frame, give the **"Extras" "Link Docu Frame"** command. Now if you document the entire project, or print parts of your project, then in the program text, there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

"Extras" "Make Docuframe File"

Use this command to create a document frame. The command is at your disposal, whenever you select an object from the global variables.

A dialog box will open for saving files under a new name. In the field for the **name file**, the *.txt extension has already been entered. Select a desired name. Now a text file has been created in which all the variables of your project are listed.

"Extras" "Link Docu File"


With this command you can select a document frame.

The dialog box for opening files is opened. Choose the desired document frame and press **OK**. Now if you document the entire project, or print parts of your project, then in the program text there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

To create a document frame, use the **"Extras" "Make Docuframe File"** command.

6.3 PLC Configuration

The PLC Configuration depends on the configuration of the corresponding hardware. Therefore, at this juncture, only the basic workings of the **CoDeSys** hardware configuration will be described.

The  PLC Configuration is found as an object in the register card **Resources** in the Object Organizer. With the PLC Configuration editor, you must describe the hardware the opened project is established for. For the program implementation, the number and position of the inputs and outputs is especially important. With this description 907 AC 1131 verifies whether the IEC addresses used in the program also actually exist in the hardware.

For inputs and outputs symbolic names can be assigned. The correct notation of an IEC address then looks like that: the symbolic name is followed by an AT and the address where the input/output can be accessed. At least the format of the input or output variable is given (for example: inputname AT %IW1.0 : INT;).

PROFIBUS-DP

907 AC 1131 supports a hardware configuration corresponding to the PROFIBUS DP standard. A profibus system consists of masters and appending slave modules. To qualify them for data exchange they have to be configured. At system start each master parametrizes the slaves which have been assigned to it during configuration. During operation the master sends and/or requests data to or from the respective slaves. The configuration of master and slave modules in 907 AC 1131 is based on the GSD files. The GSD files are delivered by the hardware manufacturer and contain a standardized description of the characteristic properties of a PROFIBUS-DP device. During the PLC configuration only those GSD files are regarded which are stored in the sub-directory PLCONF in the lib-directory. For this reason new GSD files have to be copied to this directory before. Then with the help of dialogues the corresponding devices can be inserted in the configuration tree and their parameters can be edited.

6.3.1 Working in the PLC Configuration

If a new project was created, a minimal PLC Configuration is to be considered.

- Select an element by a mouseclick or use the arrow keys to move the dotted rectangle onto the desired element.
- The words "Hardware Configuration" stand at the heading of the PLC Configuration. Elements that begin with a plus sign are organization elements and contain subelements. To open an element, select the element and doubleclick this plus sign or press <Enter>. You can close opened elements (minus sign in front of the element) the same way.
- If the cursor is located on an element, you can set an edit control box around the name by doubleclicking on the entry or by using the <Space bar>. Then you can change the designation of the input/output.

In order to set up the In-/Output modules use the command **'Extras' 'Properties'**.

- With the **'Insert' 'Insert Element'** command, you can paste the selected element in front of the selected element.
- With the **'Insert' 'Append Subelement'** command, the selected element will be appended to the selected element as the last subelement.
- The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

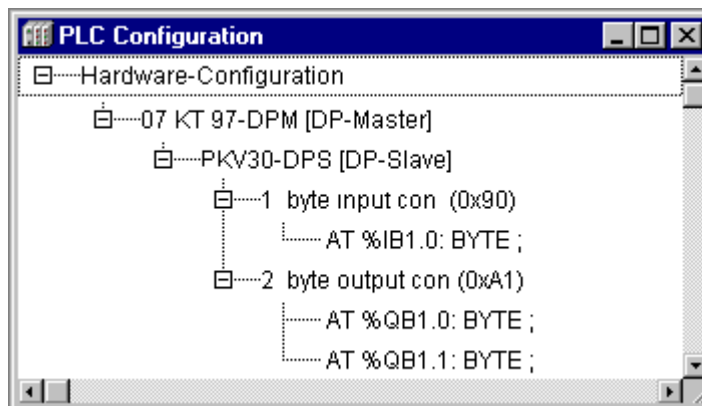


Image 6.3: PLC Configuration, Example with DP master and DP Slave

6.3.2 Doing the PROFIBUS-DP Configuration

Inserting PROFIBUS-DP devices

Select "Hardware-Configuration" which you always find in the first line of the configuration editor. Then use the command "Insert" "Insert subelement" to insert a PROFIBUS-DP device directly below. Corresponding to the hardware

you want to describe you can insert a master as well as a slave coupler at this first level position.

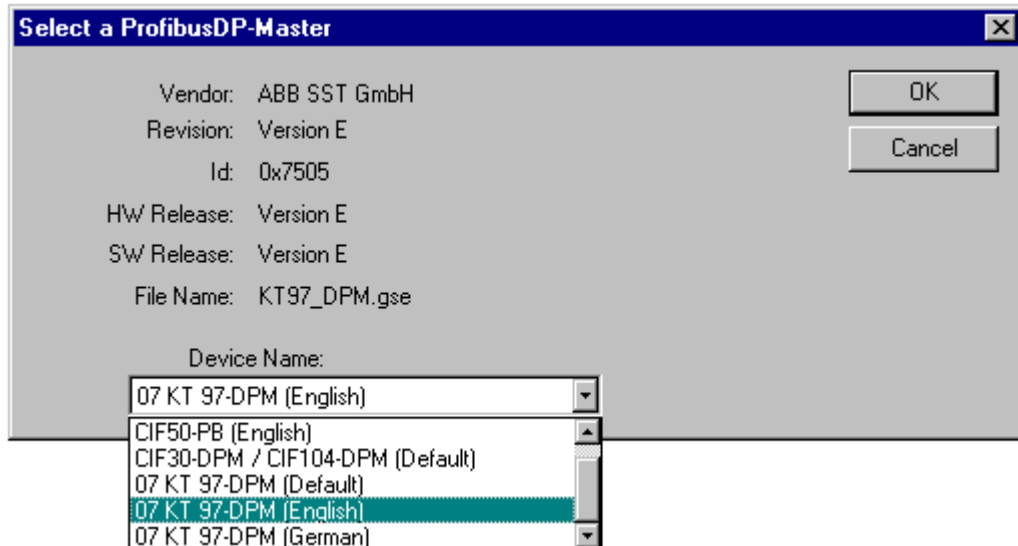


Image 6.4: Select a PROFIBUS-DP Master (resp. Slave)

The insert command opens a dialog **Select a ProfibusDP Master** or **Select a ProfibusDP Slave**. There you find a list of the available devices (**Device Name**). This list results from the selection of GSD files which are stored in the directory PLCCONF within the library directory (In this directory you also have to put the bitmaps which might be used for the design of the dialogues). Select a coupler device from the list by mouseclick. Additionally there is an input field (**Card Number**) for the slot of the coupler which should be addressed. Max. two (master or slave) cards are available, referenced by number 1 (07 KT 97 R0120) and 2 (07 KT 97 R0162). This card number is regarded during the allocation of the IEC addresses for the slave modules which are assigned to this master. Accordingly an IEC address can look like this: %1.IB0. The '1' references the card number.

The following IEC address types can be inserted:

Type		Range
Word	z.B. %QW1.4 (4 = word offset)	%IW1.0 - %IW1.1792 %IW2.0 - %IW2.1792
Byte	z.B. %IB2.3 (3 = byte offset)	%IB1.0 - %IB1.3583 %IB2.0 - %IB2.3583



Note: Bit addresses only can be used in the SPS program:

Bit	z.B. %IX1.0.15 (0 = word number, 15 = bit number)	%IX1.0.0 - %IX1.1792.15 %IX2.0.0 - %IX2.1792.15
-----	---	--

The dialog also shows some basic data of the chosen device, which are given in the GSD file: manufacturer, revision, id number, HW and SW release number, GSD file name. After closing the dialog by **OK** the module is inserted in the configuration tree.

Below the master module you can insert one or several slaves. For this purpose select the master and then use the command "Insert" as described above.

The PLC configuration as described here in the project and the parameters definition of all PROFIBUS-DP modules will be loaded into the PLC with each download and it will be stored in the flash of the PLC by "Online" "Create boot project (Flash User Program)".

Properties of 07 KT 97 as DP master

The parameters of 07 KT 97 as a master device, described by the GSD file, can be adapted to the actual demands of your configuration. For this purpose select the master in the PLC configuration tree. Use the command "**Extras**" "**Properties**" (or the right mouse key, "Properties") to open a dialog, titled with the masters device name. Here, on two registers, you can edit the **Standard Parameters** and the **Bus Parameters** of the module. These parameters result of the settings given in the GSD file.

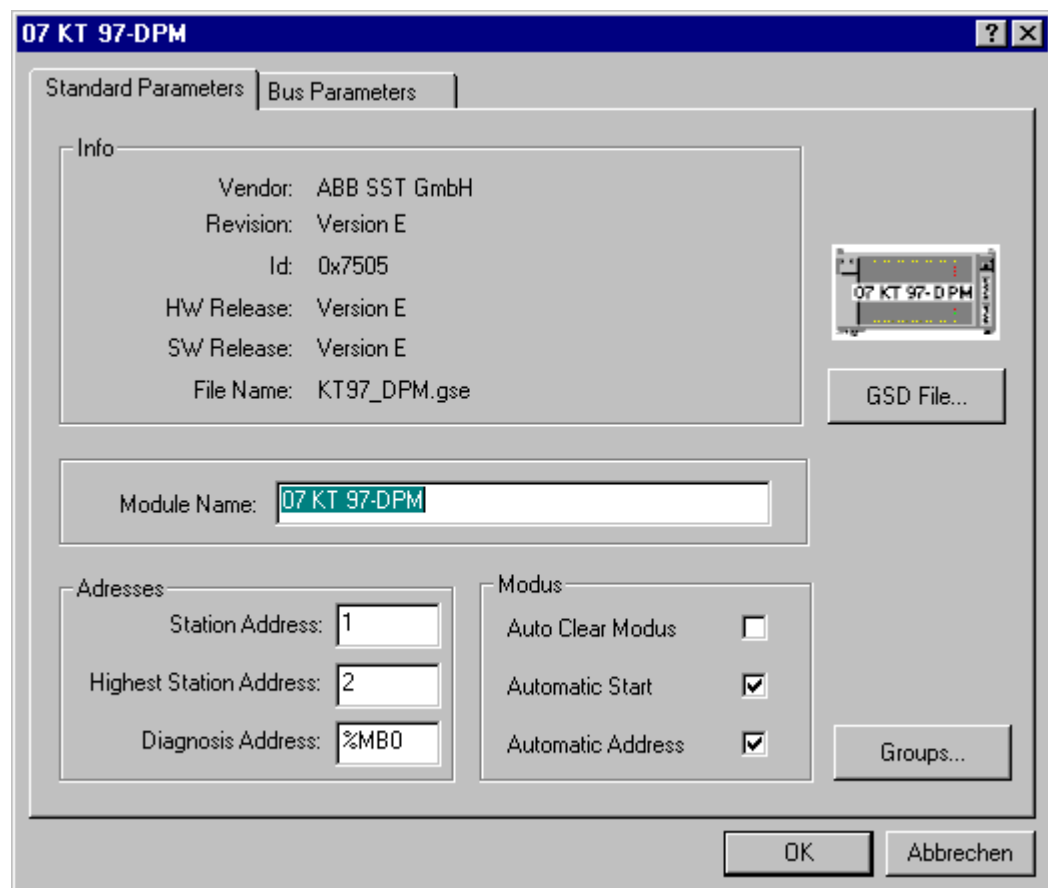


Image 6.5: Properties of the DP Master (Standard Parameters)

- The **Standard Parameters** of a master module

Info	Vendor, Revision, Id, HW and SW Release number, GSD filename
Modul Name	can be edited
Addresses	<p>Station Address: the allowed range is 0-126, each module new inserted automatically gets the next higher address (please regard: address 126 is the DP slave default address). Manual input is possible, there is a check for addresses used twice.</p> <p>Highest Station Address: number of the highest allocated station address, you can edit this address number to reduce the GAP range (i.e. the range of addresses, starting at 0, which is run through during search for active bus members)</p> <p>Diagnosis Address: diagnosis data can be called by function POU's (see chapter 'Inserting PROFIBUS-DP devices' for the IEC address types allowed for input)</p>
Mode	<p>Auto Clear Mode: If this mode is activated, all slaves are switched into the save state by the master if one slave reports "not ready for data exchange". Otherwise master and slaves remain in operating state even if one of the slaves is not ready.</p> <p>Automatic Start: Currently not supported. PROFIBUS-DP starts and stops dependent on the RUN/STOP switch</p> <p>Automatic Address: If this mode is activated, the IEC addresses of the PROFIBUS I/Os for the subsequently inserted modules will be awarded automatically in a way that they are in a row and that overlapping is avoided.</p>

Press the button **GSD file** to open the module specific GSD file.

The button **Groups** opens the dialog **Group Properties**. These properties refer to the masters slaves. Up to eight groups with different data exchange mode parameter sets can be established. Define for each group whether it should run in **Freeze Mode** and/or in **Sync Mode**. By assigning the slaves to these groups (see below, Properties of a DP Slave, Groups) the data exchange may be synchronized by a global control command of the master. By a freeze command the modules are caused to 'freeze' their actual input values and to send them synchronously during the subsequent data exchange. By a sync command the slaves are caused to transmit the data, which they receive during the subsequent data exchange from the master, synchronously to the outputs.

To switch on or off the freeze/sync option click on the corresponding position in the table and place a ,X', where you want to switch on the mode. Alternatively

you can use the right mouse button and choose "activate" or "deactivate" from the menu. Furtheron you can edit the **Group Name**.

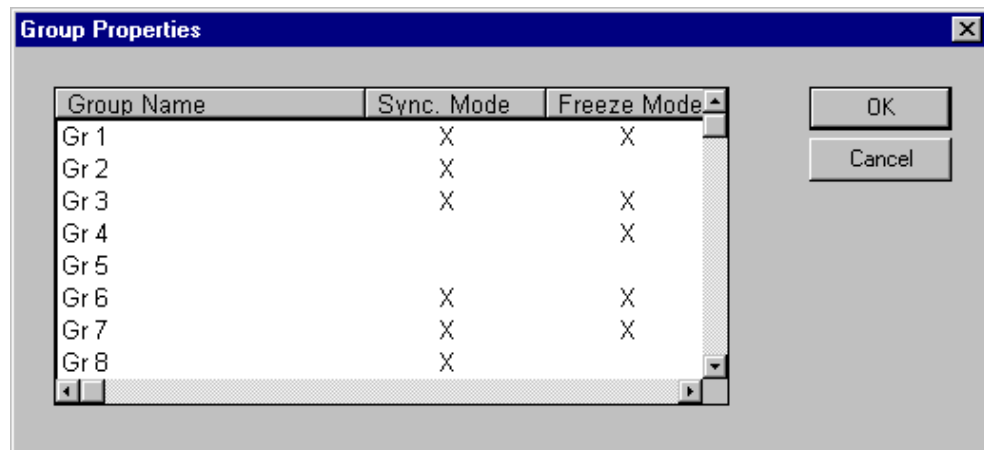


Image 6.6: Properties of a DP Master (Standard Parameters, Group Properties)

- The **Bus Parameters** of a master module

The parameters defined in this dialog describe the communication timing. The particular values are calculated automatically dependent on the baud rate and the settings given in the GSD file. Optionally all parameters can be edited manually. Please note that this should be done only by experienced PROFIBUS users, for undefined behaviour of the system could occur in case of faulty values.

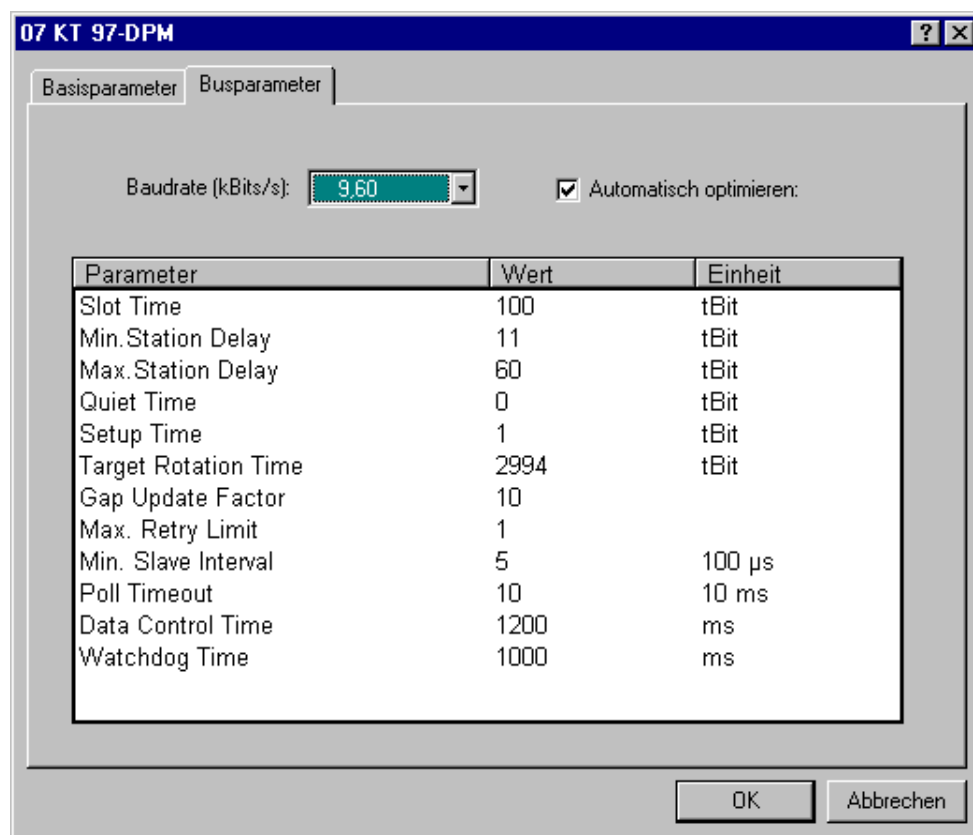


Image 6.7: Properties of a DP Master (Bus Parameters)

Baud Rate	a selection list of the settings given in the GSD file; but you only can feed in a rate, which is supported by all slaves
Optimize	if this option is activated, the parameter values given in the dialog 'Bus Parameters' get optimized automatically regarding the settings in the GSD file
Slot Time	period of time which the master at least waits for the first sign of the slaves response telegram
Min. Station Delay	min. TSDR: time in Tbit ¹ , min. period of time a bus participant has to wait before sending an answer (min. 11 TBit)
Max. Station Delay	max.TSDR: max. period of time within which a slave must answer
Quiet Time	TQUI (Tbit ¹): break period which has to be regarded when NRZ signals have to get converted into other codes. (shift time for repeaters, depends on the baud rate)
Target Rotation Time	TTR (Tbit ¹): defined period of time for the rotation of the token in the bus; this period is the sum of the times for token-holding of all masters in the bus system
Gap Update Factor	GAP update factor G: number of rotations, after which the next newly added active station is searched within the GAP (address range of the master)
Max. Retry Llimit	max. number of retries of the master to call the slave in case of not valid answers
Min Slave Interval	time period between two bus cycles, within which the slave can answer on a request from the master (time base 100 µs); the value entered here has to be coordinated with that given in the GSD file
Poll Timeout	maximum period of time, after which the requestor must have polled the answer from the requestor (DP master class 2) in a master-master communication (time base 1 ms)
Data Control Time	period of time, within which the master indicates its state towards its slaves; at the same time the master checks, whether there has occurred at least one data exchange event during this period and does an update

¹ Tbit: time unit for the transmission of a bit over a PROFIBUS; reciprocal value of the transmission rate; for example: 1 Tbit at 12Mbaud = 1/12.000.000 Bit/sek = 83ns)

of the Data_Transfer_List

Watchdog Time interval of watchdog activity; currently not used (fixed on 400 ms)

Properties of a DP slave

To describe a system completely not only the parameters of a DP master but also the configuration of the assigned slaves has to be done. For this purpose select the slave device in the configuration tree. Use the command "Extras" "Properties" (or the right mouse button, "Properties") to open a dialog, titled with the device name. Here you get four registers to do the settings for the **Standard Parameters**, the **Input/Outputs**, the additional **Parameters** and the **Groups** definitions of the slave.

- The **Standard Parameters** of the slave:

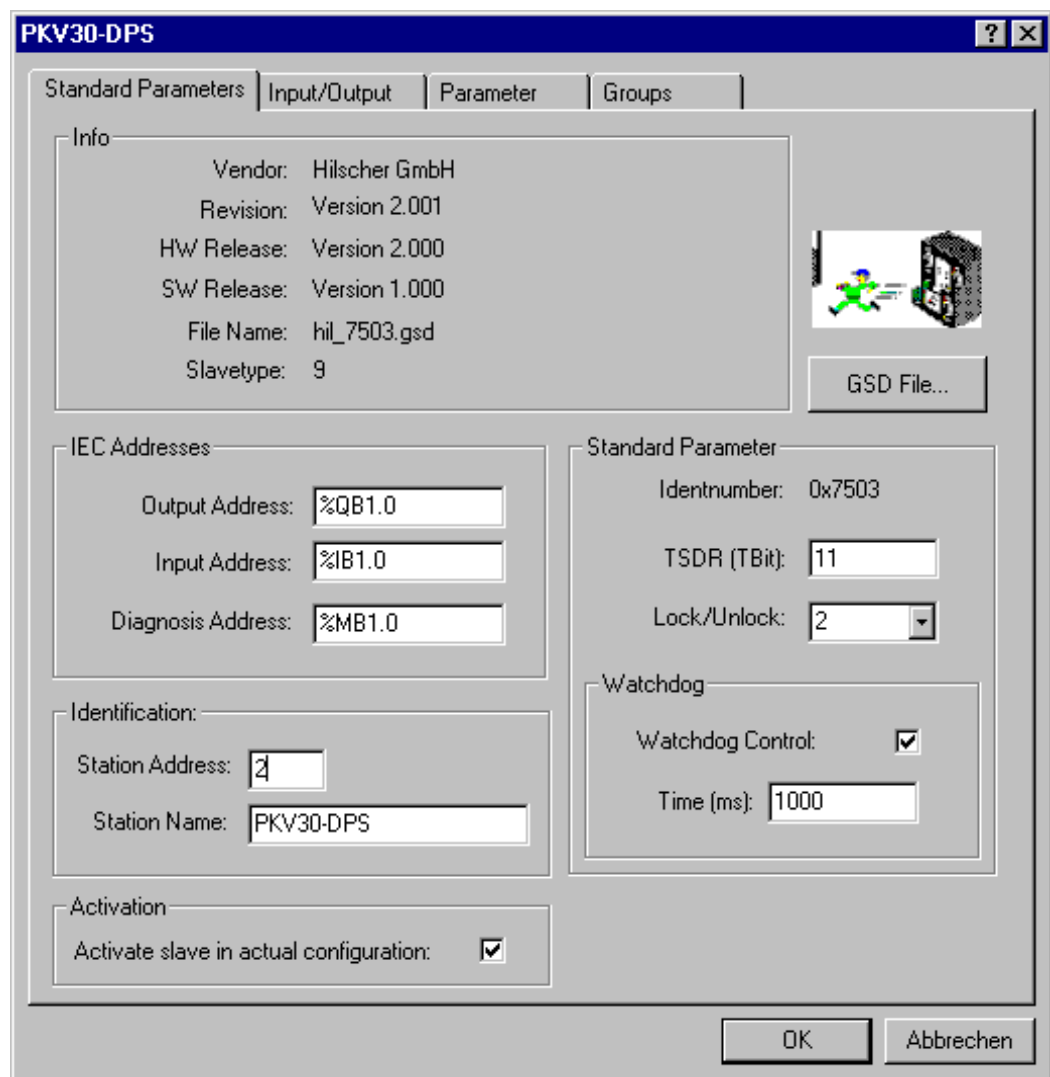


Image 6.8: Properties of a DP Slave (Standard Parameters)

Info	Vendor, Revision, HW and SW Release Number, GSD Filename, Slavetype
IEC Addresses	<p>Output Address, Input Address: If in the base parameters of the DP master the option "addresses automatically" is activated, only the first address allocated in the bus system can be here; all further addresses will be allocated subsequently. If this option is not activated, all addresses can be assigned manually. Please regard: there is no check for double allocations !</p> <p>Diagnosis Address: currently not supported; diagnosis data can be called by function POU (see chapter 'Inserting PROFIBUS-DP devices' for the IEC address types allowed for input)</p>
Standard Parameters	<p>Identnumber: given by the PNO, unique identification number for the device; definite reference between the DP slave and the corresponding GSD file</p> <p>TSDR (Tbit): Time Station Delay Response: period of time a slave at least has to wait before answering to the master (min. 11 Tbit)</p> <p>Lock/Unlock: the slave is locked/unlocked for data exchange with other masters</p> <p>0: min. TSDR and slave specific parameters can be overwritten 1: slave is unlocked for other masters 2: slave is locked for other masters, all parameters are taken over 3: slave is unlocked again for other masters</p>
Identification	Station Address (see 'Properties of a Master Module'), Station Name (= device name)
Activation	slave is activ/not activ in the actual configuration; if "not activ" is selected, the configuration data of the slaves are transmitted to the coupler, but there will be no data exchange over the bus
Watchdog	If Watchdog Control is activated, the shown watchdog time is valid (base time 10 ms); if the slave gets no message from the master within this period of time, it changes back to initialization state

Press the button **GSD file** to show the GSD file of the module.

- The **Input/Outputs** of the slave

The left window in this dialog shows all input, output and input/output modules available with the slave device. The right window shows the chosen configuration of inputs and outputs.

If you are configuring a so-called "modular" slave (i.e. a device which can be configured with various I/O modules), the selection of inputs and outputs can be done as follows: Select a module in the left window and copy it to the right window by a mouse-click on the >>.

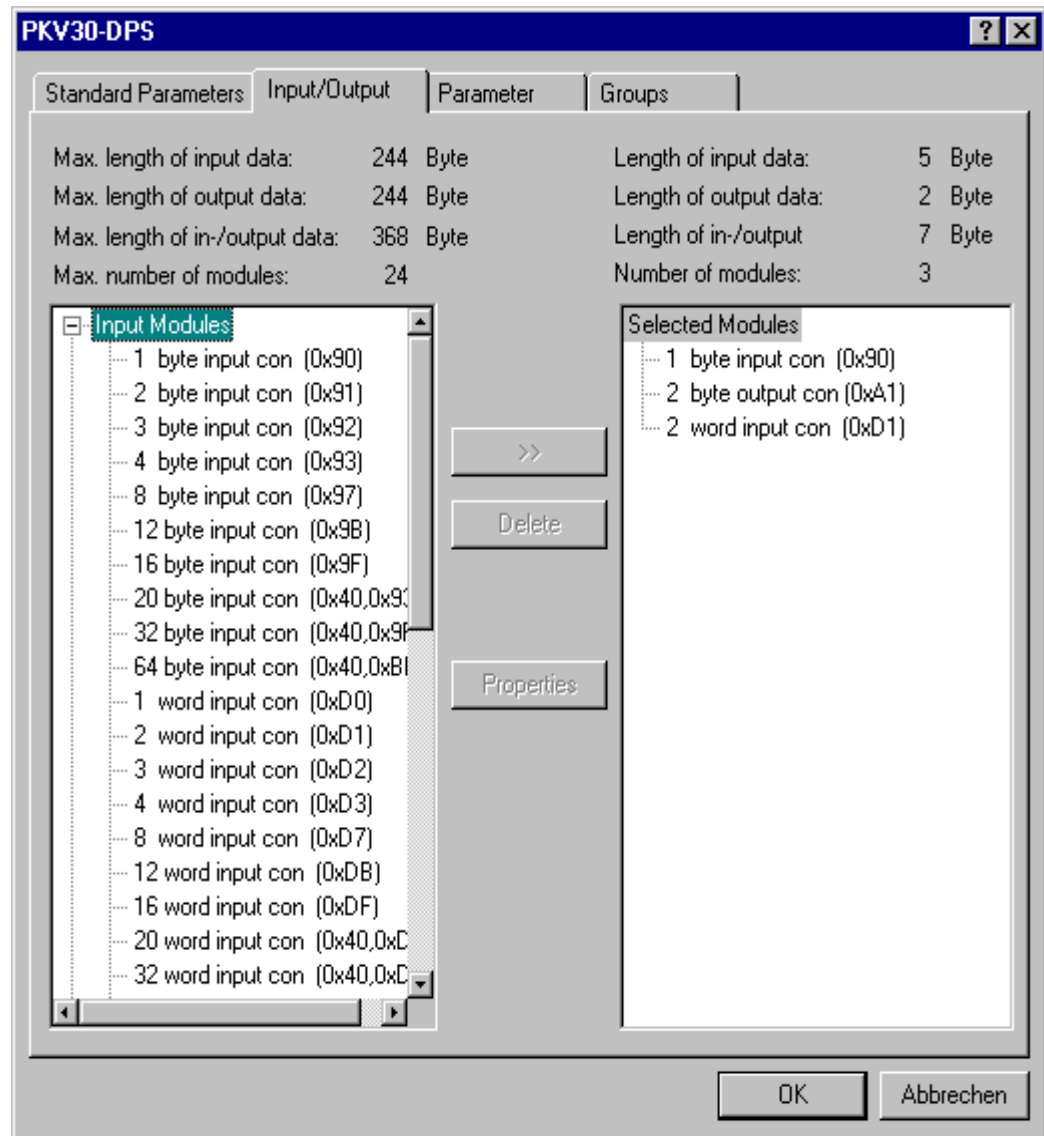


Image 6.9: Properties of a DP Slave (Input/Output)

It is not possible to create the modules list in the same way if you have a "non-modular" slave device. This type of slave first forces a closed list of inputs and outputs in the right window. But you can correct the list by selecting the modules you do not want to use and press the button **Delete** to get them out of your configuration.

Configuring your PLC you have to regard the maximum allowed data length and number of modules, which are defined in the GSD file. To make that more comfortable, these data are shown in the upper part of the dialog. The block on the left shows the allowed max. values, the right one shows the sums actually resulting of the configuration as displayed in the right window. If the actual values exceed the limit, an error message comes up.

Using the button **Properties** you get the dialog **Module Properties** for the module which has been selected last in the left or right window. Here you can see the modules **Name**, the **Config** (code of the module description according to the PROFIBUS standard) and the length of inputs and outputs (**Length Input (Byte)**, **Length Output (Byte)**). In case the GSD file lists additional parameters (**Parameter**) besides the standard set, for those the values (**Value**) and value ranges (**Allowed Values**) are displayed in a table. If the option **Symbolic Names** is activated, the symbolic names are used in this table.

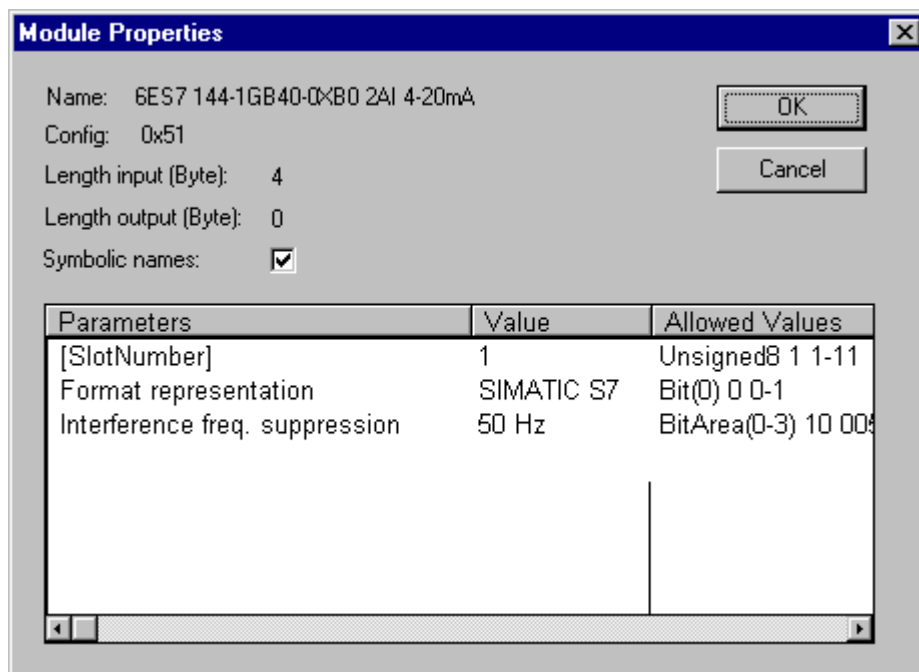


Image 6.10: Properties of a DP Slave (Input/Output, Module Properties)

- Manufacturer Specific (User) **Parameters** of a slave module:

This dialog shows various parameters which may be contained in the GSD file of the device additionally to the standard set of parameters. Their values displayed in the column **Value** can be edited by doubleclick or by using the right mouse button. The range of **Allowed Values** is shown in the right column.

If there are symbolic names defined for the parameters in the GSD file, you can make them used in this table by activating the option **Symbolic names** in the upper right corner of the dialog. Furtheron you get displayed the value of the **Length of user parameters in bytes**.

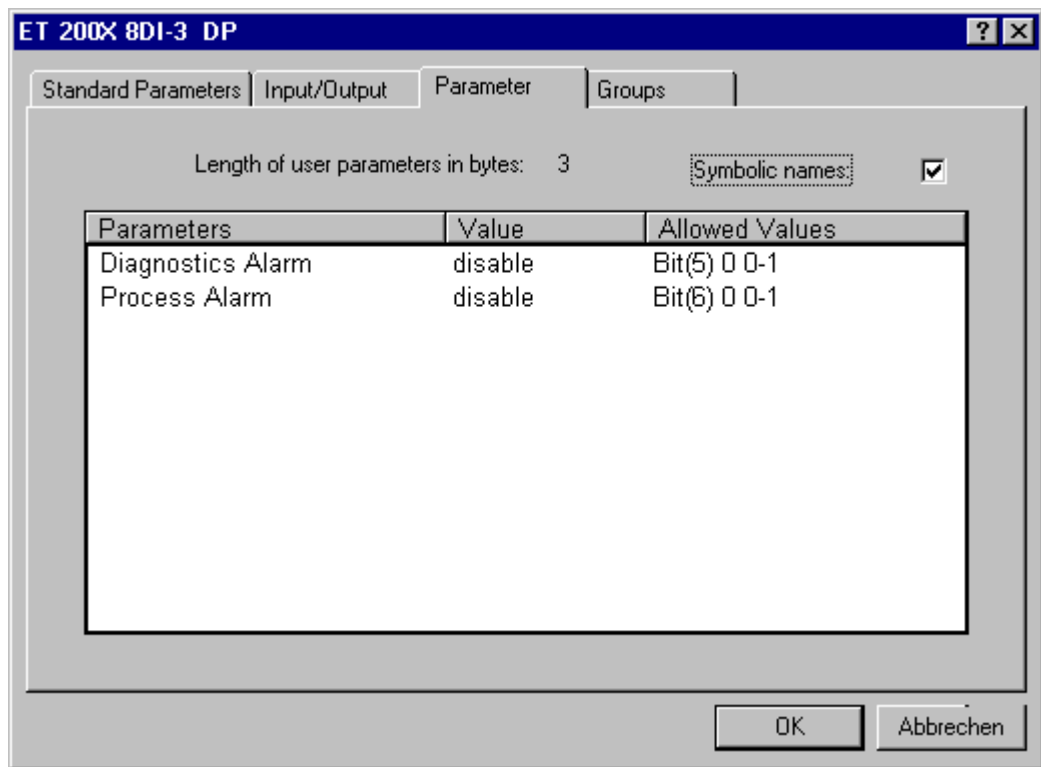


Image 6.11: Properties of a DP Slave (Parameters)

- The assignment of the slave module to **Groups**

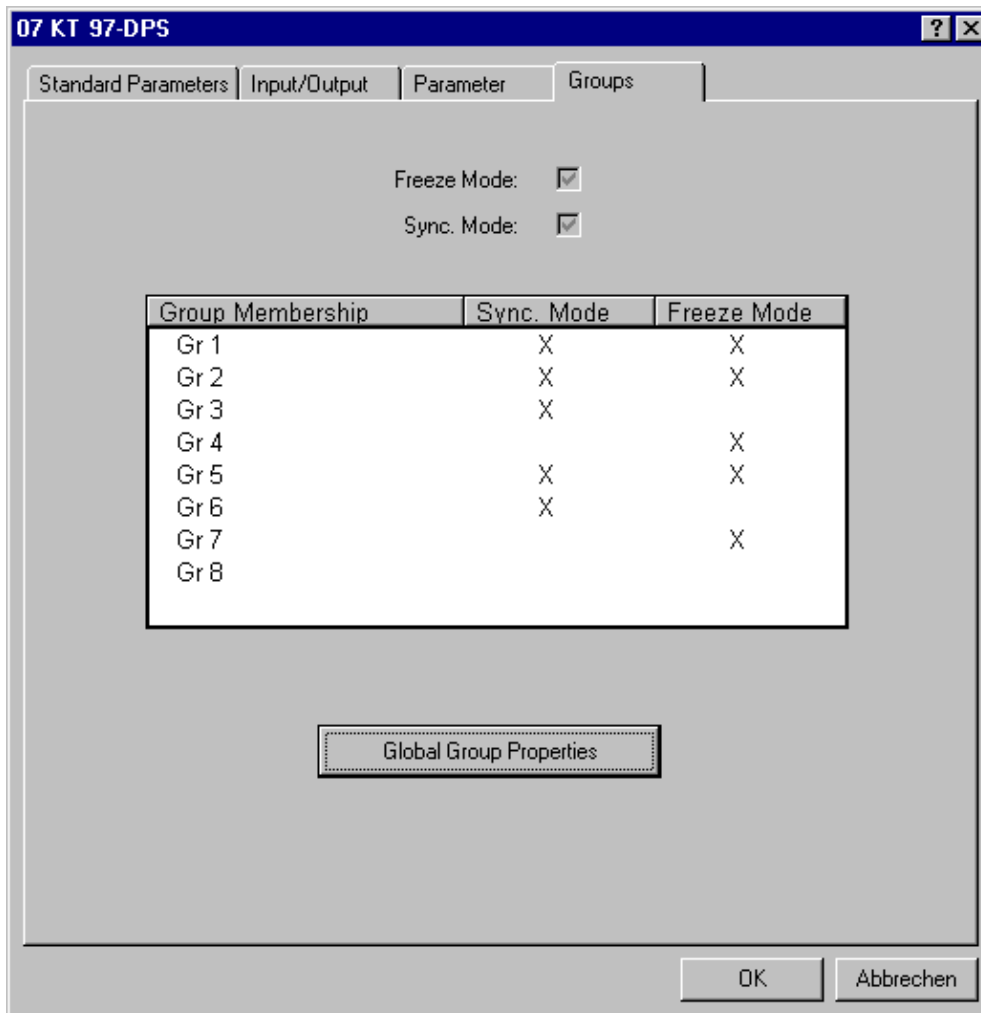


Image 6.12: Properties of a DP Slave (Groups)

In this dialog you can assign the slave to one or several of the eight groups which can be defined with various settings concerning the Sync. Mode and the Freeze Mode. These group definitions can be done in the dialog "Global Group Properties" within the configuration of the master (see chapter 'Properties of 07 KT 97 as DP master', 'Standard Parameters'). The button **Global Group Properties** as well leads to this dialog.

If the slave is assigned to a group, a plus sign appears at the left of the group number in the column **Group Membership**. To assign or to remove the slave to/from a group do the following: Select the group number by mouseclick. Then click once more at the left of the group number or alternatively choose "Insert slave into group" or "Delete slave from group" by the right mouse button.

A slave device only can be assigned to a group, if it supports the properties set for those. The corresponding properties of the slave (**Sync. Mode / Freeze Mode**) are shown above the table. Properties supported by the device are marked with a check (✓).

Properties of 07 KT 97 as a DP Slave

If 07 KT 97 is used in the slave mode the parameters can be adapted to the given requirements in the same way as if used in the master mode. For this purpose in the PLC configuration editor use "**Append Subelement**" "**DP-Slave**" (see ,Inserting PROFIBUS-DP devices'). In the dialog which opens then select 07 KT 97-DPS from the device list (**Device Name**), put in the required **Card Number** (1 for 07 KT 97 R120, 2 for 07 KT 97 R162) and confirm with **OK**. Mark the entry in the configuration tree. With the command "**Extras**" "**Properties**" (or right mouse button, properties) you get a dialog, titled with the device name. Here you find two registers where the **Standard Parameters** and the configuration of the **Input/Outputs** of the slave can be defined.

- The **Standard Parameters** of 07 KT 97 as a DP Slave (used in the slave mode):

Info	Vendor, Revision, HW and SW Release Number, GSD Filename, Slavetype
IEC Addresses	Output Address, Input Address: Input of the start addresses for the input and output data, subsequent addresses are set in ascending order Diagnosis Address: currently not supported; diagnosis data can be called by function POU's (see chapter 'Inserting PROFIBUS-DP devices' for the IEC address types allowed for input)
Identification	Station Address (see ,Properties of a Master Module'), Station Name (= device name)

Press the button **GSD file** to show the GSD file of the module.

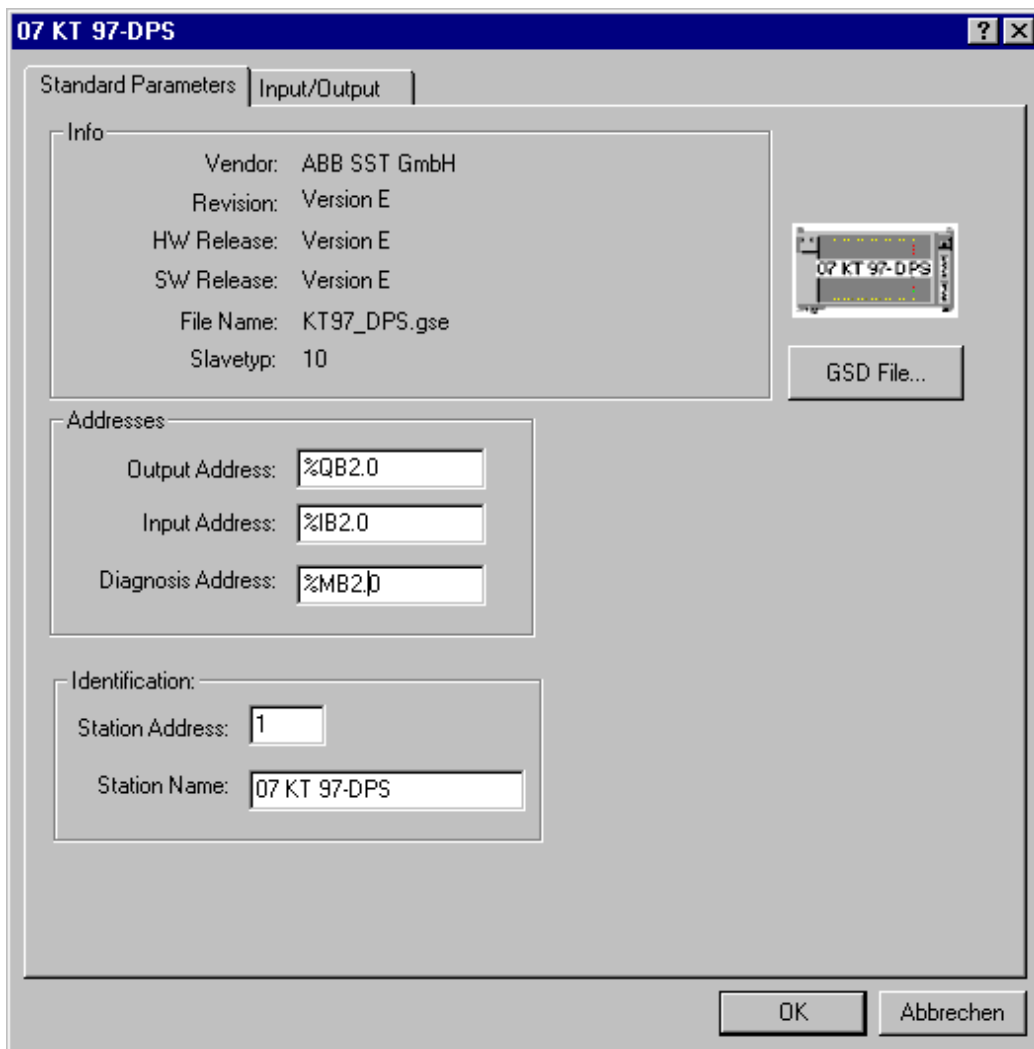


Bild 6.13: Properties of 07 KT 97 as DP-Slave (Standard Parameters)

- **Input/Outputs** of 07 KT 97 as DP Slave (used in the slave mode):

The 07 KT 97 is a modular slave in PROFIBUS DP although its inputs and outputs are not extensible. The modular description is used to reflect the arrangement of virtual modules. By that the highest possible flexibility in I/O-configuration is reached. It is not necessary that the data transferred by PROFIBUS lie directly at the inputs and outputs of the PLC. Each variable used in the program can be sent or received. The selection of virtual modules is done as described in the following: Select the desired input resp. output modul from the list on the left side of the dialog by mouseclick and copy it to the configuration list on the right side by the button >>. Wrong settings can be corrected by selecting the entry in the configuration list and pressing the button **Delete**.

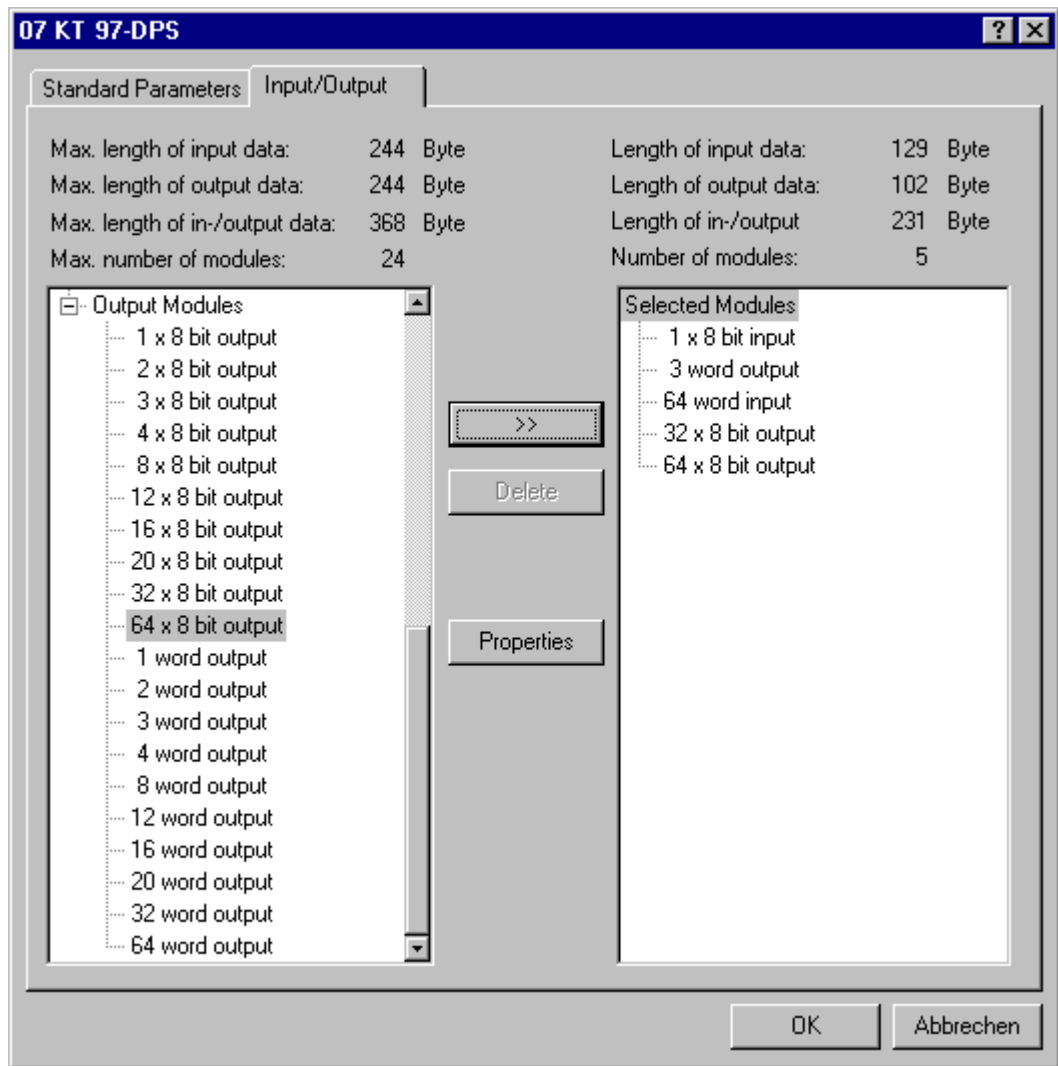


Bild 6.14: Properties of 07 KT 97 as DP Slave (Input/Output)

Configuring your PLC you have to regard the maximum allowed data length and number of modules, which are defined in the GSD file. To make that more comfortable, these data are shown in the upper part of the dialog. The block on the left shows the allowed max. values, the right one shows the sums actually resulting of the configuration as displayed in the right window. If the actual values exceed the limit, an error message comes up.

Using the button **Properties** you get the dialog **Module Properties** for the module which has been selected last in the left or right window. Here you can see the modules **Name**, the **Config** (code of the module description according to the PROFIBUS standard) and the length of inputs and outputs (**Length Input (Byte)**, **Length Output (Byte)**). In case the GSD file lists additional parameters (**Parameter**) besides the standard set, for those the values (**Value**) and value ranges (**Allowed Values**) are displayed in a table. If the option **Symbolic Names** is activated, the symbolic names are used in this table.

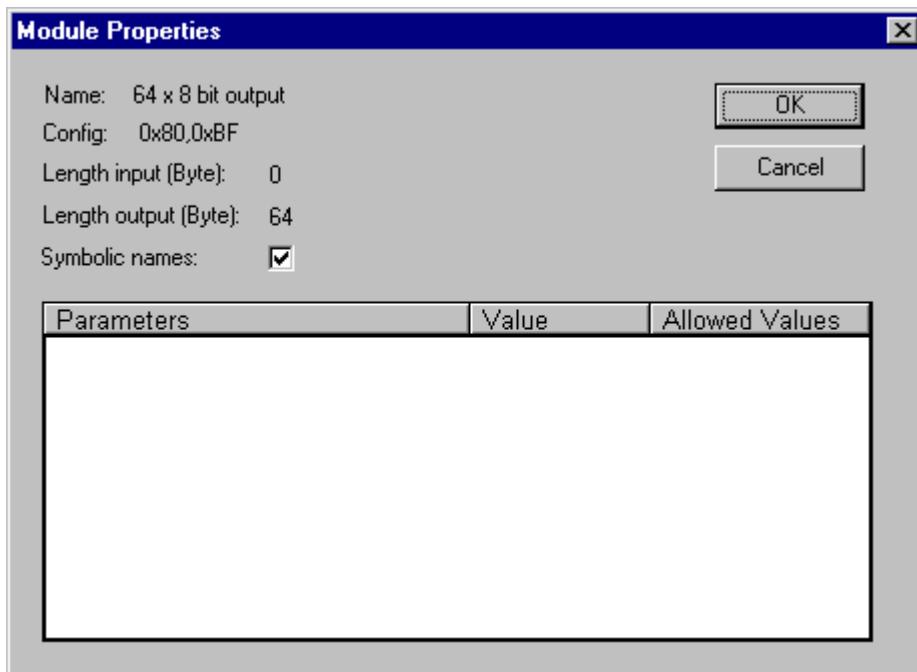



Bild 6.15: Properties of 07 KT 97 as a DP Slave (Input/Output/Module Properties)

It is not necessary to assign the 07 KT 97, when used as a DP slave, to a group (see "Properties of a DP Slave, Groups"). This is done automatically by the corresponding master during the implementing run of the system.

6.4 Task Configuration

In addition to declaring the special PLC_PRG program, you can also control the processing of your project using the task management.

The  Task Configuration is found as an object in the **Resources** register card in the Object Organizer. The task editor contains a series of tasks. The task declaration consists of the name of the task, an entry for the priority the task is to have, and an entry for the condition under which the task is to be executed. This requirement can either be a time interval, according to which the task is to be executed, or a global variable that, in the event it has a rising edge, brings about an execution.

For each task you can now specify a series of programs that will be started by the task. If the task is executed in the present cycle, then these programs will be processed the length of one cycle.

The Task Configuration is displayed in the following form:

- The Task Configuration is located in the first line.
- Underneath and indented from the Task Configuration, you will find a sequence of task entries (with name, priority, interval, and occurrence).
- Below each task entry, there is again a series of program call ups.

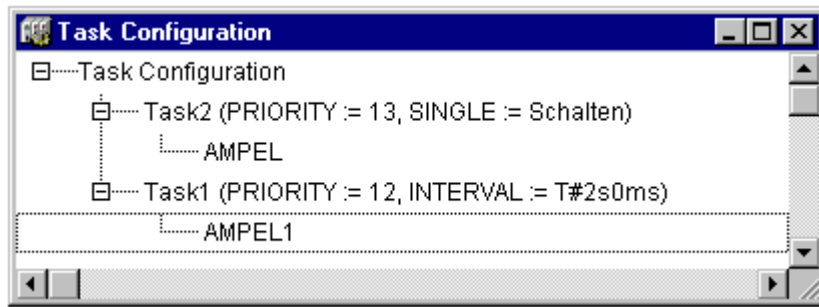


Image 6.16: Example for a Task Configuration

In this example of a Task Configuration, Task2 has a lower priority than Task1. Task1, however, is only executed every two seconds. (The entry under Single is disregarded.) Thus, in this Task Configuration, Task1 is executed every two seconds, and, in between, Task2 can be executed at any time, provided that the global variable "Schalten" has a rising edge.

Which task is being processed?

For the execution, the following rules apply:

- That task is executed, whose condition has been met; i.e., its specified time has expired, or after its condition variable exhibits a rising edge.
- If several tasks have a valid requirement, then the task with the highest priority will be executed.
- If several tasks have valid conditions and equivalent priorities, then the task that has had the longest waiting time will be executed first.
- The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Working in the Task Configuration

- At the heading of the Task Configuration are the words "Task Configuration." If a plus sign is located before the words, then the sequence list is closed. By doubleclicking on the list or pressing <Enter>, you can open the list. A minus sign now appears. By doubleclicking once more, you can close the list again.
- For every task, there is a list of program call-ups attached. Likewise, you can open and close this list the same way.
- With the "Insert" "Insert Task" command, you can insert a task.
- With the "Insert" "Insert Program Call", a program call will be inserted.
- With the "Extras" "Edit Entry" command, you can edit the task characteristics or the program call-up, depending on the selected element.

- By clicking on the task or program name, or by pressing the <Space bar>, you can set an edit control box around the name. Then you can change the designation directly in the task editor.

"Insert" "Insert Task" or "Insert"
"Append Task"

With this command you can insert a new task into the Task Configuration.

If a task is selected, then the "**Insert Task**" command will be at your disposal. The new task will be inserted in front of the cursor. If the words Task Configuration are selected, then the "**Append Task**" is available, and the new task will be appended to the end of the existing list.

The dialog box will open for you to set the task attributes.

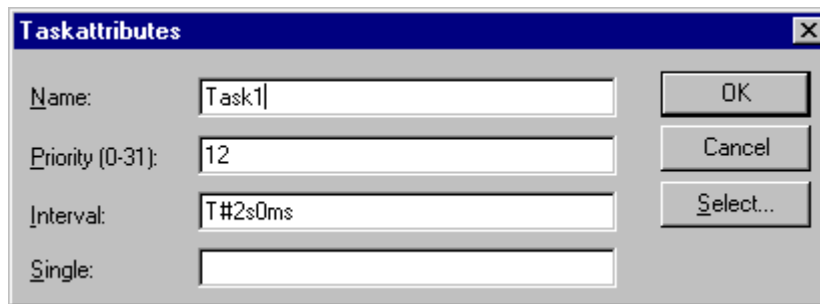


Image 6.17: Dialog Box for Setting Task Attributes

In the dialog box you can enter the desired attributes: the **Name**; the **Priority** (a number between 0 and 31, with the following validities: 0 has the highest, and, 31, the lowest priority); the **Interval** after which the task should be started again; or a variable that, following a raising edge, will cause an execution of the task (in the **Single** field). With the **Select...** button, you can open the Input Assistant to select from the declared variables.

If an entry is on hand for both the interval and for the variable, then only the interval time will be considered for the execution requirement. If an entry has not been made in either of the two fields, then only the priority with the counter will be considered. This means that in every cycle the task will be considered to be executable. It will only cease to be executed if another task of higher priority is likewise executable.

"Insert" "Insert Program Call" or
"Insert" "Append Program Call"

With these commands you will open the dialog box for entering a program call to a task in the Task Configuration.

With **Insert Program Call**", the new program call is inserted in front of the cursor, and with "**Append Program Call**", the program call is appended to the end of the existing list.

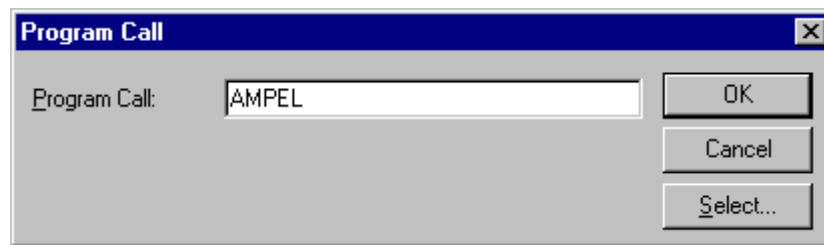


Image 6.18: Dialog box for Program Call Entry

In the field, specify a valid program name for your project, or open the Input Assistant with the **Select** button to select a valid program name. If the selected program requires input variables, then enter these in their usual form and of the declared type (for example, `prg(invar:=17)`).

"Extras" "Edit Entry"

Depending on the element selected, you can use this command in the Task Configuration to open either the dialog box for setting the task attributes (see **"Insert" "Task"**) or the dialog box for entering the program call (see **"Insert" "Program Call"**).

If the cursor is located at the task entry, and there is no list of program calls appended to the task entry, then you open the dialog for setting the doubleclicking on the entry or by pressing <Enter>.

If the cursor is located on an entry for a program call, then you can also open the dialog box for editing the program entry by doubleclicking on the entry.

By clicking on the task or program name, or by pressing the <Space bar>, you can set an edit control box around the name. Then, you can change the designation directly in the task editor.

"Extras" "Set Debug Task"

With this command a debugging task can be set in Online mode in the Task Configuration. The text [DEBUG] will appear after the set task.


The debugging capabilities apply, then, only to this task. In other words, the program only stops at a breakpoint if the program is gone through by the set task.

6.5 Sampling Trace

Sample tracing means that the progression of values for variables is traced over a certain time frame. These values are written in a ring buffer (trace buffer). If the memory is full, then the "oldest" values from the start of the memory will be overwritten. As a maximum, 20 variables can be traced at the same time. A maximum of 500 values can be traced per variable.

Since the size of the trace buffer in the PLC has a fixed value, in the event of very many or very wide variables (DWORD), fewer than 500 values can be traced.

Example: if 10 WORD variables are traced and if the memory in the PLC is 5000 bytes long, then, for every variable, 250 values can be traced.

In order to be able to perform a trace, open the object for  a **Sampling Trace** in the **Resources** register card in the Object Organizer. After this, you must enter the trace variables to be traced. (See "Extras" "Trace Configuration".) After you have sent the configuration with "**Save Trace**" to the PLC and have started the trace in the PLC ("**Start Trace**"), then the values of the variables will be traced. With "**Read Trace**", the final traced values will be read out and displayed graphically as curves.

"Extras" "Trace Configuration"

With this command you will be given the dialog box for entering the variables to be traced, as well as diverse trace parameters for the Sampling Trace. The dialog can also be opened by a double click in the grey area of the dialog Sampling Trace.

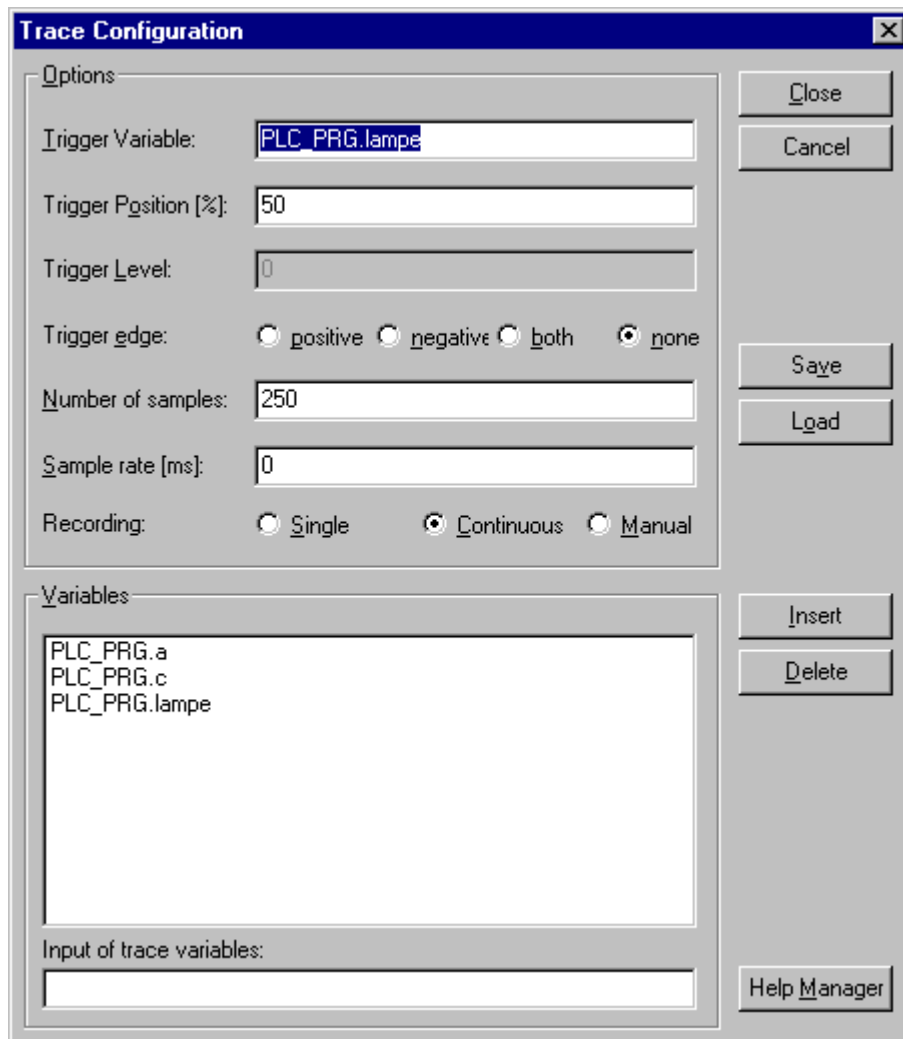


Image 6.19: Dialog Box for Trace Configuration

The list of the **Variables** to be traced is initially empty. In order to append a variable the variable must be entered in the field under the list. Following this, you can use the **Add** button or the <Enter> to append the variable to the list. You can also use the Input Assistant.

A variable is deleted from the list when you select the variable and then press the **Delete** button.

A Boolean or analogue variable can be entered into the field **Trigger Variable**. The input assistance can also be used here. The trigger variable describes the termination condition of the trace. In **Trigger Level** you enter the level of an analogue trigger variable at which the trigger event occurs. When **Trigger edge positive** is selected the trigger event occurs after an ascending edge of the Boolean trigger variable or when an analogue variable has passed through the trigger level from below to above. **Negative** causes triggering after a descending edge or when an analogue variable went from above to below. **Both** causes triggering for both descending and ascending edges or by a positive or negative pass, whereas **none** does not initiate a triggering event at all. **Trigger Position** is used to set the percentage of the measured value which will be recorded before the trigger event occurs. If, for example, you enter 25 here then 25 % of the measured values are shown before the triggering event and 75% afterwards and then the trace is terminated. The field **Sample Rate** is used set the time period between two recordings in milliseconds. The default value "0" means one scanning procedure per cycle.

Select the mode for recalling the recorded values: With **Single** the **Number of** the defined **samples** are displayed one time. With **Continuous** the reading of the recording of the defined number of measured values is initiated anew each time. If, for example, you enter the number '35' the first display contains the first measured values 1 to 35 and the recording of the next 35 measured values (36-70) will then be automatically read, etc.. **Manual** selection is used to read the trace recordings specifically with 'Extras' 'Read trace'.

The recall mode functions independently of whether a trigger variable is set or not. If no trigger variable is set the trace buffer will be filled with the defined number of measured values and the buffer contents will be read and displayed on recall.

The button **Save** is used to store the trace configuration which has been created in a file. The standard window "File save as" is opened for this purpose.

Stored trace configurations can be retrieved using the button **Load**. The standard window "File open" is opened for this purpose.



Note: Please note that **Save** and **Load** in the configuration dialog only relates to the configuration, not to the values of a trace recording (in contrast to the menu commands 'Extras' 'Save trace' and 'Extras' 'Load trace').

If the field **Trigger Variable** is empty, the trace recording will run endlessly and can be stopped by 'Extras' 'Stop Trace'.

"Extra" "Start Trace"

Symbol: 

With this command the trace configuration is transferred to the PLC and the trace sampling is started in the PLC.

"Extra" "Read Trace"

Symbol: 

With this command the present trace buffer is read from the PLC, and the values of the selected variables are displayed.

"Extra" "Auto Read"

With this command the present trace buffer is read automatically from the PLC, and the values are continuously displayed.

If the trace buffer is automatically read, then a check (✓) is located before the menu item.

"Extra" "Stop Trace"

Symbol: 

This command stops the Sampling Trace in the PLC.

Selection of the Variables to be Displayed

The comboboxes to the right, next to the window for displaying curves trace variables defined in the trace configuration. If a variable is selected from the list, then after the trace buffer has been read the variable will be displayed in the corresponding color (Var 0 green, etc.). Variables can also be selected if curves are already displayed.

A maximum of up to eight variables can be observed simultaneously in the trace window.

Display of the Sampling Trace

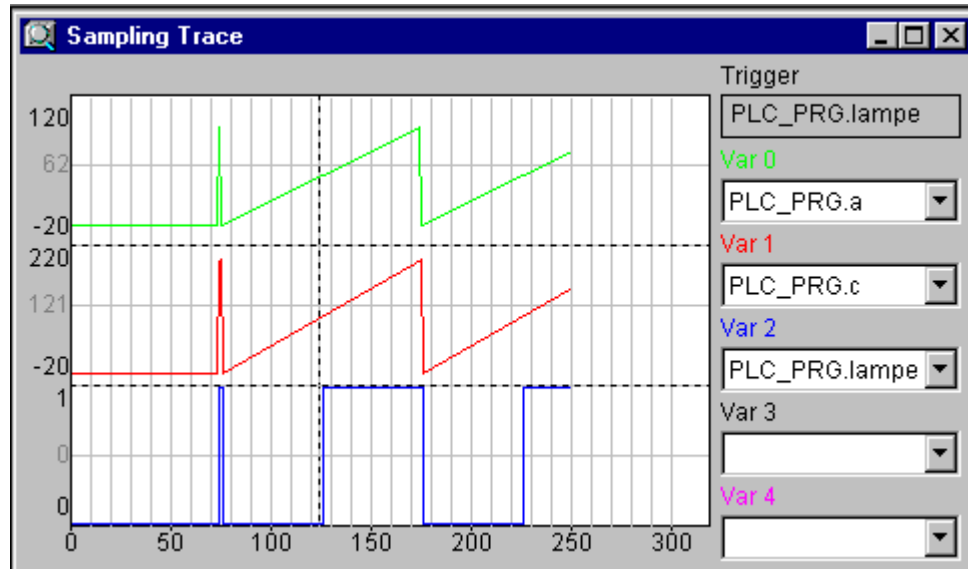


Image 6.20: Sampling Trace of Different Variables

If a trace buffer is loaded, then the values of all variables to be displayed will be read out and displayed. If no scan frequency has been set, then the X axis will be inscribed with the continuous number of the traced value. The status indicator of the trace window (first line) indicates whether the trace buffer is full and when the trace is completed.

If a value for the scan frequency was specified, then the x axis will specify the time of the traced value. The time is assigned to the "oldest" traced value. In the example, e.g., the values for the last 25 seconds are indicated.

The Y axis is inscribed with values in the appropriate data type. The scaling is laid out in a way that allows the lowest and the highest value to fit in the viewing area. In the example, Var 0 has taken on the lowest value of 6, and the highest value of 100: hence the setting of the scale at the left edge.

If the trigger requirement is met, then a vertical dotted line is displayed at the interface between the values before and after the appearance of the trigger requirement.

A memory that has been read will be preserved until you change the project or leave the system.

"Extras" "Cursor Mode"

The easiest way to set a cursor in the monitoring area is to click there with the left mouse button. A cursor appears and can be moved by the mouse. At the top of the monitoring window the current x-position of the cursor is displayed. In the fields next to 'Var 0', 'Var 1', ..., 'Var n' the value of the respective variable is shown.

Another way is the command 'Extras' 'Cursor mode'. With this command two vertical lines will appear in the Sampling Trace. First they are laying one on the other. One of the lines can be moved to the right or to the left by the arrow keys.

By pressing <Ctrl>+<left> or <Ctrl>+<right> the speed of the movement can be increased by factor 10.

If additionally the <Shift> key is pressed, the second line can be moved, showing the difference to the first one.

"Extras" "Multi Channel"

With this command you can alternate between single-channel and multi-channel display of the Sampling Trace. In the event of a multi-channel display, there is a check (✓) in front of the menu item.

The multi-channel display has been preset. Here the display window is divided into as many as eight display curves. For each curve the maximum and the minimum value are displayed at the edge.

In a single-channel display, all curves are displayed with the same scaling factor and are superimposed. This can be useful when displaying curve abnormalities.

'Extras' 'Show grid'

With this command you can switch on and off the grid in the graphic window. When the grid is switched on, a check (✓) will appear next to the menu item.

"Extras" "Y Scaling"

With this command you can change the preset Y scaling of a curve in the trace display.

In the dialog box specify the number of the desired curve (**Channel**) and the new maximum (**maximum y scale**) and the new minimum value (**minimum y scale**) on the y axis.

By doubleclicking on a curve you will also be given the dialog box. The channel and the former value are preset.

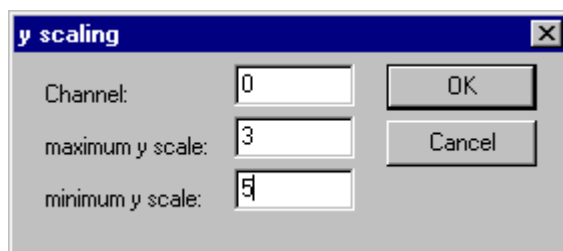


Image 6.21: Dialog Box for Setting the Y Scale

"Extras" "Stretch"

Symbol: 

With this command you can stretch (zoom) the values of the Sampling Trace that are shown. The beginning position is set with the horizontal picture adjustment bar. With repeated stretches that follow one-after-another, the trace section displayed in the window will increasingly shrink in size.

This command is the counterpart to "**Extras**" "**Compress**".

"Extras" "Compress"

Symbol: 

With this command the values shown for the Sampling Trace are compressed; i.e., after this command you can view the progression of the trace variables within a larger time frame. A multiple execution of the command is possible.

This command is the counterpart to "**Extras**" "**Stretch**".

"Extras" "Save Trace"

With this command you can save a Sampling Trace (values + configuration data). The dialog box for saving a file is opened. The file name receives the extension "*.trc".

Be aware, that here you save the traced values as well as the trace configuration, whereas **Save trace** in the configuration dialog only concerns the configuration data.

The saved Sampling Trace can be loaded again with "**Extras**" "**Load Trace**".

"Extras" "Load Trace"

With this command a saved Sampling Trace (traced values + configuration data) can be reloaded. The dialog box for opening a file is opened. Select the desired file with the "*.trc" extension.

With "**Extras**" "**Save Trace**" you can save a Sampling Trace.

"Extras" "Trace in ASCII-file"

With this command you can save a Sampling Trace in an ASCII-file. The dialog box is opened for saving a file. The file name receives the extension "*.txt". The values are deposited in the file according to the following scheme:

```
907 AC 1131 Trace
D:\907 AC 1131 \PROJECTS\TRAFFICSIGNAL.PRO
Cycle PLC_PRG.COUNTER PLC_PRG.LIGHT1
0 2 1
1 2 1
2 2 1
.....
```

If no frequency scan was set in the trace configuration, then the cycle is located in the first column; that means one value per cycle has been recorded at any given time. In the other respects, the entry here is for the point in time in ms at which the values of the variables have been saved since the Sampling Trace has been run.

In the subsequent columns, the corresponding values of the trace variables are saved. At any given time the values are separated from one another by a blank space.


The appertaining variable names are displayed next to one another in the third line, according to the sequence (PLC_PRG.COUNTER, PLC_PRG.LIGHT1).

6.6 Watch and Receipt Manager

Watch and Receipt Manager

With the help of the Watch and Receipt Manager you can view the values of selected variables. The Watch and Receipt Manager also makes it possible to preset the variables with definite values and transfer them as a group to the PLC ("**Write Receipt**"). In the same way, current PLC values can be read into and stored in the Watch and Receipt Manager ("**Read Receipt**"). These functions are helpful, for example, for setting and entering of control parameters.

All watch lists created ("**Insert**" "**New Watch List**") are indicated in the left column of the Watch and Receipt Manager. These lists can be selected with a mouse click or an arrow key. In the right area of the Watch and Receipt Manager the variables applicable at any given time are indicated.

In order to work with the Watch and Receipt Manager, open the object for the  **Watch and Receipt Manager** in the **Resources** register card in the Object Organizer.

Watch and Receipt Manager in the Offline Mode

In Offline Mode, you can create several watch lists in the Watch and Receipt Manager using the "**Insert**" "**New Watch List**".

For inputting the variables to be watched, you can call up a list of all variables with the Input Assistant, or you can enter the variables with the keyboard, according to the following notation:

<POUName>.<Variable Name>

With global variables, the POU Name is left out. You begin with a point. The variable name can, once again, contain multiple levels. Addresses can be entered directly.

Example of a multiple-level variable:

PLC_PRG.Instance1.Instance2.Structure.Componentname

Example of a global variable:

.global1.component1

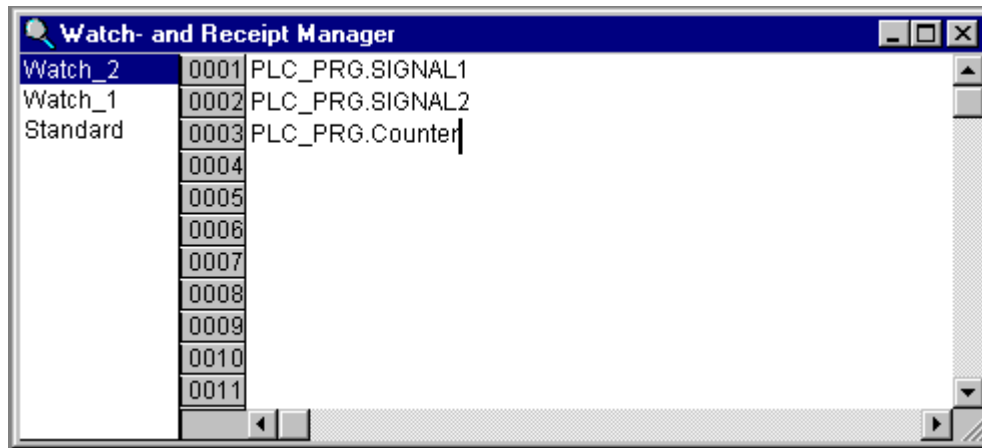


Image 6.22: Watch and Receipt Manager in the Offline Mode

The variables in the watch list can be preset with constant values. That means that in Online mode you can use the "Extras" "Write Receipt" command to write these values into the variables. To do to do must use := to assign the constant value of the variable:

Example:

`PLC_PRG.TIMER:=50`

In the example, the PLC_PRG.COUNTER variable is preset with the value 6

"Insert" "New Watch List"

With this command a new watch list can be inserted into the Watch and Receipt Manager. Enter the desired name for the watch list in the dialog box that appears.

"Extras" "Rename Watch List"

With this command you can change the name of a watch list in the Watch and Receipt Manager.

In the dialog box that appears, enter the new name of the watch list.

"Extras" "Save Watch List"

With this command you can save a watch list. The dialog box for saving a file is opened. The file name is preset with the name of the watch list and is given the extension "*.wtc".

The saved watch list can be loaded again with "Extras" "Load Watch List".

"Extras" "Load Watch List"

With this command you can reload a saved watch list. The dialog box is opened for opening a file. Select the desired file with the "*.wtc" extension. In the dialog box that appears, you can give the watch list a new name. The file name is preset without an extension.

With "Extras" "Save Watch List", you can save a watch list.

Watch and Receipt Manager in the Online Mode

In Online mode, the values of the entered variables are indicated.

Structured values (arrays, structures, or instances of function blocks) are marked by a plus sign in front of the identifier. By clicking the plus sign with the mouse or by pressing <Enter>, the variable is opened up or closed.

In order to input new variables, you can turn off the display by using the "**Extra**" "**Active Monitoring**" command. After the variables have been entered, you can use the same command again to activate the display of the values.

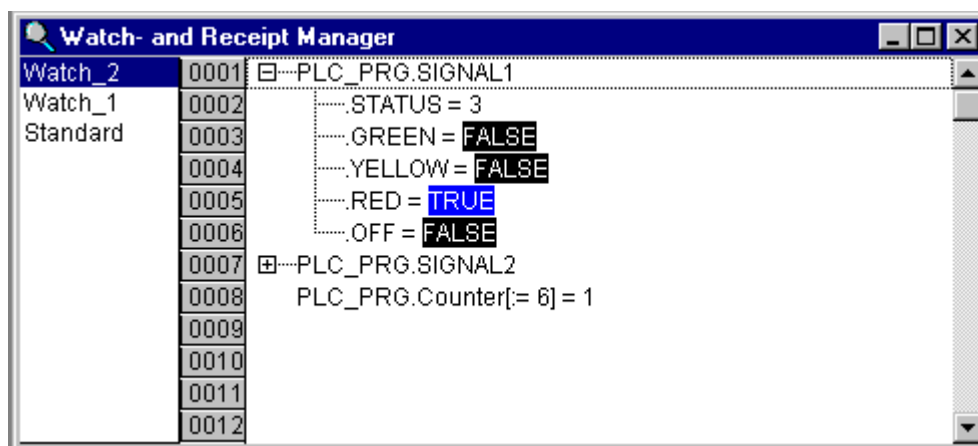


Image 6.23: Watch- and Receipt Manager in the Online Mode

In the Offline Mode you can preset variables with constant values (through inputting := <value> after the variable). In the Online Mode, these values can now be written into the variables, using the "**Extras**" "**Write Receipt**" command.

With the "**Extras**" "**Read Receipt**" command you can replace the presetting of the variable with the present value of the variable.



Note: Only those values the watch list are loaded which was selected in the Watch and Receipt Manager!

"Extra" "Monitoring Active"

With this command at the Watch and Receipt Manager in the Online mode, the display is turned on or off. If the display is active, a check (✓) will appear in front of the menu item.

In order to enter new variables or to preset a value (see Offline Mode), the display must be turned off through the command. After the variables have been entered, you can use the same command again to activate the display of the values.

"Extras" "Write Receipt"

With this command in the Online Mode of the Watch and Receipt Manager, you can write the preset values (see Offline Mode) into the variables.

"Extras" "Read Receipt"

With the command, in the Online Mode of the Watch and Receipt Manager, you can replace the presetting of the variables (see Offline Mode) with the present value of the variables.

Example:

```
PLC_PRG.Counter [:= <present value>] = <present value>
```

Force values

In the Watch and Receipt Manager you can also "**Force values**" and "**Write values**". If you click on the respective variable value, then a dialog box opens, in which you can enter the new value of the variable. Changed variables appear in red in the Watch and Receipt Manager.

7 Library Manager

The library manager shows all libraries that are connected with the current project. The POU, data types, and global variables of the libraries can be used the same way as user-defined POU, data types, and global variables.

The library manager is opened with the **"Window" "Library Manager"** command.

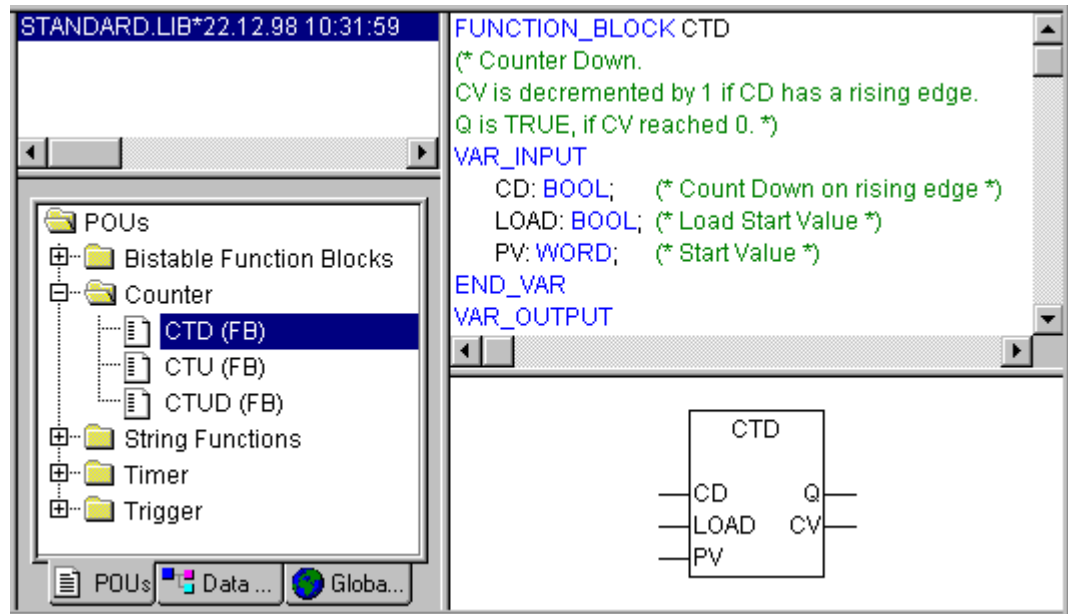


Image 7.1: Library Manager

Using the Library Manager

The window of the library manager is divided into three or four areas by screen dividers. The libraries attached to the project are listed in the upper left area.

In the area below that, depending on which register card has been selected, there is a listing of the **POUs**, **Data types**, or **Global variables** of the library selected in the upper area.

Folders are opened and closed by doubleclicking the line or pressing <Enter>. There is a plus sign in front of closed folders, and a minus sign in front of opened folders.

If a POU is selected by clicking the mouse or selecting with the arrow keys then the declaration of the POU will appear in the upper right area of the library manager; and in the lower right is the graphic display in the form of a black box with inputs and outputs.

With data types and global variables, the declaration is displayed in the right area of the library manager.

Standard Library

The library with "standard.lib" is always available. It contains all functions and function blocks that are required by the IEC1131-3 as standard POUs for an IEC programming system. The difference between a standard function and an operator is that the operator is implicitly recognized by the programming system, while the standard POUs must be tied to the project (standard.lib).

The code for these POUs exists as a C-library and is a component of **907 AC 1131** .

User-defined Libraries

If a project is to be compiled in its entirety and without errors, then it can be saved in a library with the "**Save as**" command in the "**File**" menu. The project itself will remain unchanged. Subsequently, you can gain access to the project under the entered name, just as with the standard library.

"Insert" "Additional Library"

With this command you can attach an additional library to your project.

In the dialog box for opening a file, choose the desired library with the "*.lib" extension. The library is now listed in the library manager, and you can use the objects in the library as user-defined objects.

Remove Library

With the "**Edit**" "**Delete**" command you can remove a library from a project and from the library manager.

8.1 Create Visualization

Visualization

Visualizations allow you to view your project variables. With the help of the visualization you can draw geometric elements offline. These can then change their forms or colors, in subjection to certain variable values, in Online mode. For example, it is possible to display the tendency of a variable to increase in a bar chart. You can also deal with input for the program by way of the mouse and the keyboard.

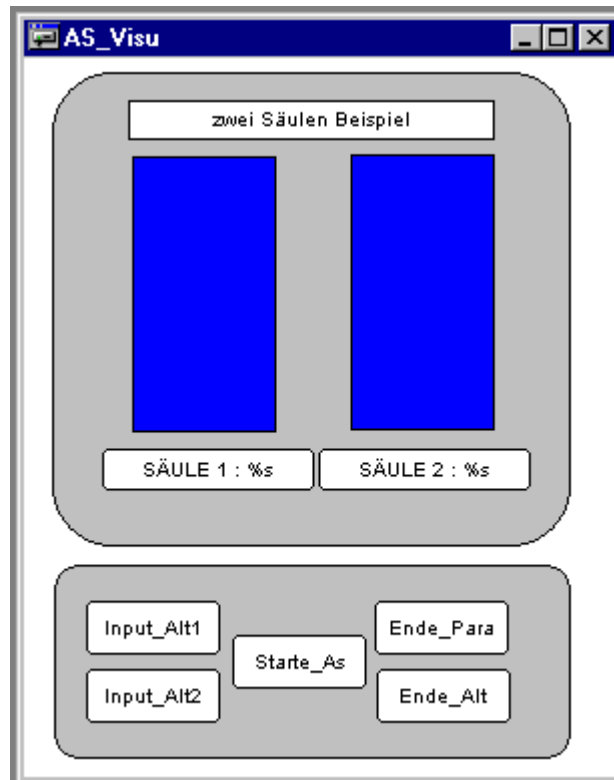



Image 8.1: Example of a Visualization

Create Visualization

In order to create a visualization, you must select the register card for  **Visualization** in the Object Organizer.










Using the "**Project**" "**Object Add**" command, you can create a new visualization object. A dialog box opens in which you can enter the name of the new visualization. If you have entered a valid name, then you can close the dialog box with **OK**. A window opens in which you can edit the new visualization.


8.2 Visualization Elements, Insert

Visualization Elements, Insert

You can insert four different geometric forms, as well as bitmaps and existing visualizations, into your visualization.

Geometric forms at your disposal include: rectangles, rounded rectangles, ellipses/circles, and polygons.

Go to the 'Insert' menu item and select freely from the following commands:  'Rectangle',  'Rounded Rectangle',  'Ellipse',  'Polygon',  'Line',  'Curve',  'Bitmap',  'Visualization'. A check appears in front of the selected command. You can also use the tool bar. The selected element appears pushed down (for example ).

If you now go to the editor window with the mouse, you will see that the mouse pointer is identified with the corresponding symbol (for example ). Click on the desired starting point of your element, and drag the pointer, while pressing the left mouse button, until the element reaches the desired size.

If you want to create a polygon or a line, first click with the mouse on the position of the first corner of the polygon resp. on the starting point of the line, and then click on the further desired corner points. By doubleclicking on the last corner point you will close the polygon and it will be completely drawn respectively the line will be completed. If you want to create a curve (Bezier curves) determine the initial and two other points with mouse clicks to define the circumscribing rectangle. An arc is drawn after the third mouse click. You can then change the position of the end point of the arc by moving the mouse and can then end the process with a double click or add another arc with additional mouse clicks.

Furthermore pay attention, to the status bar and the change from select and insert modes.

"Insert" "Rectangle"

Symbol: 

With the command you can insert a rectangle as an element into your present visualization. (Use, see Visualization Elements, Insert)

"Insert" "Rounded Rectangle"

Symbol: 

With the command you can insert a rectangle with rounded corners as an element in your present visualization. (Use, Visualization Elements, Insert)

"Insert" "Ellipse"

Symbol: 

With the command you can insert a circle or an ellipse as an element in your present visualization. (Use, see Visualization Elements, Insert)

"Insert" "Polygon"

Symbol: 

With the command you can insert a polygon as an element in your present visualization. (Use, see Visualization Elements, Insert).

"Insert" "Line"

Symbol: 

With the command you can insert a line as an element into your current visualization. (Use, see Visualization Elements, Insert).

'Insert' 'Curve'

Symbol: 

With the command you can insert a Bezier curve as an element into your current visualization. (Use, see Visualization Elements, Insert)..

"Insert" "Bitmap"

Symbol: 

With the command you can insert a bitmap as an element in your present visualization. (Use, see Visualization Elements, Insert)

While pressing the left mouse button, bring up an area in the desired size. The dialog box is opened for opening a file. Once you have selected the desired bitmap, it will be inserted into the area brought up.

"Insert" "Visualization"

Symbol: 

With the command you can insert an existing visualization as an element in your present visualization. (Use, see Visualization Elements, Insert)

While pressing the left mouse button, bring up an area in the desired size. A selection list of existing visualizations opens. After you have selected the desired visualization, it will be inserted in the defined area.

8.3 Working with Visualization Elements

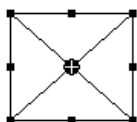
Selecting Visualization Elements

In order to select an element, click with the mouse on the element. You can also select the first element of the elements list by pressing the <Tab> key and jump to the next by each further keystroke. If you press the <Tab> key while pressing the <Shift> key, you jump backwards in the order of the elements list. In order to mark multiple elements, press and hold the <Shift> key and click the corresponding elements, one after another; or, while holding down the left mouse button, pull a window over the elements to be selected.

In order to select all the elements, use the "**Extras**" "**Select All**" command.

Modifying Visualization Elements

You can select an element which has already been inserted by a mouse click on the element or by pressing the <tab> key. A small black square will appear at each corner of each of the elements, (with ellipses at the corners of the surrounding rectangle). Except in the case of polygons, lines or curves further squares appear in the middle of the element edges between the corner points.



With a selected element, the turning point (balance point) is also displayed at the same time. You can then rotate the element around this point with a set motion/angle. The turning point is displayed as a small black circle with a white cross (⊕). You can drag the turning point with a pressed left mouse button.

You can change the size of the element by clicking on one of the black squares and, while keeping the left mouse button pressed, controlling the new outline.

With the selection of a polygon, you can drag each individual corner using the same technique. While doing this, if you press the <Ctrl>-key then an additional corner point will be inserted at the corner point, an additional corner point will be inserted, which can be dragged by moving the mouse. By pressing the <Shift>+<Ctrl>-key, you can remove a corner point.

Dragging Visualization Elements


One or more selected elements can be dragged by pressing the left mouse button or the arrow key.

Copying Visual Elements

One or more selected elements can be inserted with the "**Edit**" "**Copy**" command, the <Ctrl>+<C> key combination, or the corresponding copy symbol, and with "**Edit**" "**Paste**".

A further possibility is to select the elements and then click the mouse in an element once again, while pressing the <Ctrl>-key. Now you can remove the newly copied elements from the original ones while pressing the left mouse button.

Changing the Selection and Insert Mode

After the insertion of a visualization element, there is an automatic change back into the selection mode. If you want to insert an additional element the same way, you can once again select the corresponding command in the menu or the symbol  in the tool bar.

You can also quickly change between the selection mode and the insert mode by pressing the <Ctrl>-key and the right mouse button simultaneously.

In the insert mode, the corresponding symbol will also appear at the mouse pointer, and the name will also be indicated in black in the status bar.

Status Bar in the Visualization

In a visualization, the present **X** and **Y position** of the mouse pointer is displayed in the status bar. The position values are always relative to the upper left corner of the picture in the status bar. If the mouse pointer is located on an **Element**, or if the element is being processed, then the number of the element will be displayed. If you have selected an element to insert, then this element will also appear (for example, **Rectangle**).

8.4 Visualization Elements, Configure

"Extras" "Configure"

With this command you can open the dialog box to configure the selected visualization element.

You are given the dialog box for configuration when you doubleclick on the element.

Select a category in the left area of the dialog box, and fill out the requested information in the right area.

Depending on the visualization element selected, various categories can be selected:

- | | |
|-------------------|---|
| • Shape | Rectangle, Rounded Rectangle, Ellipse |
| • Text | All |
| • Color | Rectangle, Rounded Rectangle, Ellipse, Polygon, Line, Curve |
| • Motion absolute | All |
| • Motion relative | All, except Polygon, Line, Curve |
| • Variables | All |
| • Input | All |

- Tooltip All
- Bitmap Bitmap
- Visualization Visualization

Shape

In the visualization element configuration dialog box, you can select in the **Shape** category from among **Rectangle**, **Rounded Rectangle**, and **Ellipse** respectively **Polygon**, **Line** and **Curve**. The form will change into the size already set.

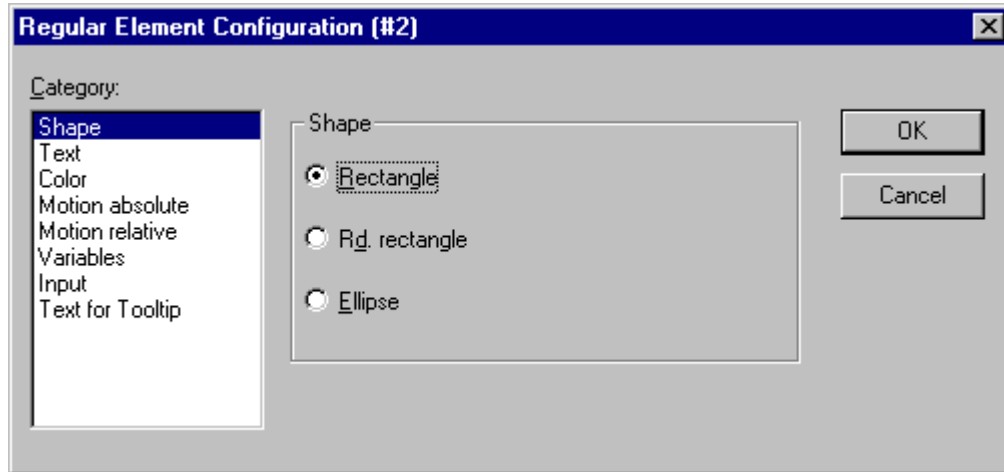


Image 8.2: Dialog Box for Configuring Visualization Elements (Shape Category)

Text

In the visualization element configuration dialog box, in the Text category you can set a text for the element.

Enter the text in the **Content** field. By pressing <Ctrl>+<Enter> you can insert line breaks.

If you enter "%s" into the text, then this location, in Online mode, will be replaced by the value of the variable from the **Text Output** field of the **Variables** category.

This text will appear in the element, according to the respective **Horizontal Left**, **Center**, or **Right** positioning, and the respective **Top**, **Center**, or **Bottom** positioning, that was specified in the element.

If you use the **Font** button, a dialog box for selection of the font will appear. Select the desired font and confirm the dialog with **OK**. With the **Standard Font** button you can set the font that is selected below in the "Project" "Options". If the font is changed there, then this font will be displayed in all elements except in those elements for which another font has explicitly been selected by using the **Font** button.

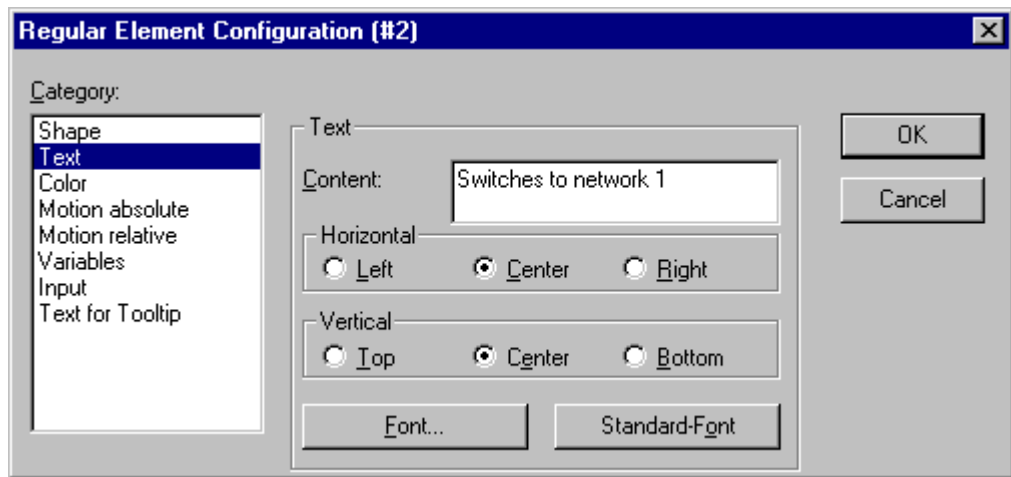


Image 8.3: Dialog Box for Configuring Visualization Elements (Text Category)

Colors

In the visualization element configuration dialog box, in the **Color** category you can select primary colors and alarm colors for the inside area and for the frame of your element. Choosing the options **no color inside** and **no frame color** you can create transparent elements.

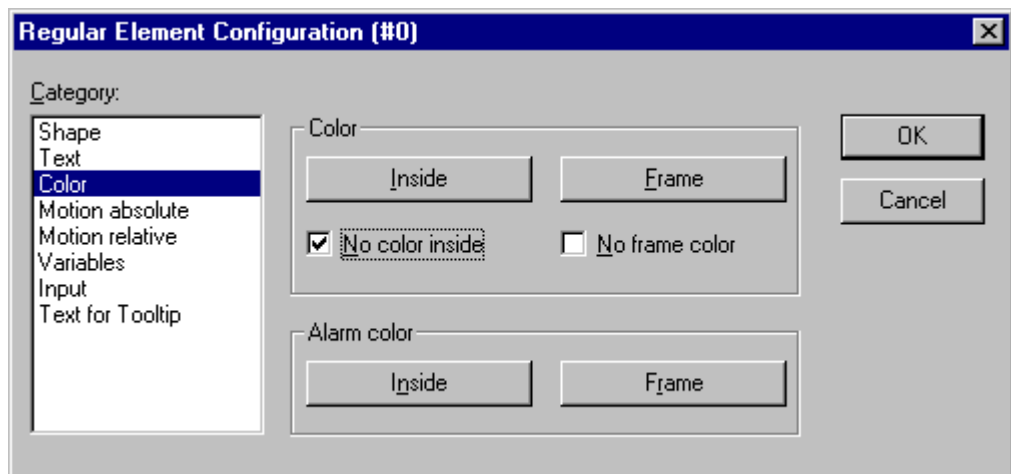


Image 8.4: Dialog Box for Configuring Visualization Elements (Color Category)

If you now enter a Boolean variable in the **Variables** category in the **Change Color** field, then the element will be displayed in the **Color** set, as long as the variable is FALSE. If the variable is TRUE, then the element will be displayed in its **Alarm Color**.



Note: The change color function only becomes active, if the PLC is in Online Mode!

If you want to change the color of the frame, then press the **Frame** button, instead of the **Inside** button. In either case, the dialog box will open for selection of the color.

Here can to choose the desired hue from the primary colors and the user-defined colors. By pressing the Define Colors you can change the user-defined colors.

Motion absolute

In the visualization element configuration dialog box, in the **Motion absolute** category, **X-** or **Y-Offset** fields variables can be entered. These variables can shift the element in the X or the Y direction, depending on the respective variable value. A variable in the **Scale** field changes the size of the element linear to the value of the variable.

A variable in the **Angle** field causes the element to turn on its turning point, depending on the value of the variable. (Positive Value = Mathematic Positive = Clockwise). The value is evaluated in degrees. With polygons, every point rotates; in other words, the polygon turns. With all other elements, the element rotates, in such a way, that the upper edge always remains on top.

The turning point appears after a single click on the element, and is displayed as a small black circle with a white cross (⊕). You can drag the turning point with a pressed left mouse button.

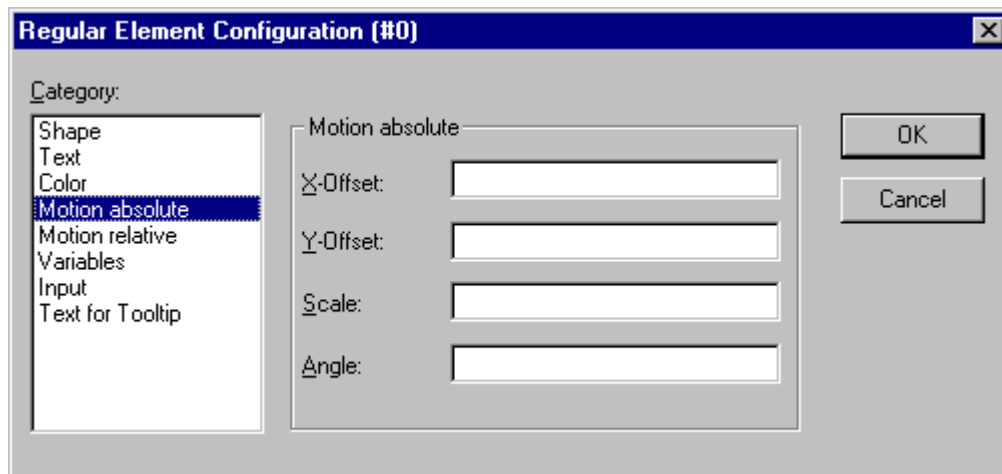


Image 8.5: Visualization Element Configuration Dialog Box (Motion Absolute Category)

Motion relative

In the dialog for configuring visualization elements in the **Motion Relative** category, you can assign variables to the individual element edges. Depending on the values of the variables, the corresponding element edges are then moved. The easiest way to enter variables into the fields is the Input Assistant.

The four entries indicate the four sides of your element. The base position of the corners is always at zero. A new value in the variables, in the corresponding column, shifts the boundary in pixels around this value. Therefore, the variables that are entered ought to be INT variables.



Note: Positive values shift the horizontal edges downward, or, the vertical edges, to the right!

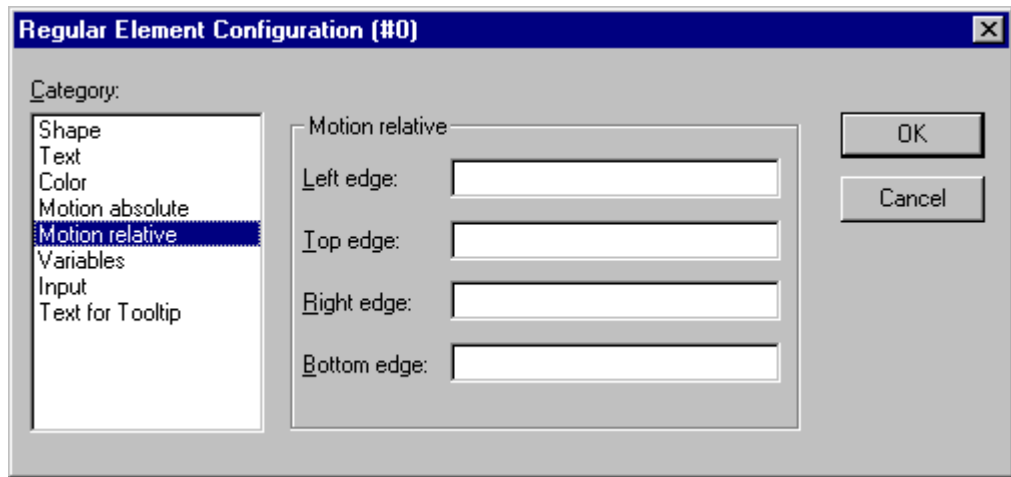


Image 8.6: Dialog Box for Configuration of Visualization Elements (Motion Relative Category)

Variables

You can enter the variables that describe the status of the visualization elements in the **Variable** category within the dialog box for configuring visualization elements. The simplest way to enter variables in the fields is to use the Input Assistant.

You can enter Boolean variables in the **Invisible** and **Change color** fields. The values in the fields determine their actions. If the variable of the **Invisible** field contains the value FALSE, the visualization element will be visible. If the variable contains the value TRUE, the element will be invisible.

If the variable at the **Change color** field contains the value FALSE, the visualization element will be displayed in its default color. If the variable is TRUE, the element will be displayed in its alarm color.

You can enter a variable in the **Textdisplay** field whose value is displayed if you have , in addition to the text, inserted %s in the **Content** field of the **Text** category. In Online mode, "%s" is replaced by the value of the variables found in **Textdisplay**.

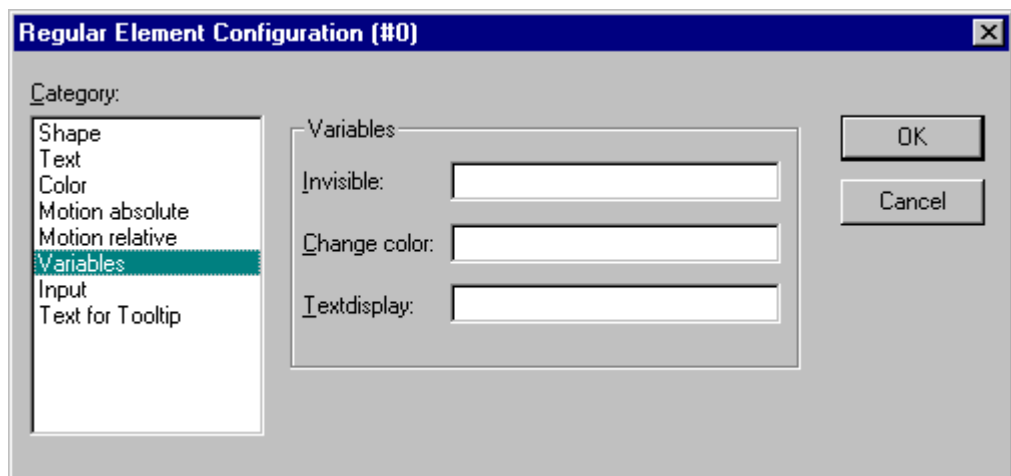


Image 8.7: Visualization Element Configuration Dialog Box (Variables Category)

Selecting the field **Toggle variable** allows you, in online mode, to toggle the value of the variables which are located in the input field with every mouse click on the element. The value of the Boolean variable changes with each mouse click from TRUE to FALSE and then back to TRUE again at the next mouse click, etc.

The option **Keying variable** allows you, in online mode, to change the value of the Boolean variable which is located in the input field, between TRUE and FALSE. Place the mouse cursor on the element, press the mousekey and hold it depressed (the value will, for example, change from TRUE to FALSE). The variable changes back to its initial value (TRUE) as soon as you release the mousekey.

Selecting the field **Zoom to Vis...** allows you, in the following field, to enter the name of a visualization object in the same project. While in online mode use a mouse click to change to the element in the window of the visualization which has been entered. If a program variable of the type STRING (e.g. PLC_PRG.xxx) has been entered, instead of a visualization object, then this variable can be used to define the name of the visualization object (e.g. ,visu1') which the system should change to when a mouse click occurs (e.g. xxx:= ,visu1).

The field **Zoom to Vis...** can be used to configure the return to the calling visualization by using the command ,ZOOMTOCALLER'.

Selecting the option **Execute program** allows you to enter any executable program in the input field and then to execute it in online mode by clicking on the element with the mouse.



Note: The configuration field **Execute program** plays a major role for the **907 AC 1131 operating version**, since 907 AC 1131 program actions can be initiated here over defined commands, which are available as menu commands in the full version (see 'Special input possibilities for the 907 AC 1131 operating version').

Selecting the option **Text input of the variable 'Text output'** allows you, in online mode, to allocate a value to a variable over this visualization element. The value which is located in the field **Text output** of the category **Variables** will be written in the variable. Clicking in online mode on the element produces an editing frame in which you can enter the new value of the variable over the keyboard. Press the <Enter> key to accept the value.

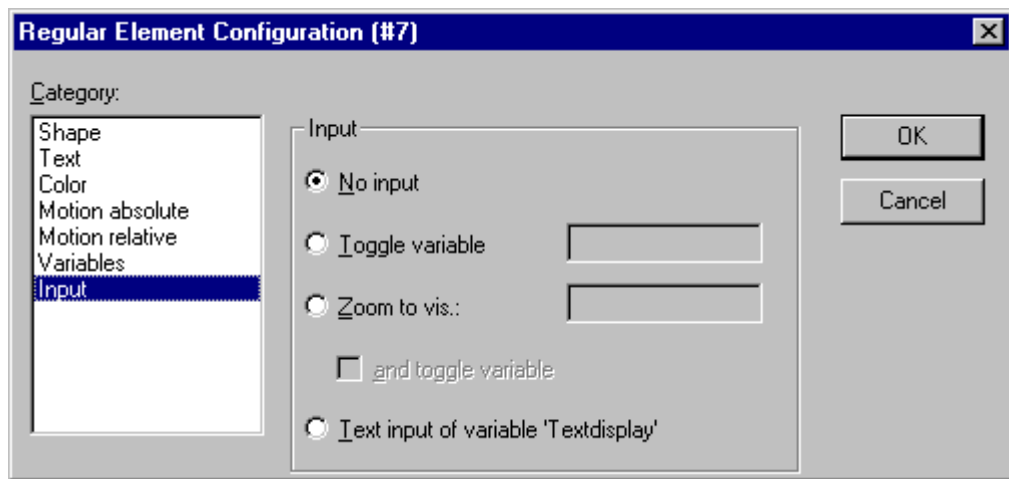


Image 8.8: Dialog to configure the visualization elements (Category Input)

Special input possibilities for the 907 AC 1131 operating version

In the operating version of 907 AC 1131 there are no menus and status and tool bars available to the user. The most important control and monitoring functions can, however, be attached to a visualization element and thus be available through a mouse click or over the keyboard. The following special input possibilities are available in the configuration dialog for a visualization element:

Enter internal commands in the field **Execute program** in the category **Input** according to the following syntax:

INTERN <COMMAND> [PARAMETER]*

The following internal commands which can accommodate a number of parameters separated by the <space bar> are presently available:

Command	The equivalent in the full version of 907 AC 1131	Explanation
LANGUAGE	<visualization settings>	The dialog for visualization settings which includes the category language gets opened.
LANGUAGE <sprache>	<visualization settings>	The desired language file is chosen without using the dialog for visualization settings.
DEFINERECEIPT name	<Select watch lists>	A watch list is selected from the receipt manager which enters your name (name) when the

		command is given. The variables in this watch list are registered and displayed.
WRITERECEIPT name	'Write receipts'	The name of a watch list of the receipt manager is expected. The receipt of this watch list will be written. A previous execution of DEFINERECEIPT is not necessary.
SAVEWATCH	'Save watch list'	The receipt will be read into the current watch list which will be stored in a file. Important: call a previous DEFINERECEIPT to define the current receipt !
LOADWATCH	'Load watch list' + 'Write receipt'	The standard window ,File open' appears, from which a previously stored receipt can be selected. This receipt will be immediately written into the controller system.
EXITPROGRAM	,File' 'Close'	The program will be exited.
PRINT	,File' 'Print'	The current visualization will be printed out online.
TRACE	<Open object trace recording>	The window for trace recording will be opened. The menu commands Trace Start, Read, Stop, Save, Load which are available in the full version of 907 AC 1131 are available in this window.

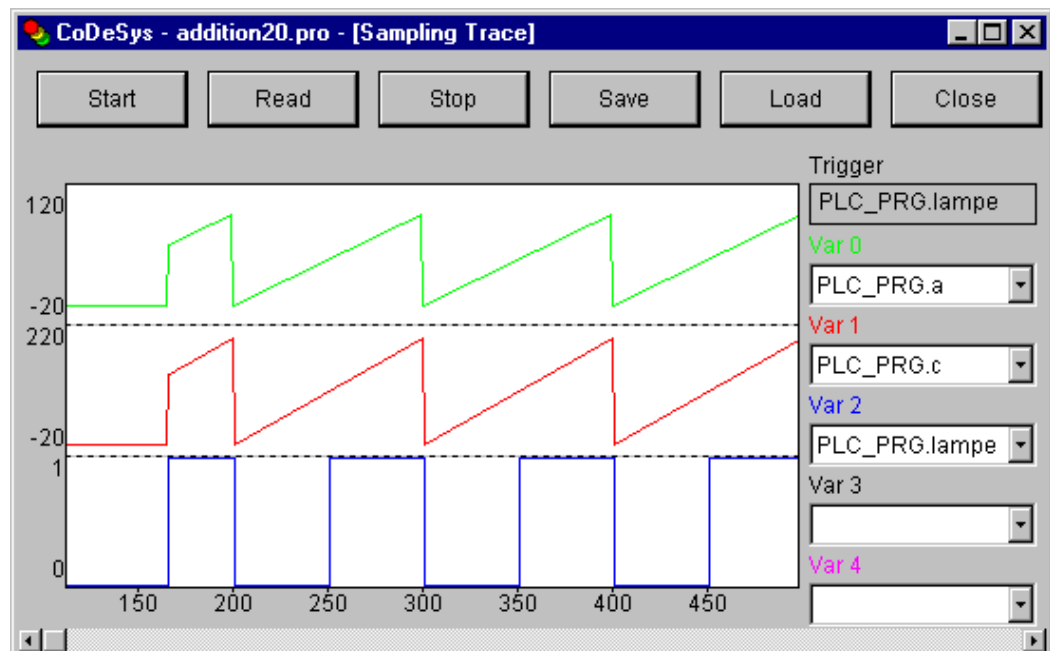


Image 8.9: Dialog for the trace recording in the operation version

ToolTip

The dialog Text for Tooltip offers an input field for text which appears in a text field as soon as the mouse cursor is passed over the object in online mode. The text can be formatted with line breaks by using the key combination <Ctrl> + <Enter>.

Bitmap

You can enter the options for a bitmap in the **Bitmap** category within the visualization element configuration dialog box.

Enter the bitmap file and its path in the **Bitmap** field. You can use the ... button to open the standard Windows Browse dialog box from which you can select the desired bitmap.

All other entries affect the **frame** of the bitmap.

By selecting **Anisotropic**, **Isotropic** or **Fixed** you specify how the bitmap should react to changes in the size of the frame. **Anisotropic** means that the bitmap remains the same size as the frame which allows you to change the height and width of the bitmap independently. **Isotropic** means that the bitmap retains the same proportions even if the overall size is changed (i.e., the relationship between height and width is maintained). If **Fixed** is selected, the original size of the bitmap will be maintained regardless of the size of the frame.

If the **Clip** option is selected together with the **Fixed** setting, only that portion of the bitmap that is contained within the frame will be displayed.

If you select the **Draw** option, the frame will be displayed in the color selected in the **Color** and **Alarm color** buttons in the color dialog boxes. The alarm color will only be used if the variable in the **Change Color** field in the **Variable** category is TRUE.

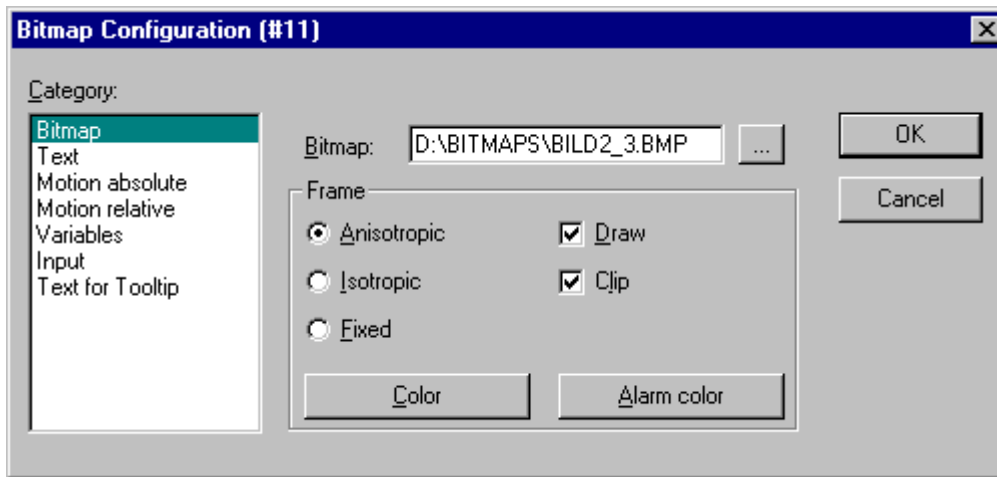


Image 8.10: Visualization Element Configuration Dialog Box (Bitmap Category)

Visualization

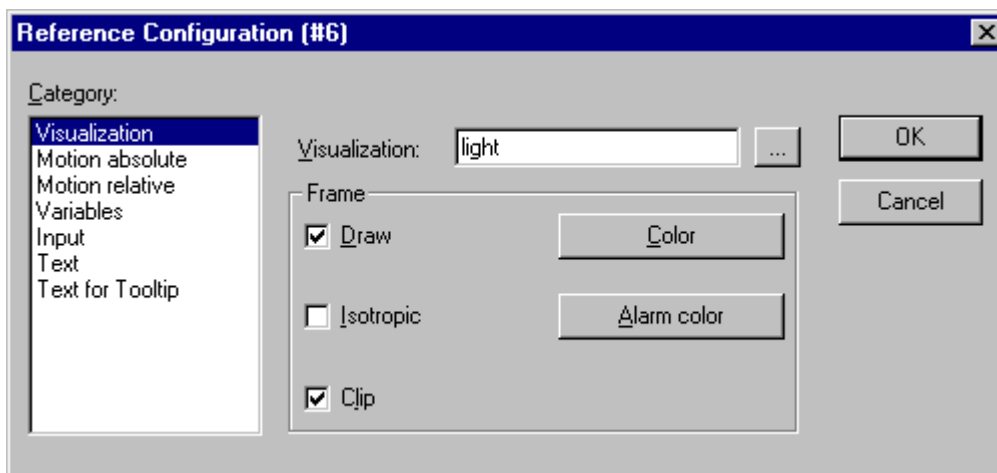
You can enter the options for a visualization as an element in another visualization in the **Visualization** category within the visualization element configuration dialog box. Enter the object name for the visualization in the **Visualization** field. Use the ... button to open a dialog box containing the visualizations available in this project. Any visualization may be used with the exception of the current one.

All other entries affect the visualization **frame**.

If you select the **Draw** option, the frame will be displayed in the color selected in the **Color** and **Alarm color** buttons in the color dialog boxes. The alarm color will only be used if the variable in the **Change Color** field in the **Variables** category is TRUE.

If **Isotropic** is selected, the proportions of the visualization will be maintained even if the size changes (i.e., the relationship between height and width will remain the same). Otherwise the proportions can be changed.

If the **Clip** option is selected in Online mode, only the original portion of the visualization will be displayed. For example, if an object extends beyond the original display area, it will be clipped and may disappear from view completely in the visualization.



8.5 Additional Visualization Element Functions

"Extras" "Send to Front"

Use this command to bring selected visualization elements to the front.

"Extras" "Send to Back"

Use this command to send selected visualization elements to the back.

"Extras" "Select Background Bitmap"

Use this command to open the dialog box for selecting files. Select a file with the extension "*.bmp". The selected bitmap will then appear as the background in your visualization.

The bitmap can be removed with the command "**Extras**" "**Clear Background Bitmap**".

"Extras" "Clear Background Bitmap"

Use this command to remove the bitmap as the background for the current visualization.

You can use the command "**Extras**" "**Select Background Bitmap**" to select a bitmap for the current visualization.

"Extras" "Align"

Use this command to align selected visualization elements.


The following alignment options are available:

- **Left:** the left edge of each of the elements will be aligned to the element that is furthest to the left
- the same is true for **Right / Top / Bottom**
- **Horizontal Center:** each of the elements will be aligned to the average horizontal center of all elements
- **Vertical Center:** each of the elements will be aligned to the average vertical center of all elements

"Extras" "Select All"

This command allows you to select all visualization elements within the current visualization object.

'Extras' 'Select Mode'

This command is used to switch the selection mode on or off. This can also be achieved using the symbol  or by pressing the right mousekey while holding down the <Ctrl> key at the same time.

"Extras" "Element list"

This command opens a dialog box containing a list of all visualization elements including their **number**, **type** and **position**. The position is given according to the x and y position of the upper left and lower right corner of the element.

When one or more items have been selected, the corresponding elements in the visualization are marked for visual control and if necessary the display will scroll to that section of the visualization that contains the elements.

Use the **To front** button to bring selected visualization elements to the front. Use the **To behind** button to move them to the back.

Use the **Delete** button to remove selected visualization elements.

Use the **Undo** and **Redo** buttons to undo or restore changes that have been made just as you would do with the commands "**Edit**" "**Undo**" and "**Edit**" "**Redo**". In the dialog box, you can observe the changes that are being made.

Click on **OK** to close the dialog box and confirm the changes.

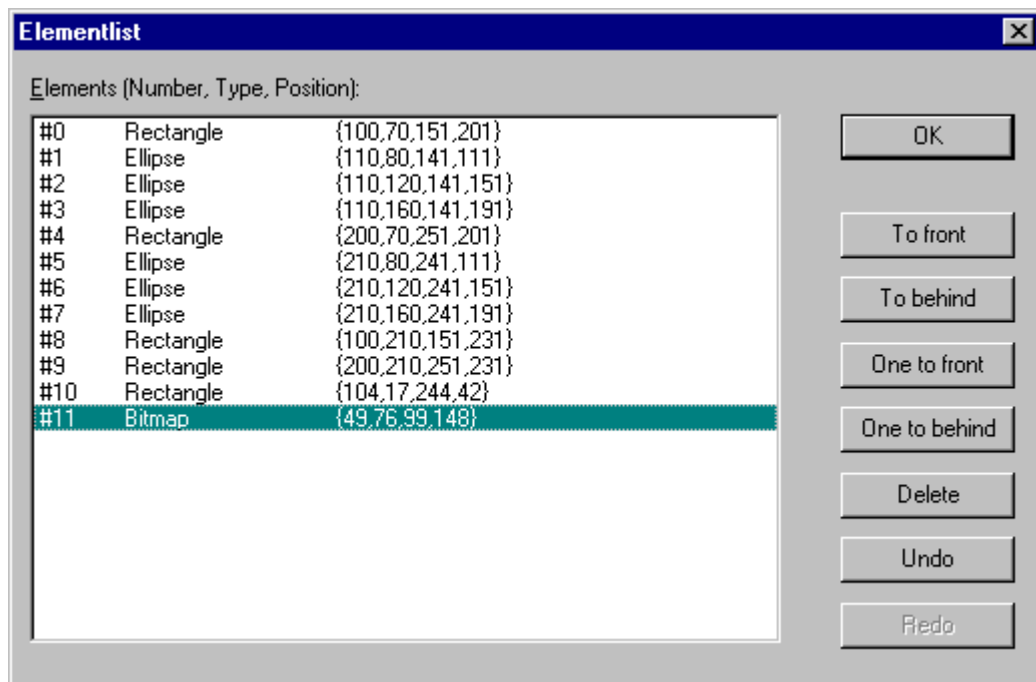


Image 8.12: Element list dialog box

"Extras" "Settings"

When this command is used, a dialog box will open in which you can make certain settings that affect the visualization.



Note: The categories **Display**, **Frame** and **Language** also can be edited in the online mode.

Enter a zoom factor into the field **Zoom** in the category **Presentation** of between 10 and 500 % in order to increase or decrease the size of the visualization display. Selecting **Element numbers** shows the numbers of the elements in each visualization element when in offline mode.

If **Auto-scrolling** is selected in the **Frame** category, the visible portion of the visualization window will move automatically when you reach the edge while drawing or moving a visualization element. If **Best fit in Online mode** is selected, the entire visualization including all elements will be shown in the window in Online mode regardless of the size of the window. When **Include Background Bitmap** is selected, the background bitmap will be fitted into the window as well, otherwise only the elements will be considered.

The category **Grid** is used to define whether the grid points are **visible** in the offline mode, whereby the spacing between the visible points is at least 10 even if the entered size is smaller than that. In this case the grid points only appear with a spacing which is a multiple of the entered size. Selecting **Active** causes the elements to be placed on the snap grid points when they are drawn and moved. The spacing of the grid points is set in the field **Size**.

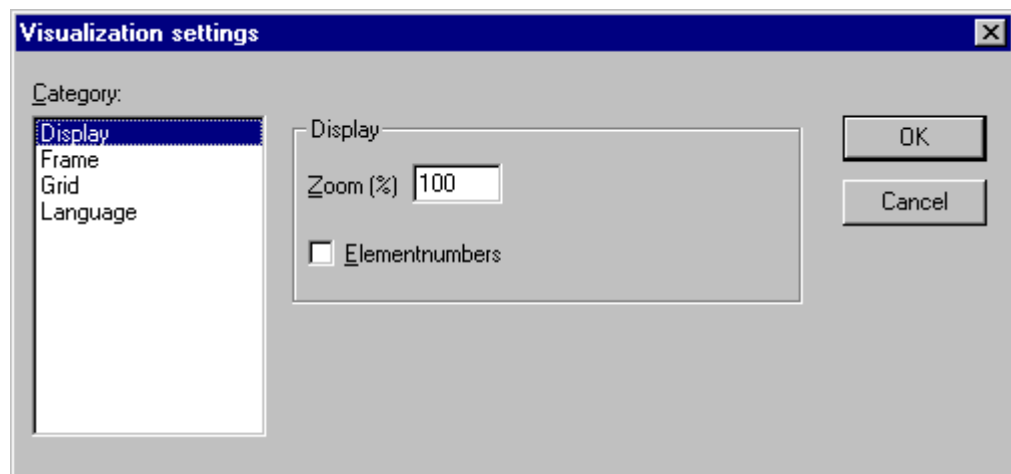



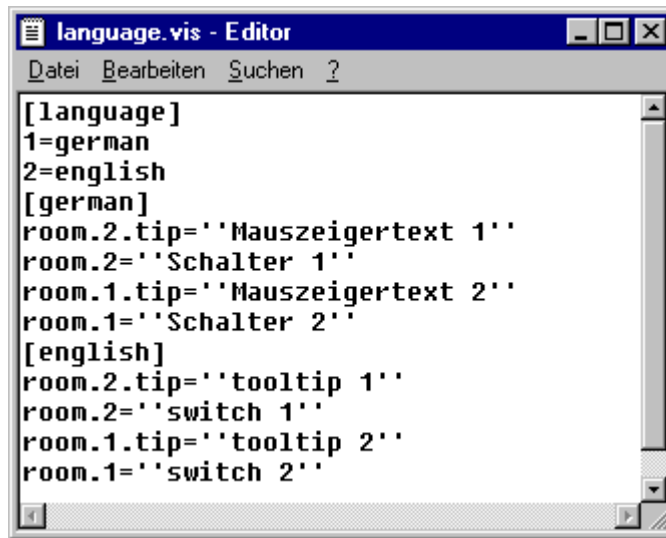
Image 8.13: Setting dialog for visualizations (Category Presentation)

The category **Language** is used to display the text which has been assigned to the visualization elements using the options **Text** and **Text for Tooltip** in a desired language. You have to prepare a language file as described in the following:

Choose option **language file**. In the associate input field you give in where you want to store the file. The extension is '.vis'. You also can use the dialog 'Open' by pressing the button . If a language file is available already, it will be offered here.

In the input field next to **Language** you fill in a keyword for the language, which is currently used in the visualization, i.e. "german" (or "D"). Then press the

button Save. A file with the extension .vis is created, which now can be edited by a normal text editor. For example you can open the file by notepad:



```
[language]
1=german
2=english
[german]
room.2.tip=''Mauszeigertext 1''
room.2=''Schalter 1''
room.1.tip=''Mauszeigertext 2''
room.1=''Schalter 2''
[english]
room.2.tip=''tooltip 1''
room.2=''switch 1''
room.1.tip=''tooltip 2''
room.1=''switch 2''
```

Bild 8.14: Example of a language file for a visulisation (Category Language)

You get a list of the text variables for the language currently used in the visualization. It includes a reference to the title of this list, for example "1=german" as reference to the title [german]. You can extend the list by copying all lines, then replacing the German by English text and setting a new title [english]. Beyond the line 1=german you accordingly have to add 2=english.

To view the visualization in one of the prepared languages, open the dialog **Language** again. In the option field beyond **Language** now you can choose between german and english (for the example described above).



Note: The text display does not change before switching to Online Mode !

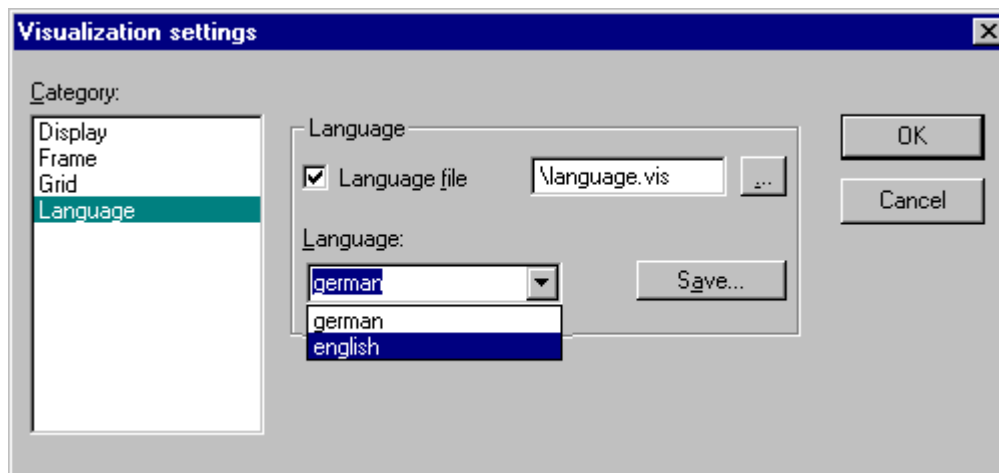


Bild 8.15: Selection of a language file for a visualization

Operation over the keyboard - in online mode

The visualization elements can be manipulated in online mode over the following key functions:

Pressing the <Tabulator> key selects the first element in the element list for which an input is configured. Each subsequent pressing of the key moves one to the next element in the list. Pressing the key while keeping the <Shift> key depressed selects the previous element.

The arrow keys can be used to change from a selected element to a neighbouring one in any direction.

The <**Space bar**> is used to execute an activity on the selected visualization element. If the element is one which has a text output variable, a text input field will be opened which displays the text contents of the variable. Pressing the <**Enter**> key writes in this value.



Note: Operation over the keyboard in online mode is of greatest significance for the operation version of 907 AC 1131 !

,File' ,Print' in online mode

'File' 'Print' is used to print out the contents of the visualization window in online mode. Visualizations which stretch over the border of the window can lead to inconsistencies particularly when there are moving elements in the visualization.

907 AC 1131 has a DDE (dynamic data exchange) interface for reading data. **907 AC 1131** uses this interface to provide other applications that also use a DDE Interface with the contents of control variables and IEC addresses.

Activating the DDE Interface

The DDE interface becomes active as soon as the PLC (or the simulation) is logged in.

General Approach to Data

A DDE inquiry can be divided into three parts:

1. Name of the program (here: **AC1131**),
2. File name and
3. Variable name to be read.

Name of the program: **AC1131**

File name: complete project path (c:\example\example.pro).

Variable name: The name of a variable as it appears in the Watch and Receipt Manager .

Which variables can be read?

All addresses and variables with the exception of global variables are readable. Variables or addresses should be entered in the format used in the Watch and Receipt Manager.

Examples:

%IX1.4.1	(* Reads the input 1.4.1*)
PLC_PRG.TES	(* Reads the variable TEST from the POU
T	PLC_PRG*)

Linking variables using WORD

In order to get the current value of the variable TEST from the POU PLC_PRG through the DDE interface in Microsoft WORD, a field (e.g., the date) must be inserted in WORD ("Insert" "Field"). Now when you click on the field with the right mouse button and select the command "Toggle Field Codes" you can change the field function for the chosen text. In our example, this would look as follows:

```
{ DDEAUTO AC1131 "C:\\AC1131\\PROJECT\\IFMBSP.PRO"  
  "PLC_PRG.TEST" }
```

Click on the field with the right mouse button again, then click on "Update Field" and the desired variable content appears in the text.

Linking variables using EXCEL

The following must be entered in Microsoft EXCEL before you can assign a variable to a cell. Do not use any special characters in the name of the variable !

```
='AC1131'|'C:\AC1131\PROJECT\IFMBSP.PRO!'PLC_PRG.TEST"
```

When you click on "Edit" "Links", the result for this link will be:

Type: AC1131
Source file: C:\AC1131\PROJECT\IFMBSP.PRO
Element: PLC_PRG.TEST

Accessing variables with Intouch

Link with your project a DDE Access Name <AccessName> with the application name 907 AC 1131 and the DDE topic name C:\AC1131\PROJECT\IFMBSP.PRO. Now you can associate DDE type variables with the access name <AccessName>. Enter the name of the variable as the Item Name (e.g., PLC_PRG.TEST).DDE-Schnittstelle

10 Appendix

Appendix A: Use of Keyboard

If you would like to run 907 AC 1131 using only the keyboard, you will find it necessary to use a few commands that are not found in the menu.

- The function key <F6> allows you to toggle back and forth within the open POU between the Declaration and the Instruction parts.
- <Alt>+<F6> allows you to move from an open object to the Object Organizer and from there to the Message window if it is open. If a Search box is open, <Alt>+<F6> allows you to switch from Object Organizer to the Search box and from there back to the object.
- Press <Tab> to move through the input fields and buttons in the dialog boxes.
- The arrow keys allow you to move through the register cards and objects within the Object Organizer and Library Manager.

All other actions can be performed using the menu commands or with the shortcuts listed after the menu commands. <Shift>+<F10> opens the context menu which contains the commands most frequently used for the selected object or for the active editor.

Key Combinations

The following is an overview of all key combinations and function keys:

General Functions	
Move between the declaration part and the instruction part of a POU	<F6>
Move between the Object Organizer, the object and the message window	<Alt>+<F6>
Context Menu	<Shift>+<F10>
Shortcut mode for declarations	<Ctrl>+<Enter>
Move from a message in the Message window back to the original position in the editor	<Enter>
Open and close multi-layered variables	<Enter>
Open and close folders	<Enter>
Switch register cards in the Object Organizer and the Library Manager	<Arrow keys>

Move to the next field within a dialog box	<Tab>
Context sensitive Help	<F1>
General Commands	
"File" "Save"	<Ctrl>+<S>
"File" "Print"	<Ctrl>+<P>
"File" "Exit"	<Alt>+<F4>
"Project" "Check"	<Strg>+<F11>
"Project" "Build"	<Umschalt>+<F11>
"Project" "Rebuild all"	<F11>
"Project" "Delete Object"	
"Project" "Add Object"	<Ins>
"Project" "Rename Object"	<Spacebar>
"Project" "Open Object"	<Enter>
"Edit" "Undo"	<Ctrl>+<Z>
"Edit" "Redo"	<Ctrl>+<Y>
"Edit" "Cut"	<Ctrl>+<X> or <Shift>+
"Edit" "Copy"	<Ctrl>+<C>
"Edit" "Paste"	<Ctrl>+<V>
"Edit" "Delete"	
"Edit" "Find next"	<F3>
"Edit" "Input Assistant"	<F2>
"Edit" "Next Error"	<F4>
"Edit" "Previous Error"	<Shift>+<F4>
"Online" "Run"	<F5>
"Online" "Toggle Breakpoint"	<F9>
"Online" "Step over"	<F10>

"Online" "Step in"	<F8>
"Online" "Single Cycle"	<Ctrl>+<F5>
"Online" "Write Values"	<Ctrl>+<F7>
"Online" "Force Values"	<F7>
"Online" "Release Force"	<Shift>+<F7>
"Window" "Messages"	<Shift>+<Esc>
FBD Editor Commands	
"Insert" "Network (after)"	<Shift>+<T>
"Insert" "Assignment"	<Ctrl>+<A>
"Insert" "Jump"	<Ctrl>+<L>
"Insert" "Return"	<Ctrl>+<R>
"Insert" "Operator"	<Ctrl>+<O>
"Insert" "Function"	<Ctrl>+<F>
"Insert" "Function Block"	<Ctrl>+
"Insert" "Input"	<Ctrl>+<U>
"Extras" "Negate"	<Ctrl>+<N>
"Extras" "Zoom"	<Alt>+<Enter>
LD Editor Commands	
"Insert" "Network (after)"	<Shift>+<T>
"Insert" "Contact"	<Ctrl>+<O>
"Insert" "Parallel Contact"	<Ctrl>+<R>
"Insert" "Function Block"	<Ctrl>+
"Insert" "Coil"	<Ctrl>+<L>
"Extras" Paste below"	<Ctrl>+<U>
"Extras" "Negate"	<Ctrl>+<N>

SFC Editor Commands	
"Insert" "Step-Transition (before)"	<Ctrl>+<T>
"Insert" "Step-Transition (after)"	<Ctrl>+<E>
"Insert" "Alternative Branch (right)"	<Ctrl>+<A>
"Insert" "Parallel Branch (right)"	<Ctrl>+<L>
"Insert" "Jump"(SFC)	<Ctrl>+<U>
"Extras" "Zoom Action/Transition"	<Alt>+<Enter>
Move back to the editor from the SFC Overview	<Enter>
Work with the PLC Configuration	
Open and close organization elements	<Enter>
Place an edit control box around the name	<Spacebar>
"Extras" "Edit Entry"	<Enter>
Work with the Task Configuration	
Place an edit control box around the task or program name	<Spacebar>

Appendix B: Data types

You can use standard data types and user-defined data types when programming. Each identifier is assigned to a data type which dictates how much memory space will be reserved and what type of values it stores.

Standard Data types

BOOL

BOOL type variables may be given the values **TRUE** and **FALSE**. 8 bits of memory space will be reserved.

Integer Data Types

BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, and UDINT are all integer data types

Each of the different number types covers a different range of values. The following range limitations apply to the integer data types:

Type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 Bit
WORD	0	65535	16 Bit
DWORD	0	4294967295	32 Bit
SINT:	-128	127	8 Bit
USINT:	0	255	8 Bit
INT:	-32768	32767	16 Bit
UINT:	0	65535	16 Bit
DINT:	-2147483648	2147483647	32 Bit
UDINT:	0	4294967295	32 Bit

As a result when larger types are converted to smaller types, information may be lost.

REAL / LREAL

REAL and **LREAL** are so-called floating-point types. They are required to represent rational numbers. 32 bits of memory space is reserved for **REAL** and 64 bits for **LREAL**.

STRING

A **STRING** type variable can contain any string of characters. The size entry in the declaration determines how much memory space should be reserved for the variable. It refers to the number of characters in the string and can be placed in parentheses or square brackets. If no size specification is given, the default size of 80 characters will be used.

Example of a String Declaration with 35 characters:

```
str:STRING(35):='This is a String';
```

Time Data Types

The data types **TIME**, **TIME_OF_DAY** (abb. **TOD**), **DATE** and **DATE_AND_TIME** (abb. **DT**) are handled internally like **DWORD**.

Time is given in milliseconds in **TIME** and **TOD**, time in **TOD** begins at 12:00 A.M.

Time is given in seconds in **DATE** and **DT** beginning with January 1, 1970 at 12:00 A.M.

The time data formats used to assign values are described in the chapter on Constants.

Defined Data Types

ARRAY

One-, two-, and three-dimensional fields (arrays) are supported as elementary data types. Arrays can be defined both in the declaration part of a POU and in the global variable lists.

Syntax:

```
<Field_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>.
```

ll1, ll2, ll3 identify the lower limit of the field range; ul1, ul2 and ul3 identify the upper limit. The range values must be integers.

Example:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Initializing Arrays:

You can initialize either all of the elements in an array or none of them.

Example for initializing arrays:

```
arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
```

```
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7);
```

(* short for 1,7,7,7 *)

```
arr3 : ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3;
```

(* short for 0,0,4,4,4,4,2,3 *)

Array components are accessed in a two-dimensional array using the following syntax:

<Field_Name>[Index1,Index2]

Example:

Card_game [9,2]



Note: If you define a function in your project with the name CheckBounds, you can use it to check for range overflows in your project (see chapter 'What is what in 907 AC 1131 ', 'Components of a project', 'Function')

Pointer

Variable or function block addresses are saved in pointers while a program is running.

Pointer declarations have the following syntax:

<Identifier>: **POINTER TO** <Datatype/Functionblock>;

A pointer can point to any data type or function block even to user-defined types.

The function of the Address Operator ADR is to assign the address of a variable or function block to the pointer.



A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example:

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
pt := ADR(var_int1);
var_int2:= pt^; (* var_int2 is now 5 *)
```

Enumeration

Enumeration is a user-defined data type that is made up of a number of string constants. These constants are referred to as enumeration values.

Enumeration values are recognized in all areas of the project even if they were locally declared within a POU. It is best to create your enumerations as objects in the Object Organizer under the register card **Data types**. They begin with the keyword TYPE and end with END_TYPE.

Syntax:

TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);
END_TYPE

A variable of the type <Identifier> can take on one of the enumeration values and will be initialized with the first one. These values are compatible with whole numbers which means that you can perform operations with them just as you would with INT. You can assign a number x to the variable. If the enumeration values are not initialized, counting will begin with 0. When initializing, make

certain the initial values are increasing. The validity of the number will be reviewed at the time it is run.

Example:


```
TYPE TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (*The initial value for
  each of the colors is red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* The value of the traffic signal is red*)
FOR i:= Red TO Green DO
  i := i + 1;
END_FOR;
```

You may not use the same enumeration value more than once.

Example:

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
Error: red may not be used for both TRAFFIC_SIGNAL and COLOR.
```

Structures

Structures are created as objects in the Object Organizer under the register card  **Data types**. They begin with the keywords TYPE and STRUCT and end with END_STRUCT and END_TYPE.

The syntax for structure declarations is as follows:

```
TYPE <Structurename>:
STRUCT
  <Declaration of Variables 1>
  .
  .
  <Declaration of Variables n>
END_STRUCT
END_TYPE
```

<Structurename> is a type that is recognized throughout the project and can be used like a standard data type.

Interlocking structures are allowed. The only restriction is that variables may not be placed at addresses (the AT declaration is not allowed!).

Example for a structure definition named Polygone:

```
TYPE Polygone:
STRUCT
  Start:ARRAY [1..2] OF INT;
  Point1:ARRAY [1..2] OF INT;
  Point2:ARRAY [1..2] OF INT;
  Point3:ARRAY [1..2] OF INT;
  Point4:ARRAY [1..2] OF INT;
  End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

Example for the initialization of a structure:

```
Poly_1:polygone := ( Start:=3,3, Point1 =5,2, Point2:=7,3, Point3:=8,5,  
Point4:=5,7, End := 3,5);
```

You can gain access to structure components using the following syntax:


```
<Structure_Name>.<Componentname>
```

For example, if you have a structure named "Week" that contains a component named "Monday", you can get to it by doing the following:

```
Week.Monday
```

References

You can use the user-defined reference data type to create an alternative name for a variable, constant or function block.

Create your references as objects in the Object Organizer under the register card  **Data types**. They begin with the keyword **TYPE** and end with **END_TYPE**.

Syntax:

```
TYPE <Identifier>: <Assignment term>;  
END_TYPE
```

Example:

```
TYPE message:STRING[50];  
END_TYPE;
```


907 AC 1131 supports all IEC operators. In contrast with the standard functions, these operators are recognized implicitly throughout the project. Operators are used like functions in POU implementation.

ADD

Addition of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

Two TIME variables can also be added together resulting in another time (e.g., t#45s + t#50s = t#1m35s)

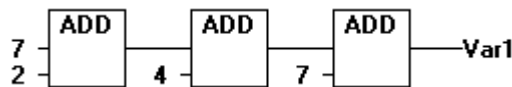
Example in IL:

```
LD    7
ADD   2,4,7
ST    Var 1
```

Example in ST:

```
var1 := 7+2+4+7;
```

Example in FBD:



MUL

Multiplication of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

Example in IL:

```
LD    7
MUL   2,4,7
ST    Var 1
```

Example in ST:

```
var1 := 7*2*4*7;
```

Example in FBD:



SUB

Subtraction of one variable from another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

A TIME variable may also be subtracted from another TIME variable resulting in third TIME type variable. Note that negative TIME values are undefined.

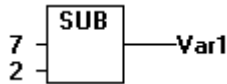
Example in IL:

```
LD      7
SUB     8
ST      Var 1
```

Example in ST:

```
var1 := 7-2;
```

Example in FBD:



DIV

Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

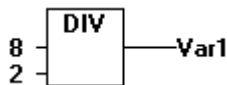
Example in IL:

```
LD      8
DIV     2
ST      Var 1      (* Result is 4 *)
```

Example in ST:

```
var1 := 8/2;
```

Example in FBD:



Hinweis: If you define functions in your project with the names CheckDivByte, CheckDivWord, CheckDivDWord and CheckDivReal, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0 (see chapter 'What is what in 907 AC 1131', 'Project components' 'Function')

MOD

Modulo Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL. The result of this function will be the remainder of the division. This result will be a whole number.

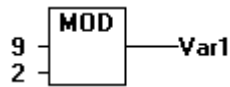
Example in IL:

```
LD      9
MOD     2
ST      Var 1      (* Result is 1 *)
```

Example in ST:

```
var1 := 9 MOD 2;
```

Example in FBD:



INDEXOF

Perform this function to find the internal index for a POU.

Example in ST:

```
var1 := INDEXOF(POU2);
```

SIZEOF

Perform this function to determine the number of bytes required by the given data type.

Example in IL:

```
arr1:ARRAY[0..4] OF INT;  
Var1      INT  
LD        arr1  
SIZEOF  
ST        Var 1      (* Result is 10 *)
```

Example in ST:

```
var1 := SIZEOF(arr1);
```

Bitstring Operators

AND

Bitwise AND of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

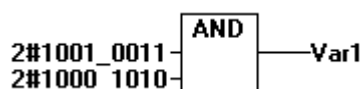
Example in IL:

```
Var1  BYTE  
LD    2#1001_0011  
AND   2#1000_1010  
ST    Var 1      (* Result is 2#1000_0010 *)
```

Example in ST:

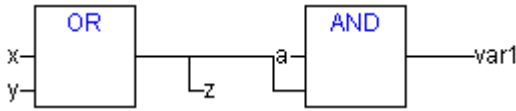
```
var1 := 2#1001_0011 AND 2#1000_1010
```

Example in FBD:





Note: If you have a program step in the SFC like the following



and if you use 68xxx- or C-code generators, please note the following: The allocation of the value of the second input variable at the AND operator module to variable z will not be executed ! This is due to the optimized processing in the SFC in case of value FALSE at the input variable.

OR

Bitwise OR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

Example in IL:

```

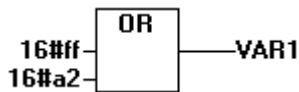
var1 :BYTE;
LD    2#1001_0011
OR    2#1000_1010
ST    var1 (* Result is 2#1001_1011 *)
  
```

Example in ST:

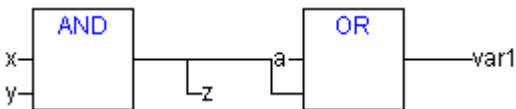
```

Var1 := 2#1001_0011 OR 2#1000_1010
  
```

Example in FBD:



Note: If you have a program step in the SFC like the following



and if you use 68xxx- or C-code generators, please note the following: The allocation of the value of the second input variable at the AND operator module to variable z will not be executed ! This is due to the optimized processing in the SFC in case of value FALSE at the input variable.

XOR

Bitwise XOR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

Example in IL:

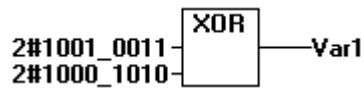
```

Var1 :BYTE;
LD    2#1001_0011
XOR   2#1000_1010
ST    Var1 (* Result is 2#0001_1001 *)
  
```


Example in ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

Example in FBD:



NOT

Bitwise NOT of a bit operand. The operand should be of the type BOOL, BYTE, WORD or DWORD.

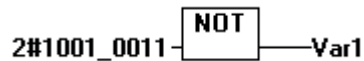
Example in IL:

```
Var1 :BYTE;  
LD    2#1001_0011  
NOT  
ST          Var1 (* Result is 2#0110_1100 *)
```

Example in ST:

```
Var1 := NOT 2#1001_0011
```

Example in FBD:



Bit-Shift Operators

SHL

Bitwise left-shift of an operand : erg:= SHL (in, n)

The input variables erg, in and n should be of the type BYTE, WORD, or DWORD. in will be shifted to the left by n bits and filled with zeros on the right.



Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

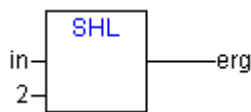
See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:

```
shl_st (PRG-ST)
0001 PROGRAM shl_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR

0001
0002 erg_byte:=SHL (in_byte,n);  (* Ergebnis ist 16#14*)
0003 erg_word:=SHL (in_word,n); (* Ergebnis ist 16#0114 *)
0004
```

Example in FBD:



Example in IL:

```
LD    16#45
SHL   2
ST    erg_byte
```

SHR

Bitwise right-shift of an operand: $erg := SHR (in, n)$

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted to the right by n bits and filled with zeros on the left.

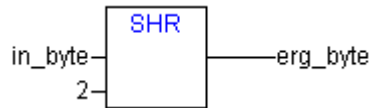
See the following example in hexadecimal notation to notice the results of the arithmetic operation depending on the type of the input variable (BYTE or WORD).

Example in ST:

```
shr_st (PRG-ST)
0001 PROGRAM shr_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR

0001
0002 erg_byte:=SHR (in_byte,n);  (* Ergebnis ist 11*)
0003 erg_word:=SHR (in_word,n); (* Ergebnis ist 0011*)
0004
```

Example in FBD:



Example in IL:

```
LD    16#45
SHL   2
ST    erg_byte
```

ROL

Bitwise rotation of an operand to the left: $erg := ROL(in, n)$

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the left n times while the bit that is furthest to the left will be reinserted from the right.



Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:

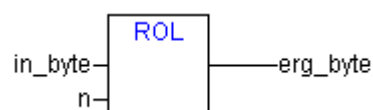
A screenshot of a software window titled 'rol_st (PRG-ST)'. The window displays two sections of code. The top section is a variable declaration block:

```
0001 PROGRAM rol_st
0002 VAR
0003   in_byte:BYTE:=16#45;
0004   in_word:WORD:=16#45;
0005   erg_byte: BYTE;
0006   erg_word: WORD;
0007   n:BYTE:=2;
0008 END_VAR
```

The bottom section shows the assignment of the ROL function to the variables:

```
0001
0002 erg_byte:= ROL (in_byte,n); (* Ergebnis ist 16#15 *)
0003 erg_word:=ROL (in_word,n); (* Ergebnis ist 16#0114 *)
```

Example in FBD:



Example in IL:

```
LD    16#45
SHL   2
ST    erg_byte
```

ROR

Bitwise rotation of an operand to the right: $erg = ROR(in, n)$

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the right n times while the bit that is furthest to the left will be reinserted from the left.

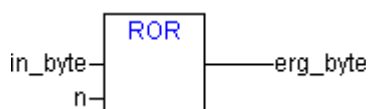


Note: Please note, that the amount of bits, which is regarded for the arithmetic operation, is pretended by the data type of the input variable !. If the input variable is a constant the smallest possible data type is regarded. The data type of the output variable has no effect at all on the arithmetic operation.

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in ST:

Example in FBD:



Example in IL:

```
LD    16#45
SHL   2
ST    erg_byte
```

Selection Operators

All selection operations can also be performed with variables. For purposes of clarity we will limit our examples to the following which use constants as operators.

SEL

Binary Selection.

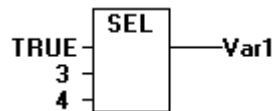
OUT := SEL(G, IN0, IN1) means:
OUT := IN0 if G=FALSE;
OUT := IN1 if G=TRUE.

IN0, IN1 and OUT can be any type of variable, G must be BOOL. The result of the selection is IN0 if G is FALSE, IN1 if G is TRUE.

Example in IL:

```
LD    TRUE
SEL   3,4
ST    Var1      (* Result ist 4 *)
```

Example in FBD:



MAX

Maximum function. Returns the greater of the two values.

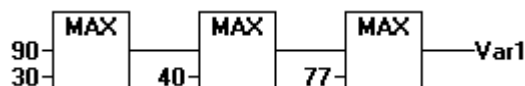
OUT := MAX(IN0, IN1)

IN0, IN1 and OUT can be any type of variable.

Example in IL:

```
LD    90
MAX   30
MAX   40
MAX   77
ST    Var1      (* Result is 90 *)
```

Example in FBD:



MIN

Minimum function. Returns the lesser of the two values.

OUT := MIN(IN0, IN1)

IN0, IN1 and OUT can be any type of variable.

Example in IL:

```
LD    90
MIN   30
MIN   40
MIN   77
ST    Var 1      (* Result is 30 *)
```

Example in FBD:



LIMIT

Limiting

OUT := LIMIT(Min, IN, Max) means:

OUT := MIN (MAX (IN, Min), Max)

Max is the upper and Min the lower limit for the result. Should the value IN exceed the upper limit Max, LIMIT will return Max. Should IN fall below Min, the result will be Min.

IN and OUT can be any type of variable.

Example in IL:

```
LD    90
LIMIT 30,80
ST    Var 1      (*Result is 80 *)
```

MUX

Multiplexer

OUT := MUX(K, IN0, ..., INn) means:

OUT := IN_K.

IN0, ..., INn and OUT can be any type of variable. K must be BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT or UDINT. MUX selects the Kth value from among a group of values.

Example in IL:

```
LD    0
MUX   30,40,50,60,70,80
ST    Var 1      (*Result is 30 *)
```

Comparison Operators

GT

Greater than

A Boolean operator which returns the value TRUE when the value of the first operand is greater than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

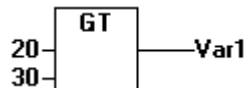
Example in IL:

```
LD    20
GT    30
ST    Var 1      (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
```

Example in FBD:



LT

Less than

A Boolean operator that returns the value TRUE when the value of the first operand is less than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

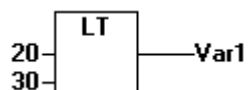
Example in IL:

```
LD    20
LT    30
ST    Var 1      (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 < 30;
```

Example in FBD:



LE

Less than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is less than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

Example in IL:

```
LD    20
LE    30
ST    Var 1      (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 <= 30;
```

Example in FBD



GE

Greater than or equal to

A Boolean operator that returns the value TRUE when the value of the first operand is greater than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

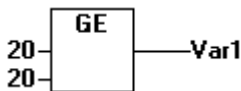
Example in IL:

```
LD    60
GE    40
ST    Var 1      (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 60 >= 40;
```

Example in FBD:



EQ

Equal to

A Boolean operator that returns the value TRUE when the operands are equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

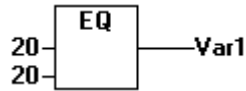
Example in IL:

```
LD    40
EQ    40
ST    Var 1      (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 40 = 40;
```

Example in FBD:



NE

Not equal to

A Boolean operator that returns that value TRUE when the operands are not equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

Example in IL:

```
LD    40
NE    40
ST    Var 1      (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 40 <> 40;
```

Example in FBD:



Address Operators

ADR

Address Function

ADR returns the address of its argument in a DWORD. This address can be sent to manufacturing functions to be treated as a pointer or it can be assigned to a pointer within the project.

Example in IL:

```
LD    Var 1
ADR
ST    Var 2
```

Content Operator

A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example in ST:

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

Calling Operator

CAL

Calling a function block or a program

Use CAL in IL to call up a function block instance. The variables that will serve as the input variables are placed in parentheses right after the name of the function block instance.

Example: Calling up the instance Inst from a function block where input variables Par1 and Par2 are 0 and TRUE respectively.

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

Type Conversion Functions

Its is forbidden to implicitly convert from a "larger" type to a "smaller" type (for example from INT to BYTE or from DINT to WORD). Special type conversions are required if one wants to do this. One can basically convert from any elementary type to any other elementary type.

Syntax:

```
<elem.Typ1>_TO_<elem.Typ2>
```

BOOL_TO Conversions

Conversion from type BOOL to any other type:

For number types the result is 1, when the operand is TRUE, and 0, when the operand is FALSE.

For the STRING type the result is ,TRUE' or ,FALSE'.

Examples in AWL:

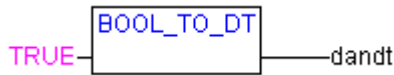
LD TRUE BOOL_TO_INT ST i	(*Result is 1 *)
LD TRUE BOOL_TO_STRING ST str	(*Result is 'TRUE' *)
LD TRUE BOOL_TO_TIME ST t	(*Result is T#1ms *)
LD TRUE BOOL_TO_TOD ST	(*Result is TOD#00:00:00.001 *)
LD FALSE BOOL_TO_DATE ST dat	(*Result is D#1970-01-01 *)
LD TRUE BOOL_TO_DT ST dandt	(*Result is DT#1970-01-01-00:00:01 *)

Examples in St:

i:=BOOL_TO_INT(TRUE);	(* Result is 1 *)
str:=BOOL_TO_STRING(TRUE);	(* Result is "TRUE" *)
t:=BOOL_TO_TIME(TRUE);	(* Result is T#1ms *)
tof:=BOOL_TO_TOD(TRUE);	(* Result is TOD#00:00:00.001 *)
dat:=BOOL_TO_DATE(FALSE);	(* Result is D#1970 *)
dandt:=BOOL_TO_DT(TRUE);	(* Result is DT#1970-01-01-00:00:01 *)

Examples in FUP:

TRUE — [BOOL_TO_INT] — i <input type="checkbox"/>	(*Result is 1 *)
TRUE — [BOOL_TO_STRING] — str	(*Result is 'TRUE' *)
TRUE — [BOOL_TO_TIME] — t	(*Result is T#1ms *)
TRUE — [BOOL_TO_TOD] — tof	(*Result is TOD#00:00:00.001 *)
FALSE — [BOOL_TO_DATE] — dat	(*Result is D#1970-01-01 *)



(*Result is
DT#1970-01-01-00:00:01 *)

TO_BOOL Conversions

Conversion from another variable type to BOOL:

The result is TRUE when the operand is not equal to 0. The result is FALSE when the operand is equal to 0.

The result is true for STRING type variables when the operand is "TRUE", otherwise the result is FALSE.

Examples in AWL:

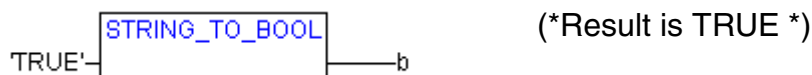
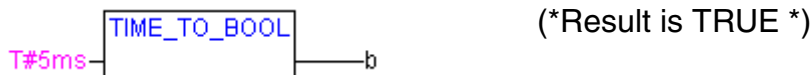
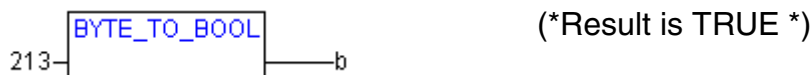
```
LD 213 (*Result is TRUE *)
BYTE_TO_BOOL
ST b
```

```
LD 0 (*Result is FALSE *)
INT_TO_BOOL
ST b
```

```
LD T#5ms (*Result is TRUE *)
TIME_TO_BOOL
ST b
```

```
LD 'TRUE' (*Result is TRUE *)
STRING_TO_BOOL
ST b
```

Examples in FUP:



Examples in St:

```
b := BYTE_TO_BOOL(2#11010101); (* Result is TRUE *)
b := INT_TO_BOOL(0); (* Result is FALSE *)
```

```

b := TIME_TO_BOOL(T#5ms);          (* Result is TRUE *)
b := STRING_TO_BOOL('TRUE');      (* Result is TRUE *)

```

Conversion between Integral Number Types

Conversion from an integral number type to another number type:

When you perform a type conversion from a larger to a smaller type, you risk losing some information. If the number you are converting exceeds the range limit, the first bytes for the number will be ignored.

Example in ST:

```

si := INT_TO_SINT(4223); (* Result is 127 *)

```

If you save the integer 4223 (16#107f represented hexadecimally) as a SINT variable, it will appear as 127 (16#7f represented hexadecimally).

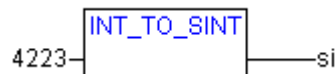
Example in IL:

```

LD          2
INT_TO_REAL
MUL        3.5

```

Example in FBD:



REAL_TO-/ LREAL_TO Conversions

Converting from the variable type REAL or LREAL to a different type:

The value will be rounded up or down to the nearest whole number and converted into the new variable type. Exceptions to this are the variable types STRING, BOOL, REAL and LREAL.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Example in ST:

```

i := REAL_TO_INT(1.5); (* Result is 2 *)
j := REAL_TO_INT(1.4); (* Result is 1 *)

```

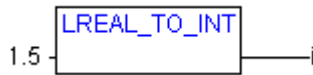
Example in IL:

```

LD          2.7
REAL_TO_INT
GE         %MW8

```

Example in FBD:



TIME_TO/TIME_OF_DAY

Conversions

Converting from the variable type TIME or TIME_OF_DAY to a different type:

The time will be stored internally in a DWORD in milliseconds (beginning with 12:00 A.M. for the TIME_OF_DAY variable). This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For the STRING type variable, the result is a time constant.

Examples in IL:

```
LD    T#12ms                (*Result is 'T#12ms' *)
TIME_TO_STRING
ST    str
```

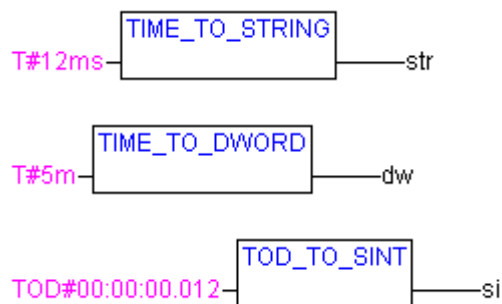
```
LD    T#300000ms           (*Result is 300000 *)
TIME_TO_DWORD
ST    dw
```

```
LD    TOD#00:00:00.012     (*Result is 12 *)
TOD_TO_SINT
ST    si
```

Examples in St:

```
str :=TIME_TO_STRING(T#12ms);      (* Result is T#12ms *)
dw:=TIME_TO_DWORD(T#5m);          (* Result is 300000 *)
si:=TOD_TO_SINT(TOD#00:00:00.012); (* Result is 12 *)
```

Examples in FBD:



DATE_TO/DT_TO Conversions

Converting from the variable type DATE or DATE_AND_TIME to a different type:

The date will be stored internally in a DWORD in seconds since Jan. 1, 1970. This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information

For STRING type variables, the result is the date constant.

Examples in St:

```
b :=DATE_TO_BOOL(D#1970-01-01);      (* Result is FALSE *)
i :=DATE_TO_INT(D#1970-01-15);      (* Result is 29952 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05); (* Result is 129 *)
str:=DT_TO_STRING(DT#1998-02-13-14:20); (* Result is 'DT#1998-02-13-14:20'*)
```

STRING_TO Conversions

Converting from the variable type STRING to a different type:

The operand from the STRING type variable must contain a value that is valid in the target variable type, otherwise the result will be 0.

Examples in St:

```
b :=STRING_TO_BOOL('TRUE');        (* Result is TRUE *)
w :=STRING_TO_WORD('abc34');       (* Result is 0 *)
t :=STRING_TO_TIME('T#127ms');     (* Result is T#127ms *)
```

TRUNC

Converting from REAL to INT. The whole number portion of the value will be used.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Examples in ST:

```
i:=TRUNC(1.9); (* Result is 1 *)
i:=TRUNC(-1.4); (* Result is 1 *)
```

Example in IL:

```
LD      2.7
TRUNC
GE      %MW8
```

Numeric Functions

ABS

Returns the absolute value of a number. ABS(-2) equals 2.

The following type combinations for input and output variables are possible:

IN	OUT
INT	INT, REAL, WORD, DWORD, DINT
REAL	REAL
BYTE	INT, REAL, BYTE, WORD, DWORD, DINT
WORD	INT, REAL, WORD, DWORD, DINT
DWORD	REAL, DWORD, DINT
SINT	REAL
USINT	REAL
UINT	INT, REAL, WORD, DWORD, DINT, UDINT, UINT
DINT	REAL, DWORD, DINT
UDINT	REAL, DWORD, DINT, UDINT

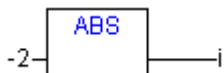
Example in IL:

```
LD    2
ABS
ST    i      (*Result is 2 *)
```

Example in ST:

```
i:=ABS(-2);
```

Example in FBD:



SQRT

Returns the square root of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

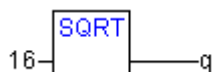
Example in IL:

```
LD    16
SQRT
ST    q      (*Result is 4 *)
```

Example in ST:

```
q:=SQRT(16);
```

Example in FBD:



LN

Returns the natural logarithm of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD    45
LN
ST    q    (*Result is 3.80666 *)
```

Example in ST:

```
q:=LN(45);
```

Example in FBD:



LOG

Returns the logarithm of a number in base 10.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

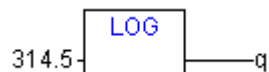
Example in IL:

```
LD    314.5
LOG
ST    q    (*Result is 2.49762 *)
```

Example in ST:

```
q:=LOG(314.5);
```

Example in FBD:



EXP

Returns the exponential function.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD    2
EXP
ST    q    (*Result is 9.7448e+009 *)
```

Example in ST:

```
q:=EXP(2);
```

Example in FBD:



SIN

Returns the sine of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

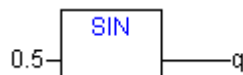
Example in IL:

```
LD    0.5
SIN
ST    q      (*Result is 0.479426 *)
```

Example in ST:

```
q:=SIN(0.5);
```

Example in FBD:



COS

Returns the cosine of number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type Typ REAL.

Example in IL:

```
LD    0.5
COS
ST    q      (*Result is 0.877583 *)
```

Example in ST:q:=COS(0.5);

Example in FBD:



TAN

Returns the tangent of a number. The value is calculated in arch minutes. IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

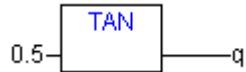
Example in IL:

```
LD    0.5
TAN
ST    q    (*Result is 0.546302 *)
```

Example in ST:

```
q:=TAN(0.5);
```

Example in FBD:



ASIN

Returns the arc sine (inverse function of sine) of a number. .

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

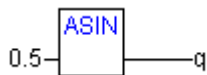
Example in IL:

```
LD    0.5
ASIN
ST    q    (*Result is 0.523599 *)
```

Example in ST:

```
q:=ASIN(0.5);
```

Example in FBD:



ACOS

Returns the arc cosine (inverse function of cosine) of a number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

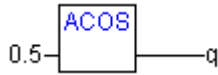
Example in IL:

```
LD    0.5
ABS
ST    q    (*Result is 1.0472 *)
```

Example in ST:

```
q:=ACOS(0.5);
```

Example in FBD:



ATAN

Returns the arc tangent (inverse function of tangent) of a number. The value is calculated in arch minutes.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

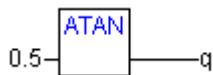
Example in IL:

```
LD    0.5
ABS
ST    q      (*Result is 0.463648 *)
```

Example in ST:

```
q:=ATAN(0.5);
```

Example in FBD:



EXPT

Exponentiation of a variable with another variable:

```
OUT = IN1IN2.
```

IN1 and IN2 can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

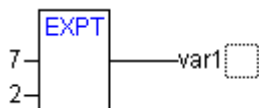
Example in IL:

```
LD    7
EXPT  2
ST    var1  (*Result is 49 *)
```

Example in ST:

```
var1 := (7,2);
```

Example in FBD:



String Functions

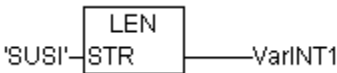
LEN

Returns the length of a string. Input STR is of type STRING, the return value of the function is type INT.

Example in IL:

```
LD      'SUSI'
LEN
ST      VarINT1   (* Ergebnis ist 4 *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := LEN ('SUSI');
```

LEFT

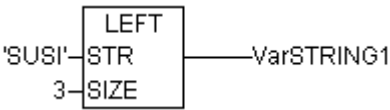
Left returns the left, initial string for a given string. Input STR is type STRING, SIZE is of type INT, the return value of the function is type STRING.

LEFT (STR, SIZE) means: Take the first SIZE character from the right in the string STR.

Example in IL:

```
LD      'SUSI'
LEFT    3
ST      VarSTRING1 (* Ergebnis ist 'SUS' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := LEFT ('SUSI',3);
```

RIGHT

Right returns the right, initial string for a given string.

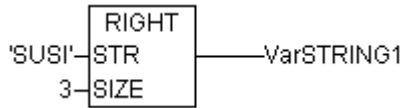
RIGHT (STR, SIZE) means: Take the first SIZE character from the right in the string STR.

Input STR is of type STRING, SIZE is of type INT, the return value of the function is of type STRING.

Example in IL:

```
LD      'SUSI'  
RIGHT  3  
ST      VarSTRING1 (* Ergebnis ist 'USI' *)
```

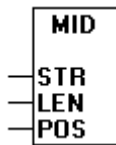
Example in FBD:



Example in ST:

```
VarSTRING1 := RIGHT ('SUSI',3);
```

MID



Mid returns a partial string from within a string.

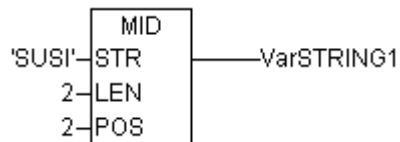
Input STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

MID (STR, LEN, POS) means: Retrieve LEN characters from the STR string beginning with the character at position POS.

Example in IL:

```
LD      'SUSI'  
RIGHT  2,2  
ST      VarSTRING1 (* Ergebnis ist 'US' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := MID ('SUSI',2,2);
```

CONCAT

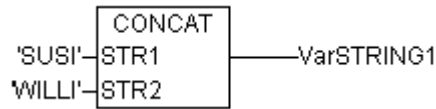
Concatenation (combination) of two strings.

The input variables STR1 and STR2 as well as the return value of the function are type STRING.

Example in IL:

```
LD      'SUSI'  
CONCAT  'WILLI'  
ST      VarSTRING1    (* Ergebnis ist 'SUSIWILLI' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := CONCAT ('SUSI','WILLI');
```

INSERT

INSERT inserts a string into another string at a defined point.

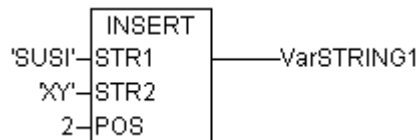
The input variables STR1 and STR2 are type STRING, POS is type INT and the return value of the function is type STRING.

INSERT(STR1, STR2, POS) means: insert STR2 into STR1 after position POS.

Example in IL:

```
LD      'SUSI'  
INSERT  'XY',2  
ST      VarSTRING1    (* Ergebnis ist 'SUXYSI' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := INSERT ('SUSI','XY',2);
```

DELETE

DELETE removes a partial string from a larger string at a defined position.

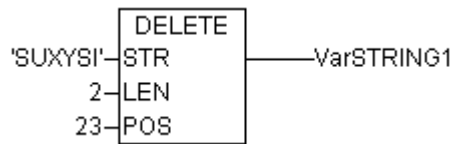
The input variable STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

DELETE(STR, L, P) means: Delete L characters from STR beginning with the character in the P position.

Example in IL:

```
LD      'SUXYSI'  
DELETE  2,23  
ST      Var1      (* Ergebnis ist 'SUSI' *)
```

Example in FBD:



Example in ST:

```
Var1 := DELETE ('SUXYSI',2,3);
```

REPLACE

REPLACE replaces a partial string from a larger string with a third string.

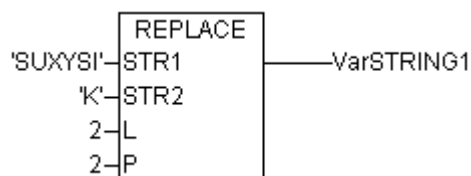
The input variable STR1 and STR2 are type STRING, LEN and POS are type INT, the return value of the function is type STRING.

REPLACE(STR1, STR2, L, P) means: Replace L characters from STR1 with STR2 beginning with the character in the P position.

Example in IL:

```
LD      'SUXYSI'  
REPLACE 'K', 2,2  
ST      VarSTRING1 (* Ergebnis ist 'SKYSI' *)
```

Example in FBD:



Example in ST:

```
VarSTRING1 := REPLACE ('SUXYSI','K',2,2);
```

FIND

FIND searches for a partial string within a string.

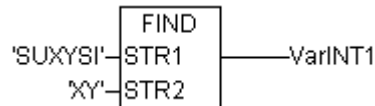
The input variable STR1 and STR2 are type STRING, the return value of the function is type STRING.

FIND(STR1, STR2) means: Find the position of the first character where STR2 appears in STR1 for the first time. If STR2 is not found in STR1, then OUT:=0.

Example in IL:

```
LD      'SUXYSI'  
FIND    'XY'  
ST      VarINT1      (* Ergebnis ist '3' *)
```

Example in FBD:



Example in ST:

```
VarINT1 := FIND ('SUXYSI','XY');
```

Bistable Function Blocks

SR

Making Bistable Function Blocks Dominant:

Q1 = SR (SET1, RESET) means:

$$Q1 = (\text{NOT RESET AND } Q1) \text{ OR SET1}$$

The input variables SET1 and RESET as well as the output variable Q1 are type BOOL.

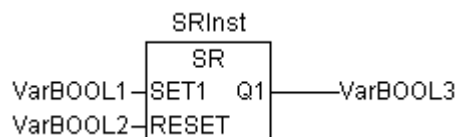
Declaration example:

```
SRInst : SR ;
```

Example in IL:

```
CAL      SRInst(SET1 := VarBOOL1, RESET := VarBOOL2)  
LD      SRInst.Q1  
ST      VarBOOL3
```

Example in FBD:



Example in ST:

```
SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2 );  
VarBOOL3 := SRInst.Q1 ;
```

RS

Resetting Bistable Function Blocks

Q1 = RS (SET, RESET1) means:

$$Q1 = \text{NOT RESET1 AND } (Q1 \text{ OR SET})$$

The input variables SET and RESET1 as well as the output variable Q1 are type BOOL.

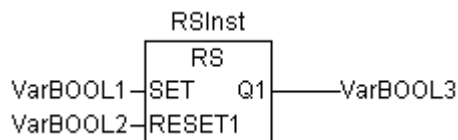
Declaration example:

```
RSInst : RS ;
```

Example in IL:

```
    CAL      RSInst(SET := VarBOOL1, RESET1 := VarBOOL2)
    LD        RSInst.Q1
    ST        VarBOOL3
```

Example in FBD:



Example in ST:

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1 ;
```

SEMA

A Software Semaphore (Interruptible)

BUSY = SEMA(CLAIM, RELEASE) means:

```
BUSY := X;
IF CLAIM THEN X:=TRUE;
ELSE IF RELEASE THEN BUSY := FALSE; X:= FALSE;
END_IF
```

X is an internal BOOL variable that is FALSE when it is initialized. The input variables CLAIM and RELEASE as well as the output variable BUSY are type BOOL.

If BUSY is TRUE when SEMA is called up, this means that a value has already been assigned to SEMA (SEMA was called up with CLAIM = TRUE). If BUSY is FALSE, SEMA has not yet been called up or it has been released (called up with RELEASE = TRUE).

Declaration example:

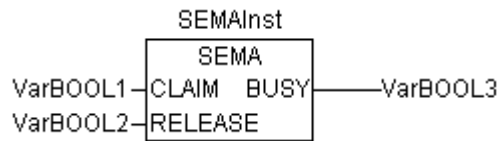
```
SEMAInst : SEMA ;
```

Example in IL:

```
    CAL      SEMAInst(CLAIM := VarBOOL1, RELEASE := VarBOOL2)
```

```
LD      SEMAInst.BUSY
ST      VarBOOL3
```

Example in FBD:



Example in ST:

```
SEMAInst(CLAIM:= VarBOOL1 , RELEASE:=VarBOOL2 );
VarBOOL3 := SEMAInst.BUSY;
```

Trigger

R_TRIG

The function block R_TRIG detects a rising edge.

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
  CLK : BOOL;
END_VAR
VAR_OUTPUT
  Q : BOOL;
END_VAR
VAR
  M : BOOL := FALSE;
END_VAR
  Q0 := CLK AND NOT M;
  M := CLK;
END_FUNCTION_BLOCK
```

The output Q0 and the help variable M will remain FALSE as long as the input variable CLK is FALSE. As soon as S1 returns TRUE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has falling edge followed by an rising edge.

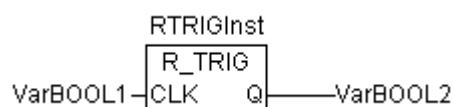
Declaration example:

```
RTRIGInst : R_TRIG ;
```

Example in IL:

```
CAL      RTRIGInst(CLK := VarBOOL1)
LD      RTRIGInst.Q
ST      VarBOOL2
```

Example in FBD:



Example in ST:

```
RTRIGInst(CLK:= VarBOOL1);  
VarBOOL2 := RTRIGInst.Q;
```

F_TRIG

The function block F_TRIG a falling edge.

```
FUNCTION_BLOCK F_TRIG  
VAR_INPUT  
  CLK: BOOL;  
END_VAR  
VAR_OUTPUT  
  Q: BOOL;  
END_VAR  
VAR  
  M: BOOL := FALSE;  
END_VAR  
  Q := NOT CLK AND NOT M;  
  M := NOT CLK;  
END_FUNCTION_BLOCK
```

The output Q and the help variable M will remain FALSE as long as the input variable CLK returns TRUE. As soon as CLK returns FALSE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has a rising followed by a falling edge.

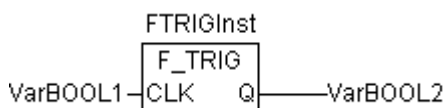
Declaration example:

```
FTRIGInst : F_TRIG ;
```

Example in IL:

```
  CAL      FTRIGInst(CLK := VarBOOL1)  
  LD      FTRIGInst.Q  
  ST      VarBOOL2
```

Example in FBD:



Example in ST:

```
FTRIGInst(CLK:= VarBOOL1);  
VarBOOL2 := FTRIGInst.Q;
```

Counter

CTU

The function block Incrementer:

The input variables CU and RESET as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type WORD.

The counter variable CV will be initialized with 0 if RESET is TRUE. If CU has a rising edge from FALSE to TRUE, CV will be raised by 1. Q will return TRUE when CV is greater than or equal to the upper limit PV.

Declaration example:

```
CTUInst : CTU ;
```

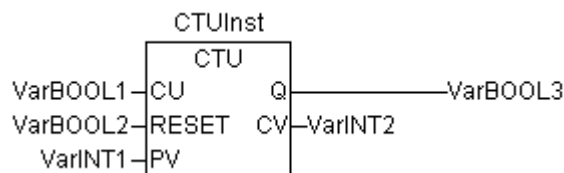
Example in IL:

```

CAL      CTUInst(CU := VarBOOL1, RESET := VarBOOL2, PV := VarINT1)
LD       CTUInst.Q
ST       VarBOOL3
LD       CTUInst.CV
ST       VarINT2

```

Example in FBD:



Example in ST:

```

CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTUInst.Q ;
VarINT2 := CTUInst.CV;

```

CTD

Function Block Decrementer:

The input variables CD and LOAD_ as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type INT.

When LOAD_ is TRUE, the counter variable CV will be initialized with the upper limit PV. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided CV is greater than 0 (i.e., it doesn't cause the value to fall below 0).

Q returns TRUE when CV is equal 0.

Declaration example:

```
CTDInst : CTD ;
```

Example in IL:

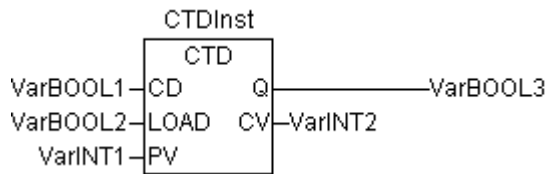
```

CAL      CTDInst(CD := VarBOOL1, LOAD := VarBOOL2, PV := VarINT1)
LD       CTDInst.Q
ST       VarBOOL3

```

```
LD      CTDInst.CV
ST      VarINT2
```

Example in FBD:



Example in ST:

```
CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTDInst.Q ;
VarINT2 := CTDInst.CV;
```

CTUD

Function Block Incrementer/Decrementer

The input variables CU, CD, RESET, LOAD_ as well as the output variables QU and QD are type BOOL, PV and CV are type WORD.

If RESET is valid, the counter variable CV will be initialized with 0. If LOAD is valid, CV will be initialized with PV.

If CU has a rising edge from FALSE to TRUE, CV will be raised by 1. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided this does not cause the value to fall below 0.

QU returns TRUE when CV has become greater than or equal to PV.

QD returns TRUE when CV has become equal to 0.

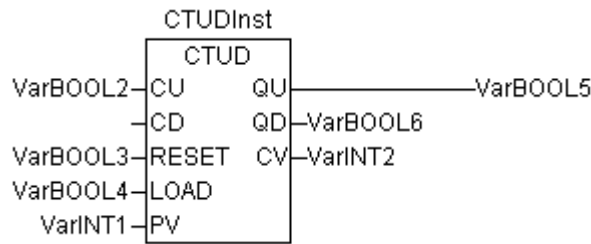
Declaration example:

```
CTUDInst : CUTD ;
```

Example in IL:

```
CAL      CTUDInst(CU := VarBOOL2, RESET := VarBOOL3, LOAD :=
           VarBOOL4, PV := VarINT1)
LD       CTUDInst.QU
ST       VarBOOL5
LD       CTUDInst.QD
ST       VarBOOL6
LD       CTUDInst.CV
ST       VarINT2
```

Example in FBD:



Example in ST:

```

CTUDInst(CU := VarBOOL1, CU:= VarBOOL2, RESET := VarBOOL3,
LOAD:=VarBOOL4 , PV:= VarINT1);
VarBOOL5 := CTUDInst.QU ;
VarBOOL6 := CTUDInst.QD ;
VarINT2 := CTUDInst.CV;

```

Timer

TP

The function blockTimer is a trigger.

TP(IN, PT, Q, ET) means:

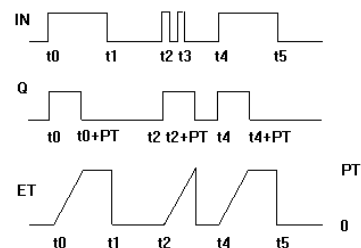
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE if IN is TRUE and ET is less than or equal to PT. Otherwise it is FALSE.

Q returns a signal for the time period given in PT.

Graphic Display of the TP Time Sequence



Declaration example:

```

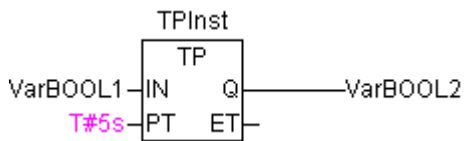
TPInst : TP ;

```

Example in IL:

```
CAL      TPIInst(IN := VarBOOL1, PT := T#5s)
LD       TPIInst.Q
ST       VarBOOL2
```

Example in FBD:



Example in ST:

```
TPIInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TPIInst.Q;
```

TON

The function block Timer On Delay implements a turn-on delay.

TON(IN, PT, Q, ET) means:

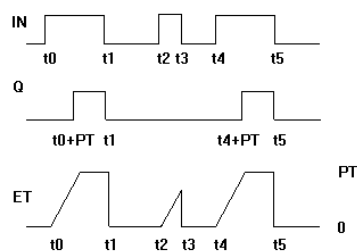
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE when IN is TRUE and ET is equal to PT. Otherwise it is FALSE.

Thus, Q has a rising edge when the time indicated in PT in milliseconds has run out.

Graphic display of TON behavior over time:



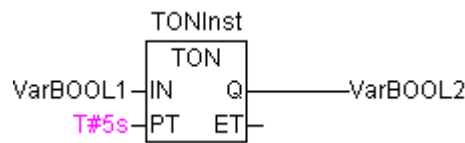
Declaration example:

```
TONInst : TON ;
```

Example in IL:

```
CAL      TONInst(IN := VarBOOL1, PT := T#5s)
LD       TONInst.Q
ST       VarBOOL2
```


Example in FBD:



Example in ST:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

TOF

The function block TOF implements a turn-off delay.

TOF(IN, PT, Q, ET) means:

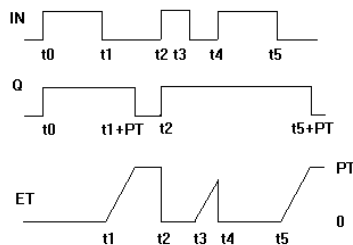
IN and PT are input variables type BOOL respectively TIME. Q and E are output variables type BOOL respectively TIME. If IN is TRUE, the outputs are TRUE respectively 0.

As soon as IN becomes FALSE, in ET the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is FALSE when IN is FALSE und ET equal PT. Otherwise it is TRUE.

Thus, Q has a falling edge when the time indicated in PT in milliseconds has run out.

Graphic display of TOF behavior over time:



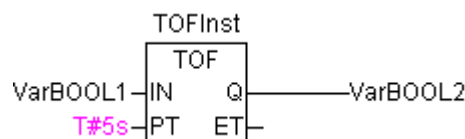
Declaration example:

```
TOFInst : TOF ;
```

Example in IL:

```
CAL      TOFInst(IN := VarBOOL1, PT := T#5s)
LD       TOFInst.Q
ST       VarBOOL2
```

Example in FBD:



Example in ST:

```
TOFInst(IN := VarBOOL1, PT:= T#5s);  
VarBOOL2 :=TOFInst.Q;
```

Operands

Constants, variables, addresses and possibly function calls can appear as operands.

Constants

BOOL Constants

BOOL constants are the logical values TRUE and FALSE.

TIME Constants

TIME constants can be declared in 907 AC 1131 . These are generally used to operate the timer in the standard library. A TIME constant is always made up of an initial "t" or "T" (or "time" or "TIME" spelled out) and a number sign "#".

This is followed by the actual time declaration which can include days (identified by "d"), hours (identified by "h"), minutes (identified by "m"), seconds (identified by "s") and milliseconds (identified by "ms"). Please note that the time entries must be given in this order according to length (d before h before m before s before m before ms) but you are not required to include all time increments.

Examples of correct TIME constants in a ST assignment:

```
TIME1 := T#14ms;  
TIME1 := T#100S12ms;    (*The highest component may be  
allowed to exceed its limit*)  
TIME1 := t#12h34m15s;
```

the following would be incorrect:

```
TIME1 := t#5m68s;      (*limit exceeded in a lower component*)  
TIME1 := 15ms;        (*T# is missing*)  
TIME1 := t#4ms13d;    (*Incorrect order of entries*)
```

DATE Constants

These constants can be used to enter dates. A DATE constant is declared beginning with a "d", "D", "DATE" or "date" followed by "#". You can then enter any date with format Year-Month-Day.

Examples:

```
DATE#1996-05-06  
d#1972-03-29
```

TIME_OF_DAY Constants

Use this type of constant to store times of the day. A TIME_OF_DAY declaration begins with "tod#", "TOD#", "TIME_OF_DAY#" or "time_of_day#" followed by a time with the format: Hour:Minute:Second. You can enter seconds as real numbers or you can enter fractions of a second.

Examples:

```
TIME_OF_DAY#15:36:30.123  
tod#00:00:00
```

DATE_AND_TIME Constants

Date constants and the time of day can also be combined to form so-called DATE_AND_TIME constants. DATE_AND_TIME constants begin with "dt#", "DT#", "DATE_AND_TIME#" or "date_and_time#". Place a hyphen after the date followed by the time.

Examples:

```
DATE_AND_TIME#1996-05-06-15:36:30  
dt#1972-03-29-00:00:00
```

Number Constants

Number values can appear as binary numbers, octal numbers, decimal numbers and hexadecimal numbers. If an integer value is not a decimal number, you must write its base followed by the number sign (#) in front of the integer constant. The values for the numbers 10-15 in hexadecimal numbers will be represented as always by the letters A-F.

You may include the underscore character within the number.

Examples:

```
14                (Decimal number)  
2#1001_0011      (Binary number)  
8#67             (Octal number)  
16#A             (Hexadecimal number)
```

These number values can be from the variable types BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL or LREAL.

Implicit conversions from "larger" to "smaller" variable types are not permitted. This means that a DINT variable cannot simply be used as an INT variable. You must use the type conversion (see the Type Conversions chapter in the appendix).

REAL/LREAL Constants

REAL and LREAL constants can be given as decimal fractions and represented exponentially. Use the standard American format with the decimal point to do this.

Example:

```
7.4 instead of 7,4  
1.64e+009 instead of 1,64e+009
```

STRING Constants

A string is a sequence of characters. STRING constants are preceded and followed by single quotation marks. You may also enter blank spaces and

special characters (umlauts for instance). They will be treated just like all other characters.

In character sequences, the combination of the dollar sign (\$) followed by two hexadecimal numbers is interpreted as a hexadecimal representation of the eight bit character code. In addition, the combination of two characters that begin with the dollar sign are interpreted as shown below when they appear in a character sequence:

\$\$	Dollar signs
\$'	Single quotation mark
\$L or \$l	Line feed
\$N or \$n	New line
\$P or \$p	Page feed
\$R or \$r	Line break
\$T or \$t	Tab

Examples:

```
'w1Wüß?'  
' Abby and Craig '  
':-)'
```

Variables

Variables can be declared either locally in the declaration part of a POU or in a global variable list.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The first 32 characters are significant.

Variables can be used anywhere the declared type allows for them.

You can access available variables through the Input Assistant.

System Flags

System flags are implicitly declared variables that are different on each specific PLC. To find out which system flags are available in your system, use the command "**Insert**" "**Operand**". An Input Assistant dialog box pops up, select the category **System Variable**.

Accessing variables for arrays,
structures and POU's.

Two-dimensional array components can be accessed using the following syntax:

<Fieldname>[Index1, Index2]

Structure variables can be accessed using the following syntax:

<Structurename>.<Variablename>

Function block and program variables can be accessed using the following syntax:

<Functionblockname>.<Variablename>

Addresses

Address

The direct display of individual memory locations is done through the use of special character sequences. These sequences are a concatenation of the percent sign "%", a range prefix, a prefix for the size and one or more natural numbers separated by blank spaces.

The following range prefixes are supported:

I	Input
Q	Output
M	Memory location

The following size prefixes are supported:

X	Single bit
None	Single bit
B	Byte (8 Bits)
W	Word (16 Bits)
D	Double word (32 Bits)

Examples:

%QX75 and %Q75	Output bit 75
%IW215	Input word 215
%QB7	Output byte 7
%MD48	Double word in memory position 48 in the memory location.
%IW2.5.7.1	depending on the PLC Configuration

The current PLC Configuration for the program determines whether or not an address is valid.

Memory location

You can use any supported size to access the memory location.

For example, the address %MD48 would address bytes numbers 192, 193, 194, and 195 in the memory location area ($48 * 4 = 192$). The number of the first byte is 0.

You can access words, bytes and even bits in the same way: the address %MX5.0 allows you to access the first bit in the fifth word (Bits are generally saved wordwise).

Functions

In ST a function call can also appear as an operand.

Example: `Result := Fct(7) + 3;`



Command Line Commands

When 907 AC 1131 is started, you can add commands in the command line which will be asserted during execution of the program. These commands start with a „/“. Capitalization/Use of small letters is not regarded. The commands will be executed sequentially from the left to the right.

/debug	Additional debug outputs like listings etc. are activated.
/online	Immediately after start 907 AC 1131 tries to go online with the current project.
/run	After login 907 AC 1131 starts the application program. Only valid in combination with /online.
/show ...	Settings for the 907 AC 1131 frame window can be made.
/show hide	The window will not be displayed, it also will not be represented in the task menu.
/show icon	The window will be minimized in display.
/show max	The window will be maximized in display.
/show normal	The window will be displayed in the same status as it was during the last closing.
/out <outfile>	All messages are displayed in the message window and additionally are written in the file <outfile>.
/cmd <cmdfile>	After starting the commands of the <cmdfile> get executed.

example for a command line:

The project ampel.pro gets opened, but no window opens. The commands included in the command file command.cmd will be executed.

```
C:\ampel.pro /show hide /cmd command.cmd
```

Command File (cmdfile) Commands

See the following table for a list of commands, which can be used in a command file (<cmdfile>). The command file you can call by a command line (see above) aufrufen können. Capitalizing/Use of small letters is not regarded. The command line will be displayed as a message in the message window and can be given out in a message file (see below). Additionally to the command a „@“ is prefixed. All signs after a semicolon (;) will be ignored (comment).

Commands of the online menu:

online login	Login with the loaded project ('Online Login')
online logout	Logout ('Online' 'Logout')

online run	Start of the application program ('Online' 'Run')
online sim	Switch on of simulation mode (✓) 'Online' 'Simulation')
online sim off	Switch off of simulation mode ('Online' 'Simulation')

Commands of the file menu

file new	A new project is created ('File' 'New')
file open <projectfile>	The project <projectfile> will be loaded ('File' 'Open')
file close	The current project will be closed ('File' 'Close')
file save	The current project will be stored ('File' 'Save')
file saveas <projectfile>	The current project will be saved with the file name <projectfile> ('File' 'Save as')
file quit	907 AC 1131 will be closed ('File' 'Exit')

Commands of the project menu:

project compile	The current project will be compiled by "Rebuild all" ('Project' 'Rebuild all')
project check	The current project will be checked ('Project' 'Check')
project build	The current project will be built ('Project' 'Build')
project import <file1> ... <fileN>	The files <file1> ... <fileN> get imported into the current project ('Project' 'Import')
project export <expfile>	The current project will be exported in the file <expfile> ('Project' 'Export')
project expmul	Each object of the current project will be exported in an own file, which gets the name of the object.

Commands for the control of the message file:

out open <msgfile>	The file <msgfile> opens as message file. New messages will be appended
out close	The currently shown message file will be closed.
out clear	All messages of the currently opened message file will be deleted.

Commands for the control of messages:

echo on	The command lines will be displayed as messages.
echo off	The command lines will not be displayed as messages.
echo <text>	<text> will be displayed in the message window.

Commands for the control of replace of objects respectively for the control of files for import, export, replace:

replace ok	Replace
-------------------	---------

replace yes	
replace no	Do not replace
replace yesall	Replace all
replace noall	Replace none

Commands for the control of the default parameters of 907 AC 1131 dialogs:

query on	Dialogs are displayed and need user input
query off ok	All dialogs respond as if the user had clicked on the 'OK' button
query off no	All dialogs respond as if the user had clicked on the 'No' button
query off cancel	All dialogs respond as if the user had clicked on the 'Cancel' button

debug command:

debug	corresponds to the command "/debug" in the command line
--------------	---

Command for calling command files as subroutines:

call <parameter1> ... <parameter10>	Command files are called as subroutines. Up to ten parameters can be consigned. In the subroutine called you can access the parameters using \$0 - \$9.
--	---

Command for setting the libraries used by 907 AC 1131 :

dir lib <libdir>	<libdir> is set as library directory
dir compile <compiledir>	<compiledir> is set as directory for compile files

Command for setting a delay time concerning execution of the CMDFILES:

delay 5000	Wartet 5 Sekunden
-------------------	-------------------

Commands for control of the Watch and Receipt Manager:

watchlist load <file>	The watch list saved in <file> will be loaded Watchliste and the appropriate window will be opened ('Extras' 'Load Watch List')
watchlist save <file>	Saves the current watch list in <file> ('Extras' 'Save Watch List')
watchlist set <text>	A previous loaded watch list gets the name <text> ('Extras' 'Rename Watch List')
watchlist read	The values of the watch variables are updated ('Extras' 'Read Receipt')
watchlist write	The values of the watch list are written to the watch variables('Extras' 'Write Receipt')

Example of a command file:

A command file like shown below will open the project file ampel.pro, will then load a watch list, which was stored as w.wtc, will then start the application program and write – after 1 second delay - the values of the variables into the watch list watch.wtc (which will be saved) and will finally close the project.

```
file open C:\work\projects\907 AC 1131 _test\ampel.pro
query off ok
watchlist load c:\work\w.wtc
online login
online run
delay 1000
watchlist read
watchlist save c:\work\watch.wtc
online logout
file close
```

Here you will find the error messages that the parser displays (*italics*) and possible causes. They are shown in alphabetical order:

*"ADR does not require an expression or constant
or addressed variable as operand"*

Replace the term or the constant with a variable.

""Function name not allowed here""

Replace the function call with a variable or a constant.

*"<Number> operands is too many for
<operator>. Exactly <number> are needed."*

Check to see how many operands the operator <operator> requires and insert the ones that are missing.

*"<Number> operands is too few for <operator>.
At least <number> are needed"*

Check to see how many operands the operator <operator> requires and remove those that aren't needed.

*"Only BOOL variables are allowed at a bit
address"*

Change the type of declaration to BOOL or change the address to a different format.

"IL Operator Expected"

Change the first word in the line to a valid operator or a valid function.

*"POU ends incorrectly: add ST or delete the last
expression."*

The POU ends with an incomplete expression. Add the correct ending or delete it.

"POU <Name> is not defined in the project"

Define a POU named <name> using the menu command "Project" "Object add" or change the name to the name of the POU defined.

"POU <Name>needs exactly <number> inputs"

Check the number of input variables this POU requires, then add or remove them as needed.

"Identifier expected"

Enter a valid identifier at the beginning of the declaration part.

"CAL, CALC, or CALN require function block instance as operand"

Declare an instance for the function block that you would like to call up.

"<Component> is not a component of <variable>"

If the variable is a structure, change the component into one of the components that are declared in this structure.

If the variable is a function block instance, change the <component> into an input or output parameter that is declared in this function block.

"Index expression of an array must be type INT"

Change the index into a constant or an INT type variable.

"Conditional Operator requires type BOOL"

The result of the previous instruction is not a BOOL type variable. Insert an operator or a function whose result is BOOL.

"Name used in interface is not identical with POU name"

Rename your POU with the menu command **"Project" "Object Rename"** or change the name of the POU in its declaration part. The name must appear directly after the keywords, PROGRAM, FUNCTION or FUNCTIONBLOCK.

"End value of FOR statement must be of type INT"

Change the variable to an INT type variable.

"Increment value of FOR statement must be of type INT"

Change the variable to an INT type variable.

"Step name is no identifier: <name>"

Change the identifier <name> to a valid identifier

"CASE requires selector of an integer type"

Change the selector to an INT type selector.

"Start value of FOR statement must be of type INT"

Change the variable to an INT type variable.

"Variable of FOR statement must be of type INT."

Change the variable to an INT type variable.

"Expression in FOR statement is not variable with write access"

Change the variable to a variable with write permission.

"It is not possible to locate an array of strings to an address"

Clear the address assignment.

"It is not possible to locate an array of an array to an address"

Clear the address assignment.

"Extra characters following valid watch expression"

Remove the extra characters.

"Function block call requires function block instance"

Insert the name of the desired instance or remove the call for the function block.

"A jump must have exactly one label"

Change the jump destination to a defined label.

"END_STRUCT or identifier expected"

A structure definition must end with the keyword END_STRUCT.

"END_VAR or identifier expected"

Write in valid identifier or END_VAR at the beginning of the declaration part.

"At most 4 numerical fields allowed in addresses"

Remove the extra address fields.

"Expression expected"

An expression must be entered at this point.

"EXIT outside a loop"

Remove EXIT

"Error in initial value"

Enter a constant (constants) for the initial value which corresponds to the declaration type.

"Too many parameters in function <Name>"

Delete the extra parameters.

"<Name> function has too few parameters"

Add the missing parameters.

*"No instance specified for call of function block
<name>"*

Change the text of the instance for function block <name> (initialized with "Instance") in the identifier of a valid instance declaration.

"Integer number or symbolic constant expected"

Only integers or symbolic constants can be used as the condition for a CASE instance. Change the incorrect condition.

"<Identifier> is not a function"

Change <identifier> into one of the functions from the libraries that are linked to the project or into one of the function declared in the project.

*"IF and ELSIF require a Boolean expression for
the condition"*

Change the expression to an expression with a BOOL type result.

"Illegal time constants"

Check to see if the time constant you wrote is correct and change any mistakes you find. Possible mistakes are:
The t or # is missing at the beginning.
A time entry appears twice (e.g., t#4d2d).

Incorrect sequence of times.

Incorrect time indicator (the d, h, m, s or ms is missing).

"[<index>] needs array variables"

Declare the identifier before the bracket as an array or change it into a declared array variable.

"INI operator needs function block instance or a data unit type instance"

Change the operands into a function block instance. To do this, declare the operand as a function block or use a previously declared function block, or use a data unit type instance.

*"No *.obj found"*

Turn on Simulation Mode.

"Label in brackets not allowed"

Remove the label or the parentheses.

"No write access to variable <Name> allowed"

Change <name> into a variable with write permission.

"Comments are only allowed at the end of the line in IL"

Write the comments at the end of the line.

"LD expected"

The instruction "LD" is the only one allowed in this line.

"It is not possible to locate a multidimensional array to an address"

Clear the address assignment.

"Duplicate definition of identifier <name>"

Rename one of the identifiers.

"Duplicate definition of label <names>"

Remove one of the defined labels.

"Multiple underscore in identifier"

Remove one of the underscore characters from the identifier.

"At least one statement is required"

Enter an instruction.

"Address expected after 'AT'"

Insert a valid address after the AT or change the keyword AT.

"Number expected after '+' or '-"

Change the word after + or - into a valid constant.

"No comma allowed after ')"

Remove the comma.

"Number is expected after ','"

Remove the comma or insert an additional number.

"<Name> is not an input variable of the called function block"

Check over the input variables for the called function block and change <name> into one of these variables.

"<Name> is no function block"

Replace <name> with the name of a valid function block.

"<Name> must be a declared instance of the function block <FBName>"

Change the text of the function block instance (initialized with "Instance") into an identifier for a valid function block instance declaration.

"N' modifier requires a BOOL type operand"

Remove the N and negate the operand explicitly with the NOT operator

"NOT requires an operand of type BOOL"

Change the operand into a BOOL type operand.

"Only VAR and VAR_GLOBAL can be located to addresses"

Copy the declaration into a VAR or VAR_GLOBAL area.

"Variable with write access or direct address required for ST, STN, S, R"

Replace the first operand with a variable that has write permission

"Operand expected"

Add an additional operand.

"<Operator> in parentheses is not allowed"

This operator is not allowed within parentheses. Either remove the parentheses or the operator.

"Operator is not extendable. Remove the surplus operands"

Check the number of operands for this operator and remove the surplus operands.

"Type mismatch in parameter <number>: Cannot convert <type> to <type>."

Check the type of the operand with the number <number> of this operator, function or function block. Change the type of the variable that caused the error to a type that is allowed or select a new variable of an allowed type.

"Closing bracket with no corresponding opening bracket"

Remove the end bracket or insert the beginning one.

"Keywords must be uppercase"

Change how the keyword is written.

"Step names are duplicated: '<Name>'"

Change one of the names.

"Jump to an undefined step: <Name>"

Change <name> into the name of an existing step or add a step named <name>.

"Jump and Return require an Boolean input"

The result of the previous instruction is not a BOOL result. Insert an operator or a function with a result of the type BOOL.

"Jump and Return are only allowed on the right side of a network"

Delete the jump or return that is not allowed.

"<Label> label is not defined"

Define a label with the name <LabelName> or change <LabelName> into a defined label.

"<string> is not an operator"

Change <string> into a valid operator

"Expecting type specification"

Write a valid type behind the identification in the declaration

"Unknown type: <string>"

Change <string> into a valid type.

"Unrecognized variable or address"

This watch variable is not declared in the project. Press <F2> to access help with declared variables.

"Unexpected End"

In the declaration part: Add the keyword END_VAR to the end of the declaration part.

In the text editor: Insert instructions that end the last instruction sequence (e.g., ST).

"Unexpected end of text in brackets"

Insert an end bracket.

"UNTIL requires a BOOL expression as condition"

Change the expression to an expression with a BOOL type result.

"Type mismatch: Cannot convert <Type1> into <Type2>."

Check the required types of operators (search for Operator in your help file) and change the variable type that produced the error into a type that is allowed or select another variable.

"Invalid address: <Address>"

Check in your PLC Configuration to see which addresses are allowed and replace the addresses with permissible addresses or change the PLC Configuration.

*"Type mismatch on input _I_variable <name>:
Cannot convert <Type1> into <Type2>."*

A value that is <Type2> (which is not allowed) is assigned to the variable <name>. Change the variable or the constant into a variable or constant of the type <Type1>.

*"Type mismatch in parameter <name> of
<name>: Cannot convert <Type1> into
<Type2>."*

Use a <Type2> type variable for the assignment to the <name> parameter or change the type of assigned variable to <Type1>.

*"Type mismatch in parameter <Parameter> of
<POU>: Cannot convert <Type1> into
<Type2>."*

Check to see what type of <parameter> parameter are required in the <POU> POU. Change the type of the variable that caused the error to <Type2> or select another variable that is <Type2>.

*"Invalid characters follow the valid expression:
'<name>'"*

Remove the extra characters.

"Identifier <name> not defined"

Declare this variable in the declaration part of the POU or in the global variable list.

*"VAR, VAR_INPUT, VAR_OUTPUT or
VAR_INOUT expected"*

The first line after the name of the POU must contain one of these keywords.

":' needs structure variable."

The identifier to the left of ':' is not a structure variable or instance for a function block. Change the identifier into a structure variable or into a instance for a function block or remove the period and the identifier to its right.

"WHILE requires a Boolean expression as its condition"

Change the expression to an expression with a BOOL type result.

"Expecting Number, ELSE or END_CASE"

The end of a CASE statement is incorrect. Add the keyword END_CASE.

"Too many indices for array"

Check how many indices are declared for the array(1, 2, or 3) and remove the extra ones.

"Overflow of identifier list"

You must learn to restrain yourself, no more than 64000 identifiers are allowed.

"Too few indices for array"

Check how many indices are declared for the array (1, 2, or 3) and add those that are needed.

"Out of Memory"

Leave the system by saving. Close Windows, restart it and then restart the compilation.

A

ABS 10-30
Absolute Value 10-30
Access rights 4-31
Access Variables 6-2
ACOS 10-33
Action 2-9, 2-20, 2-21, 3-7, 4-32
Action Init 3-6
Active step 2-21
ADD 10-11
Address Function 10-23
Addresses 10-52
ADR 10-23
ALLIAS 10-9
Alternative Branch in SFC 2-24, 5-30
AND 10-13
Arc cosine 10-33
Arc sine 10-26 10-33
Arc tangent 10-34
Argument 2-2, 2-6
ARRAY 10-6
ASIN 10-33
Assignment 2-14, 5-18
Assignment Combs 5-20
Assignment operator 2-16
AT Declaration 5-4
ATAN 10-34
Auto Load 4-6
Auto Save 4-5
Autodeclaration 4-7, 5-6
Autoformat 4-7

B

Backup, automatic 4-5
Binding of ST operators 2-13
Bitmap 8-3, 8-13
Bitvalues 4-8
Body 5-1, 5-13, 5-14, 5-23, 5-29
Bookmark in Help 4-54
BOOL 10-5
BOOL Constants 10-49
BOOL_TO Conversions 10-24
Breakpoint 1-2, 2-28, 4-42, 5-10, 5-12, 5-36
Breakpoint Dialog Box 4-43
Breakpoint position 4-42
Build 4-11, 4-19
BYTE 10-5
BYTE Constants 10-50

C

CAL 10-24
Call tree 4-20, 4-32
Calling a function 2-2
Calling a function block 2-6, 2-14

Calling function blocks in ST 2-16
CASE 2-14
CASE instruction 2-17
CheckBounds 2-2, 10-7
CheckDivByte 2-3
CheckDivDWord 2-3
CheckDivWord 2-3
Coil 2-27, 5-25
Collapse Node 4-28
Colors 4-9, 8-7
Comment 5-7, 5-8, 5-15
Communication Parameters 4-46
Communications Parameters Gateway 4-47
Comparing projects 4-22
Compress 6-31
CONCAT 10-36
Concatenation 10-36
CONSTANT 5-3, 6-3
Constants 5-3
Contact 2-26, 5-24
Content Operator 10-7, 10-24
Context menu 4-4
Context Sensitive Help 4-55
Conversion of Integral Number Types 10-27
Conversions of types 10-24
Convert object 4-30
Copy 4-36
Copy in Help 4-53
Copying in FBD 5-21
COS 10-32
Cosine 10-32
Create Backup 4-5
Cross reference list 4-21, 4-33
CTD 10-43
CTU 10-42
CTUD 10-44
Cut 4-36
Cutting in FBD 5-21

D

Data types 2-10, 4-2, 10-5
DATE 10-6
DATE Constants 10-49
DATE_AND_TIME 10-6
DATE_AND_TIME Constants 10-50
DATE_TO Conversions 10-29
DCF file 6-3
DDE Interface 9-1
Debugger 5-10
Debugging 2-28, 4-11, 5-15
Declaration 3-4
Declaration Editor 5-1
Declaration Part 2-1, 5-1, 5-13, 5-14, 5-23, 5-29
Declarations as tables 4-7, 5-7
Declare Variable 4-40
Declare, automatic 4-7, 5-6
Defined Data Types 10-6
Delete 4-37
DELETE 10-37
Deleting a Transition 5-32
Deleting an Action 5-32

Deleting in FBD 5-21
Dereferencing 10-7, 10-24
Desktop 4-8
DINT 10-5
DINT Constants 10-50
Directory 4-10
DIV 10-12
 CheckDivByte 2-3
 CheckDivDWord 2-3
 CheckDivWord 2-3
Division durch 0 2-3
Document 4-17, 4-21
Document Frame 6-5
Download 4-13, 4-42, 4-51
Drag&Drop 4-28
DT 10-6
DT_TO Conversions 10-29
DWORD 10-5
DWORD Constants 10-50

E

Edit Menu
 Copy 4-36
 Cut 4-36, 5-22
 Delete 4-37
 Find 4-38
 Find next 4-38
 Input Assistant 4-39
 Next error 4-40
 Paste 4-37, 5-22
 Previous error 4-40
 Redo 4-35
 Replace 4-38
 Undo 4-35
Editing functions 4-35
Editor options 4-6
Editors 5-1, 5-13, 5-14, 5-23, 5-29
EN Input 2-27, 5-25
EN POU 2-27
END_FUNCTION_BLOCK 2-4
END_PROGRAM 2-8
END_TYPE 10-7, 10-8, 10-9
Entering Trace Variables 6-26
Entry action 5-31
Entry action 2-21
Enumeration 10-7
EQ 10-22
Error messages 10-59
EXIT 2-15, 2-20
Exit action 5-31
Exit action 2-21
EXP 10-31
Expand Node 4-28
Exponential Function 10-31
Exponentiation 10-34
Export 4-22
export file 6-3
Expression 2-13
EXPT 10-34
External library 4-16
Extras Menu

Align 8-15
Associate Action 5-35
Auto Read 6-28
Clear Action/Transition 5-32
Clear Background Bitmap 8-12 8-15
Compress 6-31
Configure 8-5
Cursor Mode 6-29
Edit Entry 6-25
Element list 8-16
Link Docuframe File 6-6
Load Trace 6-31
Load Watch List 6-33
Make Docuframe File 6-5
Monitoring Active 6-34
Monitoring Options 5-10
Multi Channel 6-30
Negate 5-20, 5-28
Options 5-15, 5-34
Paste above 5-27
Paste after 5-27, 5-32
Paste below 5-27
Paste Parallel Branch (right) 5-31
Previous version 4-35
Read Receipt 6-35
Read Trace 6-28
Rename Watch List 6-33
Save Trace 6-31
Save Watch List 6-33
Select All 8-15
Select Background Bitmap 8-15
Send to Back 8-15
Send to Front 8-15
Set Debug Task 6-25
Set/Reset 5-21, 5-28
Settings 8-16
SFC Overview 5-34
Show grid 6-30
Start Trace 6-28
Step Attributes 5-32
Stop Trace 6-28
Stretch 6-30
Time Overview 5-33
Trace Configuration 6-26
Trace in ASCII-file 6-31
Use IEC Steps 5-35
Write Receipt 6-35
Y Scaling 6-30
Zoom 5-21
Zoom Action/Transition 5-32

F

F_TRIG 10-42
falling edge 10-42
FBD 2-2, 2-7, 2-25, 4-43, 5-16
FBD Editor 5-16
Fields 2-1, 10-6
File 4-14
File Menu
 Close 4-15
 Exit 4-19

- New 4-14
- Open 4-14
- Print 4-16
- Printer Setup 4-17
- Save 4-15
- Save as 4-15
- Find 4-38
- FIND 10-38
- Flow Control 4-46, 5-13
- Folder 4-27, 4-28
- Font 4-7
- FOR 2-18
- FOR loop 2-14, 2-18
- Forcing 4-44, 5-8, 6-35
- Formatting, automatic 4-7
- Function 2-1, 10-53
- FUNCTION 2-1
- Function Block 2-4
- Function block call 2-6
- Function Block Diagram 2-2, 2-7, 4-43, 5-16
- Function Block in FBD 5-19
- Function block in LD 2-27
- Function block, instance 2-5
- Function call 2-2
- Function declaration 2-1
- Function in FBD 5-19
- FUNCTION_BLOCK 2-4

G

- Gateway 4-47
- Gateway Server 4-47
- GE 10-22
- Global Constants 6-3
- Global Retain Variables 6-3
- Global Variables 6-1, 6-3
- Graphic Editors 5-14
- Grid 8-17
- GT 10-21

H

- Help 4-52
- Help Menu
 - Contents and Index 4-52
- Help Topics Window 4-52

I

- Identifier 5-3, 10-51
- IEC 1131-3 2-29
- IEC Step 2-22, 5-35
- IEC steps 2-23
- lecsfc.lib 2-22
- IF instruction 2-14, 2-16
- IL 2-2, 2-4, 2-6, 2-11, 4-42, 5-13
- IL Editor 5-13
- IL operator 2-11
- Implicit at load 4-13
- Implicit variables in SFC 2-23
- Import 4-22

- Index Window 4-54
- INDEXOF 10-13
- Initialization 5-3
- Input and Output Variable 5-2
- Input Assistant 4-39
- Input in FBD 5-20
- Input Variable 5-1
- INSERT 10-37
- Insert in LD 5-27
- Insert in SFC 5-30
- Insert Menu
 - Add Entry-Action 5-31
 - Add Exit-Action 5-31
 - Additional Library 7-2
 - All Instance Paths 6-4
 - Alternative Branch (left) 5-30
 - Alternative Branch (right) 5-30
 - Append Program Call 6-24
 - Append Task 6-24
 - Assignment 5-18
 - Bitmap 8-2, 8-3
 - Coil 5-25
 - Comment 5-15
 - Contact 5-24
 - Declarations Keywords 5-4
 - Ellipse 8-2, 8-3
 - Function 5-10, 5-19
 - Function Block 5-10, 5-19, 5-25
 - Function with EN 5-26
 - Input 5-20
 - Insert at Blocks 5-25, 5-26
 - Insert Program Call 6-24
 - Insert Task 6-24
 - Jump 5-18, 5-27, 5-31
 - Line 8-3
 - Network (after) 5-15
 - Network (before) 5-15
 - New Declaration 5-7
 - New Watch List 6-33
 - Operand 5-9
 - Operator 5-9, 5-18
 - Operator with EN 5-26
 - Output 5-20
 - Parallel Branch (left) 5-30
 - Parallel Branch (right) 5-30
 - Parallel Contact 5-24
 - Placeholder 4-19
 - Polygon 8-2, 8-3
 - Rectangle 8-2
 - RETURN 5-18, 5-27
 - Rounded Rectangle 8-2
 - Step Transition (after) 5-30
 - Step Transition (before) 5-29 5-30
 - Transition-Jump 5-31
 - Types 5-4
 - Visualization 8-2, 8-3
- Insert mode 5-9
- Insert Network 5-15
- Instance 2-5
- Instance name 2-5, 2-6
- Instruction 2-11, 2-14, 2-15
- Instruction List 2-2, 2-4, 2-6, 2-11, 4-42, 5-13

INT 10-5
INT Constants 10-50
Internal library 4-16

J

Jump 2-25, 5-18
Jump in SFC 5-29, 5-31

K

Keywords 5-3, 5-4

L

Label 5-15
Ladder Diagram 2-26, 4-43, 5-23
LD 2-26, 4-43, 5-23
LD Editor 5-23
LE 10-22
LEFT 10-35
LEN 10-35
Library 2-10, 4-16
Library directory 4-10
Library Manager 3-4, 7-1
Library, Define 7-2
Library, Insert 7-2
LIMIT 10-20
Line number field 4-43, 4-46, 5-11
Line Number of the Text Editor 5-12
Line numbers 5-6
LN 10-31
Load & Save 4-5
Load Trace 6-31
Load trace configuration 6-27
Load Watch List 6-33
Local Variable 5-2
LOG 10-31
Log in 4-41
Logarithm 10-31
Logout 4-41
Loop 2-13, 2-15
LREAL 10-5
LREAL Constants 10-50
LREAL_TO Conversions 10-27
LT 10-21

M

Main Help Window 4-53
Main program 2-8
Mark 4-8
Marking in SFC 5-29
MAX 10-19
Memory location 10-52
Menu Bar 4-1
Menü Edit
 Declare Variable 4-40
Menu Extras
 Select Mode 8-16
Menü Extras

Show grid 6-30
Menu Insert
 Curve 8-3
Menu Online
 Communications Parameters Gateway 4-47
 Sourcecode download 4-51
Menu Project
 Add Action 4-32
Merge 4-23
Message window 4-3
MID 10-36
MIN 10-19
MOD 10-12
Modifier 2-11
Monitoring 2-29, 4-41, 5-7, 5-10, 6-34
MUL 10-11
Multi Channel 6-30
MUX 10-20

N

NE 10-23
Negation in FBD 5-20
Negation in LD 5-28
Network 5-15, 5-16
Network Comment 5-15
Network in FBD 2-25, 3-2
Network in LD 2-26
Network in SFC 3-6
Network number field 4-43, 4-46
New Folder 4-28
Next error 4-40
NOT 10-15
Note in Help 4-53
Notice at load 4-13
Number Constants 10-50
Number of data segments 4-12

O

Object 2-1, 4-27
Object Organizer 4-2
Online 1-1, 1-2, 5-10, 5-15, 6-34
Online Change 4-11, 4-20, 4-25
Online functions 4-41
Online in Security mode 4-9
Online Menu
 Breakpoint Dialog Box 4-43
 Communication Parameters 4-46
 Download 4-42
 Force values 4-44
 Log in 4-41
 Logout 4-41
 Release Force 4-45
 Reset 4-42
 Run 4-42
 Show Call Stack 4-45
 Simulation 4-46
 Single Cycle 4-44
 Step in 4-44
 Step over 4-44
 Stop 4-42

- Toggle Breakpoint 4-42
- Write values 4-44
- Online Mode 4-41
 - Declaration Editor 5-7
 - Function Block Diagram Editor 5-22
 - Ladder Editor 5-28
 - Network Editor 5-15
 - Sequential Function Chart Editor 5-36
 - Text Editor 5-10
 - Watch- and Receipt Manager 6-34
- Operand 2-2, 5-9, 10-49
- Operating Version 8-11
- Operator in FBD 3-4, 5-18
- Operators 5-9, 10-11
- Options 4-4
- OR 10-14
- Output in FBD 5-20
- Output Variable 5-2
- Overwrite mode 5-9

P

- Parallel Branch in SFC 2-25, 5-30
- Parallel Contacts 2-26, 5-24
- Password 4-12
- Passwords for user groups 4-26
- Pasting 4-37
- Pasting in FBD 5-21
- Placeholder 4-19
- PLC 4-41, 4-45
- PLC Configuration 6-6
- PLC_PRG 2-8
- Pointer 10-7
- POINTER 10-7
- POU (Program Organization Unit) 1-1, 2-8, 3-1, 4-2
- Presentation 8-17
- Previous error 4-40
- Previous version 4-35
- Print 4-16
- Print in Help 4-53
- Program 2-7
- PROGRAM 2-8
- Program call 2-7
- Project 1-1, 2-1, 2-7, 3-1
- Project directory 4-10
- Project info 4-5, 4-23
- Project Menu
 - Add object 4-29
 - Build 4-19
 - Check 4-19
 - Convert object 4-30
 - Copy object 4-30
 - Delete Object 4-28
 - Document 4-21
 - Export 4-22
 - Global Replace 4-25
 - Global Search 4-25
 - Import 4-22
 - Merge 4-23
 - Object access rights 4-31
 - Open object 4-30
 - Options 4-4

- Passwords for user groups 4-26
- Project info 4-23
- Rebuild all 4-20
- Register Changes 4-25
- Rename object 4-29
- Show call tree 4-32
- Show cross reference list 4-33
- Show unused variables 4-34
- View instance 4-32
- Project version 1.5 4-16

Q

- Qualifier 2-23

R

- R_TRIG 10-41
- Read Receipt 6-35
- Read trace 6-27
- Read Trace 6-28
- READ_ONLY 6-2
- READ_WRITE 6-2
- REAL 10-5
- REAL Constants 10-50
- REAL_TO Conversions 10-27
- Receipt Manager 6-32
- Redo 4-35
- References 10-9
- Register Changes 4-25
- REPEAT 2-15
- REPEAT loop 2-19
- Replace 4-25, 4-38
- REPLACE 10-38
- Reset 4-42
- Reset Output 5-21, 5-28
- Resources 2-10, 4-2, 6-1
- RETAIN 5-2, 6-3
- Retain Variable 5-2
- RETURN 2-14, 2-16, 5-18
- RIGHT 10-35
- rising edge 10-41
- ROL 10-17
- ROR 10-18
- Rotation 10-17, 10-18
- RS 10-40
- Run 4-42

S

- Sample Rate 6-27
- Sampling Trace 2-28, 6-25
- Save 4-15
- Save before compile 4-12
- Save Trace 6-31
- Screen divider 4-3
- SEL 10-19
- Selecting 4-8
- SEMA 10-40
- Sequential Function Chart 2-2, 2-7, 2-20, 3-6, 4-43, 5-29

Set Output 5-21, 5-28
Set/Reset coils 2-27
SFC 2-2, 2-7, 2-20, 3-6, 4-43, 5-29
SFC Editor 5-29
SFC Flags 2-24
SFC library 2-22
SFC Overview 5-34
SFCCurrentStep 2-24
SFCEnableLimit 2-24
SFCError 2-24
SFCErrorPOU 2-24
SFCErrorStep 2-24
SFCInit 2-24
SFCPause 2-24
SFCQuitError 2-24
SFCTrans 2-24
Shift 10-15
SHL 10-15
Shortcut Mode 5-5
Show Call Stack 4-45
Show grid 6-30
SHR 10-16
Simulation 2-29, 3-14, 4-41, 4-46
SIN 10-32
Sine 10-32
Single Cycle 4-44
Single Step 2-28, 4-44, 5-36
SINT 10-5
SINT Constants 10-50
SIZEOF 10-13
Sourcecode download 4-51
Sourcedownload 4-13
SQRT 10-30
Square Root 10-30
SR 10-39
ST 2-2, 2-6, 2-13, 4-42, 5-14
ST Editor 5-14
ST operand 2-13
ST operator 2-13
Standard Function 7-2
Standard Library 3-4, 7-2
Standard POUs 2-1
Standard.lib 7-2
Start Trace 6-28
Statistics 4-24
Status bar 4-3, 4-9, 8-5
Step 2-20, 4-43, 5-29
Step Attributes 5-32
Step Init 2-21
Stepping 5-10, 5-15, 5-36
Stop 4-42
Stop Trace 6-28
Stretch 6-30
STRING 10-6
STRING Constants 10-50
String Functions 10-35
STRING_TO Conversions 10-29
STRUCT 10-8
Structured Text 2-2, 2-6, 2-13, 4-42, 5-14
Structures 2-1, 10-8
SUB 10-11
Syntax Coloring 5-1, 5-5

System Flag 10-51

T

Tab-width 4-7
TAN 10-32
Tangent 10-32
Task Configuration 6-22
Text Editors 5-8
TIME 10-6
TIME Constants 10-49
Time Management in SFC Editor 5-33
TIME_OF_DAY 10-6
TIME_OF_DAY Constants 10-49
TIME_TO Conversions 10-28
Timer 10-45
TO_BOOL Conversions 10-26
TOD 10-6
TOD_TO Conversions 10-28
TOF 10-47
TON 10-46
Tool bar 4-2, 4-9
Tooltip 4-2, 4-27, 5-8, 5-10, 5-16, 5-22, 5-28
TP 3-4, 10-45
Trace Buffer 6-25, 6-29
Trace in ASCII-file 6-31
Trace Variable 6-28
Trace, Automatically Read 6-28
Transition 2-21, 5-29
Trigger 2-28, 6-27, 10-41
Trigger Edge 6-27
Trigger Level 6-27
Trigger Position 6-27
TRUNC 10-29
TYPE 10-7, 10-8, 10-9
Type Conversions 10-24
Types 5-4

U

UDINT 10-5
UDINT Constants 10-50
UINT 10-5
UINT Constants 10-50
Undo 4-35
Unused variables 4-21
User group 4-26
User information 4-6
USINT 10-5
USINT Constants 10-50

V

VAR 5-2, 5-6
VAR_ACCESS 6-2
VAR_CONFIG 6-2, 6-4
VAR_GLOBAL 5-6, 6-2, 6-3
VAR_IN_OUT 5-2
VAR_INOUT 5-6
VAR_INPUT 5-1, 5-6
VAR_OUTPUT 5-2, 5-6

- Variable Configuration 6-3
- Variables 10-51
- Variables declaration 5-3
- Visualization 2-11, 4-2, 8-1
 - Bitmap 8-13
 - Colors 8-7
 - File - Print 8-19
 - Input 8-10
 - Input possibilities for the operating version 8-11
 - Motion absolute 8-8
 - Motion relative 8-8
 - Operation in online mode 8-19
 - Operation over the keyboard 8-19
 - Shape 8-6
 - Text 8-6
 - Tooltip 8-13
 - Variables 8-9
- Visualization Elements, Configure 8-5
- Visualization Elements, Copy 8-4
- Visualization Elements, Insert 8-2
- Visualization Elements, Shift, Move 8-4

W

- Watch and Receipt Manager 6-32
- Watch and Receipt Manager Offline 6-32
- Watch and Receipt Manager Online 6-34
- Watch List 6-32
- Watch Variable 5-8, 5-22
- WHILE loop 2-14, 2-18
- Window 4-51
- Window Menu
 - Arrange symbols 4-52
 - Cascade 4-51
 - Close all 4-52
 - Library Manager 7-1
 - Messages 4-52
 - Tile Horizontal 4-51
 - Tile Vertical 4-51
- WORD 10-5
- WORD Constants 10-50
- Work space 4-3
- Write protection password 4-12
- Write Receipt 6-35

X

- XOR 10-14

Y

- Y Scaling 6-30

Z

- Zoom 5-21
- Zoom Action 5-32
- Zoom Transition 5-32



ABB STOTZ-KONTAKT GmbH

Eppelheimer Straße 82 Postfach 101680
D-69123 Heidelberg D-69006 Heidelberg

Telephone +49 6221 701-0
Telefax +49 6221 701-1111
E-Mail desst.help@de.abb.com
Internet <http://www.abb.de/sst>