Training document for the company-wide automation solution Totally Integrated Automation (T I A)

Appendix II

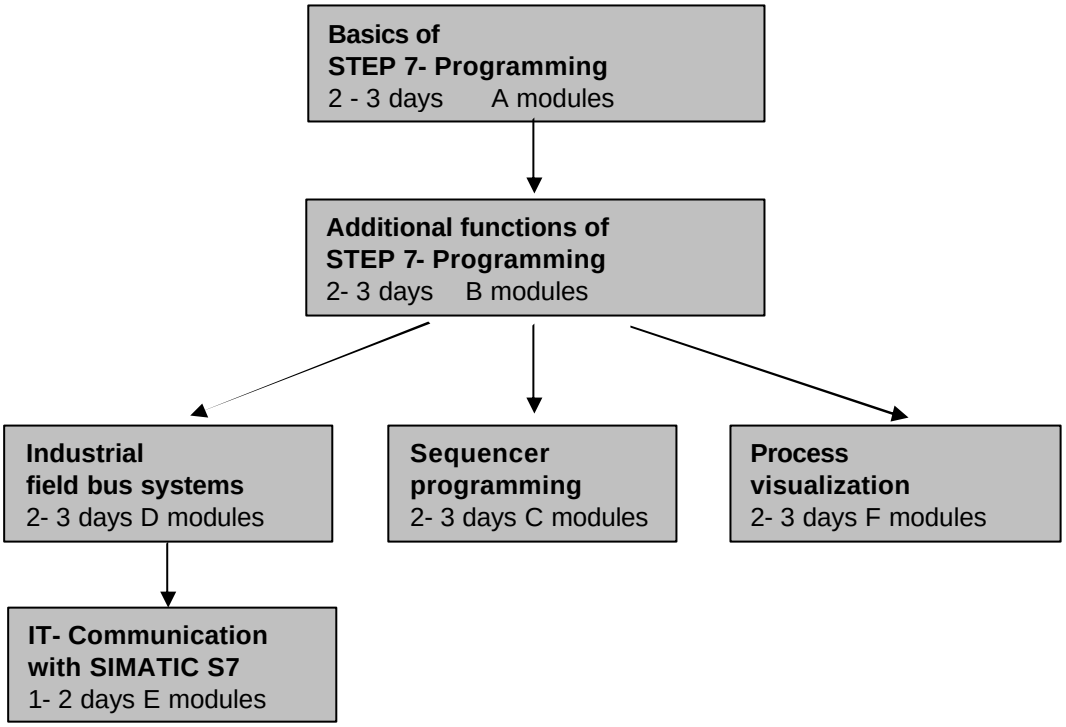
IEC 61131

We thank the company Michael Dziallas Engineering and the instructors of vocational schools as well as further persons for the support with the production of the document..

PAGE:

1. FORWARD

Appendix II represents a theoretical foundation for the processing of all modules.



Learning goal:

The reader obtains information for the international Norm IEC 61131 with this appendix.

Requirements:

Since the theoretical foundation is presented here, no special requirements are necessary.

2. NOTES FOR NORM IEC 61131

The Norm EN 661131-3 specifies the syntax and semantics of a unique array of program languages for programmable logic control (PLC).
In this module, the reader should receive an overview in conventions of the Norm. A complete report about the Norm is not contained here, bur rather only a general description of the most important elements of the languages.

- Statement list (STL)
- Function block diagram (FBD)
- Ladder diagram (LAD)
- Sequential function chart (SFC)
- Strukturierter Text (ST)

The document is based on excerpts and interpretations from the norm DIN EN 661131: 1993. An exact description of the several definitions of the program languages can be gathered from the Norm documents.

Notes for the Norm compliance

The program languages LAD and FBD comply to the specified languages “Ladder diagram“ and “Function block diagram“ in the Norm DIN EN 661131-3 (int. IEC 61131-3). You find the exact propositions in the Norm compliance table “NORM.TAB“. The data is found in the folder “Siemens“ under the directory “STEP 7“.

STL complies to the specified language “Statement list“ in the Norm DIN EN 661131-3 (int. IEC 61131-3), whereas essential differences exist in terms of operations.
You find the exact propositions in the Norm compliance table “NORM.TAB“. The data is found in the folder “Siemens“ under the directory “STEP 7“.

The sequencer program S7-GRAPH complies to the specified language “Sequential function chart“ in the Norm DIN EN 661131-3 (int. IEC 61131-3).

3. INTRODUCTION TO THE NORM DIN 661131

The document represents a manual that deals with the application of programmable logic controllers (PLC) and the associated peripheral devices.
This international Norm was accepted after the rules of the European Community in Germany as DIN EN 661131, in France as NF EN 661131 and in England as BS EN 661131.

3.1. DIN EN 661131 SECTION 1, GENERAL INFORMATION

The Norm DIN EN 661131 section 1 (IEC 61131-1) applies to programmable logic control in low voltage systems, in which the rated voltage of the network does not exceed 1000V AC (50/60 Hz) or 1500V DC and whose appointed for the controlling of machines and discontinued processes. The terms in section 1 are defined for use in the other sections of the Norm as well.

3.2. DIN EN 661131 SECTION 2, EQUIPMENT REQUIREMENTS

This section of the Norm DIN EN 661131 section 2 (IEC 61131-2) specifies

- The electrical, mechanical and functional requirements for programmable logic controllers and their associated peripheral devices such as the applicable operations, bearing and transport conditions;
- The information that must be provided from the manufacturer.
- The inspection methods and policies that are applied for the programmable logic controllers and the associated peripheral devices.

3.3. DIN EN 661131 SECTION 3, PROGRAM LANGUAGES (IEC 61131-3)

The document handles the programming languages for the programmable logic controllers like they are defined in DIN EN 661131-1. Therefore the ISO 646 character set is applicable to be represented with a printer and monitor. Graphical and semi-graphical representations of the language elements which are defined in this section are acceptable but not defined here.
No new program languages were defined, but rather the most used programming languages STL, LAD, FBD, SFC, and ST were brought into consideration.

3.4. DIN EN 661131 SECTION 4, USER GUIDELINES (BY IEC IN PROCESSING)

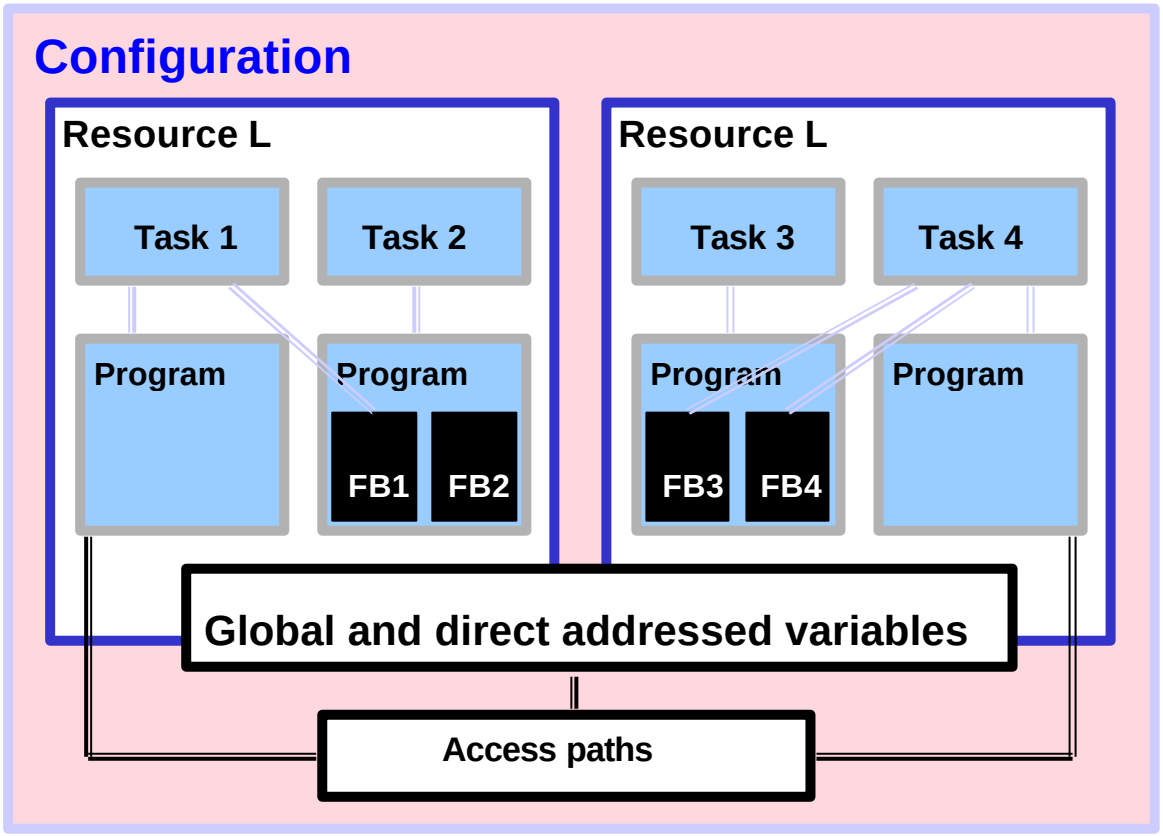
This section processes the user guidelines of PLC systems. Notes for all phases of a project are given; Starting from the system analysis over the phases of the specification and choice of devices to the application and maintenance of the devices.

3.5. DIN EN 661131 SECTION 5, COMMUNICATION (BY IEC IN PROCESSING)

This section processes the communication between PLCs of different manufacturers and the communication of arbitrary devices with the PLC.
The communication services of a PLC as a supplement Norm for ISO/IEC 9506-1/2 are specified based on the MAP-standard. The communication blocks are described for standard read and write access.

4. DIN EN 661131 SECTION 3, PROGRAM LANGUAGES

4.1. THE IEC 61131-3- SOFTWARE MODEL



Overview One of the most important points in a programmable system is the possibility to divide a large program into smaller operations. These operations must communicate with your data "modules" and should only communication with other software components over clearly defined interfaces
For this purpose, during the development of the IEC61131-3 Norm, the passed on context of a PLC program was included and an abstract software model with several hierarchy levels was developed.

Configuration The highest level is a PLC user program contained in a configuration.
A configuration is for general use in a PLC controller.
Real automation systems are generally constructed from more PLCs and/or IEC configurations. An IEC can thus communicate with another IEC configuration over clearly defined interfaces that must already be declared.

Resource There is one or more resources inside of a configuration. A resource provides the important support for the execution of PLC user program inside of a configuration.
An IEC resource in a control system is i.e. a real CPU module or a virtual PLC-CPU on a PC(SOFT-PLC).

4.2. PROGRAMS

A program is defined in IEC 61131 as "a logical arrangement of all program language elements and designs that are necessary for the intended signal processing for the controlling of a machine or a process with a PLC system".

A program which the Norm complies with

1. Should only use properties that are specified in this section for the use of current languages.
2. Should not use properties that are considered to be expansions of a language.
3. Should not be dependent on any specific interpretation of implement defined properties.

A Norm orientated program must provide the same results on each Norm orientated system.
The only exemptions are programs that are dependent

- On time responses of the program execution or application
- On implementation dependent properties in the program
- On the type of procedures for error handling

Extract from the program declaration:

1. The restricted keywords for the program declaration must be PROGRAM...END-PROGRAM
2. A program can contain a design VAR_ACCESS...END_VAR, which offers a mean for the demonstration of identified variables, which can be accessed through one of the communication services. An access path links each variable with an input, output or an internal variable of a program. The format and its use must be of Norm IEC 61131-5.
3. Programs can only be instanced inside of resources whereas function blocks can only be instanced inside of programs.

Summary

A Norm orientated program

- Must be created in a Norm orientated language without expansions
- Has a variable declaration component at it‘s disposal
- Is limited through the keywords PROGRAM...END-PROGRAM
- Is only executable inside of resources
- Must provide the same results from each Norm orientated system. Exceptions are possible and must be documented

4.3. VARIABLES

A variable serves as identification of data objects. The content of a variable is alterable. The variable is connected with data, inputs, outputs or memory of the PLC. Variables can be declared as elements or as predetermined types.

Representation of a single element-variable

A single element-variable is a single data element from one of the element data types (i.e. an input or output). The direct representation of a single element-variable takes place through the following symbols:

- A percent character
- A prefix for the location
- A prefix for the length
- One or more unsigned whole numbers

Examples:

%QX75 and %Q75	Output bit 75
%IW215	Input word-location 15

Eigenschaften der Präfixe

Nr.	Prefix	Definition
1	I	Input location
2	Q	Output location
3	M	Memory bit location
4	X	(Single-) bit-length
5	None	(Single-) bit-length
6	B	Byte-(8 bit) length
7	W	Word-(16 bit) length
8	D	Double word-(32 bit) length
9	L	Long word-(64 bit) length

Note: In the case that nothing else is declared, the data type of a direct addressed variable of "(Single-)bit"-length must be of type bool.
National Norm organization can issue tables with translations of the prefixes

Representation of the multiple element-variable

The multiple element-variable consists of arrays and structures.
An array is a collection of data elements of the same type which are addressed through one or more array indices. The indices are enclosed in square brackets and separated through commas.

Example:

OUTARY[%MB6,SYM] := INARY[0] + INARY[7] - INARY[%MB6] * %IW62;

A structured variable is a variable that is declared as a type which was already specified as a data structure. This structure consists of a collection of identified elements. An element of a structured variable must be represented through one or more identifiers or array accesses. The identifiers are separated through points whereby the first identifier represents the name of the structured element. The following symbols represent the access of a specific data element inside of the data structure.

Example:

MODULE_5_CONFIG.SIGNAL_TYP := SINGLE_ENDED;

Initialization of variables

By the initialization, a variable can take on various values:

- The value that the variable had was yielded as the configuration element (buffer value)
- A user specified initial value
- The default initial value for the assigned initial value of the variable

The user can choose to use the specified characters RETAIN, if he/she wants to buffer the variable. The initial value must be determined at the beginning according to the following rules:

- By a “Warm start“, the buffered (remenent) variables must accept the saved value
- By a “Cold start“, the variables must accept the previous value or the specific initial value of the specified data type
- Variables not buffered must be initialized with the previous initial value or with the specified initial value of the data type

Declaration

Each declaration type of a PLC program oriented unit must contain at least one declarations part at its beginning which specifies the types of the variables to be used in this organization unit. The declaration part must contain the text from of keyword VAR, VAR_INPUT or VAR_OUTPUT. The declaration is ended through VAR_END.

Example:

Program puncher Task 2

```
VAR
    B1 AT %IX0.0        :BOOL;  (*  Proximity switch B1   *)
    B2 AT %IX0.1        :BOOL;  (*  Proximity switch B2   *)
    B3 AT %IX0.2        :BOOL;  (*  Proximity switch B3   *)
    B4 AT %IX0.3        :BOOL;  (*  Proximity switch B4   *)
    Y1 AT %QX4.0        :BOOL;  (*  Cylinder 1.0 extension  *)
END_VAR
```

4.4. STATEMENT LIST (STL)

The statement list is composed of a sequence of statements (instructions). The following conventions are complied:

- Each statement must begin on a new line
- Each statement contains an operation with additional modifications (i.e. I 0.0 for the input bit 0 in the input byte 0)
- More operands are separated by commas (i.e. cal FB1, DB2)
- An identification tag of the statement proceeds, followed by a double point
- A defined comment is the last element of a line
- Empty lines can exist

Operators, modifiers and operands

The value of the expression which is calculated, is replaced by the actual generated value.
Result := Result OP Operand

Example: Result := Result AND %IX1

The modifier „N“ shows the Boolean negation of the operand.
The statement ANDN%IX2 means:

Result := Result AND NOT %IX2

The modifier left parenthesis “(“ indicates that the evaluation of the operator is postponed until the right parenthesis “)” is seen.
The statement sequence

AND(%IX1
OR %IX2
)

means Result := Result AND (%IX1 OR %IX2)

The modifier “C“ indicates that the associated statement is only allowed to be executed when the value of the evaluated result is a Boolean 1. If the modifier “N” is used with the operator, the result must be a Boolean 0.

Operands of the statement list

Nr.	Operator	Modifier	Operand	Definition
1	LD	N	Note 1	Sets the actual result of the operand
2	ST	N	Note 1	Stores the actual result in the operand address
3	S	Note 2	**BOOL**	Set Boolean operator to 1
	R	Note 2	BOOL	Reset Boolean operator to 0
4	AND	N, (	BOOL	Boolean AND
5	&	N, (	BOOL	Boolean AND
6	OR	N, (	BOOL	Boolean OR
7	XOR	N, (	BOOL	Boolean Exclusive-OR
8	ADD	(	Note 1	Addition
9	SUB	(	Note 1	Subtraction
10	MUL	(	Note. 1	Multiplication
11	DIV	(	Note 1	Division
12	GT	(	Note 1	Comparison: >
13	GE	(	Note 1	Comparison: >=
14	EQ	(	Note 1	Comparison: =
15	NE	(	Note 1	Comparison: <>
16	LE	(	Note 1	Comparison: <=
17	LT	(	Note 1	Comparison: <
18	JMP	C,N	**MARK**	Jump to the Mark
19	CAL	C,N	NAME	Call function block (Note 3)
20	RET	C,N		Return to a function or a function block
21	)			Processing reset operations

Note 1: The operations must be either loaded or given with a type. The actual result and the operand must have the same type.

Note 2: The operations are only executed when the value of the actual result is a Boolean 1.

Note 3: A list of arguments in parenthesis follow the name of the function block

Properties of the function block call

Nr.	Description/Example
1	CAL with a list of the input parameters: CAL C10(CU:=%IX10, PV:=15)
2	CAL with Load/Save of the input parameters: LD 15 ST C10.PV LD %IX10 ST C10.CU CAL C10
3	Use of the input operators: LD 15 PV C10 LD %IX10 CU C10

Note: A declaration in the form of VAR C10: CTU; END_VAR is assumed in the examples.

Representation of the statement list in STEP 7

```
FC2 : Example block
Comment:

Network 1: AND logic operation in STL
Comment:

      A     "B1"                    I0.0            -- Proximity switch B1
      A     "B2"                    I0.1            -- Proximity switch B2
      =     "Y1"                    Q4.0            -- Cylinder extension

Network 2: OR logic operation in STL
Comment:

      O     "B3"                    I0.2            -- Proximity switch B3
      O     "B4"                    I0.3            -- Proximity switch B4
      =     "Y1"                    Q4.0            -- Cylinder extension
```

4.5. LADDER DIAGRAM (LAD)

Definition of the network

- A LAD-Network is left and right defined though a vertical power rail
- A connection element is recognized through a horizontal line
- The state of the connection element is identified as “ON“ and “OFF“ according to the current flow and corresponds to the Boolean values 0 and 1
- The state of the left connection line is always “ON“
- The state of the right connection line is undefined
- The horizontal connection element transmits the state of the immediate left side to the immediate right side of the element
- The vertical connection element crosses itself with one or more horizontal elements. The state of the vertical connection must be “OFF“ when all horizontal connections on the left side of the element are “OFF“. The state must be “ON“ when one or more of the horizontal connections on the left side of the element are “ON“
- The state of the vertical connection must be copied to all horizontal connections to the right side of the element. A copy of the state is not allowed on the left side

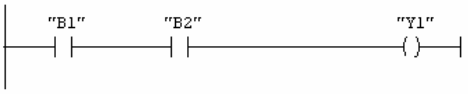
Representation of the ladder diagram in STEP 7

FC2 : Example block

Comment:

Network 1: AND logic operation in LAD

Comment:



Symbol information:

I0.0	B1	Proximity switch B1
I0.1	B2	Proximity switch B2
Q4.0	Y1	Cylinder extension

Network 2: OR logic operation in LAD

Comment:

Symbol information:

I0.2	B3	Proximity switch B3
I0.3	B4	Proximity switch B4
Q4.0	Y1	Cylinder extension

4.6. FUNCTION BLOCK DIAGRAM (FBD)

Definition of the network

- Elements of the language FBD must be defined through signal flow lines
- The outputs of the function blocks are not allowed to be connected with one another
- The evaluation of a network must be completed before the evaluation of the next network is allowed to begin. This occurs when the next network uses the outputs of the previous network

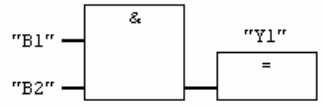
Representation of the FBD in STEP 7

FC2 : Example block

Comment:

Network 1: AND logic operation in FBD

Comment:

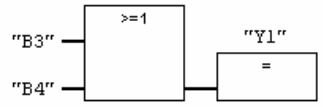


Symbol information:

I0.0	B1	Proximity switch B1
I0.1	B2	Proximity switch B2
Q4.0	Y1	Cylinder extension

Network 2 : OR logic operation in FBD

Comment:



Symbol information:

I0.2	B3	Proximity switch B3
I0.3	B4	Proximity switch B4
Q4.0	Y1	Cylinder extension

4.7. SEQUENTIAL FUNCTION CHART (SFC)

The elements of the sequence function chart offer aid for the structuring of a PLC program organized unit when dealing with transitions and steps which are connected with one another. A great deal of actions belong to each step and each transition contains a transition condition. The created program is saved in a function block with an instance data block.

Steps

A step is graphically represented through a block. The block contains a step name in the form of an identifier or in text form (i.e. Step 1 or S1). The arranged connections are displayed through vertical lines which begin at the step above. The connections can also be displayed in the form of transitions.
A number of various actions (i.e. Set or Reset from an in or output) belong to a step.
A step flag shows the state of the step, active or inactive, which can be displayed through the Boolean value 1, when the step is active, and 0, when the step is inactive.
The time "Time" begins by the activation of a step with the value "0" and stops by deactivation. The time value remains until the next activation stands on the reached value. Each sequential function chart has a program organization unit. The initial values of your internal and output variables are stored here. In addition, a definition of the active initial steps is that each SFC-network must have exactly one initial step.
The system initialization takes place over the default initial time for steps. The activation of the next step must take place inside of given time otherwise the system changes into the state "Error" or the activation takes place over the initial state for common steps during the fulfilling of the additional switch condition.

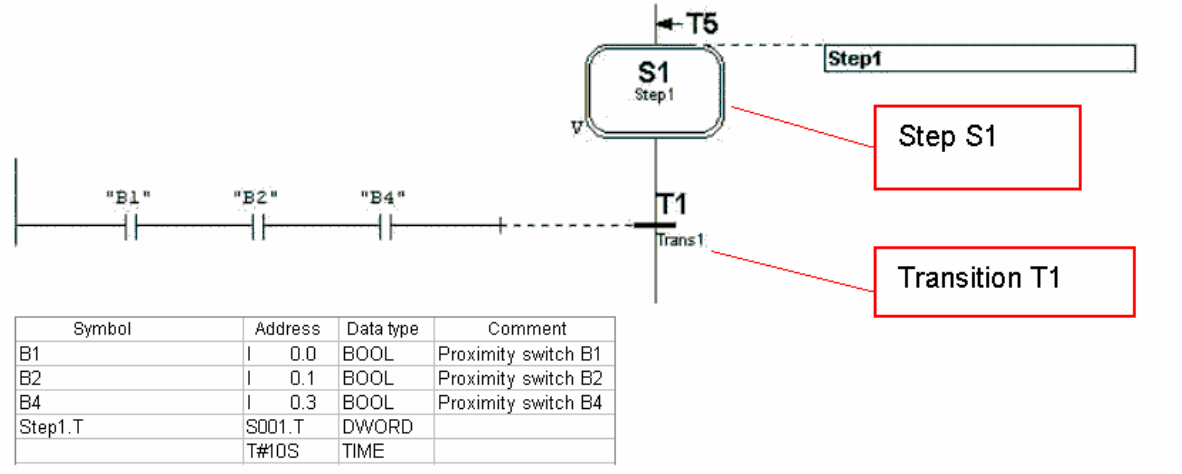
Transitions

The transition is displayed through a horizontal line which goes though the vertical connection between two steps. A transition is an additional switch condition defined in the conditions that must be fulfilled before the next step can be switched to active. If no conditions are defined in a transition, the transition is deemed to be fulfilled. The transition condition can be defined in the languages LAD, ST, STL or FBD.

Actions

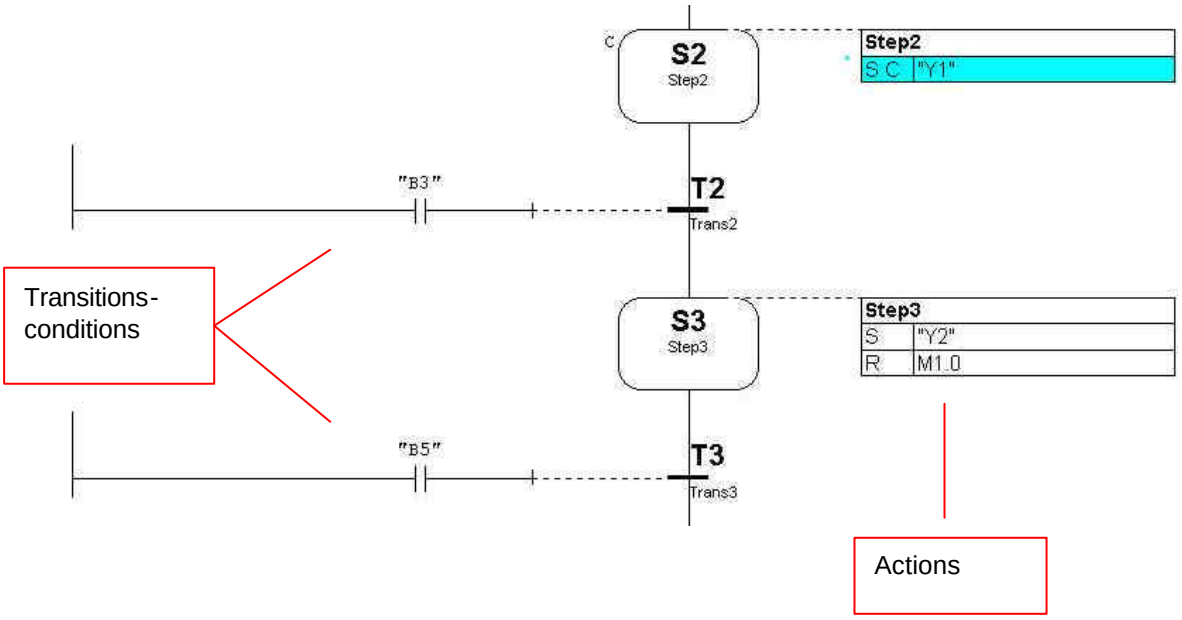
A step consists of 0 or more actions. An action can be a Boolean variable or also a sequence of statements in one of the norm oriented program languages. The function of the action must be recognizable through a control indicator.

Representation of a SFC in S7-GRAPH



Symbol	Address	Data type	Comment
B1	I 0.0	BOOL	Proximity switch B1
B2	I 0.1	BOOL	Proximity switch B2
B4	I 0.3	BOOL	Proximity switch B4
Step1.T	S001.T	DWORD	
	T#10S	TIME	

Portion of a sequence function chart



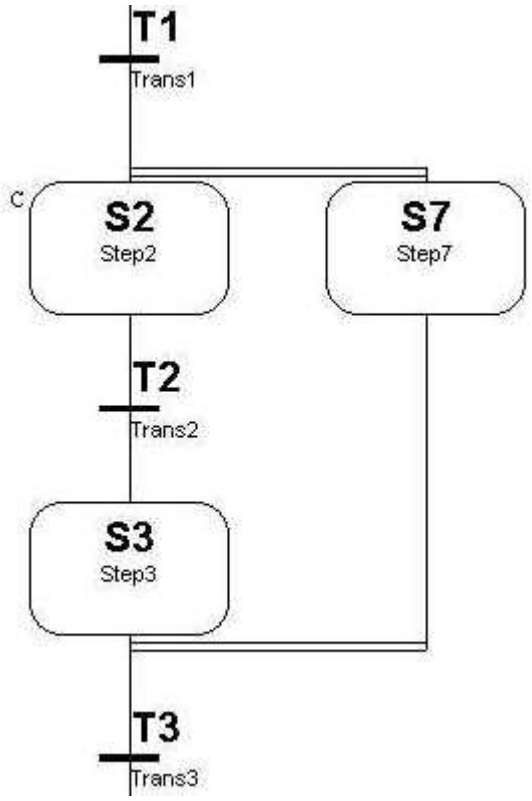
Representation of the specified characters for actions

Nr.	Specified characters	Explanation
1	None	Not saved (no character)
2	N	Not saved
3	R	Reset dominant
4	S	Set (saved)
5	L	Time limit
6	D	Time delay
7	P	Impulse (Flank)
8	SD	Saved and time delay
9	DS	Delayed and saved
10	SL	Saved and time limit

Sequence rules

- The initial state of a SFC network is characterized through an initial step which is in the active state by program initialization
- A sequence is released when the preceding step and the required transition condition are fulfilled
- The activation of a transition deactivates all previous steps
- Two steps are not allowed to be directly connected. They are divided by a transition
- Two transitions are not allowed to be directly connected. Thy are divided by a step
- For an activation of more steps, there exists a simultaneous branch, where the junction of the simultaneous sequence must be recognized through a double horizontal line.

Representation of a simultaneous branch



4.8. STRUCTURED TEXT (ST)

4.8.1. EXPRESSIONS

Expressions consist of operators and operands. The operators of the language ST are displayed in table 5.

Nr.	Operation	Symbol	Priority
1	Parenthesis	(Expression)	Highest
2	Function evaluation Examples:	Identifier (Argument-List) LN(A), MAX(X,Y) etc.	
3	Potentiation	**	
4 5	Negation Complement	- NOT	
6 7 8	Multiplication Division Modulus	* / MOD	
9 10	Addition Subtraction	+ -	
11	Comparison	<,>,<=,>=	
12 13	Equality Inequality	= <>	
14 15	Boolean AND Boolean AND	& AND	
16	Boolean Exclusive OR	XOR	
17	Boolean OR	OR	Lowest

The evaluation of an expression is from the application of the operations to the operands, in the sequence that is defined through the priority of operators. The operator with the highest priority in an expression must first be applied, followed by the operator of the next lowest priority and so forth until the expression is completed. Operators with equal priorities must be applied how they are written from left to right in the expression.

For example, when A, B, C and D of type INT (Integer), corresponding to the values 1, 2, 3 and 4, then

A+B-C*ABS(D) must be calculated as –9 ((1+2)-(3*4)= -9)

and

(A+B-C)*ABS(D) results in 0 . **((1+2-3)*4=0)**

When an operator has two operands, the left operand must first be evaluated. For example:
SIN(A)*COS(B) the expression SIN(A) must first be calculated followed by COS(B), followed by the evaluation of the product.
Boolean expressions need to be evaluated only to the extent, which is necessary, in order to determine the resulting result.
When, for example, A<=B, then it is enough to only evaluate the expression (A>B), in order to decide, that the value of the expression (A>B)&(C<D) has the Boolean value 0.
Functions must be called as elements of expressions that are from a function name, which follows a parenthetical list of arguments.

4.8.2. STATEMENTS

The statements of the language ST are outlined in table 6. Statements must be ended through semi-colons.

Nr.	Statement type/reference	Examples
1	Assignment	A:=B; CV:=CV+1; C:=SIN(X);
2	Function block call and use of the FB output	CMD_TMR(IN:=%IX5, PT:=T#300ms); A:=CMD_TMR.Q;
3	RETURN	RETURN;
4	IF	D:=B*B-4*A*C; IF D< 0.0 THEN NROOTS:=0; ELSIF D=0.0 THEN NROOTS:= 1; X1:=-B/(2.0*A); ELSE NROOTS:= 2; X1:=(-B+SQRT(D))/(2.0*A); X1:=(-B-SQRT(D))/(2.0*A); END_IF;
5	CASE	TW:=BCD_TO_INT(THUMBWHEEL); TW_ERROR:=0; CASE TW OF 1,5:DISPLAY:=OVEN_TEMP; 2: DISPLAY:=MOTOR_SPEED; 3: DISPLAY:=GROSS_TARE; 4,6..10:DISPLAY:=STATUS(TW-4); ELSE DISPLAY:=0; TW_ERROR:=1; END_CASE; QW100:=INT_TO_BCD(DISPLAY);

6	FOR	J:=101;
		FOR I:=1TO 100BY 2 DO
		IF WORDS[I]='KEY' THEN
		J:=I;
		EXIT;
		END_IF
		END_FOR;

7	WHILE	J:=1; WHILE J<= 100 & WORDS[J] <> ,KEY' DO J:= J+2; END_WHILE;
8	REPEAT	J:= -1; REPEAT J:= J+2; UNTIL J= 101 OR WORDS[J]= ,KEY' END_REPEAT;
9	EXIT	EXIT;
10	Empty statement	,

4.8.2. ASSIGNMENTS

The assignment sets the actual value of a single or multi element variable through the result of the evaluation of an expression. An assignment must be made of a variable declaration on the left side, followed by an assignment operation ":=" and then an expression to be evaluated. For example, the statement A:=B is used in order to set the single data value of variable A through the actual value of variable B, in the case that both variables have the type INT.

The assignment must also be used in order to assign a value that is returned from a function. This executes through the setting of the function name on the left side of the assignment operation in the body of the function declaration. The value that is returned from the function must be the result of the last expression of the function. It is an error when the "ENO" output from the expression of the function is returned not equal to 0 because no assignment was made.

4.8.3. CONTROL STATEMENTS FOR FUNCTIONS AND FUNCTION BLOCKS

Control statements for functions and function blocks are made of mechanisms for the calling of function blocks and for the controlling of the return to the called operation before the physical end of a function or a function block is reached.
Function blocks must be called through a statement that consists of the name of the function block. This function block follows a parenthetical list of value assignments to specific input parameters similar to those outlined in table 6. It is not necessary that by all calls of a function, all input parameters are assigned values. In the case of a specific parameter where no value is assigned to a function block call, the previously assigned value must be applied (or the initial value, in case no previous assignment was made). The statement RETURN must cause an early exit of the function or function block.

4.8.4. SELECTION STATEMENTS

Selection statements contain the IF and CASE statements. A selection statement chooses a) or a group) as its assignments on the basis of a specified condition from which the assignments are composed of. Examples of selection statements are given in table 6.
The IF statement specifies that a group of assignments is only executed when the associated Boolean expression provides the value 1 (TRUE). If the condition is false, either no assignment was allowed to be executed or the following statement group with KEYWORD ELSE was executed (or the keyword ELSIF, if its assigned Boolean condition is true).

The CASE statement consists of an expression which must be evaluated as a variable of type INT and of a list of statement groups, where each group is accepted with a tag that consists of one or more whole numbers or a range of integer values. This specifies that the first group of statements must be executed, followed by the keyword ELSE. Otherwise no statement sequence is allowed to be executed.

4.8.5. REPEAT STATEMENTS

Repeat statements specify that a group of associated statements must be executed repeatedly. The FOR statement is used so the number of repetitions can be specified beforehand, otherwise the construction WHILE and REPEAT are used.

The EXIT statement must be used in order to end a repeat before the end condition is fulfilled.
When the EXIT statement lies inside of a nest loop repeat design, the innermost loop where the EXIT lies, must be exited. The control must skip to the next statement after the first loop end, followed by the EXIT statement.

The FOR statement shows that a control sequence must be repeated until the keyword END_FOR is executed. Therefore, a progressive value of the control variable of the FOR loop is assigned. The control variable, the initial value and the end value must be expressions of the same whole number data types and are not allowed to be altered through one of the repeat statements. The FOR statement increments the control variable from the initial value to or up to the end value in increments that are specified through the value of an expression. This value is initialized to 1.

The checking of the end condition is executed at the beginning of each repeat so that the statement sequence is not executed if the initial value exceeds the end value. The value of the control variable after the end of the FOR loop is dependent on the implementation.

The WHILE statement causes the sequence of statements to be repeatedly executed until the keyword END_WHILE or until the associated Boolean expression is false. If the expression is false from the beginning, the group of statements is not executed at all.

The REPEAT statement cause the sequence of statements to be repeatedly executed (a minimum of one time) until the keyword UNTIL or until the associated Boolean condition is true.

The statements WHILE and REPEAT are not allowed to be used to undertake synchronization between processes (i.e. a “while loop“ with an externally specified end condition). For this purpose the SFC elements must be used.
It is a mistake to use a WHILE or REPEAT statement in an algorithm for which the fulfilling of a loop end condition or the execution of an EXIT statement cannot be guaranteed.

4.9. DOCUMENTATION

The documentation of a project should consist of:

- A description of the hardware configuration with project specific designations
- User program documentation consist of:
- Expression of the user program with possible designations for the handled signals and data
- Cross-reference list for all processing data (In-/Outputs, internal functions like internally saved data, timers, counters, etc.)
- Comments
- Description of modifications
- Maintenance guide