IEC 61131-3 3rd Edition "PLC Programming Languages"

Implementer: CODESYS Development GmbH
Tobias-Dannheimer-Str. 5 | 87439 Kempten
Product: CODESYS V3.5 SP 17
Date: 2021-07-14

This Product complies with the requirements of the standard for the following language features:

Feature No.	Table Number and Title / Feature Description		Compliantly implemented (✓)				Implementer's note
			LD	FBD	ST	IL	
Table 1 – Character set							
1	"ISO-646 IRV" characters Table 1 - Row 00 of ISO/IEC 10646-1		✓	✓	✓	✓	
2	"Latin-1 Supplement" Table 2 - Row 00 of ISO/IEC 10646-1		✓	✓	✓	✓	
3a	Lower case characters a:	a, b, c, …	✓	✓	✓	✓	
3b	Number sign:	#	✓	✓	✓	✓	
3c	Dollar sign:	$	✓	✓	✓	✓	
Table 2 - Identifiers							
1	Upper case letters and numbers:	`IW215`	✓	✓	✓	✓	
2	Upper and lower case letters, numbers, embedded underscores		✓	✓	✓	✓	
3	Upper and lower case, numbers, leading or embedded underscores		✓	✓	✓	✓	
Table 3 - Comments							
1	Single-line comment	//…	✓	✓	✓	✓	
2a	Multi-line comment	(* … *)	✓	✓	✓	✓	
2b	Multi-line comment	/* … */					
3a	Nested comment	(* ..(* .. *) ..*)	✓	✓	✓	✓	
3b	Nested comment	/* .. /* .. */ .. */					
Table 4 - Pragma							
1	Pragma with curly brackets	{ … }	✓	✓	✓	✓	In LD FBD only for declaration and in label field
Table 5 – Numeric literals							
1	Integer literal:	`-12`	✓	✓	✓	✓	
2	Real literal:	`-12.0`	✓	✓	✓	✓	
3	Real literals with exponent:	`-1.34E-12`	✓	✓	✓	✓	
4	Binary literal:	`2#1111_1111`	✓	✓	✓	✓	
5	Octal literal:	`8#377`					
6	Hexadecimal literal:	`16#FF`	✓	✓	✓	✓	
7	Boolean zero and one		✓	✓	✓	✓	

8	Boolean `FALSE` and `TRUE`	✓	✓	✓	✓	
9	Typed literal: `INT#-123`	✓	✓	✓	✓	
Table 6 – Character string literals						
	Single-byte characters or character strings with ‘ ‘					
1a	Empty string (length zero)	✓	✓	✓	✓	
1b	String of length one or character `CHAR` containing a single character	✓	✓	✓	✓	CHAR is not supported
1c	String of length one or character `CHAR` containing the “space” character	✓	✓	✓	✓	CHAR is not supported
1d	String of length one or character `CHAR` containing the “single quote” character	✓	✓	✓	✓	CHAR is not supported
1e	String of length one or character `CHAR` containing the “double quote” character	✓	✓	✓	✓	CHAR is not supported
1f	Support of two character combinations of **Fehler! Verweisquelle konnte nicht gefunden werden.**	✓	✓	✓	✓	
1g	Support of a character representation with ‘$’ and two hexadecimal characters	✓	✓	✓	✓	
	Double-byte characters or character strings with `""` (NO					
2a	Empty string (length zero)	✓	✓	✓	✓	
2b	String of length one or character `WCHAR` containing a single character	✓	✓	✓	✓	WCHAR is not supported
2c	String of length one or character `WCHAR` containing the “space” character	✓	✓	✓	✓	WCHAR is not supported
2d	String of length one or character `WCHAR` containing the “single quote” character	✓	✓	✓	✓	WCHAR is not supported
2e	String of length one or character `WCHAR` containing the “double quote” character	✓	✓	✓	✓	WCHAR is not supported
2f	Support of two character combinations of **Fehler! Verweisquelle konnte nicht gefunden werden.**	✓	✓	✓	✓	
2g	Support of the character representation with ‘$’ and four hexadecimal characters	✓	✓	✓	✓	
	Single-byte typed characters or string literals with #					
3a	Typed string					
3b	Typed character					
3c	Typed character (using hexadecimal representation)					
	Double-byte typed string literals with # (NOTE)					
4a	Typed double-byte string (using “double quote” character)					
4b	Typed double-byte character (using “double quote” character)					
4c	Typed double-byte string (using “single quote” character)					
4d	Typed double-byte character (using “single quote” character)					

Table 7 – Two-character combinations in character strings						
1	Dollar sign	✓	✓	✓	✓	
2	Single quote	✓	✓	✓	✓	
3	Line feed	✓	✓	✓	✓	
4	Newline	✓	✓	✓	✓	
5	Form feed (page)	✓	✓	✓	✓	
6	Carriage return	✓	✓	✓	✓	
7	Tabulator	✓	✓	✓	✓	
8	Double quote	✓	✓	✓	✓	
Table 8 – Duration literals						
	Duration abbreviations	✓	✓	✓	✓	
1a	`d`	✓	✓	✓	✓	
1b	`h`	✓	✓	✓	✓	
1c	`m`	✓	✓	✓	✓	
1d	`s`	✓	✓	✓	✓	
1e	`ms`	✓	✓	✓	✓	
1f	`us`	✓	✓	✓	✓	
1g	`ns`	✓	✓	✓	✓	
	Duration literals without underscore					
2a	short prefix	✓	✓	✓	✓	
2b	long prefix	✓	✓	✓	✓	
	Duration literals with underscore					
3a	short prefix	✓	✓	✓	✓	
3b	long prefix	✓	✓	✓	✓	
Table 9 – Date and Time of Day literals						
1a	Date literal (long prefix)	✓	✓	✓	✓	
1b	Date literal (short prefix)	✓	✓	✓	✓	
2a	Long date literal (long prefix)	✓	✓	✓	✓	
2b	Long date literal (short prefix)	✓	✓	✓	✓	
3a	Time of day literal (long prefix)	✓	✓	✓	✓	
3b	Time of day literal (short prefix)	✓	✓	✓	✓	
4a	Long time of day literal (short prefix)	✓	✓	✓	✓	
4b	Long time of day literal (long prefix)	✓	✓	✓	✓	
5a	Date and time literal (long prefix)	✓	✓	✓	✓	
5b	Date and time literal (short prefix)	✓	✓	✓	✓	
6a	Long date and time literal (long prefix)	✓	✓	✓	✓	
6b	Long date and time literal (short prefix)	✓	✓	✓	✓	

Table 10 – Elementary data types						
1	Boolean	✓	✓	✓	✓	
2	Short integer	✓	✓	✓	✓	
3	Integer	✓	✓	✓	✓	
4	Double integer	✓	✓	✓	✓	
5	Long integer	✓	✓	✓	✓	
6	Unsigned short integer	✓	✓	✓	✓	
7	Unsigned integer	✓	✓	✓	✓	
8	Unsigned double integer	✓	✓	✓	✓	
9	Unsigned long integer	✓	✓	✓	✓	
10	Real numbers	✓	✓	✓	✓	
11	Long reals	✓	✓	✓	✓	
12a	Duration	✓	✓	✓	✓	
12b	Duration	✓	✓	✓	✓	
13a	Date (only)	✓	✓	✓	✓	
13b	Long Date (only)	✓	✓	✓	✓	
14a	Time of day (only)	✓	✓	✓	✓	
14b	Time of day (only)	✓	✓	✓	✓	
15a	Date and time of Day	✓	✓	✓	✓	
15b	Date and time of Day	✓	✓	✓	✓	
16a	Variable-length single-byte character string	✓	✓	✓	✓	
16b	Variable-length double-byte character string	✓	✓	✓	✓	
17a	Single-byte character					
17b	Double-byte character					
18	Bit string of length 8	✓	✓	✓	✓	
19	Bit string of length 16	✓	✓	✓	✓	
20	Bit string of length 32	✓	✓	✓	✓	
21	Bit string of length 64	✓	✓	✓	✓	
Table 11 – Declaration of user-defined data types and initialization						
1a 1b	Enumerated data type	✓	✓	✓	✓	
2a 2b	Data type with named values	✓	✓	✓	✓	
3a 3b	Subrange data type	✓	✓	✓	✓	
4a 4b	Array data type	✓	✓	✓	✓	
5a 5b	Function block type and class as array element	✓	✓	✓	✓	Only FB-Types since Classes are not available
6a 6b	Structured data type	✓	✓	✓	✓	
7a 7b	Function block type and class as structure elements	✓	✓	✓	✓	Only FB-Types since Classes are not available
8a	Structured data type with relative addressing `AT`					With attribute {attribute

8b						'local_offset'}
9a	Structured data type with relative addressing `AT` and `OVERLAP`					UNION instead
10a 10b	Directly represented elements of a structure - partly specified using " * "	✓	✓	✓	✓	
11a 11b	Directly derived data type					
12	Initialization using constant expression	✓	✓	✓	✓	

Table 12 – Reference operations

	Declaration					
1	Declaration of a reference type	✓	✓	✓	✓	We have two concepts, REFERENCE TO and POINTER TO in combination those two support all features of this reference, with a little different syntax.
	Assignment and comparison					
2a	Assignment reference to reference <reference> := <reference>	✓	✓	✓	✓	
2b	Assignment reference to parameter of function, function block and method	✓	✓	✓	✓	For POINTER TO no type check
2c	Comparison with NULL	✓	✓	✓	✓	IsValid(Reference) for POINTER TO yes
	Referencing					
3a	`REF(<variable>)` Provides of the typed reference to the variable	✓	✓	✓	✓	REF= / ADR()
3b	`REF(<function block instance>)` Provides the typed reference to the function block or class instance	✓	✓	✓	✓	REF= / ADR()
	Dereferencing					
4	<reference> ^ Provides the content of the variable or content of the instance to which the reference variable contains the reference	✓	✓	✓	✓	For POINTERS, implicitly for references

Table 13 – Declaration of variables

1	Variable with elementary data type	✓	✓	✓	✓	
2	Variable with user-defined data type	✓	✓	✓	✓	
3	Array	✓	✓	✓	✓	
4	Reference	✓	✓	✓	✓	REFERENCE TO / POINTER TO

Table 14 – Initialization of variables

1	Initialization of a variable with elementary data type	✓	✓	✓	✓	
2	Initialization of a variable with user-defined data type	✓	✓	✓	✓	
3	Array	✓	✓	✓	✓	
4	Declaration and initialization of constants	✓	✓	✓	✓	
5	Initialization using constant expressions	✓	✓	✓	✓	
6	Initialization of a reference	✓	✓	✓	✓	REF= / := ADR(var)

Table 15 – Variable-length ARRAY variables						
1	Declaration `using *` `ARRAY [*, *, . . . ] OF data type`	✓	✓	✓	✓	
	Standard functions LOWER_BOUND / UPPER_BOUND					
2a	Graphical representation	✓	✓	✓	✓	
2b	Textual representation	✓	✓	✓	✓	
Table 16 – Directly represented variables						
	Location (NOTE 1)					
1	Input location `I`	✓	✓	✓	✓	
2	Output location `Q`	✓	✓	✓	✓	
3	Memory location `M`	✓	✓	✓	✓	
	Size and data type					
4a	Single bit size X	✓	✓	✓	✓	
4b	Single bit size None					
5	Byte (8 bits) size `B`	✓	✓	✓	✓	
6	Word (16 bits) size `W`	✓	✓	✓	✓	
7	Double word (32 bits) size `D`	✓	✓	✓	✓	
8	Long (quad) word (64 bits) size `L`	✓	✓	✓	✓	
	Addressing					
9	Simple addressing `%IX1`	✓	✓	✓	✓	Not for bits
10	Hierarchical addressing using “**.**” (NOTE 3) `%QX7.5`	✓	✓	✓	✓	Bits are addressed by two elements. The evaluation of addresses can be done by Automation Platform Plugins.
11	partly specified direct representation using asterisk ”*****”	✓	✓	✓	✓	
Table 17 – Partial access to ANY_BIT variables						
	Data Type - Access to					Simple bit access on any kind of ANY_BIT variable with Var.<Bit_Nr>
1a	`BYTE` – bit `VB2.%X0`					
1b	`WORD` – bit `VW3.%X15`					
1c	`DWORD` – bit					
1d	`LWORD` – bit					
2a	`WORD` – byte `VW4.%B0`					
2b	`DWORD` – byte					
2c	`LWORD` – byte					
3a	`DWORD` – word					
3b	`LWORD` – word					
4	`LWORD` – dword `VL5.%D1`					

Table 18 – Execution control graphically using EN and ENO						
1	Usage **without** EN and ENO	✓	✓			Feature of the graphic languages, the EN/ENO is always evaluated outside the call. ENO is not used for error determination but is only a copied EN-value.
2	Usage of EN **only** (without ENO)	✓	✓			
3	Usage of ENO **only** (without EN)					
4	Usage of EN **and** ENO	✓	✓			
Table 19 – Function declaration						
1a	Without result FUNCTION ... END_FUNCTION	✓	✓	✓	✓	
1b	With result FUNCTION <name> : <data type> END _FUNCTION	✓	✓	✓	✓	
2a	Inputs VAR_INPUT...END_VAR	✓	✓	✓	✓	
2b	Outputs VAR_OUTPUT...END_VAR	✓	✓	✓	✓	
2c	In-outs VAR_IN_OUT...END_VAR	✓	✓	✓	✓	
2d	Temporary variables VAR_TEMP...END_VAR					
2e	Temporary variables VAR...END_VAR	✓	✓	✓	✓	All functions variables are temporary
2f	External variables VAR_EXTERNAL...END_VAR	✓	✓	✓	✓	Not necessary, but possible
2g	External constants VAR_EXTERNAL CONSTANT...END_VAR	✓	✓	✓	✓	Not necessary, but possible
3a	Initialization of inputs	✓	✓	✓	✓	Initialized inputs are used as default value for the call
3b	Initialization of outputs	✓	✓	✓	✓	
3c	Initialization of temporary variables	✓	✓	✓	✓	
Table 20 – Function call						
1a	Complete formal call (textual only) NOTE Shall be used if EN/ENO is necessary in the call.	✓	✓	✓	✓	No EN/ENO in textual languages
1b	Incomplete formal call (textual only) NOTE May be used if EN/ENO is not necessary in the call.					
2	Non-formal call (textual only) (fix order and complete) NOTE Shall be used for call of standard functions without formal names.	✓	✓	✓	✓	
3	Function without function result	✓	✓	✓	✓	

4	Graphical representation	✓	✓	✓	✓	
5	Graphical usage of negated boolean input and output in graphical representation	✓	✓	✓	✓	
6	Graphical usage of `VAR_IN_OUT`					No assignment Out for VAR_IN_OUT
Table 21 – Typed and overloaded functions						
1a	Overloaded function `ADD` (`ANY`_Num to `ANY`_Num)	✓	✓	✓	✓	All built-in Operators are overloaded
1b	Conversion of inputs `ANY_ELEMENT TO_INT`	✓	✓	✓	✓	
2a	Typed function `ADD_INT`					
2b	Conversion `WORD_TO_INT`	✓	✓	✓	✓	
Table 22 – Data type conversion function						
1a	Typed conversion input_TO_output	✓	✓	✓	✓	
1b	Overloaded conversion TO_output	✓	✓	✓	✓	
2a	“Old” overloaded truncation TRUNC	✓	✓	✓	✓	TRUNC accepts REAL and LREAL and produces DINT
2b	Typed truncation input_TRUNC_output					
2c	Overloaded truncation TRUNC_output	✓	✓	✓	✓	TRUNC_INT for compatibility reasons
3a	Typed input_BCD_TO_output					
3b	Overloaded BCD_TO_output					
4a	Typed input_TO_BCD_output					
4b	Overloaded `TO_BCD`_output					
Table 23 – Data type conversion of numeric data types						
1	LREAL TO REAL	✓	✓	✓	✓	
2	LREAL TO LINT	✓	✓	✓	✓	
3	LREAL TO DINT	✓	✓	✓	✓	
4	LREAL TO INT	✓	✓	✓	✓	
5	LREAL TO SINT	✓	✓	✓	✓	
6	LREAL TO ULINT	✓	✓	✓	✓	
7	LREAL TO UDINT	✓	✓	✓	✓	
8	LREAL TO UINT	✓	✓	✓	✓	
9	LREAL TO USINT	✓	✓	✓	✓	
10	REAL TO LREAL	✓	✓	✓	✓	
11	REAL TO LINT	✓	✓	✓	✓	
12	REAL TO DINT	✓	✓	✓	✓	
13	REAL TO INT	✓	✓	✓	✓	

14	REAL TO SINT	✓	✓	✓	✓	
15	REAL TO ULINT	✓	✓	✓	✓	
16	REAL TO UDINT	✓	✓	✓	✓	
17	REAL TO UINT	✓	✓	✓	✓	
18	REAL TO USINT	✓	✓	✓	✓	
19	LINT TO LREAL	✓	✓	✓	✓	
20	LINT TO REAL	✓	✓	✓	✓	
21	LINT TO DINT	✓	✓	✓	✓	
22	LINT TO INT	✓	✓	✓	✓	
23	LINT TO SINT	✓	✓	✓	✓	
24	LINT TO ULINT	✓	✓	✓	✓	
25	LINT TO UDINT	✓	✓	✓	✓	
26	LINT TO UINT	✓	✓	✓	✓	
27	LINT TO USINT	✓	✓	✓	✓	
28	DINT TO LREAL	✓	✓	✓	✓	
29	DINT TO REAL	✓	✓	✓	✓	
30	DINT TO LINT	✓	✓	✓	✓	
31	DINT TO INT	✓	✓	✓	✓	
32	DINT TO SINT	✓	✓	✓	✓	
33	DINT TO ULINT	✓	✓	✓	✓	
34	DINT TO UDINT	✓	✓	✓	✓	
35	DINT TO UINT	✓	✓	✓	✓	
36	DINT TO USINT	✓	✓	✓	✓	
37	INT TO LREAL	✓	✓	✓	✓	
38	INT TO REAL	✓	✓	✓	✓	
39	INT TO LINT	✓	✓	✓	✓	
40	INT TO DINT	✓	✓	✓	✓	
41	INT TO SINT	✓	✓	✓	✓	
42	INT TO ULINT	✓	✓	✓	✓	
43	INT TO UDINT	✓	✓	✓	✓	
44	INT TO UINT	✓	✓	✓	✓	
45	INT TO USINT	✓	✓	✓	✓	
46	SINT TO LREAL	✓	✓	✓	✓	
47	SINT TO REAL	✓	✓	✓	✓	
48	SINT TO LINT	✓	✓	✓	✓	
49	SINT TO DINT	✓	✓	✓	✓	
50	SINT TO INT	✓	✓	✓	✓	
51	SINT TO ULINT	✓	✓	✓	✓	
52	SINT TO UDINT	✓	✓	✓	✓	
53	SINT TO UINT	✓	✓	✓	✓	
54	SINT TO USINT	✓	✓	✓	✓	
55	ULINT TO LREAL	✓	✓	✓	✓	
56	ULINT TO REAL	✓	✓	✓	✓	
57	ULINT TO LINT	✓	✓	✓	✓	
58	ULINT TO DINT	✓	✓	✓	✓	

59	ULINT TO INT	✓	✓	✓	✓	
60	ULINT TO SINT	✓	✓	✓	✓	
61	ULINT TO UDINT	✓	✓	✓	✓	
62	ULINT TO UINT	✓	✓	✓	✓	
63	ULINT TO USINT	✓	✓	✓	✓	
64	UDINT TO LREAL	✓	✓	✓	✓	
65	UDINT TO REAL	✓	✓	✓	✓	
66	UDINT TO LINT	✓	✓	✓	✓	
67	UDINT TO DINT	✓	✓	✓	✓	
68	UDINT TO INT	✓	✓	✓	✓	
69	UDINT TO SINT	✓	✓	✓	✓	
70	UDINT TO ULINT	✓	✓	✓	✓	
71	UDINT TO UINT	✓	✓	✓	✓	
72	UDINT TO USINT	✓	✓	✓	✓	
73	UINT TO LREAL	✓	✓	✓	✓	
74	UINT TO REAL	✓	✓	✓	✓	
75	UINT TO LINT	✓	✓	✓	✓	
76	UINT TO DINT	✓	✓	✓	✓	
77	UINT TO INT	✓	✓	✓	✓	
78	UINT TO SINT	✓	✓	✓	✓	
79	UINT TO ULINT	✓	✓	✓	✓	
80	UINT TO UDINT	✓	✓	✓	✓	
81	UINT TO USINT	✓	✓	✓	✓	
82	USINT TO LREAL	✓	✓	✓	✓	
83	USINT TO REAL	✓	✓	✓	✓	
84	USINT TO LINT	✓	✓	✓	✓	
85	USINT TO DINT	✓	✓	✓	✓	
86	USINT TO INT	✓	✓	✓	✓	
87	USINT TO SINT	✓	✓	✓	✓	
88	USINT TO ULINT	✓	✓	✓	✓	
89	USINT TO UDINT	✓	✓	✓	✓	
90	USINT TO UINT	✓	✓	✓	✓	

Table 24 – Data type conversion of bit data types

1	LWORD TO DWORD	✓	✓	✓	✓	
2	LWORD TO WORD	✓	✓	✓	✓	
3	LWORD TO BYTE	✓	✓	✓	✓	
4	LWORD TO BOOL	✓	✓	✓	✓	
5	DWORD TO LWORD	✓	✓	✓	✓	
6	DWORD TO WORD	✓	✓	✓	✓	
7	DWORD TO BYTE	✓	✓	✓	✓	
8	DWORD TO BOOL	✓	✓	✓	✓	
9	WORD TO LWORD	✓	✓	✓	✓	
10	WORD TO DWORD	✓	✓	✓	✓	
11	WORD TO BYTE	✓	✓	✓	✓	

12	WORD TO BOOL	✓	✓	✓	✓	
13	BYTE TO LWORD	✓	✓	✓	✓	
14	BYTE TO DWORD	✓	✓	✓	✓	
15	BYTE TO WORD	✓	✓	✓	✓	
16	BYTE TO BOOL	✓	✓	✓	✓	
17	BYTE TO CHAR	✓	✓	✓	✓	
18	BOOL TO LWORD	✓	✓	✓	✓	
19	BOOL TO DWORD	✓	✓	✓	✓	
20	BOOL TO WORD	✓	✓	✓	✓	
21	BOOL TO BYTE	✓	✓	✓	✓	
22	CHAR TO BYTE					
23	CHAR TO WORD					
24	CHAR TO DWORD					
25	CHAR TO LWORD					
26	WCHAR TO WORD					
27	WCHAR TO DWORD					
28	WCHAR TO LWORD					

Table 25 – Data type conversion of bit types to numeric types

1	LWORD TO LREAL	✓	✓	✓	✓	
2	DWORD TO REAL	✓	✓	✓	✓	
3	LWORD TO LINT	✓	✓	✓	✓	
4	LWORD TO DINT	✓	✓	✓	✓	
5	LWORD TO INT	✓	✓	✓	✓	
6	LWORD TO SINT	✓	✓	✓	✓	
7	LWORD TO ULINT	✓	✓	✓	✓	
8	LWORD TO UDINT	✓	✓	✓	✓	
9	LWORD TO UINT	✓	✓	✓	✓	
10	LWORD TO USINT	✓	✓	✓	✓	
11	DWORD TO LINT	✓	✓	✓	✓	
12	DWORD TO DINT	✓	✓	✓	✓	
13	DWORD TO INT	✓	✓	✓	✓	
14	DWORD TO SINT	✓	✓	✓	✓	
15	DWORD TO ULINT	✓	✓	✓	✓	
16	DWORD TO UDINT	✓	✓	✓	✓	
17	DWORD TO UINT	✓	✓	✓	✓	
18	DWORD TO USINT	✓	✓	✓	✓	
19	WORD TO LINT	✓	✓	✓	✓	
20	WORD TO DINT	✓	✓	✓	✓	
21	WORD TO INT	✓	✓	✓	✓	
22	WORD TO SINT	✓	✓	✓	✓	
23	WORD TO ULINT	✓	✓	✓	✓	
24	WORD TO UDINT	✓	✓	✓	✓	
25	WORD TO UINT	✓	✓	✓	✓	
26	WORD TO USINT	✓	✓	✓	✓	

27	BYTE TO LINT	✓	✓	✓	✓	
28	BYTE TO DINT	✓	✓	✓	✓	
29	BYTE TO INT	✓	✓	✓	✓	
30	BYTE TO SINT	✓	✓	✓	✓	
31	BYTE TO ULINT	✓	✓	✓	✓	
32	BYTE TO UDINT	✓	✓	✓	✓	
33	BYTE TO UINT	✓	✓	✓	✓	
34	BYTE TO USINT	✓	✓	✓	✓	
35	BOOL TO LINT	✓	✓	✓	✓	
36	BOOL TO DINT	✓	✓	✓	✓	
37	BOOL TO INT	✓	✓	✓	✓	
38	BOOL TO SINT	✓	✓	✓	✓	
39	BOOL TO ULINT	✓	✓	✓	✓	
40	BOOL TO UDINT	✓	✓	✓	✓	
41	BOOL TO UINT	✓	✓	✓	✓	
42	BOOL TO USINT	✓	✓	✓	✓	
43	LREAL TO LWORD	✓	✓	✓	✓	
44	REAL TO DWORD	✓	✓	✓	✓	
45	LINT TO LWORD	✓	✓	✓	✓	
46	LINT TO DWORD	✓	✓	✓	✓	
47	LINT TO WORD	✓	✓	✓	✓	
48	LINT TO BYTE	✓	✓	✓	✓	
49	DINT TO LWORD	✓	✓	✓	✓	
50	DINT TO DWORD	✓	✓	✓	✓	
51	DINT TO WORD	✓	✓	✓	✓	
52	DINT TO BYTE	✓	✓	✓	✓	
53	INT TO LWORD	✓	✓	✓	✓	
54	INT TO DWORD	✓	✓	✓	✓	
55	INT TO WORD	✓	✓	✓	✓	
56	INT TO BYTE	✓	✓	✓	✓	
57	SINT TO LWORD	✓	✓	✓	✓	
58	SINT TO DWORD	✓	✓	✓	✓	
59	SINT TO WORD	✓	✓	✓	✓	
60	SINT TO BYTE	✓	✓	✓	✓	
61	ULINT TO LWORD	✓	✓	✓	✓	
62	ULINT TO DWORD	✓	✓	✓	✓	
63	ULINT TO WORD	✓	✓	✓	✓	
64	ULINT TO BYTE	✓	✓	✓	✓	
65	UDINT TO LWORD	✓	✓	✓	✓	
66	UDINT TO DWORD	✓	✓	✓	✓	
67	UDINT TO WORD	✓	✓	✓	✓	
68	UDINT TO BYTE	✓	✓	✓	✓	
69	UINT TO LWORD	✓	✓	✓	✓	
70	UINT TO DWORD	✓	✓	✓	✓	
71	UINT TO WORD	✓	✓	✓	✓	

72	UINT TO BYTE	✓	✓	✓	✓	
73	USINT TO LWORD	✓	✓	✓	✓	
74	USINT TO DWORD	✓	✓	✓	✓	
75	USINT TO WORD	✓	✓	✓	✓	
76	USINT TO BYTE	✓	✓	✓	✓	
Table 26 – Data type conversion of date and time types						
1	LTIME TO TIME	✓	✓	✓	✓	
2	TIME TO LTIME	✓	✓	✓	✓	
3	LDT TO DT	✓	✓	✓	✓	
4	LDT TO DATE	✓	✓	✓	✓	
5	LDT TO LTOD	✓	✓	✓	✓	
6	LDT TO TOD	✓	✓	✓	✓	
7	DT TO LDT	✓	✓	✓	✓	
8	DT TO DATE	✓	✓	✓	✓	
9	DT TO LTOD	✓	✓	✓	✓	
10	DT TO TOD	✓	✓	✓	✓	
11	LTOD TO TOD	✓	✓	✓	✓	
12	TOD TO LTOD	✓	✓	✓	✓	
Table 27 – Data type conversion of character types						
1	WSTRING TO STRING	✓	✓	✓	✓	
2	WSTRING TO WCHAR					
3	STRING TO WSTRING	✓	✓	✓	✓	
4	STRING TO CHAR					
5	WCHAR TO WSTRING					
6	WCHAR TO CHAR					
7	CHAR TO STRING					
8	CHAR TO WCHAR					

Table 28 – Numerical and arithmetic Functions

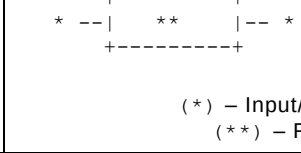
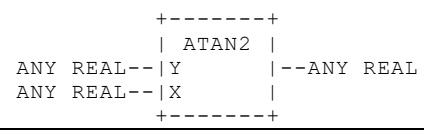
	Graphical form					
		✓	✓	✓	✓	
	General functions					
1	ABS(x)	✓	✓	✓	✓	
2	SQRT(x)	✓	✓	✓	✓	
	Logarithmic functions					
3	LN(x)	✓	✓	✓	✓	
4	LOG(x)	✓	✓	✓	✓	
5	EXP(x)	✓	✓	✓	✓	
	Trigonometric functions					
6	SIN(x)	✓	✓	✓	✓	
7	COS(x)	✓	✓	✓	✓	
8	TAN(x)	✓	✓	✓	✓	
9	ASIN(x)	✓	✓	✓	✓	
10	ACOS(x)	✓	✓	✓	✓	
11	ATAN(x)	✓	✓	✓	✓	
12	ATAN2(y, x) 					

Table 29 – Arithmetic functions

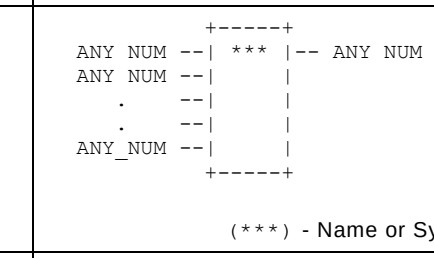
	Graphical form					
						
	Extensible arithmetic functions					
1	Addition	✓	✓	✓	✓	
2	Multiplication	✓	✓	✓	✓	
	Non-extensible arithmetic functions					
3	Subtraction	✓	✓	✓	✓	
4	Division	✓	✓	✓	✓	
5	Modulo	✓	✓	✓	✓	
6	Exponentiation	✓	✓	✓	✓	
7	Move	✓	✓	✓	✓	

Table 30 – Bit shift functions

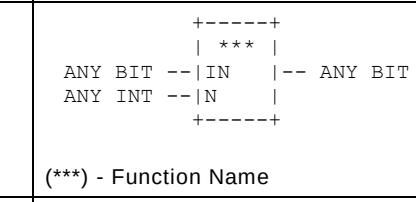
	Graphical form					
						
1	Shift left	✓	✓	✓	✓	
2	Shift right	✓	✓	✓	✓	
3	Rotation left	✓	✓	✓	✓	
4	Rotation right	✓	✓	✓	✓	

Table 31 – Bitwise Boolean functions

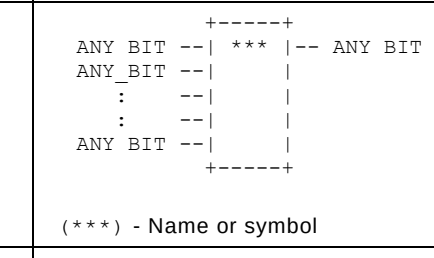
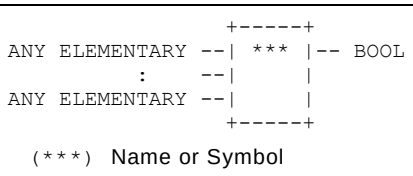
	Graphical form					
						
1	And	✓	✓	✓	✓	
2	Or	✓	✓	✓	✓	
3	Exclusive Or	✓	✓	✓	✓	
4	Not	✓	✓	✓	✓	

Table 32 – Selection functions						
1	Move a,d (assignment)	✓	✓	✓	✓	
2	Binary selection	✓	✓	✓	✓	
3	Extensible maximum function	✓	✓	✓	✓	
4	Extensible minimum function	✓	✓	✓	✓	
5	Limiter	✓	✓	✓	✓	
6	Extensible multiplexer	✓	✓	✓	✓	
Table 33 – Comparison functions						
	Graphical form					
						We only support comparison with two operands
1	Decreasing sequence	✓	✓	✓	✓	Only symbolic not extensible
2	Monotonic sequence:	✓	✓	✓	✓	Only symbolic not extensible
3	Equality	✓	✓	✓	✓	Only symbolic not extensible
4	Monotonic sequence	✓	✓	✓	✓	Only symbolic not extensible
5	Increasing sequence	✓	✓	✓	✓	Only symbolic not extensible
6	Inequality	✓	✓	✓	✓	Only symbolic not extensible
Table 34 – Character string functions						
1	String length	✓	✓	✓	✓	Standard.LEN Standard64.WLEN
2	Left	✓	✓	✓	✓	Standard.Left Standard64.WLeft
3	Right	✓	✓	✓	✓	Standard.Right Standard64.WRight
4	Middle	✓	✓	✓	✓	Standard.Middle Standard64.WMiddle
5	Extensible concatenation	✓	✓	✓	✓	Standard.Concat Standard64.WConcat Not extensible
6	Insert	✓	✓	✓	✓	Standard.Insert Standard64.WInsert
7	Delete	✓	✓	✓	✓	Standard.Delete Standard64.WDelete
8	Replace	✓	✓	✓	✓	Standard.Replace Standard64.WReplace
9	Find	✓	✓	✓	✓	Standard.Find Standard64.WFind
Table 35 – Numerical functions of time and duration data types						
1a	ADD	✓	✓	✓	✓	Overloaded for TIME and LTIME
1b	ADD_TIME					

1c	ADD_LTIME					
2a	ADD	✓	✓	✓	✓	Overloaded for TIME and LTIME
2b	ADD_TOD_TIME					
2c	ADD_LTOD_LTIME					
3a	ADD	✓	✓	✓	✓	Overloaded for TIME and LTIME
3b	ADD_DT_TIME					
3c	ADD_LDT_LTIME					
4a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
4b	SUB_TIME					
4c	SUB_LTIME					
5a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
5b	SUB_DATE_DATE					
5c	SUB_LDATE_LDATE					
6a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
6b	SUB_TOD_TIME					
6c	SUB_LTOD_LTIME					
7a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
7b	SUB_TOD_TOD					
7c	SUB_LTOD_LTOD					
8a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
8b	SUB_DT_TIME					
8c	SUB_LDT_LTIME					
9a	SUB	✓	✓	✓	✓	Overloaded for TIME and LTIME
9b	SUB_DT_DT					
9b	SUB_LDT_LDT					
10a	MUL	✓	✓	✓	✓	Overloaded for TIME and LTIME
10b	MUL_TIME					
10c	MUL_LTIME					
11a	DIV	✓	✓	✓	✓	Overloaded for TIME and LTIME
11b	DIV_TIME					
11c	DIV_LTIME					

Table 36 – Additional functions of time data types CONCAT and SPLIT

	Concatenate time data types					
1a	CONCAT_DATE_TOD					
1b	CONCAT_LDATE_LTOD					
2	CONCAT_DATE					
3a	CONCAT_TOD					
3b	CONCAT_LTOD					
4a	CONCAT_DT					
4b	CONCAT_LDT					
	Split time data types					

5	SPLIT_DATE					
6a	SPLIT_TOD					
6b	SPLIT_LTOD					
7a	SPLIT_DT					
7b	SPLIT_LDT					
	Get day of the week					
8	DAY_OF_WEEK					
Table 37 – Functions for Endianess Conversion						
1	TO_BIG_ENDIAN					
2	TO_LITTLE_ENDIAN					
3	BIG_ENDIAN_TO					
4	LITTLE_ENDIAN_TO					
Table 38 – Functions of enumerated data types						
1	SEL	✓	✓	✓	✓	
2	MUX	✓	✓	✓	✓	
3	EQ	✓	✓	✓	✓	
4	NE	✓	✓	✓	✓	
Table 39 – Validate functions						
1	IS_VALID					
2	IS_VALID_BCD					
Table 40 – Function block type declaration						
1	Declaration of function block type FUNCTION_BLOCK ... END_FUNCTION_BLOCK	✓	✓	✓	✓	
2a	Declaration of inputs VAR_INPUT ... END_VAR	✓	✓	✓	✓	
2b	Declaration of outputs VAR_OUTPUT ... END_VAR	✓	✓	✓	✓	
2c	Declaration of in-outs VAR_IN_OUT ... END_VAR	✓	✓	✓	✓	
2d	Declaration of temporary variables VAR_TEMP ... END_VAR	✓	✓	✓	✓	
2e	Declaration of **static** variables VAR ... END_VAR	✓	✓	✓	✓	
2f	Declaration of external variables VAR_EXTERNAL ... END_VAR	✓	✓	✓	✓	
2g	Declaration of external constants VAR_EXTERNAL CONSTANT ... END_VAR	✓	✓	✓	✓	
3a	Initialization of inputs	✓	✓	✓	✓	
3b	Initialization of outputs	✓	✓	✓	✓	
3c	Initialization of static variables	✓	✓	✓	✓	
3d	Initialization of temporary variables	✓	✓	✓	✓	
4a	Declaration of RETAIN qualifier on input variables	✓	✓	✓	✓	
4b	Declaration of RETAIN qualifier on output variables	✓	✓	✓	✓	
4c	Declaration of NON_RETAIN qualifier on input variables					

4d	Declaration of NON_RETAIN qualifier on output variables					
4e	Declaration of RETAIN qualifier on static variables	✓	✓	✓	✓	
4f	Declaration of NON_RETAIN qualifier on static variables					
5a	Declaration of RETAIN qualifier on local FB instances	✓	✓	✓	✓	
5b	Declaration of NON_RETAIN qualifier on local FB instances					
6a	Textual declaration of - rising edge inputs (`R_EDGE`)					
6b	Textual declaration of - falling edge inputs (`F_EDGE`)					
7a	Graphical declaration of - rising edge inputs (>)	✓	✓			This is implemented as a property of the call, not of the called function block
7b	Graphical declaration of - falling edge inputs (<)	✓	✓			This is implemented as a property of the call, not of the called function block
Table 41 – Function block instance declaration						
1	Declaration of function block instance(s)	✓	✓	✓	✓	
2	Declaration of function block instance with initialization of its variables	✓	✓	✓	✓	
Table 42 – Function block call						
1	Complete formal call (textual only) NOTE - Shall be used if EN/ENO is necessary in calls.	✓	✓	✓	✓	
2	Incomplete formal call (textual only)	✓	✓	✓	✓	
3	Graphical call	✓	✓	✓	✓	
4	Graphical call with negated boolean input and output	✓	✓	✓	✓	
5a	Graphical call with usage of `VAR_IN_OUT`	✓	✓	✓	✓	
5b	Graphical call with assignment of `VAR IN OUT` to a variable					
6a	Textual call with separate assignment of input `FB_Instance.Input := x;`	✓	✓	✓	✓	
6b	Graphical call with separate assignment of input	✓	✓	✓	✓	
7	Textual output read after function block call `x:= FB_Instance.Output;`	✓	✓	✓	✓	
8a	Textual output assignment in function block call	✓	✓	✓	✓	
8b	Textual output assignment in function block call with negation	✓	✓	✓	✓	
9a	Textuall call with function block instance name as input	✓	✓	✓	✓	
9b	Graphical call with function block instance name as input	✓	✓	✓	✓	
10a	Textual call with function block instance name as `VAR_IN_OUT`	✓	✓	✓	✓	
10b	Graphical call with function block instance name as `VAR_IN_OUT`	✓	✓	✓	✓	
11a	Textual call with function block instance name as external variable	✓	✓	✓	✓	
11b	Graphical call with function block instance name as external variable	✓	✓	✓	✓	

Table 43 – Standard bistable function blocks

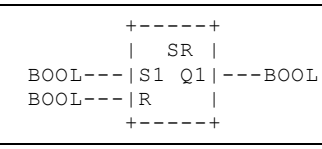
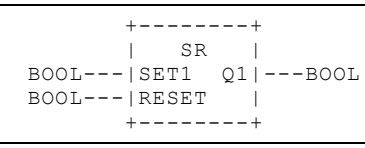
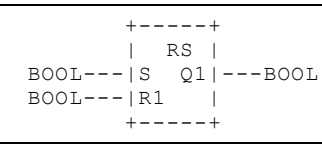
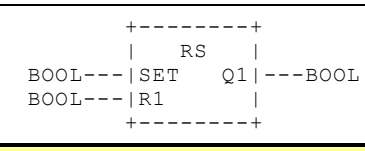
1a	Bistable function block (set dominant): `SR(S1,R,Q1)`					
						
1b	Bistable function block (set dominant) with long input names: `SR(SET1, RESET, Q1)`	✓	✓	✓	✓	
		✓	✓	✓	✓	
2a	Bistable function block (reset dominant): `RS(S, R1, Q1)`					
						
2b	Bistable function block (reset dominant) with long input names: `RS(SET,RESET1, Q1)`	✓	✓	✓	✓	
		✓	✓	✓	✓	

Table 44 – Standard edge detection function blocks

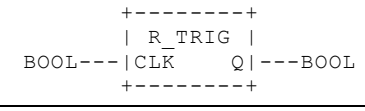
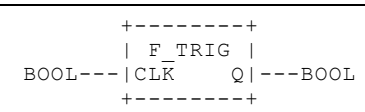
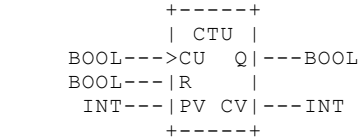
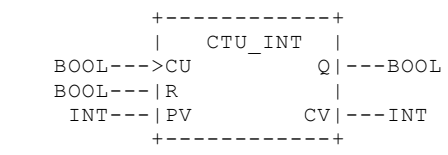
1	Rising edge detector: `R_TRIG(CLK, Q)`	✓	✓	✓	✓	
		✓	✓	✓	✓	
2	Falling edge detector: `F_TRIG(CLK, Q)`	✓	✓	✓	✓	
		✓	✓	✓	✓	

Table 45 – Standard counter function blocks

	Up-Counter					
1a	`CTU_INT(CU, R, PV, Q, CV)` or `CTU(..)`	✓	✓	✓	✓	Uses WORD
	 and also: 	✓	✓	✓	✓	Not-Typed version
1b	`CTU_DINT PV, CV: DINT`					
1c	`CTU_LINT PV, CV: LINT`					
1d	`CTU_UDINT PV, CV: UDINT`					

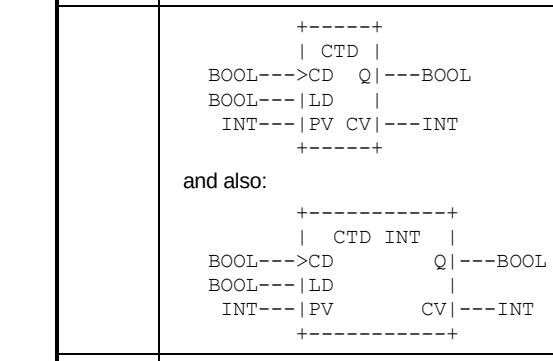
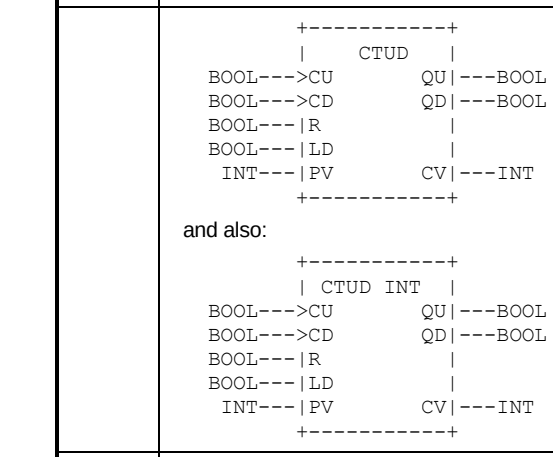
1e	`CTU_ULINT(CD, LD, PV, CV) PV, CV: ULINT`					
	Down-counters					
2a	`CTD_INT(CD, LD, PV, Q, CV)` or `CTD`	✓	✓	✓	✓	Uses WORD
		✓	✓	✓	✓	Not-Typed version
2b	`CTD_DINT PV, CV: DINT`					
2c	`CTD_LINT PV, CV: LINT`					
2d	`CTD_UDINT PV, CV: UDINT`					
2e	`CTD_ULINT PV, CV: UDINT`					
	Up-down counters					
3a	`CTUD_INT(CD, LD, PV, Q, CV)` or `CTUD(..)`	✓	✓	✓	✓	Uses WORD
		✓	✓	✓	✓	Not-Typed version
3b	`CTUD_DINT PV, CV: DINT`					
3c	`CTUD_LINT PV, CV: LINT`					
3d	`CTUD_UDINT PV, CV: UDINT`					
3e	`CTUD_ULINT PV, CV: ULINT`					

Table 46 – Standard timer function blocks						
1a	Pulse, overloaded **TP**					
1b	Pulse using TIME	✓	✓	✓	✓	
1c	Pulse using LTIME	✓	✓	✓	✓	Standard64.LTP
2a	On-delay, overloaded **TON**					
2b	On-delay using TIME	✓	✓	✓	✓	
2c	On-delay using LTIME	✓	✓	✓	✓	Standard64.LTON
2d	On-delay, overloaded (Graphical)					
3a	Off-delay, overloaded **TOF**					
3b	Off-delay using TIME	✓	✓	✓	✓	
3c	Off-delay using LTIME	✓	✓	✓	✓	Standard64.LTOF
3d	Off-delay, overloaded (Graphical)					
Table 47 – Program declaration						
1	Declaration of a program PROGRAM ... END_PROGRAM	✓	✓	✓	✓	
2a	Declaration of inputs VAR_INPUT ... END_VAR	✓	✓	✓	✓	
2b	Declaration of outputs VAR_OUTPUT ... END_VAR	✓	✓	✓	✓	
2c	Declaration of in-outs VAR_IN_OUT ... END_VAR	✓	✓	✓	✓	
2d	Declaration of temporary variables VAR_TEMP ... END_VAR	✓	✓	✓	✓	
2e	Declaration of static variables VAR ... END_VAR	✓	✓	✓	✓	
2f	Declaration of external variables VAR_EXTERNAL ... END_VAR	✓	✓	✓	✓	
2g	Declaration of external constants VAR_EXTERNAL CONSTANT ... END_VAR	✓	✓	✓	✓	
3a	Initialization of inputs	✓	✓	✓	✓	
3b	Initialization of outputs	✓	✓	✓	✓	
3c	Initialization of static variables	✓	✓	✓	✓	
3d	Initialization of temporary variables	✓	✓	✓	✓	
4a	Declaration of RETAIN qualifier on input variables	✓	✓	✓	✓	
4b	Declaration of RETAIN qualifier on output variables	✓	✓	✓	✓	
4c	Declaration of NON_RETAIN qualifier on input variables					
4d	Declaration of NON_RETAIN qualifier on output variables					
4e	Declaration of RETAIN qualifier on static variables	✓	✓	✓	✓	
4f	Declaration of NON_RETAIN qualifier on static variables					
5a	Declaration of RETAIN qualifier on local Function block instances	✓	✓	✓	✓	

5b	Declaration of NON RETAIN qualifier on local FB instances					
6a	Textual declaration of - rising edge inputs					
6b	- falling edge inputs (textual)					
7a	Graphical declaration of - rising edge inputs (>)	✓	✓			This is implemented as a property of the call, not of the called function block
7b	- falling edge inputs (<)	✓	✓			This is implemented as a property of the call, not of the called function block
8a	VAR GLOBAL...END VAR declaration within a PROGRAM					In own object
8b	VAR_GLOBAL CONSTANT declarations within PROGRAM type declarations					In own object
9	VAR_ACCESS...END_VAR declaration within a PROGRAM					

Table 48 - Class

1	CLASS ... END_CLASS					
1a	FINAL specifier					
	Adapted from function block					
2a	Declaration of variables VAR ... END_VAR					
2b	Initialization of variables					
3a	RETAIN qualifier on internal variables					
3b	NON_RETAIN qualifier on internal variables					
4a	VAR_EXTERNAL declarations within class declarations					
4b	VAR_EXTERNAL CONSTANT declarations within class declarations					
	Methods and specifiers					
5	METHOD...END_METHOD					
5a	PUBLIC specifier					
5b	PRIVATE specifier					
5c	INTERNAL specifier					
5d	PROTECTED specifier					
5e	FINAL specifier					
	Inheritance					
6	EXTENDS					
7	OVERRIDE					
8	ABSTRACT					
	Access reference					
9a	THIS					
9b	SUPER					
	Variable access specifiers					
10a	PUBLIC specifier					
10b	PRIVATE specifier					

10c	INTERNAL specifier					
10d	PROTECTED specifier					
	Polymorphism					
11a	with VAR_IN_OUT					
11b	with reference					

Table 49 – Class instance declaration

1	Declaration of class instance(s) with default initialization					
2	Declaration of class instance with initialization of its public variables					

Table 50 – Textual call of methods – Formal and non-formal parameter list

1a	Complete formal call (textual only) NOTE Shall be used if EN/ENO is necessary in calls.	✓	✓	✓	✓	
1b	Incomplete formal call (textual only) NOTE Shall be used if EN/ENO is not necessary in calls.	✓	✓	✓	✓	
2	Non-formal call (textual only) (fix order and complete)	✓	✓	✓	✓	

Table 51 - Interface

1	INTERFACE ... END_INTERFACE	✓	✓	✓	✓	
	Methods and specifiers					
2	METHOD...END_METHOD	✓	✓	✓	✓	
	Inheritance					
3	EXTENDS	✓	✓	✓	✓	
	Usage of interface					
4a	IMPLEMENTS interface	✓	✓	✓	✓	
4b	IMPLEMENTS multi-interfaces	✓	✓	✓	✓	
4c	Interface as type of a variable	✓	✓	✓	✓	

Table 52 – Assignment Attempt

1	Assignment attempt with interfaces using ?=					QueryInterface supports the same functionality
2	Assignment attempt with references using ?=					QueryPointer supports the same functionality

Table 53 – Object oriented function block

1	Object oriented Function block	✓	✓	✓	✓	
1a	FINAL specifier	✓	✓	✓	✓	
	Methods and specifiers					
5	METHOD...END_METHOD	✓	✓	✓	✓	
5a	PUBLIC specifier	✓	✓	✓	✓	
5b	PRIVATE specifier	✓	✓	✓	✓	
5c	INTERNAL specifier	✓	✓	✓	✓	

5d	PROTECTED specifier	✓	✓	✓	✓	
5e	FINAL specifier	✓	✓	✓	✓	
	Usage of interface					
6a	IMPLEMENTS interface	✓	✓	✓	✓	
6b	IMPLEMENTS multi-interfaces	✓	✓	✓	✓	
6c	Interface as type of a variable	✓	✓	✓	✓	
	Inheritance					
7a	EXTENDS	✓	✓	✓	✓	
7b	EXTENDS	✓	✓	✓	✓	
8	OVERRIDE					Keyword override not supported. Same name overrides automatically.
9	ABSTRACT	✓	✓	✓	✓	
	Access reference					
10a	THIS	✓	✓	✓	✓	THIS is a pointer in codesys
10b	SUPER	✓	✓	✓	✓	SUPER is a pointer in codesys
10c	SUPER()	✓	✓	✓	✓	SUPER^()
	Variable access specifiers					
11a	PUBLIC specifier					
11b	PRIVATE specifier					
11c	INTERNAL specifier					
11d	PROTECTED specifier					
	Polymorphism					
12a	with VAR IN OUT with equal signature	✓	✓	✓	✓	
12b	With VAR IN OUT with compatible signature	✓	✓	✓	✓	
12c	with reference with equal signature	✓	✓	✓	✓	
12d	with reference with compatible signature	✓	✓	✓	✓	

Table 54 – SFC step

1a	Step - graphical form with directed links	✓	✓	✓	✓	
1b	Initial step - graphical form with directed link	✓	✓	✓	✓	
2a	Step - textual form without directed links	✓	✓	✓	✓	
2b	Initial step - textual form without directed links	✓	✓	✓	✓	
3a	Step flag - general form ***.X = BOOL#1 when *** is active, BOOL#0 otherwise	✓	✓	✓	✓	
3b	Step flag - direct connection of Boolean variable ***.X to right side of step	✓	✓	✓	✓	
4	Step elapsed time - general form ***.T = a variable of type TIME	✓	✓	✓	✓	

Table 55 – SFC transition and transition condition						
1	Transition condition physically or logically adjacent to the transition using **ST** language	✓	✓	✓	✓	
2	Transition condition physically or logically adjacent to the transition using **LD** language					
3	Transition condition physically or logically adjacent to the transition using **FBD** language					
4	Use of connector					
5	Transition condition: Using **LD** language	✓	✓	✓	✓	
6	Transition condition: Using **FBD** language	✓	✓	✓	✓	
7	Textual equivalent of feature 1 using **ST** language	✓	✓	✓	✓	
8	Textual equivalent of feature 1 using **IL** language	✓	✓	✓	✓	
9	Use of transition name	✓	✓	✓	✓	Transition name must be a valid identifier
10	Transition condition using **LD** language	✓	✓	✓	✓	
11	Transition condition using **FBD** language	✓	✓	✓	✓	
12	Transition condition using **IL** language	✓	✓	✓	✓	
13	Transition condition using **ST** language	✓	✓	✓	✓	
Table 56 – SFC declaration of actions						
1	Any Boolean variable declared in a `VAR` or `VAR_OUTPUT` block, or their graphical equivalents, can be an action.	✓	✓	✓	✓	
2l	Graphical declaration in **LD** language					
2s	Inclusion of SFC elements in **action**					
2f	Graphical declaration in **FBD** language					
3s	Textual declaration in **ST** language					
3i	Textual declaration in **IL** language					
Table 57 – Step/action association						
1	Action block physically or logically adjacent to the step	✓	✓	✓	✓	
2	Concatenated action blocks physically or logically adjacent to the step	✓	✓	✓	✓	
3	Textual step body	✓	✓	✓	✓	
4	Action block "d" field					

Table 58 – Action block						
1	"a" : Qualifier as per **Fehler! Verweisquelle konnte nicht gefunden werden.**	✓	✓	✓	✓	
2	"b" : Action name	✓	✓	✓	✓	
3	"c" : Boolean "indicator" variables (**deprecated**)					
	"d" : Action using:					
4i	IL language					
4s	ST language					
4l	LD language					
4f	FBD language					
5l	Use of action blocks LD					
5f	Use of action blocks in FBD					
Table 59 – Action qualifiers						
1	Non-stored (null qualifier)	✓	✓	✓	✓	
2	**N**on-stored	✓	✓	✓	✓	
3	overriding **R**eset	✓	✓	✓	✓	
4	**S**et (**S**tored)	✓	✓	✓	✓	
5	time **L**imited	✓	✓	✓	✓	
6	time **D**elayed	✓	✓	✓	✓	
7	**P**ulse	✓	✓	✓	✓	
8	**S**tored and time **D**elayed	✓	✓	✓	✓	
9	**D**elayed and **S**tored	✓	✓	✓	✓	
10	**S**tored and time **L**imited	✓	✓	✓	✓	
11	**P**ulse (rising edge)	✓	✓	✓	✓	
12	**P**ulse (falling edge)	✓	✓	✓	✓	
Table 60 – Action control features						
1	With final scan	✓	✓	✓	✓	
2	Without final scan					
Table 61 – Sequence evolution – graphical						
1	Single sequence	✓	✓	✓	✓	
2a	Divergence of sequence with left to right priority	✓	✓	✓	✓	
2b	Divergence of sequence with numbered branches					
2c	Divergence of sequence with mutual exclusion					
3	Convergence of sequence	✓	✓	✓	✓	
4a	Simultaneous divergence after a single transition	✓	✓	✓	✓	
4b	Simultaneous convergence before one transition					
4c	Simultaneous divergence after conversion					
4d	Simultaneous convergence					

	before a sequence selection					
5a,b,c	Sequence skip	✓	✓	✓	✓	
6a,b,c	Sequence loop					
7	Directional arrows	✓	✓	✓	✓	With jumps

Table 62 – Configuration and resource declaration

1	CONFIGURATION...END_CONFIGURATION					Resource configuration and Application configuration is done with specialized editors
2	VAR_GLOBAL...END_VAR within CONFIGURATION					
3	RESOURCE...ON ...END_RESOURCE					
4	VAR_GLOBAL...END_VAR within RESOURCE					
5a	Periodic TASK (see NOTE 1)					
5b	Non-periodic TASK (see NOTE 1)					
6a	WITH for PROGRAM to TASK association (see NOTE 1)					
6b	WITH for FUNCTION BLOCK to TASK association (see NOTE 1)					
6c	PROGRAM with no TASK association (see NOTE 1)					
7	Directly represented variables in VAR_GLOBAL					
8a	Connection of directly represented variables to PROGRAM inputs					
8b	Connection of GLOBAL variables to PROGRAM inputs					
9a	Connection of PROGRAM outputs to directly represented variables					
9b	Connection of PROGRAM outputs to GLOBAL variables					
10a	VAR_ACCESS...END_VAR					
10b	Access paths to directly represented variables					
10c	Access paths to PROGRAM inputs					
10d	Access paths to GLOBAL variables in RESOURCEs					
10e	Access paths to GLOBAL variables in CONFIGURATIONs					
10f	Access paths to PROGRAM outputs					
10g	Access paths to PROGRAM internal variables					
10h	Access paths to function block inputs					
10i	Access paths to function block outputs					
11a	VAR_CONFIG...END_VAR to variables [a]	✓	✓	✓	✓	
11b	VAR_CONFIG...END_VAR to components of structures	✓	✓	✓	✓	
12a	VAR_GLOBAL CONSTANT in RESOURCE					
12b	VAR_GLOBAL CONSTANT in CONFIGURATION					
13a	VAR_EXTERNAL in RESOURCE					
13b	VAR_EXTERNAL CONSTANT in RESOURCE					

Table 63 - Task						
1a	Textual declaration of periodic TASK					Task declaration is done with a specialized editor.
1b	Textual declaration of non-periodic TASK					
	Graphical representation of TASKs (general form)					
2a	Graphical representation of periodic TASKs (witrh INTERVAL)					
2b	Graphical representation of non-periodic TASK (with SINGLE)					
3a	Textual association with PROGRAMs					
3b	Textual association with function blocks					
4a	Graphical association with PROGRAMs					
4b	Graphical association with function blocks within PROGRAMs					
5a	Non-preemptive scheduling					Scheduling is depending on OS on Runtime, typically preemptive scheduling
5b	Preemptive scheduling	✓	✓	✓	✓	Scheduling is depending on OS on Runtime, typically preemptive scheduling
Table 64 – Namespace						
1a	Public namespace (without access specifier)	✓	✓	✓	✓	Namespaces are a feature of our library concept, when properly used (qualified-only)
1b	Internal namespace (with INTERNAL specifier)	✓	✓	✓	✓	
2	Nested namespaces	✓	✓	✓	✓	
3	Variable access specifier INTERNAL Equivalent to feature in Table 48	✓	✓	✓	✓	
4	Method access specifier INTERNAL Equivalent to feature in Table 48	✓	✓	✓	✓	
5	Language element with access specifier INTERNAL: • User-defined data types - using keyword TYPE • Functions • Function block types • Classes • Interfaces	✓ ✓ ✓ ✓	✓	✓	✓	
Table 65 – Nested namespace declaration options						
1	Lexically nested namespace declaration Equivalent to feature 2 of Table 64	✓	✓	✓	✓	
2	Nested namespace declaration by fully qualified name	✓	✓	✓	✓	
3	Mixed lexically nested namespace and namespace nested by fully qualified name	✓	✓	✓	✓	

Table 66 – Namespace directive USING							
1	`USING` in global namespace						
2	`USING` in other namespace						
3	`USING` in POUs ▪ Functions ▪ Function block types ▪ Classes ▪ Methods ▪ Interfaces						
Table 67– Parenthesized expression for IL language							
1	Parenthesized expression beginning with explicit operator:				✓		
2	Parenthesized expression (short form)				✓		
Table 68 (65) – Instruction list operators							
1	`LD`				✓		
2	`ST`				✓		
3	`S ' R`				✓		
4	`AND`				✓		
5	`&`				✓		
6	`OR`				✓		
7	`XOR`				✓		
8	`NOT`				✓		
9	`ADD`				✓		
10	`SUB`				✓		
11	`MUL`				✓		
12	`DIV`				✓		
13	`MOD`				✓		
14	`GT`				✓		
15	`GE`				✓		
16	`EQ`				✓		
17	`NE`				✓		
18	`LE`				✓		
9	`LT`				✓		
20	`JMP`				✓		
21	`CAL`				✓		
22	`RET`				✓		
23	`)`				✓		
24	`ST?`						
Table 69 – Calls for IL language							
1a	Function block call with **non-formal** parameter list				✓		
1b	Function block call with **formal** parameter list				✓		
2	Function block call with **load/stor**e of standard input				✓		

	parameters					
3a	Function call with **formal** parameter list				✓	
3b	Function call with **non-formal** parameter list				✓	
4a	Method call with **formal** parameter list				✓	
4b	Method call with **non-formal** parameter list				✓	

Table 70 – Standard function block operators for IL language

1	SR				✓	
2	RS				✓	
3	F/R_TRIG				✓	
4	CTU				✓	
5	CTD				✓	
6	CTUD				✓	
7	TP				✓	
8	TON				✓	
9	TOF				✓	

Table 71 – Operators of the ST language

1	Parentheses			✓		
2	Evaluation of result of function and method – if a result is declared			✓		
3	Dereference			✓		
4	Negation			✓		
5	Unary Plus			✓		
6	Complement			✓		
7	Exponentiation			✓		
8	Multiply			✓		
9	Divide			✓		
10	Modulo			✓		
11	Add			✓		
12	Subtract			✓		
13	Comparison			✓		
14	Equality			✓		
15	Inequality			✓		
16a	Boolean AND					
16b	Boolean AND			✓		
17	Boolean Exclusive OR			✓		
18	Boolean OR			✓		

Table 72 – ST language statements

1	**Assignment** Variable := expression;			✓		
1a	Variable and expression of elementary data type			✓		
1b	Variables and expression of different elementary data types with implicit type conversion according **Fehler! Verweisquelle konnte nicht gefunden werden.**			✓		
1c	Variable and expression of user-defined type			✓		
1d	Instances of function block type			✓		

	Call					
2a	Function call			✓		
2b	Function block call and function block output usage			✓		
2c	Method call			✓		
3	`RETURN`			✓		
	Selection					

4	`IF ...`
	`THEN ...`
	`  ELSIF ...`
	`  THEN ...`
	`ELSE ...END_IF`
				✓		

5	`CASE ... OF`
	`  ...`
	`  ELSE ...`
	`END_CASE`
				✓		

	Iteration					

6	`FOR ... TO ... BY ... DO`
	`  ...`
	`END_FOR`
				✓		

7	`WHILE ... DO`
	`...`
	`END_WHILE`
				✓		

8	`REPEAT ...`
	`UNTIL ...`
	`END_REPEAT`
				✓		

9	`CONTINUE`			✓		
10	`EXIT` an iteration			✓		
11	Empty Statement			✓		

Table 73 – Graphic execution control elements

	Unconditional jump					
1a	FBD language	✓				
1b	LD language		✓			
	Conditional jump					
2a	FBD language	✓				
2b	LD language		✓			
	Conditional return					
3a	LD language	✓				
3b	FBD language		✓			
	Unconditional return					
4	LD language	✓	✓			

Table 74 – Power rails and link elements

1	Left power rail (with attached horizontal link)	✓				
2	Right power rail (with attached horizontal link)	✓				
3	Horizontal link	✓				

4	Vertical link (with attached horizontal links)	✓				
Table 75 - Contacts						
	Static contacts					
1	Normally open contact	✓				
2	Normally closed contact	✓				
	Transition-sensing contacts					
3	Positive transition-sensing contact	✓				
4	Negative transition-sensing contact	✓				
5a	Compare contact (typed)					
5b	Compare contact (overloaded)					
Table 76 - Coils						
	Momentary coils					
1	Coil	✓				
2	Negated coil	✓				
	Latched Coils					
3	Set (latch) coil	✓				
4	Reset (unlatch) coil .	✓				
	Transition-sensing coils					
8	Positive transition-sensing coil					
9	Negative transition-sensing coil					