Merged feedback for corrigendum for 3rd edition of the IEC 61131-3
Jan. 2007. Editor: EvdWal / PLCopen.org

Table of Content

1. Comments by Andy Verwer MMU

I have had one difficulty with the 2002 version of '1131-3 which you may be interested in. It concerns the SFC action qualifiers "S" and "R" when used with simple Boolean actions. The latest version clearly describes how the qualifiers operate with the action control function block - using a RS latch within the block. unfortunately, this method if implementing the qualifiers has an undesirable side effect: After the execution of an "S" action, the latch will write TRUE to the variable every program cycle even if that step is no longer active. This means that another step or program etc cannot write a FALSE to the variable - i.e it can ONLY be reset by an "R" step in the same POU.

I feel that this is an unnecessary restriction. Wouldn't life have been simpler if the qualified actions were described in ST as:

```
IF S_Qualifier THEN Q:=TRUE;
IF R_Qualifier THEN Q:=FALSE;
```

This simple solution is simpler to implement, behaves in the same way as the S & R qualifiers in LD and overcomes the problem described above. A similar simple description can be formulated for the other qualifiers.

If you are interested in following up this last point, I can supply you with some simple examples that illustrate the problem and work arounds we have had to use.

2. Multiple Feedback KirchnerSoft

Table 4 - Numeric literals

****

bug : BOOL#TRUE, BOOL#FALSE

9) Typed Literals

BOOL#0 BOOL#1 BOOL#TRUE BOOL#FALSE

but:
B.1.2.1   Numeric literals
boolean_literal ::= (['BOOL#'] ('1' | '0'))| 'TRUE' | 'FALSE'

so, according to B.1.2.1 BOOL#TRUE is not allowed, but according to Tbl4) it is a typedLiteral.

TABLE14

****

bug: 3) defines ANALOG_DATAZ, 4) uses ANALOG_DATA

2.5.2.2 Declaration - Figure 10

****

bug : output of functionblock SR is Q1 and not Q !

```
   DB_FF : SR ;
   ...
   OUT := DB_FF.Q ;
```

should be

```
   OUT := DB_FF.Q1 ;
```

2.7 Configuration elements- Figure 19b

****

bug : missing ";" after END_VAR

after all usages of END_VAR in Fig19b the ";" is missing.

3.3.2 Statements - Table 56.4

****
bug : 4 is not a real constant.

D := B*B - 4*A*C ;

A,B,C and D are REAL constants, but 4 is not a REAL constant. Type-Error!

3.3.2.4 Iteration statements Fig. 22

****
bug : missing ; after END_IF

IF FLAG THEN EXIT ; END_IF

B.1.2.1   Numeric literals

****
bug: undefined unsigned_integer

bit_string_literal ::= [('BYTE' | 'WORD' | 'DWORD' | 'LWORD') '#']
          (unsigned_integer | binary_integer | octal_integer | hex_integer)

B.1.3.3 Derived data types vs. 2.3.3.1

****
bug : enumeration list with A#B.

enumerated_type_declaration ::= enumerated_type_name ':' enumerated_spec_init
enumerated_spec_init ::= enumerated_specification [':=' enumerated_value]
enumerated_specification ::=
('(' enumerated_value {',' enumerated_value} ')') | enumerated_type_name
enumerated_value ::= [enumerated_type_name '#'] identifier

so,
TYPE SPEED: (SLOW, SPEED#MEDIUM, FAST, VERY_FAST); END_TYPE
is valid.

but:

2.3.3.1 Declaration

Derived (i.e., user- or manufacturer-specified) data types can be declared using the TYPE...END_TYPE textual construction shown in table 12. These derived data types can then be

used, in addition to the elementary data types defined in 2.3.1, in variable declarations as defined in 2.4.3.
An enumerated data type declaration specifies that the value of any data element of that type can only take on one of the values given in the associated list of identifiers, as illustrated in table 12. The enumeration list defines an ordered set of enumerated values, starting with the first identifier of the list, and ending with the last. Different enumerated data types may use the same identifiers for enumerated values. The maximum allowed number of enumerated values is an implementation-dependent parameter.
To enable unique identification when used in a particular context, enumerated literals may be qualified by a prefix consisting of their associated data type name and the '#' sign, similar to typed literals defined in 2.2.1.

!!!!!!!!
Such a prefix shall not be used inside an enumeration list. It is an error if sufficient
!!!!!!!!

so,
TYPE SPEED: (SLOW, SPEED#MEDIUM, FAST, VERY_FAST); END_TYPE
SPEED#MEDIUM is not ok.
so what?

possible fix:

enumerated_spec_init ::= enumerated_specification [':=' enumerated_value]
enumerated_specification ::=
    ('(' enumerated_value_spec {',' enumerated_value_spec} ')') | enumerated_type_name
enumerated_value_spec ::= identifier
enumerated_value ::= [enumerated_type_name '#'] identifier

B.1.4.3 Declaration and initialization

****
bug : VAR_GLOBAL A: ; END_VAR; ok???

global_var_decl ::= global_var_spec ':' [located_var_spec_init | function_block_type_name]
(and the type is optional).

B.1.5.2 Function blocks

****
bug : undefined non_retentive_var_declarations

other_var_declarations ::= external_var_declarations | var_declarations |
retentive_var_declarations | non_retentive_var_declarations | temp_var_decls |
incompl_located_var_declarations

and non_retentive_var_declarations is nowhere defined in the norm.

B.3.1 Expressions

****

bug: it is not possible to implement a function with no parameter to just returns a constant.

primary_expression ::=
constant | enumerated_value | variable | '(' expression ')' | function_name '(' param_assignment {','
param_assignment} ')'
param_assignment ::= ([variable_name ':='] expression) | (['NOT'] variable_name '=>' variable)

ANNEX F.6.3

****

bug: mixing of real/int types in one statement.

```
FUNCTION_BLOCK AVERAGE
VAR_INPUT
RUN : BOOL ;      (* 1 = run, 0 = reset *)
XIN : REAL ;     (* Input variable *)
N   : INT ;      (* 0 <= N < 128 or manufacturer- *)
END_VAR          (*      specified maximum value  *)
VAR_OUTPUT
XOUT : REAL ;
END_VAR (* Averaged output *)
VAR
SUM  : REAL := 0.0; (* Running sum *)
FIFO : DELAY ;      (* N-Element FIFO *)
END_VAR

SUM := SUM - FIFO.XOUT ;
FIFO (RUN := RUN , XIN := XIN, N := N) ;
SUM := SUM + FIFO.XOUT ;
IF RUN THEN
  //??XOUT := SUM/N ;
  //??>> SUM is REAL, N is INT -> Typeclash!
  //fix:
  XOUT := SUM/INT_TO_REAL(N) ;
ELSE
  //??SUM := N*XIN ;
  //??>> SUM is REAL, N is INT -> Typeclash!
  //fix:
  SUM := INT_TO_REAL(N)*XIN ;
  XOUT := XIN ;
  END_IF ;
END_FUNCTION_BLOCK
```

F.11 Function block ALRM_INT

****

bug a) TLO is defined, but THL will be used.
    b) logic error.

```
FUNCTION ALRM_INT : BOOL
VAR_INPUT
 IN : INT ;
 THI : INT ;(* High threshold *)
//?? TLO : INT ; (* Low threshold *)
//fix: define THL
 THL : INT ; (* Low threshold *)
END_VAR
VAR_OUTPUT
 HI: BOOL; (* High level alarm *)
 LO: BOOL; (* Low level alarm *)
END_VAR

 HI := IN > THI ;
 LO := IN < THL ;
//?? ALRM_INT := THI OR THL ;
//?? what should this be?
//fix:
 ALRM_INT := HI OR LO ;

END_FUNCTION
```

3. Corr./Add. W. Zeilinger

10. RTC

What happened with the function block RTC in IEC 61131-3 2nd edition (IEC 61131-3 1st edition Kapitel 2.5.2.3.4).

Proposals 1 with set and get functions:
SET_RTC_DT with VAR_INPUT IN of type DATE_AND_TIME
GET_RTC_DT with VAR_OUTPUT OUT of type DATE_AND_TIME
SET_RTC_TIME with VAR_INPUT IN of type TIME
GET_RTC_TIME with VAR_OUTPUT OUT of type TIME
SET_RTC_DATE with VAR_INPUT IN of type DATE
GET_RTC_DATE with VAR_OUTPUT OUT of type DATE

Usage in ST:

```
if (RTRIG(SetNewRtcTime)) then
  SET_RTC_DT(NewRtcTime);
end_if;
CurrentRtcTime:=GET_RTC_DT();
```

Usage in LD:

```
                  +--------------+
     SetNewRtcTime      | SET_RTC_TIME |
-----------| P |----------|EN        ENO|---
         NewRtcTime----|              |
                  +--------------+


          +-----------------+
          |  GET_RTC_TIME   |---CurrentRtcTime
          +-----------------+
```

Proposals 2 with overloaded set and get functions:
SET_RTC with VAR_INPUT IN of the types DATE, TIME, DATE_AND_TIME
GET_RTC with VAR_OUTPUT OUT of the types DATE, TIME, DATE_AND_TIME

Proposals 3 as conversion functions:
DT_TO_RTC with VAR_INPUT IN of type DATE_AND_TIME
RTC_TO_DT with VAR_OUTPUT OUT of type DATE_AND_TIME
TIME_TO_RTC with VAR_INPUT IN of type TIME
RTC_TO_TIME with VAR_OUTPUT OUT of type TIME
DATE_TO_RTC with VAR_INPUT IN of type TIME

RTC_TO_DATE with VAR_OUTPUT OUT of type TIME

Proposals 4 as overloaded conversion functions:
TO_RTC with VAR_INPUT IN of the types DATE, TIME, DATE_AND_TIME
RTC_TO with VAR_OUTPUT OUT of the types DATE, TIME, DATE_AND_TIME

3. Feedback Mario de`Sousa

This concerns an older .pdf file, dealing till the compiler errors.
See separate doc.