Differences between IEC 61131-3 standard and CPDev.STComp05 compiler for version 0.4.14.63 (final)

1. Local declaration `VAR` cannot involve modifier `AT` to locate a variable (local variables are located automatically).
2. Global variable declaration cannot begin with `AT`.
3. `VAR_EXTERNAL` can be used in programs, function blocks and functions.
4. Global arrays are accessible everywhere without `VAR_EXTERNAL`.
5. Access paths declared by `VAR_ACCESS` are not handled.
6. Identifiers are associated with name spaces (ranges of identifier names). Identifier name must be unique in any space. Each project or imported library is a separate name space. Short names of identifiers in different spaces may be the same. Finding short name is executed as follows:
   a) directly in current space, in which the name is called,
   b) in imported space (library), but at the main level only and depending on call type (call with determined type category, call with any type category); name matching in other spaces is also checked. If the same name is found, an error of identifier ambiguity is raised („Ambiguous ...", „Multiple name found ...", etc.). If this happens, full name in the form `NAMESPACE.IDENTIFIER` must be used. Name spaces of the system involves dollar sign (`$`), e.g. `$DEFAULT`, `$VMSYS`. To get into such space in case of ambiguity, use directive `NS` (NameSpace) for instance `(*$NS $DEFAULT*)ADD(3, 4)`.
7. Numbers may be entered also with other radix than described in the standard. Standard allows for 2, 8, 10 and 16 radix. Compiler is more convinient and use radix from 2 to 36. If the radix is different than 10, input should have all following points: 1) base of number system (in decimal form), 2) the sign #, 3) alphanumeric string. For instance `5#24` means 14 decimal (also may be written like 10#14 or 16#E).
8. Numeric literal may appear in declaration of non-numeric types, e.g. `BOOL#1`, `WORD#16#55AA`. Numeric literal without the dot (`.`) is always of the type `INT`, and with the dot of the type `REAL`. To change default type of constant INT to DINT use a compilation switch or write it with type prefix like DINT#1. Type of a constant may be implied in some cases.
9. Optional inputs to function blocks are defined by: 1) initial value of the input, 2) and/or directive `PAROPT` after name of the input.
10. Configuration elements declared by the following keyword constructions are not allowed: `CONFIGURATION`, `RESOURCE ... ON ...` , `TASK ...(INTERVAL := ..., PRIORITY := ...)` , `PROGRAM ... WITH ... : ...` etc. Tasks are declared by means of directive `TASKS`.
11. So far, user types declared by `TYPE` ... `END_TYPE` are permitted for structures and arrays only. Other types, such as aliases (e.g. USINT, UINT, UDINT), enumeration and subrange are not implemented yet. Elementary data types must be used instead.
12. `STRING` type is now handled by the compiler but not WSTRING.
13. Entry point of `CASE` instruction (sequence before the colon `:`) requires constant or constants. Identifiers declared as constants (or expressions) are not allowed. Compiler does not check, whether ranges of entry points do not overlap, and whether types of entries correspond to the type of `CASE` control variable.
14. Type of `FOR` loop is determined by incrementation constant that follows `BY` keyword (if used). For negative constant the system checks whether loop control variable is greater or equal to final value; for positive constant it checks whether the variable is less or equal to the final value. Incrementation constant can not be 0.
15. Contrary to function blocks, assignments to parameter names are not allowed while calling a

function (no matter whether the function is built-in, user-defined, or *in-line*). For instance, function call may look like `X := SEL(BWY, P, 1);` , whereas function block call is `BSWI1(S:=BWY, IN1 := Pb, IN2 := TRUE);` Parameters of function block call may appear in any order (names matter), whereas order of function parameter is fixed (as defined by the programmer).

16. Communication blocks `COM_SMx` (for SMC controller) are specific examples of function blocks not being called during program execution. The blocks merely reserve memory space for data of communication subsystem. Assignment of outputs directly in the block call is not permitted. Inputs denoted as constants (e.g. slave number) must have the same value in each of corresponding instances.
17. **This point is subject of change in future compiler 0.4.15.*.**
    Multidimensional arrays are treated as one-dimension, but indexing value are computed in special way (its semantics is different like in CoDeSys or other compilers). Details are follow:

```
   VAR
     TRAN2   : ARRAY[1..4,0..20] OF INT :=
 ( 22,24,26,27,33,34,36,37,42,43,44,46,47,62,63,64,72,73,74,0, (*PAD*)  0,
   22,24,26,27,33,34,36,37,42,43,44,46,47,62,63,64,72,73,74,0, (*PAD*)  0,
   22,24,26,27,33,34,36,37,42,43,44,46,47,62,63,64,72,73,74,0, (*PAD*)  0,
   22,24,26,27,33,34,36,37,42,43,44,46,47,62,63,64,72,73,74,0, (*PAD*)  0
 );
   END_VAR
```

    Name `TRAN2` is treated like 21 arrays of 4-elements array, than 4 array of 21 elements. So values obtained in CPDev may be quite different from values in CoDeSys. In fact value `TRAN2[1,5]` indicates to cell with value `72` instead of expected `34` in CoDeSys. To access elements close to each other it is necessary to change the first index in array, not as in CoDeSys the last one. Above padding mark `(*PAD*)` in declaration was included to show that value in row is missing.
    Iterating each value in such array is following:

```
FOR I := 0 TO 20 DO
   FOR J := 1 TO 4 DO
     TRAN2[J, I] := expr;
   END_FOR
END_FOR
```

18. Global instances of function blocks are now handled.
19. Changes to 0.4.14.* compiler series will be made only when serious bug will appear. Now focus is moved to develop 0.4.15.* compiler.