BC SW – Style Guide

Version overview

Version	Author	Chapter	Remarks/Changes
1.0	Lucka		First version

Shortcuts und glossary

Contents

1 Naming Rules

1.1 Requirement 1: Use of prefixes

Priority: High

Description: To improve readability and to reduce programming mistakes the use of prefixes shall be introduced. A variable should include several types of information which include the scope, datatype, and control information.

Following prefixes shall be used to provide the information mentioned above.

Datatype indication:

Type	Prefix
BOOL	bo
SINT	si
USINT	usi
INT	i
DINT	di
UINT	ui
UDINT	udi
REAL	r
LREAL	lr
TIME	t
DATE	dt
TIME_OF_DAY	tod
DATE_AND_TIME	dtt
STRING	str
WSTRING	wstr
BYTE	by
WORD	w
DWORD	dw

Custom datatype indication:

Type	Prefix
STRUCT	st
ARRAY	a
FUNCTION BLOCK	fb
PROGRAM	prg

Scope indication:

Type	Prefix
Global	glb
Retain	ret
Temporary	tmp

Control indication:

Type	Prefix software	Prefix hardware
Input	i	I
Output	o	O

If the use of multiple prefixes is required, they shall be used as followed:

<Scope>_<Control>_<Datatype><Name>
Example: glb_I_boStartButton

Example:

```
DON'T:
VAR_GLOBAL
    Error          : BOOL;
END_VAR

VAR_INPUT
    Enable         : BOOL;
END_VAR

VAR
    Start          : BOOL;
    Count          : INT;
    State          : UINT;

    Autoload       : AUTOLOAD_FB;
END_VAR
```

```
DO:
VAR_GLOBAL
    glb_boError    : BOOL;
END_VAR

VAR_INPUT
    i_boEnable     : BOOL;
END_VAR

VAR
    boStart        : BOOL;
    iCount         : INT;
    uiState        : UINT;

    fbAutoload     : AUTOLOAD_FB;
END_VAR
```

1.2 Requirement 2: Use of capitalization

Priority: High

Description: To further improve readability and understanding of code the use of capitalization is a small but powerful way of naming all variables, constants, functions, function blocks, etc.
From now on CamelCase shall be used for all non-constant objects. For constant objects use UPPER_SNAKE_CASE.

Exception: When beginning an object name with a prefix, the prefix is all lowercase.

Example:

```
DON'T:
VAR_CONSTANT
    Numberofagvs        : INT;
END_VAR

FORK_REG();
HydraulicPressure();
AUTOLOAD_FB();
```

```
DO:
VAR_CONSTANT
    iNUMBER_OF_AGVS     : INT;
END_VAR

fbForkReg();
fbHydraulicPressure();
fbAutoload();
```

1.3 Requirement 3: Names to avoid

Priority: High

Description: The use of keywords and reserved words must be avoided to prevent build errors. Even words the currently used compiler does not mark as a keyword or reserved word should not be used to maintain portability. To prevent the use of such words the following table lists all words that are prohibited to be used as an object name (not for use in a name):

ABS	LTOD	LTOD	SQRT
ABSTRACT	END_REPEAT	LWORD	SR
ACOS	END_RESOURCE	MAX	STEP
ACTION	END_STEP	METHOD	STRING
ADD	END_STRUCT	MID	STRING#
AND	END_TRANSITION	MIN	STRUCT
ARRAY	END_TYPE	MOD	SUB
ASIN	END_VAR	MOVE	SUPER
AT	END_WHILE	MUL	T
ATAN	EQ	MUX	TAN
ATAN2	EXIT	NAMESPACE	TASK
BOOL	EXP	NE	THEN
BY	EXPT	NON_RETAIN	THIS
BYTE	EXTENDS	NOT	TIME
CASE	F_EDGE	NULL	TIME_OF_DAY
CHAR	F_TRIG	OF	TO
CLASS	FALSE	ON	TOD
CONCAT	FINAL	OR	TOF
CONFIGURATION	FIND	OVERLAP	TON
CONSTANT	FOR	OVERRIDE	TP
CONTINUE	FROM	PRIORITY	TRANSITION
COS	FUNCTION	PRIVATE	TRUE
CTD	FUNCTION_BLOCK	PROGRAM	TRUNC
CTU	GE	PROTECTED	TYPE
CTUD	GT	PUBLIC	UDINT
DATE	IF	R_EDGE	UINT
DATE_AND_TIME	IMPLEMENTS	R_TRIG	ULINT
DELETE	INITIAL_	READ_ONLY	UNTIL
DINT	STEP	READ_WRITE	USING
DIV	INSERT	REAL	USINT
DO	INT	REF	VAR
DT	INTERFACE	REF_TO	VAR_ACCESS
DWORD	INTERNAL	REPEAT	VAR_CONFIG
ELSE	INTERVAL	REPLACE	VAR_EXTERNAL
ELSIF	LD	RESOURCE	VAR_GLOBAL
END_ACTION	LDATE	RETAIN	VAR_IN_OUT
END_CASE	LDATE_AND_TIME	RETURN	VAR_INPUT
END_CLASS	LDT	RIGHT	VAR_OUTPUT
END_CONFIGURATION	LE	ROL	VAR_TEMP
END_FOR	LEFT	ROR	WCHAR
END_FUNCTION	LEN	RS	WHILE
END_FUNCTION_BLOCK	LIMIT	SEL	WITH
END_IF	LINT	SHL	WORD
END_INTERFACE	LN	SHR	WSTRING
END_METHOD	LOG	SIN	XOR
END_NAMESPACE	LREAL LT LTIME	SINGLE	
END_PROGRAM	LTIME_OF_DAY	SINT	

1.4 Requirement 4: Acceptable name length

Priority: Medium

Description: To further ensure an easier understanding and a better maintainability of code this rule defines mnemonics and a maximum and a minimum length range for object names to use. The following guidelines should be followed:

1. Don't use less than
   a. 8 characters for object names
   b. 3 characters for local variables
2. A maximum length of 25 characters for objects should not be exceeded
3. The average length of names should be around 15 characters
4. Abbreviations should only be used if well known
5. Names shouldn't be very similar

The following table shows a list of well-known abbreviations:

Abbrev.	Meaning
Min	Minimum
Max	Maximum
Act	Actual, Current
Next	Next value
Prev	Previous value
Avg	Average
Sum	Total sum
Diff	Difference
Cnt	Count
Len	Length
Pos	Position
Ris	Rising edge
Fal	Falling edge
Old	Old value
Sim	Simulated
Dir	Direction
Err	Error
Warn	Warning
Cmd	Command
Addr	Address

Exceptions: This rule can be ignored if for example loop indexes are used.

Example:

```
DON'T:
abc             : INT;
Go              : BOOL;

MaximumTemepratureForThermocoupleOutput   : REAL;
```

```
DO:
boStart                   : BOOL;

rMaxTCTemeperature        : REAL;
```

1.5 Requirement 5: Acceptable character set

Priority: Medium

Description: The use of special characters could lead to problems in portability of developing environments or platforms and leads to worse readability. To prevent those problems only alphanumeric and underscore characters should be used.

Example:

```
DON'T:
boDépart     : BOOL;
boBöse       : BOOL;
```

```
DO:
boDepart     : BOOL;
boBoese      : BOOL;
```

1.6 Requirement 6: Similarity of names

Priority: Medium

Description: The use of nearly identical names for objects should be avoided even if compilation is possible. This prevents readability issues.

Example:

```
DON'T:
VAR_GLOBAL
    MyResult       : REAL;
END_VAR

FUNCTION_BLOCK MyResult
    MyResult       : REAL;
END_FUNCTION_BLOCK
```

```
DO:
VAR_GLOBAL
    glb_rMyResult : REAL;
END_VAR

FUNCTION_BLOCK FBNAME_FB
    rMyResult      : REAL;
END_FUNCTION_BLOCK
```

2 Coding Practice

2.1 Requirement 7: All variables shall be initialized before being used

Priority: High

Description: All variables shall be initialized before being read by another part of code. This does not apply for variables that need to be initialized as 0, because the PLC initializes all variables as 0. Variables that are linked to physical inputs don't need to be initialized.

Example:

```
DON'T
VAR
      LoadDist                : INT := 360;
      SavePara                : BOOL;
      TargetSpeed             : INT;
      TargetPosition          : DINT;
      SteerSetAngleEnabled    : BOOL := 0;
      SteerMethod             : DINT := 0;
      SteerMethodMan          : DINT := 0;
END_VAR
```

```
DO
VAR
      iLoadDist               : INT := 360;
      boSavePara              : BOOL;
      iTargetSpeed            : INT;
      diTargetPosition        : DINT;
      boSteerSetAngleEnabled  : BOOL;
      diSteerMethod           : DINT;
      diSteerMethodMan        : DINT;
END_VAR
```

2.2 Requirement 8: Direct addressing should be done in one location to avoid overlapping

Priority: High

Description: When assigning a memory location to an object, it should be taken care that the memory is not already assigned. To ensure no overlapping of memory usage, all direct memory addressing should be done in one location.
Direct memory addressing is only allowed in STOPSYMBOLS.POE

Example:

DON'T

```
PROGRAM StopSymbols
      VAR_GLOBAL
            StopButton  AT %MX0.0    : BOOL;      (* soft stop button is pressed *)
      END_VAR
END_PROGRAM

PROGRAM GLOBAL
      VAR_GLOBAL
            HostStop    AT %MX0.0    : BOOL;      (* stop by host *)
      END_VAR
END_PROGRAM
```

DO

```
PROGRAM prgStopSymbols
      VAR_GLOBAL
            glb_O_dwStopWord        AT %MX0.0    : DWORD;     (*Sum of the four
                                                                 first bytes*)

            glb_O_boStopButton      AT %MX0.0    : BOOL;      (* soft stop button
                                                                 is pressed *)
            glb_O_boHostStop        AT %MX0.1    : BOOL;      (* stop by host *)
      END_VAR
END_PROGRAM
```

2.3 Requirement 9: Avoid external variables in functions, function blocks and classes

Priority: extreme

Description: The use of external variables referencing global variables in functions and function blocks shall be avoided. This means do not use VAR_EXTERNAL inside the definition of a function or function block.
Instead of using external references to global variables, the input and output parameter list can be extended.
Encapsulation of data can minimize integration testing and remove functional testing for pre-tested POUs as the known behavior can't change.
Using In- and Output variables can also help grasping the variable access and determine where a variable is written.

Example:

```
DON'T
FUNCTION_BLOCK CheckTemperature
VAR_EXTERNAL
      boModeAuto        : BOOL;     (* AGV is in Automode *)
      iCurrentTemp      : INT;      (* Temperature of CVC *)
END_VAR;
VAR_INPUT
      i_boEnable        : BOOL;     (* Enable Function *)
      i_iMaxTemp        : INT;      (* Maximum CVC Temperature *)
END_VAR;
VAR_OUTPUT
      o_boTempOK        : BOOL;     (* Temperature Status OK *)
END_VAR;

o_boTempOK := i_boEnable AND boModeAuto AND (iCurrentTemp < i_iMaxTemp);
```

```
DO
FUNCTION_BLOCK CheckTemperature
VAR_INPUT
      i_boEnable        : BOOL;     (* Enable Function *)
      i_iMaxTemp        : INT;      (* Maximum CVC Temperature *)
      i_boModeAuto      : BOOL;     (* AGV is in Automode *)
      i_iCurrentTemp    : INT;      (* Current Temperature of CVC *)

END_VAR;
VAR_OUTPUT
      o_boTempOK        : BOOL;     (* Temperature Status OK *)
END_VAR;

o_boTempOK := i_boEnable AND i_boModeAuto AND (i_iCurrentTemp < i_iMaxTemp);
```

2.4 Requirement 10: Floating point comparison shall not be equality or inequality

Priority: High

Description: Using equality or inequality operators to detect a threshold with floating point variable is prohibited. Instead use only the following operators:

- strict less than (<)
- less than or equal (<=)
- strict greater than (>)
- greater than or equal (>=).

2.5 Requirement 11: Time and physical measures comparison shall not be equality or inequality

Priority: High

Description: Using equality or inequality operators to detect a threshold with time information or physical measure even in Integer format is prohibited. Instead use only the following operators:

- strict less than (<)
- less than or equal (<=)
- strict greater than (>)
- greater than or equal (>=).

2.6 Requirement 12: Physical inputs and outputs shall be read and written once per plc cycle

Priority: High

Description: The physical inputs shall be read only once per PLC cycle. Reading the physical inputs at the beginning of the task ensures that every POU uses the same information each cycle.

The physical outputs shall be written only once per PLC cycle. The physical outputs should be written in one location at the end of the task.

2.7 Requirement 13: POUs shall have a single point of exit

Priority: High

Description: For testability, readability, and maintainability reasons and to provide easy debugging there should only be one point of exit in any POU created. This rule prohibits the use of RETURN and forces to create conditional instructions.

Example:

```
DON'T:
IF iResult = 1 THEN
    RETURN;
END_IF;

…some more code
```

```
DO:
IF iResult = 1 THEN
    boDone := TRUE;
END_IF;

IF NOT boDone THEN
    …some more code
END_IF;
```

2.8 Requirement 14: Read a variable written by another task once per cycle

Priority: High

Description: To avoid reading the same variable of one task in a cycle of another task more than once and therefore getting different results, objects of another task should only be read once in the beginning of a cycle. This prevents an unexpected change in value during the same cycle of program execution. Those variables should be treated as inputs or outputs.

Example:

```
DON'T
fbDoSomething(iValue := glb_iTask2Value);

…some more code

fbDoMore(iValue := glb_iTask2Value);
```

```
DO
tmp_iTask2Value := glb_iTask2Value;
fbDoSomething(iValue := tmp_iTask2Value);

…some more code

fbDoMore(iValue := tmp_iTask2Value);
```

2.10 Requirement 15: Usage of parameters shall match their declaration mode

Priority: High

Description: When using input, output, or in-out variables they must be used as declared. The following rules must be followed when using such variables:

- Each input variable must be read at least once in a cycle
- No input variable should be written
- Each output variable must be written once in a cycle
- Each in-out-variable should be either written or read once a cycle

Example:

```
DON'T
VAR INPUT
    i_iValue     : INT;
END_VAR

VAR OUTPUT
    o_boDone     : BOOL;
END_VAR

FUNCTION_BLOCK MYFB_FB
    i_iValue     := 10;
END_FUNCTION_BLOCK
```

```
DO
VAR INPUT
    i_iValue     : INT;
END_VAR

VAR OUTPUT
    o_boDone     : BOOL;
END_VAR

FUNCTION_BLOCK MYFB_FB
    IF i_iValue = 10 THEN
        o_boDone := TRUE;
    END_IF;
END_FUNCTION_BLOCK
```

2.11 Requirement 16: Function Block instances should only be called once

Priority: Medium

Description: It is advised to call function block instances only once per program cycle to keep code maintainable. During development by frequently using the copy and paste function the occasions of error increases which may lead to the use of an instance twice and therefore unwanted behavior within the code.

Exceptions: There may be scenarios where the call of the same instance could be efficient. If such a use of function block instances is well sophisticated it can be used.

2.12 Requirement 17: Define maximum Input/Output/In-Out variables for a POU

Priority: Medium

Description: To increase the readability of code a maximum amount of **10 Input/Output/In-Out variables** overall should not be exceeded for a single function block or function. If there is a need to exceed this amount the use of user defined structures to group information is advised.

2.13 Requirement 18: Do not declare variables that are not used

Priority: Medium

Description: During program development it may occur that removal of unused objects after removal of the same object in the code is not as important. Nevertheless, the removal of such unused object should be done at the latest before deployment. Unremoved and unused object take up unnecessary memory which can be avoided.

2.14 Requirement 19: Datatype conversion should be explicit

Priority: Medium

Description: If operations between different datatypes are used the conversion of the different datatypes should be done by the developer and not the compiler to avoid precision loss where precision is needed. The developer should keep that in mind, keep the original datatype as long as possible, and purposefully use conversions.

Example:

```
DON'T
iValue1 := REAL_TO_INT(rValue1);
iValue2 := REAL_TO_INT(rValue2);
iResult := (iValue1 + iValue2) / iValue3;
```

```
DO
iResult := REAL_TO_INT(rValue1 + rValue2) / iValue3;
```

2.15 Requirement 20: A global variable shall be written by only one program

Priority: Medium

Description: If global variables are used in different Programs each global variable should only be written by one program to keep an easy understanding of dataflow.

2.16 Requirement 21: Call timer outside of switch-cases or if-instructions

Priority: High

Description: Timer should always be called outside of switch-cases or if-instructions. Only the start condition and initial time value should be used inside a switch-case or if-instruction.

Example:

```
DON'T:
CASE iState OF
      0 :
            IF iHeight < iMaxHeight THEN
                  iState := 1;
            END_IF;

      1 :   (* move up *)
            T_Up(IN := TRUE, PT := T#1000MS);
            IF T_Up.Q THEN
                  (* do move up *)
                  iState := 0;
            END_IF;
END_CASE;
```

```
DO:
T_Up(IN := boUp, PT := T#1000MS);

CASE iState OF
      0 :
            IF iHeight < iMaxHeight THEN
                  boUp := TRUE;
                  iState := 1;
            ELSE
                  boUp := FALSE;
            END_IF;

      1 :   (* move up *)
            IF T_Up.Q THEN
                  (* do move up *)
                  iState := 0;
            END_IF;
END_CASE;
```

3 Language

3.1 Requirement 22: Indentation

Priority: Medium

Description: To keep the code readable especially for nested cases and loops an indentation level of 4 spaces or one Tab should be used for each case or loop.

Example:

```
DON'T
IF boEnable THEN
…some code
ELSE
IF boTest THEN
…some code
END_IF;
END_IF;
```

```
DO
IF boEnable THEN
    …some code
ELSE
    IF boTest THEN
        …some code
    END_IF;
END_IF;
```

3.2 Requirement 23: Loop variables should not be modified inside a FOR loop

Priority: Medium

Description: Modifying loop variables inside a FOR loop is forbidden. The FOR statement is used if the number of iterations can be determined in advance; otherwise, the WHILE or REPEAT constructs are used. Modifying during execution can cause unexpected behavior, including infinite loops, and can be difficult to debug and maintain.

Example:

```
DON'T
FOR i := 0 TO 100 DO
     IF a_array[i] = i_value THEN
          i := 100;
     END_IF;
END_FOR;
```

```
DO
WHILE i <= 100 AND a_array[i] <> i_value DO
     i := i + 1;
END_WHILE;
```

3.3 Requirement 24: Use parenthesis to explicitly express operation precedence

Priority: Medium

Description: When using operators with a similar precedence, like AND/OR/= or +/-, use parenthesis to clarify the intention of the code.

Example:

```
DON'T
IF A AND B OR C AND D THEN
… Do something
END_IF;
```

```
DO
IF (A AND B) OR (C AND D) THEN
… Do something
END_IF;
```

3.4 Requirement 25: Each IF instruction should have an ELSE clause

Priority: Low

Description: For every IF/SWITCH-CASE instruction in the code, an ELSE clause should be added to ensure that all cases are managed. The developer should always take into account what will happen if the condition is false.

Example:

```
DON'T
IF iValue = 1 THEN
      … Do something
ELSIF iValue = 5 THEN
      … Do something else
END_IF;
```

```
DO
IF iValue = 1 THEN
      … Do something
ELSIF iValue = 5 THEN
      … Do something else
ELSE
      … Handle unexpected values
END_IF;
```