



1. Purpose

The purpose of this document is to guide persons involved in developing software, so that good practices and standardization can be achieved to improve software quality and maintainability.

2. Scope

This document is intended to be a guideline. It was created to help explain general expectations and to make recommendations. The usage of this guideline applies to internal equipment developers and suppliers to Bosch who create custom software.

3. References

The following documents and references should be used for additional insight and guidance related to the purpose of this document:

- **Motor Industry Software Reliability Association (MISRA)** <http://www.misra.org.uk/>

4. Definitions

n/a


5. Goals

5.1 The main goals for establishing software coding guidelines are the following:

- **Reliability:** We expect that our software is developed to be reliable and robust. These guidelines ensure that good practices are used and common pitfalls are avoided.
- **Maintainability:** We expect that our software is easy to maintain. All members of the department will have a common understanding of these guidelines and can more easily follow the construction of software developed by others. Following ones own work after periods of inactivity is also improved.
- **Testability:** Being able to test software at a point prior to system integration requires that good design and architecture are used.
- **Debug-ability:** It is necessary to be able to trace and debug the software developed by others. Following these guidelines makes it easier to understand and follow through the structure and logic.

5.2 By following these guidelines it is expected that:

- Source code has consistent style and structure.
- Maintenance and future development costs are reduced.
- Source code is easier to read and understand.
- Source code quality and robustness are improved.
- Source code is less complex and easier to test.
- Developers can read and understand each others code more easily.
- Source code contains fewer defects.
- Source code is easier to maintain and correct.
- Learning curves for new developers are reduced.

 BOSCH AUTOMOTIVE GROUP	GUIDELINE SOFTWARE CODING GUIDELINE	GUIDELINE NO: FG4.7.03.078 Page 2 of 7
---	--	--

- Functionality can be traced back to requirements more easily.

6. Coding Guidelines

6.1. Modular software design

The expectations of modular software design are that related functions and operations are grouped together in a logical collective manner. Concepts which contribute to good modularity are those of cohesion, coupling, encapsulation, abstraction, information hiding, and object orientation.

6.1.1 Basic definitions:

- Cohesion is the degree of interaction within a module and should be maximized.
- Coupling is the degree of interaction between two or more modules and should be minimized.
- Encapsulation is the method of combining data structures and the operations to be performed on those data structures. Internal techniques, computations, and logic are contained within the module and do not need to be known in order to use the module.
- Abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time.
- Information Hiding is the hiding of design decisions in a program that are most likely to change, thus protecting other parts of the program from change if the design decision is changed.
- Object Orientation is a collection of methods and techniques to analyze, describe, and implement software. It encompasses several of the items already listed as well as inheritance, polymorphism, and the use of classes.

Note: There is a significant body of knowledge available on these topics and the reader is encouraged to research these in order to gain further insight.

6.2. Elimination of redundant code

Any code which is repeated more than one time shall be considered for optimization and packaging as a sub or function.

6.3. Use of global variables


The use of global variables shall be minimized and used only when necessary for system-wide information passing. A global variable can potentially be modified from anywhere, and any part of the program may depend on it. A global variable therefore has an unlimited potential for creating mutual dependencies, and adding mutual dependencies increases complexity.

6.4. Module level header

Every module / source file shall start with a descriptive header containing the following information:

- File name
- General description of contents
- Copyright
- History

The History of changes need not be described directly in the software or header. A separate change log

 BOSCH AUTOMOTIVE GROUP	GUIDELINE SOFTWARE CODING GUIDELINE	GUIDELINE NO: FG4.7.03.078 Page 3 of 7
---	--	--

can be maintained and a tab like Help/About may to be included in the software to capture the history. A history of changes must be maintained regardless of the method.

Example:

```

/*****
/* COPYRIGHT
/* Robert Bosch GmbH reserve all rights even in the event of industrial
/* property. We reserve all rights of disposal such as copying and passing
/* on to third parties.
*****
/* File name      : COMCAN_DataLinkLayer_COMMON.ESC
/* -----
/* Description    : Data Link Layer CAN-Software
/* -----
/* History
/* -----
/* Revision      : 1.19
/* Changed by    : mrz2si
/* Change date   : 12.Aug.2002 16:38:11
/* Changes       : Update Clear ES register after reinitialisation
/* Reasons       : Busoff Error passiv monitoring for customer X
/* -----

```

6.5. Sub/Function level header

Every sub and function shall start with a descriptive header containing the following information:

- Description of purpose
- Description of input parameters
- Description of output parameters and return values

Example:

```

/* Description:
Query the corresponding bit in the RMP-Register, if any message received
the bit is set. After function call the bit is cleared
RETURN : TRUE   if the bit is set (! 0)
        FALSE  if the bit is reset (= 0)
PARMS  : Connection number [0 .. sizeof (CONNECTION Table)]
Comments :
ULONG will be used as return type, because this produce shorter code.
*/

```

6.6. Comments

Comments shall be used throughout to describe key concepts and sections of code. Looping and logic constructs (do..while, if..then..else, etc.) shall have comments describing their purpose. Complex calculations and logic shall have comments describing the desired result.

Source code that has been commented out because it is no longer needed shall be considered for deletion. It may be kept in for one revision in case it needs to be conveniently reinstated.

In the .Net development environment comments should be formatted using XML tags for at least these tag types: Remarks, Parameters, and Summary. These XML comments can be used to create documentation and help files.



6.7. Revision Control

It is important to maintain the integrity of software source code using an appropriate method of revision control. File and program versioning must be applied as a minimum level of control. Changes to these files are identified by incrementing a primary or secondary revision number. A single number running version number (v01, v02, ...) or major.minor numbering (2.01, 2.02, 3.00, etc) may be used. The minor number is incremented when small changes, adjustments, or bug fixes are applied to a file/module. The major number is incremented when significant rework or requirements change.

6.8. Data types

All variables shall have their type declared before use. In Visual Basic the statement Option Explicit shall be used to ensure all variables are declared.

The following standard data types shall be used:

- Byte, Char
- Integer, word; signed and unsigned
- Long integer; signed and unsigned
- Float, Single
- Double
- String

Object types may be used when appropriate. Use of variant or indeterminate data types shall be avoided as much as possible. Declaration of the variable shall include a descriptive comment as to the need for declaring as such a type.

6.9. Variable naming convention

Standard naming convention shall be used for local variables, global variables, constants, parameters, and function names. The below definition is a recommended for naming conventions, but there can be deviations within the scope of the programming language being used. The naming conventions shall be described in the project document or source code. It is recommended that a prefix and suffix be used with every definition. The prefix shall describe the scope of the variable and the suffix shall describe the data type of the variable. Variable names must appropriately describe the purpose or content of the variable. Upper and lower case shall be used to make variables and function names more readable. Example: calculatePeakTorque()

Prefix definitions

- Local variable l_ (optional)
- Global variable g_
- Constants, #define All capitals
- Object variables txt, lbl, cmb, btn, frm, and others as appropriate

Suffix definitions (optional if development environment supports quick definition look-up, tool tips, or object browser)

Data type	Signed Suffix	Unsigned Suffix
Byte, Char, Boolean	n/a	_byt, _chr, _bln
Word, Integer	_int	_uint
Long Word, Long Integer	_lng	_ulng



Float, Single	_sng	n/a
Double Float	_dbl	n/a
String	n/a	_str

Simple 1 character variables i, j, and k may be used as char/integer loop counters.

6.10 Use of goto, continue, break

The inappropriate use of goto, continue, and break can produce unreadable and generally unmaintainable "spaghetti code", and therefore their use shall be avoided except in these cases.

- Goto may be used in the Visual Basic error trapping construct "On Error Goto ..."
- Break statements are required elements in the C/C++ switch/case construct
- Continue statements can always be replaced with a structured construct

6.11 Error trapping and recovery

If a function returns error codes they shall be evaluated and properly handled. Fault conditions must be anticipated and recovery or retry loops shall be used. Appropriate fault messages shall be presented to the user to assist in system operation. Generic error messages are to be avoided.

In the .Net development environment Try/Catch/Finally constructs shall be used for error trapping.

6.12 Process progress and status

Informational messages for task and process status shall be logged to the screen in a text or list box and also to a file in order to provide sufficient information as to the state of the system.

6.13 Coding layout

Code shall be indenting 3 spaces at every block statement (if, do, etc.). The length of any line shall be limited to 120 characters max, 80 characters preferred.

6.14 Abbreviated statements

Compact and abbreviated statements such as possible with C++ shall be avoided. All statements shall be written in a manner that makes them clearly understood and unambiguous.

Example: `array[i++] += --j;`


6.15 Use of TRUE and FALSE

Boolean constants TRUE and FALSE may only be used with Boolean variables. Do not mix Boolean and arithmetic operators. Binary (bit-wise) operations may not be performed with TRUE or FALSE.

- If (myvar == FALSE) {ok}
- If (!myvar) {ok but not recommended}
- If (Not myvar) {ok}
- If (~myvar) {Not ok, where ~ is binary NOT}

6.16 Arithmetic operators

In case of division the denominator shall be tested to be not equal zero. If it can be assured that zero is not possible so that the test can be omitted, it shall be justified sufficiently in the comments.

 BOSCH AUTOMOTIVE GROUP	GUIDELINE SOFTWARE CODING GUIDELINE	GUIDELINE NO: FG4.7.03.078 Page 6 of 7
--	--	--

6.17 Single point of exit from a sub/function

There should only be one point of exit from a sub or function and this shall be at the end of the function. If an exit from any other point is necessary it shall be sufficiently documented as to why it is necessary.

6.18 Function Size/Length


A single subroutine or function should not exceed more than 1 page or about 60 lines. Functions larger than this shall be considered for further decomposition in order to improve readability / maintainability.

6.19 Include Files

The use of include files shall be avoided. If they must be used they shall be maintained in a common directory. If a common directory is not possible the explicit locations must be sufficiently defined and described in comments.

7 Revision History

“If printed, it is the user’s responsibility to check that this document reflects the latest revision level”

 BOSCH AUTOMOTIVE GROUP	GUIDELINE SOFTWARE CODING GUIDELINE	GUIDELINE NO: FG4.7.03.078 Page 7 of 7
---	--	--

Revision Level	Description	Effective Date
2	Updates according to peer review with Sbd regarding location of source code history (6.4), relaxed variable naming (6.9), and mixing of boolean and arithmetic operators (6.15)	07/21/2009
1	Changed guideline format; updated acronym from AB/EDT to AE/ETC	07/21/2008
Original	Original Release	2/20/2007

8 Reviewed and Approved By:

Name / Acronym	Date reviewed
Mike Schwedt – RBNA/ETC4-NA	07/21/2009
Martin Boos – RBNA /ETC-NA	07/21/2009

“If printed, it is the user’s responsibility to check that this document reflects the latest revision level”