Coding Style Guide

[C Language Edition]

**Embedded Software Developers' Guide to Coding Practices for SQT Projects**

Version 1.0.0

# Revision History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Revision History | | Embedded Software Developers' Guide to Coding Practices for SQT Projects | | |  |  |
| **Rev.** | Date | Description | | |  |  |
| **Page** | | Summary | **Approval** | **Creation** |
| 1.0.0 | 2020/4/10 | - | Create new | | AnhLT74 | HaoNP2 |
|  |  |  |  | |  |  |

[Revision History 2](#_Toc168692614)

[はじめに 6](#_Toc168692615)

[1 文字コード・改行コード 7](#_Toc168692616)

[1.1 文字コード 7](#_Toc168692617)

[1.2 改行コード 7](#_Toc168692618)

[2 コーディングスタイルの統一 8](#_Toc168692619)

[2.1 波括弧（{ }）の位置 8](#_Toc168692620)

[2.2 字下げ（インデンテーション） 10](#_Toc168692621)

[2.3 空白の入れ方 11](#_Toc168692622)

[2.4 継続行における改行の位置 12](#_Toc168692623)

[2.5 変数の初期化 12](#_Toc168692624)

[2.6 グローバル変数使用時の注意 12](#_Toc168692625)

[2.7 暗黙の型変換禁止 12](#_Toc168692626)

[2.8 if 文における判断条件 13](#_Toc168692627)

[2.9 無条件のジャンプ禁止 13](#_Toc168692628)

[2.10 再帰処理禁止 13](#_Toc168692629)

[2.11 void 関数のreturn 文 13](#_Toc168692630)

[2.12 関数の途中 return の許可 13](#_Toc168692631)

[3 コメントの書き方の統一 15](#_Toc168692632)

[3.1 コメントの書き方 15](#_Toc168692633)

[3.2 特殊なケース 15](#_Toc168692634)

[3.3 コード部のコメントアウト禁止 17](#_Toc168692635)

[3.4 コメントブロック 18](#_Toc168692636)

[3.5 トレーサビリティ対応 21](#_Toc168692637)

[4 名前の付け方の統一 22](#_Toc168692638)

[5 ポインタの書き方の統一 23](#_Toc168692639)

[5.1 ポインタ 23](#_Toc168692640)

[5.2 ヌルポインタ 23](#_Toc168692641)

[6 ダミー関数の書き方の統一 24](#_Toc168692642)

[6.1 ダミーヘッダファイル 24](#_Toc168692643)

[6.2 ダミープログラムファイル 24](#_Toc168692644)

[7 ソフトウェアバージョン更新ルールの統一 25](#_Toc168692645)

[7.1 ソフトウェアバージョン 25](#_Toc168692646)

[7.2 ソースファイルバージョン 25](#_Toc168692647)

[8 機能安全対応 26](#_Toc168692648)

[8.1 処理系定義の文書化 26](#_Toc168692649)

[8.2 依存する処理系定義の動作を、以下に定義をする。 26](#_Toc168692650)

[8.3 コンパイルエラー・ワーニング検出 26](#_Toc168692651)

[8.4 エラーチェックの実施 27](#_Toc168692652)

[8.4.1 実行時エラーのチェック 27](#_Toc168692653)

[8.4.2 エラー情報を戻す関数のエラーチェック 29](#_Toc168692654)

[8.4.3 引数のエラーチェック 31](#_Toc168692655)

[8.5 アセンブリ言語使用禁止 33](#_Toc168692656)

[8.6 基本型使用禁 33](#_Toc168692657)

[8.7 関数・関数マクロ作成時の注意 34](#_Toc168692658)

[8.8 ポインタ使用時の注意 34](#_Toc168692659)

[8.9 多重インクルード禁止 34](#_Toc168692660)

[8.10 標準ライブラリ使用禁止 35](#_Toc168692661)

[8.11 動的なオブジェクト・メモリ確保禁止 35](#_Toc168692662)

[8.12 ファイルの操作禁止 36](#_Toc168692663)

# Introduction

This document defines a coding style guide aimed at standardizing the source code for automotive development. For coding regulations that ensure software safety, we use MISRA-C:2012, and this document mainly focuses on the style of writing source code, serving as a supplementary position to the aforementioned regulation.

This document defines SQT project unique coding style guide, using "Revised Edition: Coding Practices Guide for Embedded Software Developers [C Language Edition] Ver1.0"

## Character Encoding and Line Break Code

### Character Encoding

The character encoding for creating automotive program code shall be UTF-8.

### Line Break Code

The line break code for creating automotive program code shall be CR+LF.

## Unified Coding Style

### Position of Curly Braces ({ })

The position of curly braces should be standardized to make the beginning and end of blocks easier to see.

1. **Functions**

The opening brace should be placed on the same line as the function or on a separate line, and the closing brace should be placed on a separate line from the code.

Example:

|  |
| --- |
| void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo ) {  } |

Example:

|  |
| --- |
| void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo )  {  } |

1. **Control Statements**

The opening brace should be placed on the same line as the control statement, and the closing brace should be placed on a separate line from the code.

Example:

|  |
| --- |
| if ( NUM\_OF\_EVENT\_DATA == data ){  } |

1. **else Statement**

The else keyword and the opening brace should be placed on the same line as the closing brace of the control statement, and the closing brace should be placed on a separate line from the code.

Example:

|  |
| --- |
| if ( NUM\_OF\_EVENT\_DATA == data ) {  } else if ( MIN\_EVENT\_DATA == data ) {  } else {  } |

### Indentation

Indentation should be used to make declarations and blocks of code easier to read. The indentation should be 4 spaces (^^^^). Tabs should not be used because they can cause misalignment in different editors, so spaces should be used instead.

**Note:** "^" represents a half-width space

Example:

|  |
| --- |
| if ( 1 == data ) {  ^^^^p = &list[0];  } |

### Use of Spaces

Insert a single space to make the code easier to read and to help spot coding errors.

1. **Functions**

Example:

|  |
| --- |
| void⌃Dem\_GetVersionInfo(⌃Std\_VersionInfoType\*⌃versioninfo⌃)⌃{ |

1. **Control statement**

Example:

|  |
| --- |
| if⌃(⌃MAX\_DATA⌃==⌃data⌃)⌃{  } |

Example:

|  |
| --- |
| for⌃(⌃i=0;⌃i<max;⌃i++⌃)⌃{  } |

1. **Expression**

Example:

|  |
| --- |
| SEventStatusBuffer[i].faultDetectionCounter⌃=⌃0U; |

### Line Breaks in Continuation Lines

When an expression becomes long and readability is compromised, break the line at an appropriate position. However, to improve readability on the monitor display and when printing, if a line exceeds 100 characters, break the line at 100 characters.

Place the operator at the beginning of the continuation line.

The indentation of the continuation line after a line break should be adjusted with spaces (half-width spaces) to a position that enhances readability, and the number of indentation spaces is not specified.

**Note:** "^" represents a half-width space

Example:

|  |
| --- |
| dataNum = DemConfigData.FreezeFrameData.FreezeFrameDataNum  ⌃⌃⌃⌃⌃⌃⌃⌃+ DemConfigData.ExtendedData.ExtendedDataNum  ⌃⌃⌃⌃⌃⌃⌃⌃+ DemConfigData.ExtendedData.OtherDataNum; |

Example:

|  |
| --- |
| if (( SWC\_E\_SYSTEM\_POWER\_BROWNOUT == data1 )  ⌃&& ( SWC\_E\_SENSOR\_POWER\_BROWNOUT == data2 )) { |

### Variable Initialization

All variables must be initialized.

### Precautions When Using Global Variables

The use of global variables must be clearly identified, and their necessity and safety must be explained.

### Prohibition of Implicit Type Conversion

All type conversions must be explicit.

Example:

|  |
| --- |
| uint8 data; data = (uint8)1U; |

### Conditions in If Statements

When comparing constants and variables in if statements, write the constant on the left side and the variable on the right side, as shown below.

Example:

|  |
| --- |
| /\* MAX\_DATA: constant / data: variable \*/  if ( MAX\_DATA == data ) {  } |

When the condition expression in an if statement "if (variable == constant)" is mistakenly written as "if (variable = constant)", it is interpreted as an assignment to the variable and does not result in a compile error. This can delay the detection of bugs. By writing the condition expression in the if statement as "if (constant == variable)", a mistaken "if (constant = variable)" will result in a compile error, allowing the bug to be detected at compile time.

### Prohibition of Unconditional Jumps

Do not use goto statements or jump processing.

### Prohibition of Recursive Processing

Do not use recursive processing.

### Return Statements in void Functions

Do not write return statements in void functions. Writing return statements in void functions can sometimes cause the compiler to generate unexpected code.

Example:

|  |
| --- |
|  |

### Permission for Mid-Function Return

MISRA-C:2012 Rule 15.5 recommends that "a function should have a single exit point at the end of the function." This recommendation is based on the consideration that having multiple exit points in a function can cause control flow to be interrupted in the middle of processing or require different handling for each exit point, which can potentially induce bugs.

However, in cases where code becomes heavily nested (such as during configuration checks), readability can suffer, which in turn might induce bugs. Additionally, modularizing code to reduce nesting can impact execution speed.

Therefore, if there is a justification to prevent increased nesting or a corresponding reason, multiple exit points (returns) within a function are permitted but must be subject to review, and the results should be documented as evidence.

Example: /\* If configuration data 1-3 are out of range, return an error and exit processing \*/

|  |
| --- |
| /\* If configuration data 1-3 are out of range, return an error and exit processing \*/  if (((s\_config.data1 < MIN1) || (s\_config.data1 > MAX1))   || ((s\_config.data2 < MIN2) || (s\_config.data2 > MAX2))   || ((s\_config.data3 < MIN3) || (s\_config.data3 > MAX3))) {      return (E\_NOT\_OK);  }  /\* If configuration data 4-5 are out of range, return an error and exit processing \*/  if (((s\_config.data4 < MIN4) || (s\_config.data4 > MAX4))   || ((s\_config.data5 < MIN5) || (s\_config.data5 > MAX5))) {      return (E\_NOT\_OK);  }  /\* Normal processing \*/  ...  return (E\_OK); |

## Standardization of Comment Writing

### How to Write Comments

To improve the readability of the source code, write comments as follows:

1. **Write comments on the line preceding the source code in the same column, independent of the source code.**

For long source code lines or control statements, it is better for readability to have comments on a separate line.

Example:

|  |
| --- |
| /\* check DTC par event \*/  if ( dtc == eventdata.dtc ) {  } |

Example:

|  |
| --- |
| /\* check EventId  \* NOTE: 0xFFFF is not valid  \*/  if ( NO\_EVENT\_ID != eventId ) {  } |

1. **Write comments on the same line as the source code, following the code.**

For data assignments or local variable declarations, it is better for readability to write comments after the source code.

Example:

|  |
| --- |
| dtc = eventdata.dtc; /\* set DTC \*/ |

### Special Cases

For cases where it is difficult to determine whether it is a coding mistake, comments are mandatory.

1. When not writing break in a switch case statement

Provide a comment explaining why break is not needed. Generally, break should be written.

1. When there is no processing in an if statement

Provide a comment explaining why processing is not needed.

1. #endif

Include a comment indicating which #if it corresponds to.

### Prohibition of Commenting Out Code Sections

Leaving unused code sections commented out in the source file makes the code harder to read. Therefore, do not comment out parts of the code. If it is necessary to disable a code section, use "#if 0".

[MISRA-C:2012 Dir] 4.4 (recommended)

The following is an example of prohibited code commenting:

Example:

|  |
| --- |
| /\* if ( TRUE == flag ) { \*/ |

If you want to leave a code section in the source file, use "#if 0" as shown below.

Example:

|  |
| --- |
| #if 0  If ( TRUE == flag ) {  }}  #endif |

### Comment Blocks

1. **Header Comment Block**

At the beginning of the source code, include a description of the source code using the following format.

Example:

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  /\* Copyright : 2014 SCSK Corporation \*/①  /\* System Name : AUTOSAR BSW 16bit \*/②  /\* File Name : Dem.c \*/③  /\* Version : v1.00.00 \*/④  /\* Contents : The service component Diagnostic Event Manager (Dem) is \*/⑤  /\* responsible for processing and storing diagnostic events \*/  /\* (errors) and associated data. Further, the Dem provides \*/  /\* fault information to the Dcm (e.g. read all stored DTCs from \*/  /\* the event memory). The Dem offers interfaces to the \*/  /\* application layer and to other BSW modules. \*/  /\* \*/  /\* The basic target of the Dem specification document is to \*/  /\* define the ability for a common approach of “diagnostic \*/  /\* fault memory” for automotive manufacturers and component \*/  /\* suppliers. \*/  /\* \*/  /\* This specification defines the functionality, API and the \*/  /\* configuration of the AUTOSAR basic software module \*/  /\* Diagnostic Event Manager (Dem). Parts of the internal \*/  /\* behavior are manufacturer specific and described in the \*/  /\* Limitations chapter. \*/  /\* Author : t.sasaki \*/⑥  /\* Note : \*/⑦  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

① Copyright holder and the initial year of publication (if revised in 2015, write as "2014-2015")

② System name, product name

③ File name

④ Version of the file

Refer to "8.2 Software Version of Source Files" for details

⑤ Description of the file

⑥ Author's name

⑦ Remarks (content is optional)

1. Footer Comment Block

At the end of the source code, include a comment indicating the end of the source code using the following format.

Example:

|  |
| --- |
| /\* EOF Dem.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

1. Function Comment Block

At the beginning of a function, include a description of the function using the following format.

Example:

|  |
| --- |
|  |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| /\*  /\*  /\*  /\* | ModuleID  ServiceID  Name Param | : MODULE\_ID\_DEM (054)  : DEM\_INIT\_ID (0x02)  : Dem\_Init  : (in) ConfigPtr Pointer to the configuration set | in | \*/①  \*/➁  \*/③  \*/④ |
| /\* |  | VARIANT-POST-BUILD. |  | \*/ |
| /\* | Return | : void |  | \*/⑤ |
| /\* | Contents | : Initializes or reinitializes this module. |  | \*/⑥ |
| /\* | Author | : t.sasaki |  | \*/⑦ |
| /\* | Note | : [SWS\_Dem\_00181] |  | \*/⑧ |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

Ⓒ モジュール名、およびモジュール ID

Ⓒ サービス名、およびサービス ID

③ 関数名

④ 関数パラメータ（in/out/inout を明記すること）

⑤ 戻り値

⑥ 当該関数の説明

⑦ 当該関数の作成者名

⑧ 備考（SRS/SWS Item 等）

1. 変更履歴

ヘッダコメントブロック、および関数コメントブロックに、以下のフォーマットで変更履歴を記述する。

※履歴は、ファイルのマイナー及びパッチバージョンが更新されたタイミングで記載する。ファイルメジャーバージョ ンが更新されたとき、新規作成されるため履歴は残らない。

ヘッダコメントブロックには、日付とファイルバージョン、変更した関数名（もしくは変数名）を記述する。

/\* History : 2014.04.01 v1.00.00 t.sasaki Dem\_ReportErrorStatus \*/

関数コメントブロックには、変更理由を記述する。

|  |  |  |  |
| --- | --- | --- | --- |
| /\* | History | : 2014.04.01 v1.00.00 t.sasaki | \*/ |
| /\* |  | Fixed problem the return code is not returned | \*/ |
| /\* |  | correctly. | \*/ |

### Traceability Compliance

To confirm whether the Software Requirements Specification (SRS) or AUTOSAR requirements (SWS) are linked to the API, a traceability matrix must be documented. 【MISRA-C:2012 Dir】 3.1 (Required)

Additionally, based on the created traceability matrix, requirement IDs need to be embedded in the source code.

1. Function Comment Block

If requirement IDs are defined for the function itself, the requirement ID should be noted in the "Note" section of the function comment block. The requirement ID should be enclosed in brackets [] for extraction by tools.

Example:

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  ：  ：  /\* Note : [SWS\_Dem\_00181] \*/  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/ |

1. Code Comments

If requirement IDs are defined for specific functionalities, the requirement ID should be noted in the comments of the code implementing that functionality. The requirement ID should be enclosed in brackets [] for extraction by tools.

Example:

|  |
| --- |
| /\* [SWS\_Dem\_00181] \*/  /\* check EventId \*/  if ( NO\_EVENT\_ID != eventId ) { |

## Standardization of Naming Conventions

Establish naming conventions for variables, constants, etc., within the project and ensure uniformity in naming. Also, make sure that identifiers within the same namespace (e.g., "label names", "tags of structures, unions, and enumerations", "members of structures or unions", and "all other identifiers") are distinguishable from each other.

For Latin alphabet identifiers, ensure that there are no identifiers with different combinations of the following: [MISRA-C:2012 Dir] 4.5 (recommended)

**Table 1** Identifiers That Are Not Distinguishable

|  |  |  |
| --- | --- | --- |
| 区別がつかない識別子の組み合わせ | 例 | |
| 小文字/大文字 | 小文字 | returnCode |
| 大文字 | ReturnCode |
| アンダースコアの有/無 | アンダースコア有り | returnCode |
| アンダースコア無し | return\_Code |
| 文字の「O」/数字の「0」 | 文字の「O」 | ChannelO |
| 数字の「0」 | Channel0 |
| 文字の「I」/数字の「1」 | 文字の「I」 | counter\_I |
| 数字の「1」 | counter\_1 |
| 文字の「I」/「l」(エル) | 文字の「I」 | counter\_I |
| 文字の「l」(エル) | counter\_l |
| 文字の「l」(エル)/数字の「1」 | 文字の「l」(エル) | counter\_l |
| 数字の「1」 | counter\_1 |
| 文字の「S」/数字の「5」 | 文字の「S」 | counter\_S |
| 数字の「5」 | counter\_5 |
| 文字の「Z」/数字の「2」 | 文字の「Z」 | counter\_Z |
| 数字の「2」 | counter\_2 |
| 「n」(エヌ)/「h」(エイチ) | 文字の「n」(エヌ) | counter\_n |
| 数字の「h」(エイチ) | counter\_h |
| 文字の「B」/数字の「8」 | 文字の「B」 | counter\_B |
| 数字の「8」 | counter\_8 |
| 文字の並び「rn」(r とそれに続くn)  /文字の「m」(エム) | 文字の並び「rn」(rとそれに続くn) | Dcrn |
| 文字の「m」(エム) | Dcm |

## Standardization of Pointer Writing

### Pointers

Declare pointers without adding a space after the type.

Example:

|  |
| --- |
| char\*⌃p; |

### Null Pointer

Write null pointers as "NULL". Do not use "NULL" for anything other than null pointers.

Example:

|  |
| --- |
| char\* p; p = NULL; |

## Standardization of Writing Dummy Functions

When implementing a program, there may be instances where you need to implement function (API) calls for modules that are not yet implemented. The following defines how to code in such cases.

### Dummy Header File

To compile code that calls functions (APIs) of unimplemented modules, create a dummy header file (module\_name.h) for the unimplemented module, and declare the functions (APIs) there. Do not write the declarations of functions (APIs) in the header files or source files of the module you are trying to compile.

Example:

|  |
| --- |
| void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo );  Std\_ReturnType NvM\_RestorePRAMBlockDefaults( NvM\_BlockIdType BlockId ); |

### Dummy Program File

To link (build) code that calls functions (APIs) of unimplemented modules, create a dummy source file (module\_name.c) for the unimplemented module, and implement the bodies of the functions (APIs) there. Do not write the bodies of functions (APIs) in the header files or source files of the module you are trying to link.

The bodies of the functions (APIs) should be empty (no processing). For functions (APIs) with return values, return a fixed normal value (such as E\_OK). For functions (APIs) that return set values through return values or arguments, you can either return fixed values within the valid range or not set anything.

Example:

|  |
| --- |
| void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo ) {}  Std\_ReturnType NvM\_RestorePRAMBlockDefaults( NvM\_BlockIdType BlockId ) {  return E\_OK;  } |

## Standardization of Software Version Update Rules

The rules for updating software versions are defined as follows.

### Software Version

Define and manage the software version within the project.

### Source File Version

Manage the version of each source file according to the following rules, and include it in the header comment block. (Refer to ④ in "(1) Header Comment Block" of "4.3 Comment Blocks")

v1. 00. 00

Patch Version

Minor Version

Major Version

* Major Version

Update when adding new features.

* Minor Version

Update for incompatible changes such as modifications to functions involving changes to the interface (IF). Set to "0" when the major version is updated.

* Patch Version

Update for compatible changes such as modifications to functions not involving changes to the interface (IF). Set to "0" when the major or minor version is updated.

**Note**: The version of this source file should be updated for each file (this is different from the software version)

## Functional Safety Compliance

The guidelines and rules of MISRA-C:2012 related to functional safety compliance are defined as follows.

### Documentation of System Definitions

The behavior of system definitions on which program output depends must be documented and understood. Therefore, this will be described in this section.

[MISRA-C:2012 Dir] 1.1 (Required)

### The behavior of dependent system definitions is defined as follows:

* + - C Language Standard

Targeting C99.

### Detection of Compile Errors and Warnings

Compile errors and warnings must not be detected in any source files.

Depending on the functionality being implemented, fixing warning points might cause issues that prevent the intended purpose from being achieved. In such cases, deviations should be made, and the reasons for the exceptions should be clearly stated in the release notes. 【MISRA-C:2012 Dir】 2.1 (Required)

### Error Checking Implementation

#### Runtime Error Checking

Perform error checks at places where runtime errors are likely to occur.

MISRA-C:2012 Dir】 4.1 (Required)

Below are examples of locations where error checks should be performed.

* + - * Consider overflow

Example:

|  |
| --- |
| /\* Left side: buffer size, Right side: actual data length \*/  /\* Check if the actual data is larger than the buffer size \*/  if ( s\_Dlt\_MBuff\_RemainSize < ( message\_length + header\_length + 1U ) ) {      /\* Error handling \*/  } |

* + - * Guard against division by zero

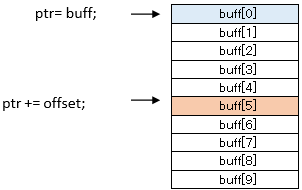
Example:

|  |
| --- |
| /\* data: denominator, Division: quotient \*/  /\* Check if the denominator is 0 \*/  if ( 0U == data ) {      /\* If the denominator is 0, do not perform the calculation \*/      Division = 0;  } else {      /\* Perform the calculation only if the denominator is not 0 (normal processing) \*/      Division = 100 / data;  } |

* + - * For pointer arithmetic, ensure that the results are as intended by reviewing the code to confirm that the intended operations can be performed correctly.

Example:

|  |
| --- |
| uint8 buff[10];  uint8 offset = 5;  uint8\* ptr = buff;  ptr += offset; |



In this case, during code review, it is necessary to confirm that ptr correctly points to buff[5] as intended by the operation.

* + - * Check the parameters of functions

For details, refer to '9.3.3 Argument Error Checking.

* + - * Do not access NULL pointers

Example:

|  |
| --- |
| /\* Check if the address is not NULL when using a pointer \*/  if ( NULL == PduInfoPtr ) {      /\* Error handling \*/  } |

* + - * Ensure that array indices are not negative or out of range

Below is an example of checking to ensure that array indices are not negative.

Example:

|  |
| --- |
| uint8 buff[200];  /\* Check that the index is 0 or above \*/  for ( i = (sint8)100 ; i >= 0 ; i-- ) {      /\* Normal processing \*/      buff[i] = TRUE;  } |

Below is an example of checking to ensure that array indices are not out of range.

Example:

|  |
| --- |
| /\* CANIF\_Q\_NUM\_OF\_TRCV: Number of elements in the array \*/  CanIf\_TrcvInfoType s\_TrcvInfo[CANIF\_Q\_NUM\_OF\_TRCV];  /\* Check that the index is less than the number of elements \*/  for ( i = (uint32\_least)0x0U; i < CANIF\_Q\_NUM\_OF\_TRCV; i++ ) {      /\* Normal processing \*/      s\_TrcvInfo[i].WakeupValidationCtrl = FALSE;  } |

* + - * 動的なメモリ確保をしない

詳細は『9.10 動的なメモリ確保禁止』を参照のこと

#### Error Checking for Functions that Return Error Information

If a function returns error information, it is considered that an error might occur internally. Therefore, when calling a function that returns error information, proper checks of the output values should be performed. 【MISRA-C:2012 Dir】 4.7 (Required)

Below are examples of proper error checking for return values.

Example:

|  |
| --- |
|  |

/\* Func1 の戻り値はエラー情報を含む(= E\_NOT\_OK) \*/ Std\_ReturnType Func1 ( uint8 Num );

void main( void ) { uint8 num = 0x01U;

Std\_ReturnType retCode = E\_OK;

retCode = Func1( num );

/\* 関数 Func1 の戻り値がエラー情報かどうかのチェック \*/ if ( E\_NOT\_OK == retCode ) {

/\* 異常処理 \*/

}

/\* Func2 の戻り値はポインタ \*/ uint8\* Func2 ( uint8 Num );

void main( void ) { uint8 num = 0x01U;

uint8\* retPtr = NULL;

retCode = Func2( num );

/\* 関数 Func2 の戻り値のアドレスが NULL かどうかのチェック \*/ if ( NULL == retCode ) {

/\* 異常処理 \*/

}

}

引数の適切な判定を行っている例を以下に示す。

/\* Func3 の引数のポインタにエラー情報を含む\*/ void Func3 ( uint8\* Status );

void main( void ) { uint8 status; Func3( &status );

/\* 関数 Func3 の引数のポインタがエラー情報かどうかのチェック \*/ if ( XXX\_ERROR\_STATUS == status ) {

/\* 異常処理 \*/

}

/\* Func4 の引数のポインタアドレスが NULL の場合\*/ void Func4 ( uint8\* Ptr );

void main( void ) { uint8\* ptr = NULL; Func4( &ptr );

/\* 関数 Func4 の引数のポインタアドレスが NULL かのチェック \*/ if ( NULL == ptr ) {

/\* 異常処理 \*/

}

#### 引数のエラーチェック

各 API の処理の始めに、入力値の妥当性チェックを行う（内部関数は除く）。

【MISRA-C:2012 Dir】 4.14 (必要)

1. 入力値の妥当性を閾値で確認する場合

void Func1 ( uint8 Number) {

/\* 引数が閾値を越えていないかのチェック \*/ if ( NUM\_OF\_CONFIG < Number ) {

/\* 異常処理 \*/

}

1. 入力値のポインタが NULL でないか確認する場合void Func2 ( uint8\* Ptr) {

/\* 引数が NULL でないかのチェック \*/ if ( NULL == Ptr) {

/\* 異常処理 \*/

}

1. 入力値が列挙型の場合

列挙型に設定されていない数値で、関数をコールする場合のためにエラーチェックが必要である。

typedef enum { BSWM\_FALSE = ( 0U ), BSWM\_TRUE, BSWM\_UNDEFINED

} BswM\_Q\_RuleStateType;

void Func2 ( BswM\_Q\_RuleStateType Status ) { switch ( Status ) {

case BSWM\_FALSE:

/\* 処理 \*/ break;

case BSWM\_TRUE:

/\* 処理 \*/ break;

case BSWM\_UNDEFINED:

/\* 処理 \*/ break;

default:

/\* 引数が宣言されている列挙型以外の場合の異常処理 \*/ break;

}

### Prohibition of Using Assembly Language

Assembly language should not be used as it is dependent on the compiler.

If it is absolutely necessary to use assembly language, the files using assembly language should be isolated in separate files from the C language files to clearly identify the parts that need to be rewritten during porting (prohibition of inline assembly). 【MISRA-C:2012 Dir】 4.3 (Required)

Additionally, when using assembly language, the compiler dependency and the reasons for using assembly language must be documented as comments in the source file. 【MISRA-C:2012 Dir】 4.2 (Recommended)

### Prohibition of Using Basic Types

The size of types may vary depending on the system. Therefore, basic types (such as "int") should not be used directly. Instead, use typedefs (such as "uint8") that clearly indicate the size and whether the type is signed or unsigned. 【MISRA-C:2012 Dir】 4.6 (Recommended)

Below are specific examples of types on a 32-bit machine.

Table 2: Specific Examples of Types on a 32-bit Machine

|  |  |  |  |
| --- | --- | --- | --- |
| typedef | 型名 | typedef | 型名 |
| unsigned char | uint8 | unsigned long | uint8\_least |
| unsigned short | uint16 | unsigned long | uint16\_least |
| unsigned long | uint32 | unsigned long | uint32\_least |
| unsigned long long | uint64 | signed long | sint8\_least |
| signed char | sint8 | signed long | sint16\_least |
| signed short | sint16 | signed long | sint32\_least |
| signed long | sint32 | float | float32 |
| signed long long | sint64 | double | float64 |

**Note:** Align with the ECU being used. For loop variables, use the type defined as least to optimize processing speed. (Defined in SWS\_Platform)

### Caution When Creating Functions and Function Macros

When using functions or function-like macros, consider whether functions or function-like macros are more appropriate for the specific processing.

【MISRA-C:2012 Dir】 4.9 (Recommended)

Ensure the code complies with the following guidelines:

* If the function is called frequently, functions are more suitable than function macros.
* If speed is a priority, function macros are more suitable than functions. *Specific values will be considered separately.*

### Caution When Using Pointers

When using pointers, it must be clear what type of pointer variables are being used, and the necessity and safety must be explainable.

Additionally, it is necessary to consider whether it is better to declare an incomplete type pointer or a complete type pointer. 【MISRA-C:2012 Dir】 4.8 (Recommended)

Below are examples of incomplete and complete type pointers.

Example:

|  |
| --- |
| struct ImpfType; /\* Incomplete type \*/  struct ImpfType\* ptr1; /\* Pointer to an incomplete type = Incomplete type pointer \*/  uint8\* ptr2; /\* Pointer to a complete type = Complete type pointer \*/ |

When an address is assigned to the above pointers, an incomplete type pointer (ptr1) cannot reference the members of the assigned address, whereas a complete type pointer (ptr2) can reference the members of the assigned address. By using an incomplete type pointer only for passing addresses, the type can be hidden from external access.

### Prohibition of Multiple Includes

Multiple includes may result in undefined or incorrect behavior; therefore, multiple includes are prohibited. Accordingly, it must be ensured that multiple includes are prevented. However, multiple includes are allowed when used for memory mapping. 【MISRA-C:2012 Dir】 4.10 (Required)

Below is an example of preventing multiple includes in a header file.

Example:

|  |
| --- |
| #ifndef BSWM\_H  #define BSWM\_H  /\* Processing \*/  #endif /\* BSWM\_H \*/ |

Below is an example of an exception where multiple includes are used for memory mapping.

Example:

|  |
| --- |
| #define BSWM\_START\_SEC\_VAR\_INIT\_LOCAL\_8  #include "BswM\_MemMap.h"  static bool\_t s\_BswMInit[ BSWM\_Q\_P\_NUM ] = {FALSE};  #define BSWM\_STOP\_SEC\_VAR\_INIT\_LOCAL\_8  #include "BswM\_MemMap.h" |

### Prohibition of Standard Library Usage

The standard library shall not be used. Therefore, standard library header files shall not be included.

【MISRA-C:2012 Dir】 4.11 (Required)

【MISRA-C:2012 Rule】 22.1, 22.3 (Required)

### Prohibition of Dynamic Object Memory Allocation

All objects shall be statically allocated, ensuring no objects are created only for a specific period. Dynamic memory allocation shall not be performed.

【MISRA-C:2012 Dir】 4.12 (Required)

The following are prohibited, and it must be confirmed during code review that no dynamic memory allocation is used:

* Use of standard library functions for dynamic memory allocation. Representative examples: calloc, malloc, realloc, and free functions.
* Custom dynamic memory allocation (including third-party packages) that does not use standard library functions.

### Prohibition of File Operations

No operations shall be performed on files.

【MISRA-C:2012 Dir】 4.13 (Recommended)

【MISRA-C:2012 Rule】 22.3 (Required), 22.4, 22.6 (Mandatory))

The following operations are prohibited:

* Resource allocation (e.g., opening a file).
* Resource deallocation (e.g., closing a file).
* Other operations (e.g., reading from a file).

— 以上―