


コーディングスタイルガイド

[C 言語版]



SCSK 株式会社
モビリティシステム事業部門
モビリティシステム第一事業本部
システム第三部 第一課

発行日：2020 年 05 月 25 日

●改訂履歴

版	変更日	変更内容・理由	承認	作成
1.0.0	2020/4/14	・新規作成	森谷	藤田
1.0.1	2020/5/25	・「3.5 変数の初期化」を追加 ・「3.6 グローバル変数使用時の注意」を追加 ・「3.7 暗黙の型変換禁止」を追加 ・「3.9 無条件のジャンプ禁止」を追加 ・「3.10 再起処理禁止」を追加 ・「9.7 ポインタ使用時の注意」に注意事項を追加 ・「9.10 動的なメモリ確保禁止」のタイトルを「9.10 動的なオブジェクト・メモリ確保禁止」に変更し、内容にオブジェクトの記載を追記。	森谷	藤田

●目次

1 はじめに	5
2 文字コード・改行コード	6
2.1 文字コード	6
2.2 改行コード	6
3 コーディングスタイルの統一	7
3.1 波括弧({ })の位置	7
3.2 字下げ(インデンテーション)	8
3.3 空白の入れ方	9
3.4 継続行における改行の位置	10
3.5 変数の初期化	10
3.6 グローバル変数使用時の注意	10
3.7 暗黙の型変換禁止	10
3.8 if 文における判断条件	11
3.9 無条件のジャンプ禁止	11
3.10 再帰処理禁止	11
3.11 void 関数の return 文	11
3.12 関数の途中 return の許可	12
4 コメントの書き方の統一	13
4.1 コメントの書き方	13
4.2 特殊なケース	13
4.3 コード部のコメントアウト禁止	14
4.4 コメントブロック	15
4.5 トレーサビリティ対応	17
5 名前の付け方の統一	18
6 ポインタの書き方の統一	19
6.1 ポインタ	19
6.2 ヌルポインタ	19
7 ダミー関数の書き方の統一	20
7.1 ダミーヘッダファイル	20
7.2 ダミープログラムファイル	20
8 ソフトウェアバージョン更新ルール of the 統一	21
8.1 ソフトウェアバージョン	21
8.2 ソースファイルバージョン	21
9 機能安全対応	22
9.1 処理系定義の文書化	22
9.2 コンパイルエラー・ワーニング検出	22
9.3 エラーチェックの実施	23
9.3.1 実行時エラーのチェック	23
9.3.2 エラー情報を戻す関数のエラーチェック	25
9.3.3 引数のエラーチェック	26

9.4 アセンブリ言語使用禁止

9.5 基本型使用禁止

9.6 関数・関数マクロ作成時の注意

9.7 ポインタ使用時の注意

9.8 多重インクルード禁止

9.9 標準ライブラリ使用禁止

9.10 動的なオブジェクト・メモリ確保禁止

9.11 ファイルの操作禁止

28

28

29

29

30

30

31

31

1 はじめに

本ドキュメントは、車載開発のソースコードの標準化を目的とした、コーディングスタイルガイドを定義する。

なお、ソフトウェアの安全性を確保するためのコーディング規約としては MISRA-C:2012 を利用し、これを補完する位置づけとして、本ドキュメントでは主にソースコードの書き方に着目している。

本ドキュメントは、「改訂版 組込みソフトウェア開発者向け コーディング作法ガイド[C 言語版] Ver1.1」(独立行政法人 情報処理推進機構 ソフトウェア・エンジニアリング・センター編)の「保守性 4 統一した書き方にする。」をテンプレートとして、SCSK 固有のコーディングスタイルガイドを定義する。

2 文字コード・改行コード

2.1 文字コード

車載プログラムコードを作成するための文字コードは、UTF-8 とする。

2.2 改行コード

車載プログラムコードを作成するための改行コードは、CR+LF とする。

3 コーディングスタイルの統一

3.1 波括弧（{ }）の位置

波括弧の位置は、ブロックの始まりと終わりを見やすくするために統一する。

(1) 関数

開き括弧は関数と同じ行、または独立した行に、閉じ括弧は処理とは独立した行に記述する。

```
void Dem_GetVersionInfo( Std_VersionInfoType* versioninfo ) {  
}
```

```
void Dem_GetVersionInfo( Std_VersionInfoType* versioninfo )  
{  
}
```

(2) 制御文

開き括弧は制御文と同じ行に、閉じ括弧は処理とは独立した行に記述する。

```
if ( NUM_OF_EVENT_DATA == data ) {  
}
```

(3) else 文

制御文の閉じ括弧と同じ行に、else と開き括弧を記述し、閉じ括弧は処理とは独立した行に記述する。

```
if ( NUM_OF_EVENT_DATA == data ) {  
} else if ( MIN_EVENT_DATA == data ) {  
} else {  
}
```

3.2 字下げ（インデントーション）

宣言や処理のまとまりを見やすくするために字下げを行う。

字下げは空白(半角スペース)4 つとする。タブを使うとエディタによって表示がずれるケースがあるため、空白(半角スペース)を使うこととする。

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

※「^」が半角スペースを示す

```
switch ( type ) {  
case 1:  
    ^^^^data = 1;  
    ^^^^break;  
case 2:  
    ^^^^data = 2;  
    ^^^^break;  
default:  
    ^^^^break;  
}
```

※「^」が半角スペースを示す

※「case」は字下げしない

3.3 空白の入れ方

コードを見やすくし、また、コーディングミスを発見しやすくするために空白(スペース)1 つを挿入する。

(1) 関数

```
void Dem_GetVersionInfo(^Std_VersionInfoType*^versioninfo)^{  
    ※「^」が半角スペースを示す
```

(2) 制御文

```
if(^MAX_DATA^==^data^){  
    ※「^」が半角スペースを示す
```

```
for(^i=0;^i<max;^i++)^{  
    ※「^」が半角スペースを示す
```

(3) 式

```
SEventStatusBuffer[i].faultDetectionCounter^=^0U;  
    ※「^」が半角スペースを示す
```

3.4 継続行における改行の位置

式が長くなり可読性が悪くなったとき、適当な位置で改行する。ただし、コードのモニタ画面表示時の見易さや、印刷時の可読性を良くするために、1 行の文字数が 100 文字を超えた場合は、100 文字で改行する。

演算子は継続行の先頭に記述する。

改行した後の継続行の字下げは、空白 (半角スペース) で見やすくなる位置に調整するものとし、字下げ数は規定しない。

```
dataNum = DemConfigData.FreezeFrameData.FreezeFrameDataNum
^^^^^^ + DemConfigData.ExtendedData.ExtendedDataNum
^^^^^^ + DemConfigData.ExtendedData.OtherDataNum;
```

※「^」が半角スペースを示す

```
if ( ( SWC_E_SYSTEM_POWER_BROWNOUT == data1 )
^&& ( SWC_E_SENSOR_POWER_BROWNOUT == data2 ) ) {
```

※「^」が半角スペースを示す

3.5 変数の初期化

全ての変数は初期化を行う。

3.6 グローバル変数使用時の注意

使用するグローバル変数が明らかになっており、必要性和安全性が説明できること。

3.7 暗黙の型変換禁止

型変換は全て明示的に行っていること。

```
uint8    data;
data = (uint8)1U;
```

3.8 if 文における判断条件

定数と変数を比較する if 文の条件は、以下のように左辺に定数を、右辺に変数を記述する。

```
/* MAX_DATA:定数/data:変数 */  
if ( MAX_DATA == data ) {
```

if 文の条件式「if (変数 == 定数)」を「if (変数 = 定数)」と誤記しても、変数への代入とみなされて、コンパイルエラーとならない。このため不具合の検出が遅れる可能性がある。if 文の条件式を「if (定数 == 変数)」と記述することで、「if (定数 = 変数)」と誤記したときにコンパイルエラーとなり、不具合をコンパイル時に発見することができる。

3.9 無条件のジャンプ禁止

goto 文やジャンプ処理を使用しない。

3.10 再帰処理禁止

再帰処理を使用しない。

3.11 void 関数の return 文

void 関数には return 文を記述しない。

void 関数に return 文を記述すると、コンパイラによっては予測しないコードを出力することがあるため。

3.12 関数の途中 return の許可

MISRA-C:2012 ルール 15.5 にて「関数では、関数の最後に唯一の出口がなくてはならない」ことが推奨されている。

これは、関数の出口が複数あると、制御フローが処理の途中で中断されたり、出口ごとに処理が必要であったりしたときに、不具合を誘発する可能性を想定していることによる。

しかし、コードのネストが多くなってしまうような場合 (コンフィグチェック時など) は、可読性が悪くなって逆に不具合を誘発する可能性がある。またネストを減らすために処理を関数化すると、実行速度に影響が出る。

このためネストの増加を抑止したり、それに相当する理由がある場合は、関数の途中での return (複数の出口) を許可するがレビュー対象とし、結果をエビデンスとして残す。

```
/* 設定データ 1～3 が範囲外の場合はエラーを返して処理を抜ける */
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 );
}

/* 設定データ 4～5 が範囲外の場合はエラーを返して処理を抜ける */
if ((( s_config.data4 < MIN4 ) || ( s_config.data4 > MAX4 ))
    || (( s_config.data5 < MIN5 ) || ( s_config.data5 > MAX5 ))) {
    return( E_NOT_OK );
}

/* 正常処理 */
. . .
return ( E_OK );
```

4 コメントの書き方の統一

4.1 コメントの書き方

ソースコードの可読性を向上させるために、以下のようにコメントを記載する。

- (1) ソースコードと同じカラムの前行に、ソースコードと独立してコメントを記載する

ソースコードが長い場合や、制御文の場合などは、コメント行を独立させた方が可読性が良い。

```
/* check DTC par event */  
if ( dtc == eventdata.dtc ) {  
  
/* check EventId  
 * NOTE: 0xFFFF is not valid  
 */  
if ( NO_EVENT_ID != eventId ) {
```

- (2) ソースコード同じ行の後ろに、コメントを記載する

データの代入や、ローカル変数の宣言などは、ソースコードの後にコメントを記載した方が可読性が良い。

```
dtc = eventdata.dtc;    /* set DTC */
```

4.2 特殊なケース

コーディングミスかどうかの判断がつきづらいケースに関しては、コメントの記載を必須とする。

- (1) switch case 文で break を書かない場合

break が必要でない理由をコメントで記載すること。

基本的に break は記述する。

- (2) if 文で処理がない場合

処理が必要でない理由をコメントで記載すること。

- (3) #endif

どの#ifに対応しているのかをコメントで記載すること。

4.3 コード部のコメントアウト禁止

使用しないコード部をコメントアウトしてソースファイルに残すことは、コードを読みにくくするため、コードの一部をコメントアウトしてはいけない。

コード部の無効化が必要な場合は、「#if 0」を使用する。

【MISRA-C:2012 Dir】 4.4 (推奨)

以下のようにコード部をコメントアウトするのは禁止する。

```
/* if ( TRUE == flag ) { */
```

コード部をソースファイルに残しておきたいときは、以下のように「#if 0」を使用する。

```
#if 0
```

```
If ( TRUE == flag ) {
```

```
#endif
```

4.4 コメントブロック

(1) ヘッダコメントブロック

ソースコードの先頭に、以下のフォーマットで当該ソースコードの説明を記述する。

```

/*****/
/* 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 著作権者と最初の発行年(2015 年に改訂があった場合は、“2014-2015”のように記述する)
- ② システム名、商品名
- ③ ファイル名
- ④ 当該ファイルのバージョン
詳細は『8.2 ソースファイルのソフトウェアバージョン』を参照のこと
- ⑤ 当該ファイルの説明
- ⑥ 作成者名
- ⑦ 備考(内容は書かなくても良い)

(2) フッタコメントブロック

ソースコードの最後に、以下のフォーマットで当該ソースコードの末端を示すコメントを記述する。

```
/* EOF Dem. c *****/
```

(3) 関数コメントブロック

関数の先頭に、以下のフォーマットで当該関数の説明を記述する。

```
/* *****/
/* ModuleID      : MODULE_ID_DEM (054)                */①
/* ServiceID     : DEM_INIT_ID (0x02)                  */②
/* Name          : Dem_Init                             */③
/* Param         : (in) ConfigPtr      Pointer to the configuration set in  */④
/*               :                               VARIANT-POST-BUILD.         */
/* Return        : void                                */⑤
/* Contents      : Initializes or reinitializes this module.                */⑥
/* Author        : t.sasaki                                           */⑦
/* Note          : [SWS_Dem_00181]                                       */⑧
/* *****/
```

- ① モジュール名、およびモジュール ID
- ② 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)

(4) 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.                                                     */
```

4.5 トレーサビリティ対応

ソフトウェア要求仕様 (SRS) または AUTOSAR 要件 (SWS) が API に紐づけられているかを確認するため、トレーサビリティマトリックスを文書化しなければならない。

【MISRA-C:2012 Dir】 3.1 (必要)

また、作成したトレーサビリティマトリックスをもとに、ソースコードに要件 ID を埋め込む必要がある。

(1) 関数コメントブロック

関数そのものに要件 ID を定義しているものがある場合、関数コメントブロックの「Note」の項に要件 ID を記載する。要件 ID はツールで抽出することを想定し、□で括る。

```
/* *****
:
:
/* Note      : [SWS_Dem_00181]
/* *****
```

(2) コード部のコメント

機能に対して要件 ID が定義されている場合、その機能を実現しているコードのコメントに要件 ID を記載する。要件 ID はツールで抽出することを想定し、□で括る。

```
/* [SWS_Dem_00181] */
/* check EventId */
if ( NO_EVENT_ID != eventId ) {
```

5 名前の付け方の統一

変数、定数等の命名規則をプロジェクト内で定め、名前の付け方を統一すること。

また同じ名前空間(「ラベル名」「構造体、共用体及び列挙体のタグ」「構造体又は共用体のメンバ」「その他のすべての識別子」)にある識別子の名前は区別がつくようにする。

ラテンアルファベットに関しては、以下の組み合わせの異なる識別子がないこと。

【MISRA-C:2012 Dir】 4.5 (推奨)

表 1 区別がつかない識別子

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	<code>returnCode</code>
	大文字	<code>ReturnCode</code>
アンダースコアの有/無	アンダースコア有り	<code>returnCode</code>
	アンダースコア無し	<code>return_Code</code>
文字の「O」/数字の「0」	文字の「O」	<code>Channel0</code>
	数字の「0」	<code>Channel0</code>
文字の「I」/数字の「1」	文字の「I」	<code>counter_I</code>
	数字の「1」	<code>counter_1</code>
文字の「I」/「l」(エル)	文字の「I」	<code>counter_I</code>
	文字の「l」(エル)	<code>counter_l</code>
文字の「l」(エル)/数字の「1」	文字の「l」(エル)	<code>counter_l</code>
	数字の「1」	<code>counter_1</code>
文字の「S」/数字の「5」	文字の「S」	<code>counter_S</code>
	数字の「5」	<code>counter_5</code>
文字の「Z」/数字の「2」	文字の「Z」	<code>counter_Z</code>
	数字の「2」	<code>counter_2</code>
「n」(エヌ)/「h」(エイチ)	文字の「n」(エヌ)	<code>counter_n</code>
	数字の「h」(エイチ)	<code>counter_h</code>
文字の「B」/数字の「8」	文字の「B」	<code>counter_B</code>
	数字の「8」	<code>counter_8</code>
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)	<code>Dcrn</code>
	文字の「m」(エム)	<code>Dcm</code>

6 ポインタの書き方の統一

6.1 ポインタ

ポインタ宣言子は、型の後ろにスペースを空けずに記述する。

```
char*^p;
```

※「^」が半角スペースを示す

6.2 ヌルポインタ

ヌルポインタの書き方は、「NULL」とする。「NULL」はヌルポインタ以外に使用しない。

```
char* p;
```

```
p = NULL;
```

7 ダミー関数の書き方の統一

プログラム実装する際に、まだ未実装のモジュールの関数(API)コールを実装する場合がある。このときのコーディングの仕方を以下に定義する。

7.1 ダミーヘッダファイル

未実装のモジュールの関数(API)をコールしているコードをコンパイルするために、未実装モジュールのダミーヘッダファイル(モジュール名.h)を作成し、そこで関数(API)を宣言する。

関数(API)の宣言を自身の(コンパイルを通そうとしている)モジュールのヘッダファイルやソースファイルに記述してはならない。

```
void Dem_GetVersionInfo( Std_VersionInfoType* versioninfo );  
Std_ReturnType NvM_RestorePRAMBlockDefaults( NvM_BlockIdType BlockId );
```

7.2 ダミープログラムファイル

未実装のモジュールの関数(API)をコールしているコードをリンク(ビルド)するために、未実装モジュールのダミーソースファイル(モジュール名.c)を作成し、そこで関数(API)のボディを実装する。

関数(API)のボディを自身の(リンクを通そうとしている)モジュールのヘッダファイルやソースファイルに記述してはならない。

関数(API)のボディは、空(処理がない状態)とする。戻り値のある関数(API)に関しては、正常値(E_OK 等)を固定で返却するものとする。また、戻り値や引数で設定値を返す関数(API)に関しては、有効範囲内の値を固定で返却しても良いし、何も設定しなくても良いものとする。

```
void Dem_GetVersionInfo( Std_VersionInfoType* versioninfo ) {}  
Std_ReturnType NvM_RestorePRAMBlockDefaults( NvM_BlockIdType BlockId ) { return E_OK; }
```

8 ソフトウェアバージョン更新ルールの統一

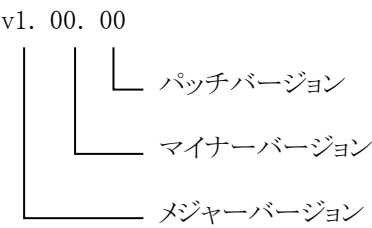
ソフトウェアバージョンの更新ルールを以下に定義する。

8.1 ソフトウェアバージョン

ソフトウェアバージョンについてプロジェクト内で定義し、管理すること。

8.2 ソースファイルバージョン

各ソースファイルのバージョンを、以下のルールにて管理し、ヘッダコメントブロックに記載すること。
(『4.3 コメントブロック』の「(1) ヘッダコメントブロック」の④を参照)



- メジャーバージョン
新規機能追加時に更新。
- マイナーバージョン
IF の変更を伴う関数の修正など、互換性のない変更時に更新。
メジャーバージョン更新時は「0」とする。
- パッチバージョン
IF の変更を伴わない関数の修正など、互換性のある変更時に更新。
メジャーバージョンまたはマイナーバージョン更新時は「0」とする。

※このソースファイルバージョンはファイルごとに更新する。(ソフトウェアバージョンとは異なる。)

9 機能安全対応

機能安全対応に関する MISRA-C:2012 の指針・ルールを以下に定義する。

9.1 処理系定義の文書化

プログラムの出力が依存する処理系定義の動作は、文書化され、理解されなければならない。そのため、本項内に記載を行う。

【MISRA-C :2012 Dir】 1.1 (必要)

依存する処理系定義の動作を、以下に定義をする。

- C 言語の規格
C99 を対象とする。

9.2 コンパイルエラー・ワーニング検出

すべてのソースファイルのコンパイルエラー、ワーニングが検出されてはいけない。

実現する機能により、ワーニング箇所を修正することで、目的が達せられないなどの不具合になる可能性が考えられる。そのような場合は、逸脱を行いリリースノートに除外理由を明記する。

【MISRA-C:2012 Dir】 2.1 (必要)

9.3 エラーチェックの実施

9.3.1 実行時エラーのチェック

実行時にエラーが発生する可能性がある場所では、エラーチェックを行う。

【MISRA-C:2012 Dir】 4.1 (必要)

以下に、エラーチェックを行うべき箇所の例を示す。

- オーバーフローを考慮する

```
/* 左辺 : バッファサイズ 右辺 : 実データ長 */
/* 実データがバッファサイズより大きい確認*/
if ( s_Dlt_MBuff_RemainSize < ( message_length + header_length + 1U ) ) {
    /* 異常処理 */
}
```

- 0除算のガードをする

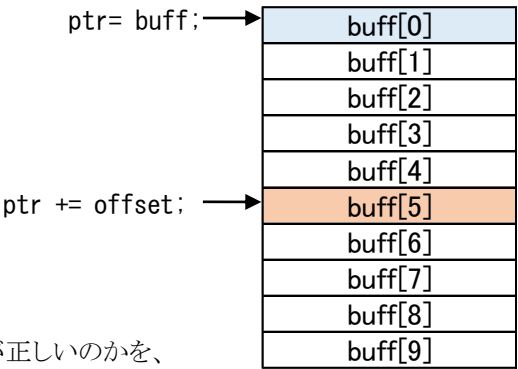
```
/* data:分母 Division: 商 */
/* 分母が0 かの確認 */
if ( 0U == data ) {
    /* 分母が0 の場合は、計算をしてはいけない */
    Division = 0;
} else {
    /* 分母が0 以外の場合のみ、計算をする(正常処理) */
    Division = 100 / data;
}
```

- ポインタ演算が意図された結果にならなくてはいけない
ポインタ演算を行っている箇所については、
コードレビュー時に意図した処理が行えるかを確認する。

以下に例を示す。

```
uint8 buff[10];
uint8 offset = 5;
uint8* ptr = buff;

ptr += offset;
```



この場合、処理として ptr が buff[5]を指し示すことが正しいのかを、
コードレビュー時に確認する必要がある。

- 関数のパラメータのチェックをする
詳細は『9.3.3 引数のエラーチェック』を参照のこと

- NULL ポインタにはアクセスしない

/* ポインタ使用時にアドレスが NULL でないか確認 */

```
if ( NULL == PduInfoPtr ) {
```

```
    /* 異常処理 */
```

```
}
```

- 配列の添え字がマイナスや範囲外にならないようにする

配列の添え字がマイナスにならないようにチェックをしている例。

```
uint8 buff[200];
```

/* 添え字が 0 以上であるか確認 */

```
for ( i = (sint8)100 ; i >= 0 ; i-- ) {
```

```
    /* 正常処理 */
```

```
    buff[i] = TRUE;
```

```
}
```

配列の添え字が範囲外にならないようにチェックをしている例。

/* CANIF_Q_NUM_OF_TRCV : 配列の要素数 */

```
CanIf_TrcvInfoType s_TrcvInfo[CANIF_Q_NUM_OF_TRCV];
```

/* 添え字が要素数より小さいか確認 */

```
for ( i=(uint32_least)0x0U; i < CANIF_Q_NUM_OF_TRCV; i++ ) {
```

```
    /* 正常処理 */
```

```
    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

9.3.2 エラー情報を戻す関数のエラーチェック

関数がエラー情報を戻す場合、内部でエラーが発生する可能性があると考えられる。そのため、エラー情報を戻す関数を呼び出す処理において、出力値の適切なチェックを行う。

【MISRA-C:2012 Dir】 4.7 (必要)

戻り値の適切なエラーチェックを行っている例を以下に示す。

```
/* 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 */
```

```
}
```

9.3.3 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 */
```

```
}
```

(2) 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 */  
    }  
}
```

(3) 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;  
    }  
}
```

9.4 アセンブリ言語使用禁止

アセンブリ言語はコンパイラ依存の言語のため使用はしてはいけない。

止むを得ず、アセンブリ言語を使用する場合、移植時の書き直し対象の明確化のために、アセンブリ言語使用ファイルは、C 言語のファイルとは別ファイルに隔離する。(Inline アセンブラの禁止)

【MISRA-C:2012 Dir】 4.3 (必要)

また、アセンブリ言語を使用した場合、依存しているコンパイラ、アセンブリ言語を使用した理由を、ソースファイルにコメントとして記述する。

【MISRA-C:2012 Dir】 4.2 (推奨)

9.5 基本型使用禁止

処理系により型のサイズが異なる可能性がある。そのため、基本型("int"等)を直接使用しない。代わりに一目でサイズと符号の有無を示す typedef("uint8"等)を使用する。

【MISRA-C:2012 Dir】 4.6 (推奨)

以下に、32bit マシンでの型の具体例を示す。

表 2 32bit マシン時の型

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

※使用する ECU に合わせること。

なお、ループ変数は、処理速度の効率化のため、least で定義された型を使用する。
(SWS_Platform に定義されている)

9.6 関数・関数マクロ作成時の注意

関数や、関数形式のマクロを使用する場合、該当箇所の処理には関数・関数形式マクロのどちらが適しているか検討すること。

【MISRA-C:2012 Dir】 4.9 (推奨)

以下に記載する指針に合致するコードとなっていること。

- コールされる箇所が多い場合は、関数マクロではなく関数の方が適している
- スピード重視をする場合は、関数ではなく関数マクロの方が適している

※具体的な数値については別途検討

9.7 ポインタ使用時の注意

ポインタを使用する際は、使用するポインタ型変数が明らかになっており、必要性和安全性が説明できること。

また、不完全型のポインタ宣言が良いか、完全型のポインタ宣言が良いかを検討する必要がある。

【MISRA-C:2012 Dir】 4.8 (推奨)

以下に不完全型、完全型ポインタの例を示す。

```
struct ImpfType;           /* 不完全な型 */
struct ImpfType* ptr1; /* 不完全な型を指すポインタ = 不完全型ポインタ */
uint8* ptr2; /* 完全な型を指すポインタ = 完全型ポインタ */
```

上記ポインタにアドレスを代入した際、代入したアドレスのメンバを参照不可能なのが不完全型ポインタ(ptr1)、代入したアドレスのメンバを参照可能なのが完全型ポインタ(ptr2)。アドレス渡しだけを行う処理に不完全型ポインタを使用することで、外部から型を隠蔽することができる。

9.8 多重インクルード禁止

多重インクルードをした場合、結果は不確定か誤ったふるまいとなるおそれがあるため、多重インクルードをしてはいけない。それに伴い、多重インクルードが防止されていること

メモリマップ対応で使用する場合は、多重インクルードしても良い。

【MISRA-C:2012 Dir】 4.10 (必要)

以下に、ヘッダファイルの多重インクルード防止例を示す。

```
#ifndef BSWM_H
#define BSWM_H
    /* 処理 */
#endif /* BSWM_H */
```

以下に例外である、メモリマップ対応での多重インクルード例を示す。

```
#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"
```

9.9 標準ライブラリ使用禁止

標準ライブラリは使用してはいけない。

そのため、標準ライブラリのヘッダファイルはインクルードしてはいけない。

【MISRA-C:2012 Dir】 4.11 (必要)

【MISRA-C:2012 Rule】 22.1, 22.3 (必要)

9.10 動的なオブジェクト・メモリ確保禁止

オブジェクトは全て静的に配置され、特定の期間のみ生成されるオブジェクトが存在しないこと。

動的にメモリ確保を行ってはいけない。

【MISRA-C:2012 Dir】 4.12 (必要)

以下を禁止し、コードレビュー時に動的なメモリ確保をしていないことを確認する。

- 動的なメモリ割り当てを行う標準ライブラリ関数の使用
代表的な例: calloc、malloc、realloc 及び free 関数
- 標準ライブラリ関数を使用しない、自作による動的なメモリ割り当て(サードパーティのパッケージを含む)

9.11 ファイルの操作禁止

ファイルに対する操作は行わない。

【MISRA-C:2012 Dir】 4.13 (推奨)

【MISRA-C:2012 Rule】 22.3 (必要) 22.4, 22.6 (義務)

以下のような処理を行うことを禁止する。

- ・リソースの割当て(例:ファイルを開く)。
- ・リソースの割当て解除(例:ファイルを閉じる)。
- ・他の操作(例:ファイルからの読み取り)。

— 以上 —