

Coding Style Guide

[C language version]



SCSK Corporation

Mobility System Business Division

Mobility System 1st Business division, System 3rd department, 1st Section

Ngày phát hành: 25/05/2020

●Revision history

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Ngày thay đổi | Nội dung và lý do thay đổi | Approve | Create |
| 1.0.0 | 2020/4/14 | ・Tạo mới | 森谷 | 藤田 |
| 1.0.1 | 2020/5/25 | ・Add 「3.5 変数の初期化/Khởi tạo biến số」  ・Add 「3.6 グローバル変数使用時の注意/Chú ý khi sử dụng biến global」  ・Add 「3.7 暗黙の型変換禁止/Cấm chuyển đổi type kiểu ngầm định (type coercion)」  ・Add 「3.9 無条件のジャンプ禁止/Cấm jump không điều kiện」  ・Add 「3.10 再帰処理禁止/Cấm xử lý đệ quy (recursion) 」  ・Add mục cần lưu ý vào 「9.7 ポインタ使用時の注意/Chú ý khi sử dụng pointer」  ・Thay đổi title của 「9.10 動的なメモリ確保禁止」thành 「9.10 動的なオブジェクト・メモリ確保禁止」 và add thêm mô tả object vào nội dung. | 森谷 | 藤田 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

●Mục lục

[1 Mở đầu 4](#_Toc58598188)

[2 Character code/Newline code 5](#_Toc58598189)

[2.1 Character code 5](#_Toc58598190)

[2.2 Newline code 5](#_Toc58598191)

[3 Thống nhất coding style 6](#_Toc58598192)

[3.1 Vị trí của ngoặc nhọn （{ }） 6](#_Toc58598193)

[3.2 Thụt đầu dòng (Indentation) 7](#_Toc58598194)

[3.3 Cách chèn space 7](#_Toc58598195)

[3.4 Vị trí xuống dòng ở dòng tiếp theo 8](#_Toc58598196)

[3.5 Khởi tạo biến số 8](#_Toc58598197)

[3.6 Chú ý khi sử dụng biến global 8](#_Toc58598198)

[3.7 Cấm chuyển đổi type kiểu ngầm định (type coercion) 8](#_Toc58598199)

[3.8 Điều kiệu phán đoán ở câu lệnh if 10](#_Toc58598200)

[3.9 Cấm jump không điều kiện 10](#_Toc58598201)

[3.10 Cấm xử lý đệ quy 10](#_Toc58598202)

[3.11 Câu lệnh return của hàm void 10](#_Toc58598203)

[3.12 Cho phép return giữa chừng của hàm 10](#_Toc58598204)

[4 Thống nhất cách viết comment 11](#_Toc58598205)

[4.1 Cách viết comment 11](#_Toc58598206)

[4.2 Case đặc thù 12](#_Toc58598207)

[4.3 Cấm comment out phần code 13](#_Toc58598208)

[4.4 Comment block 14](#_Toc58598209)

[4.5 Hỗ trợ traceability 16](#_Toc58598210)

[5 Thống nhất cách đặt tên 17](#_Toc58598211)

[6 Thống nhất cách ghi pointer 18](#_Toc58598212)

[6.1 Pointer 18](#_Toc58598213)

[6.2 Null Pointer 18](#_Toc58598214)

[7 Thống nhất cách ghi dummy function 18](#_Toc58598215)

[7.1 Dummy header file 18](#_Toc58598216)

[7.2 Dummy Program File 19](#_Toc58598217)

[8 Thống nhất rule update software version 19](#_Toc58598218)

[8.1 Software Version 19](#_Toc58598219)

[8.2 Source File Version 20](#_Toc58598220)

[9 Đối ứng an toàn chức năng 20](#_Toc58598221)

[9.1 Tạo văn bản đối với định nghĩa hệ thống xử lý 20](#_Toc58598222)

[9.2 Phát hiện compile error và warning 21](#_Toc58598223)

[9.3 Thực hiện check error 22](#_Toc58598224)

[9.3.1 Check error khi chạy 22](#_Toc58598225)

[9.3.2 Check error cho hàm trả về thông tin error 23](#_Toc58598226)

[9.3.3 Check error đối với đối số 24](#_Toc58598227)

[9.4 Cấm sử dụng hợp ngữ (assembly language) 25](#_Toc58598228)

[9.5 Cấm sử dụng kiểu cơ bản 26](#_Toc58598229)

[9.6 Chú ý khi tạo hàm và macro hàm 26](#_Toc58598230)

[9.7 Chú ý khi sử dụng pointer 27](#_Toc58598231)

[9.8 Cấm multiple include 27](#_Toc58598232)

[9.9 Cấm sử dụng standard library 28](#_Toc58598233)

[9.10 Cấm cấp phát bộ nhớ và object động 28](#_Toc58598234)

[9.11 Cấm thao tác đối với file 28](#_Toc58598235)

# 1 Mở đầu

Tài liệu này định nghĩa về hướng dẫn coding style với mục đích chuẩn hóa source code của in-vehicle development.

Ngoài ra sẽ sử dụng MISRA-C:2012 như là coding convention để đảm bảo tính an toàn của software, tài liệu này chủ yếu tập trung vào cách viết source code với vai trò hoàn thiện bổ sung.

Tài liệu này định nghĩa về hướng đẫn coding style có sẵn của SCSK lấy template là 「保守性 4 統一した書き方にする。」 của 「Revised version Hướng dẫn tạo coding dành cho embedded software developer [C language version] Ver1.1」(Tổ chức xúc tiến xử lý thông tin thể chế hành chính độc lập Software/Engineering/Center)

# 2 Character code/Newline code

### 2.1 Character code

Character code để tạo in-vehicle program code sẽ là UTF-8.

### 2.2 Newline code

Newline code để tạo in-vehicle program code sẽ là CR+LF.

# 3 Thống nhất coding style

### 3.1 Vị trí của ngoặc nhọn （{ }）

Thống nhất vị trí của ngoặc nhọn để dễ nhìn được bắt đầu và kết thúc của block.

1. Dấu ngoặc mở hàm sẽ ghi ở cùng dòng với hàm hoặc dòng riêng, dấu ngoặc đóng hàm sẽ ghi ở dòng riêng biệt so với xử lý.

void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo ) {

}

void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo )

{

}

1. Dấu ngoặc mở câu lệnh control sẽ ghi ở cùng dòng với hàm hoặc dòng riêng, dấu ngoặc đóng sẽ ghi ở dòng riêng biệt so với xử lý.

if ( NUM\_OF\_EVENT\_DATA == data ) {

}

1. Ghi else và dấu ngoặc mở ở cùng dòng với dấu ngoặc đóng của câu lệnh control câu lệnh else, ghi dấu ngoặc đóng vào dòng riêng biệt so với xử lý.

if ( NUM\_OF\_EVENT\_DATA == data ) {

} else if ( MIN\_EVENT\_DATA == data ) {

} else {

}

### 3.2 Thụt đầu dòng (Indentation)

Thực hiện thụt đầu dòng để dễ nhìn conclusion của việc khai báo và xử lý.

Thụt đầu dòng 4 space (half-size space). Do có case là nếu sử dụng tab thì hiển thị sẽ bị lệch do editor nên sẽ sử dụng space (half-size space).

if ( 1 == data ) {

^^^^p = &list[0];

}

※「^」 chỉ half-size space

switch ( type ) { case 1:

^^^^data = 1; ^^^^break; case 2:

^^^^data = 2; ^^^^break; default:

^^^^break;

}

※「^」 chỉ half-size space

※「case」 thì không thụt đầu dòng

### 3.3 Cách chèn space

Chèn 1 space để dễ nhìn code và dễ phát hiện lỗi coding.

1. Function

void^Dem\_GetVersionInfo(^Std\_VersionInfoType\*^versioninfo^)^{

※「^」 chỉ half-size space

1. Control statement

if^(^MAX\_DATA^==^data^)^{

※「^」 chỉ half-size space

for^(^i=0;^i<max;^i++^)^{

※「^」 chỉ half-size space

1. Công thức

SEventStatusBuffer[i].faultDetectionCounter^=^0U;

※「^」 chỉ half-size space

### 3.4 Vị trí xuống dòng ở dòng tiếp theo

Khi biểu thức dài trở nên khó đọc thì sẽ xuống dòng ở vị trí thích hợp. Tuy nhiên, để dễ nhìn khi hiển thị màn hình monitor code và dễ đọc khi in thì có trường hợp số ký tự của 1 dòng vượt quá 100 ký thì sẽ xuống dòng ở ký tự 100.

Ghi operator ở đầu của dòng tiếp theo.

Thụt đầu dòng ở dòng tiếp theo sau khi đã xuống dòng sẽ điều chỉnh ở vị trí dễ nhìn bằng space (half-size space), không quy định số lượng thụt đầu dòng.

dataNum = DemConfigData.FreezeFrameData.FreezeFrameDataNum

^^^^^^^^+ DemConfigData.ExtendedData.ExtendedDataNum

^^^^^^^^+ DemConfigData.ExtendedData.OtherDataNum;

※「^」 chỉ half-size space

if (( SWC\_E\_SYSTEM\_POWER\_BROWNOUT == data1 )

^&& ( SWC\_E\_SENSOR\_POWER\_BROWNOUT == data2 )) {

※「^」 chỉ half-size space

### 3.5 Khởi tạo biến số

Thực hiện khởi tạo toàn bộ biến số.

### 3.6 Chú ý khi sử dụng biến global

Để các biến global sẽ sử dụng trở nên rõ ràng và có thể giải thích được tính cần thiết và tính an toàn.

### 3.7 Cấm chuyển đổi type kiểu ngầm định (type coercion)

Toàn bộ việc chuyển đổi type phải được thực hiện một cách rõ ràng.

uint8 data; data = (uint8)1U;

### 3.8 Điều kiệu phán đoán ở câu lệnh if

Điều kiện của câu lệnh if so sánh hằng số và biến số sẽ ghi hằng số ở vùng bên trái và ghi biến số ở vùng bên phải như dưới đây.

/\* MAX\_DATA:hằng số／data:Biến số \*/ if ( MAX\_DATA == data ) {

Công thức điều kiện của câu lệnh if 「if ( biến số == hằng số )」 dù có ghi sai thành 「if ( biến số = hằng số )」 đi nữa thì vẫn được xem là gán cho biến số và sẽ không thành compile error. Do đó có khả năng việc detect bug bị chậm. Bằng việc ghi công thức điều kiện của câu lệnh if thành 「if ( hằng số == biến số )」 thì khi có ghi sai thành 「if ( hằng số = biến số )」 thì sẽ thành compile error và có thể phát hiện bug khi compile.

### 3.9 Cấm jump không điều kiện

Không sử dụng câu lệnh goto và xử lý jump.

### 3.10 Cấm xử lý đệ quy

Không sử dụng xử lý đệ quy.

### 3.11 Câu lệnh return của hàm void

Ở hàm void thì không ghi câu lệnh return.

Do nếu ghi câu lệnh return ở hàm void thì có trường hợp một số compiler output ra code không mong muốn.

### 3.12 Cho phép return giữa chừng của hàm

Ở MISRA-C:2012 Rule 15.5 thì đang khuyến khích việc 「đối với hàm số thì phải có một đầu ra duy nhất ở cuối hàm」. Việc này là do đang hình dung khi có nhiều đầu ra của hàm thì khi control flow bị gián đoạn ở giữa chừng xử lý và cần xử lý cho từng đầu ra thì có khả năng gây ra bug.

Tuy nhiên, trường hợp code lồng nhau trở nên nhiều (khi check config chẳng hạn) thì sẽ khó đọc, ngược lại còn có khả năng gây ra bug.

Ngoài ra, nếu hàm số hóa xử lý để giảm lồng nhau thì sẽ gây ảnh hưởng đến tốc độ thực hiện.

Do đó, trường hợp ngăn chặn sự gia tăng của việc lồng nhau này hay có lý do chính đáng cho việc đó thì việc cho phép return giữa chừng của hàm (nhiều đầu ra) sẽ đối tượng review và sẽ lưu lại kết quả để làm evidence.

/\* Trường hợp setting data 1～3 là ngoài phạm vi thì sẽ trả về error và thoát khỏi xử lý \*/

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 );

}

/\* Trường hợp setting data 4～5 là ngoài phạm vi thì sẽ trả về error và thoát khỏi xử lý \*/

if ((( s\_config.data4 < MIN4 ) || ( s\_config.data4 > MAX4 )) || (( s\_config.data5 < MIN5 ) || ( s\_config.data5 > MAX5 ))) { return( E\_NOT\_OK );

}

/\* Xử lý bình thường \*/

・・・

return ( E\_OK );

# 4 Thống nhất cách viết comment

### 4.1 Cách viết comment

Để dễ đọc source code thì sẽ ghi comment như dưới đây.

1. Ở dòng trước của cùng một cột với source code thì sẽ ghi comment độc lập so với source code.

Trường hợp source code dài và trường hợp câu lệnh control... thì nên để dòng comment độc lập thì sẽ dễ đọc hơn.

/\* check DTC par event \*/

if ( dtc == eventdata.dtc ) {

/\* check EventId

\* NOTE: 0xFFFF is not valid

\*/

if ( NO\_EVENT\_ID != eventId ) {

1. Ghi comment ở phía sau của cùng dòng với source code.

Việc gán data và khai báo biến local... thì nên ghi comment vào sau source code thì sẽ dễ đọc hơn.

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

### 4.2 Case đặc thù

Liên quan đến case khó phán đoán là lỗi coding hay không thì bắt buộc phải ghi comment.

1. Trường hợp không ghi break ở câu lệnh switch case

Phải ghi lý do break không cần thiết ở trong comment.

Về cơ bản thì sẽ ghi break.

1. Trường hợp không xử lý bằng câu lệnh if.

Phải ghi lý do xử lý không cần thiết ở trong comment.

1. #endif

Ghi đang đối ứng cho #if nào ở comment.

### 4.3 Cấm comment out phần code

Việc comment out phần code không sử dụng rồi lưu lại ở file source thì sẽ làm khó đọc code nên không được comment out một phần của code.

Trường hợp cần disable phần code thì sẽ sử dụng 「#if 0」.

【MISRA-C:2012 Dir】 4.4 (Recommended)

Cấm comment out phần code như sau.

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

Khi muốn lưu lại phần code ở file source thì sẽ sử dụng 「#if 0」 như dưới đây.

#if 0

If ( TRUE == flag ) {

#endif

### 4.4 Comment block

(1) Ghi giải thích source code tương ứng bằng format dưới đây vào phía đầu của header comment block source code.

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

/\* 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. Người giữ bản quyền và năm phát hành đầu tiên (trường hợp đã có sửa đổi vào năm 2015 thì sẽ ghi là "2014-2015")
2. System name, product name
3. File name
4. Version của file tương ứng

Chi tiết tham khảo 『8.2 ソースファイルのソフトウェアバージョン/Software version của file source』

1. Giải thích file tương ứng
2. Tên người tạo
3. Note (không cần ghi nội dung cũng được)

1. Ghi comment cho biết phần cuối của source code tương ứng với format dưới đây vào cuối cùng của footer comment block source code.

/\* EOF Dem.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1. Ghi giải thích hàm tương ứng bằng format dưới đây vào phần đầu của function comment block function

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

/\* 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. Module name và Module ID
2. Service name và Service ID
3. Tên hàm
4. Function parameter (phải ghi rõ in/out/inout)
5. Giá trị trả về
6. Giải thích hàm tương ứng
7. Tên người tạo hàm tương ứng
8. Note (SRS/SWS Item...)

(4) Lịch sử thay đổi

Ghi lịch sử thay đổi với format dưới đây vào header comment block và function comment block.

※Đối với history thì sẽ ghi ở timing mà minor và patch version của file đã được update. Khi file major version được update thì sẽ không lưu history do được tạo mới.

Ở header comment block sẽ ghi date và file version và tên hàm (hoặc tên biến) đã thay đổi.

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

Ở function comment block sẽ ghi lý do thay đổi.

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

/\* Fixed problem the return code is not returned \*/

/\* correctly. \*/

### 4.5 Hỗ trợ traceability

Do confirm việc software requirement spec (SRS) hoặc AUTOSAR requirement (SWS) đang liên kết với API nên phải tạo document cho traceability matrix.

【MISRA-C:2012 Dir】 3.1 (Necessary)

Ngoài ra, cần chèn Requirement ID vào source code dựa vào traceability matrix đã tạo.

1. Trường hợp ở function comment block có cái đang định nghĩa requirement ID thì sẽ ghi requirement ID vào item [Note] của function comment block. Đang assume việc extract requirement ID bằng tool và bao quanh bằng [].

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

：

：

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

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

1. Comment phần code

Trường hợp đang định nghĩa requirement ID theo chức năng thì sẽ ghi requirement ID vào comment của code đang thực hiện chức năng đó. Đang assume việc extract requirement ID bằng tool và bao quanh bằng [].

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

# 5 Thống nhất cách đặt tên

Phải quy định quy ước đặt tên biến số, hằng số... ở trong project và thống nhất cách đặt tên.

Ngoài ra, sẽ phân biệt tên của Identifier có trong cùng name space ([Label name], [Tag của Structure, Union và enumeration], [member của Structure và Union], [toàn bộ Identifier khác])

Liên quan đến bảng chữ cái latin thì sẽ không có Identifier khác nhau trong các kết hợp dưới đây.

【MISRA-C:2012 Dir】 4.5 (Recommended)

Bảng 1 Identifier không thể phân biệt

|  |  |  |
| --- | --- | --- |
| Kết hợp Indentifier không thể phân biệt | Ví dụ | |
| Chữ thường /Chữ in hoa | Chữ thường | returnCode |
| Chữ in hoa | ReturnCode |
| Có/không có underscore | Có underscore | returnCode |
| Không có underscore | return\_Code |
| Ký tự 「O」/Chữ số 「0」 | Ký tự 「O」 | ChannelO |
| Chữ số 「0」 | Channel0 |
| Ký tự 「I」/Chữ số 「1」 | Ký tự 「I」 | counter\_I |
| Chữ số 「1」 | counter\_1 |
| Ký tự 「I」/「l」(chữ l) | Ký tự 「I」 | counter\_I |
| Ký tự 「l」(chữ l) | counter\_l |
| Ký tự 「l」(chữ l)/Chữ số 「1」 | Ký tự 「l」(chữ l) | counter\_l |
| Chữ số 「1」 | counter\_1 |
| Ký tự 「S」/Chữ số 「5」 | Ký tự 「S」 | counter\_S |
| Chữ số 「5」 | counter\_5 |
| Ký tự 「Z」/Chữ số 「2」 | Ký tự 「Z」 | counter\_Z |
| Chữ số 「2」 | counter\_2 |
| 「n」(chữ n)/「h」(chữ h) | Ký tự 「n」 (chữ n) | counter\_n |
| Chữ số 「h」 (chữ h) | counter\_h |
| Ký tự 「B」/Chữ số 「8」 | Ký tự 「B」 | counter\_B |
| Chữ số 「8」 | counter\_8 |
| Sắp xếp ký tự 「rn」 (r rồi tiếp theo nó là n)  /Ký tự 「m」(chữ m) | Sắp xếp ký tự 「rn」 (r rồi tiếp theo nó là n) | Dcrn |
| Ký tự 「m」(chữ m) | Dcm |

# 6 Thống nhất cách ghi pointer

### 6.1 Pointer

Ghi lệnh khai báo pointer mà không có space ở phía sau type.

char\*^p;

※「^」 chỉ half-size space

### 6.2 Null Pointer

Cách ghi Null Pointer sẽ là 「NULL」. Không sử dụng 「NULL」 đối với những cái khác Null Pointer.

char\* p; p = NULL;

# 7 Thống nhất cách ghi dummy function

Khi coding program thì vẫn có trường hợp coding hàm (API) của module chưa coding. Định nghĩa cách coding lúc này ở dưới đây.

### 7.1 Dummy header file

Để compile code mà đang call hàm (API) của module chưa implement thì sẽ tạo dummy header file của module chưa implement (module name.h) rồi khai báo hàm (API) ở đó.

Không được viết khai báo hàm (API) vào header file và source file thuộc module (mà bạn đang định compile) của bạn.

void Dem\_GetVersionInfo( Std\_VersionInfoType\* versioninfo );

Std\_ReturnType NvM\_RestorePRAMBlockDefaults( NvM\_BlockIdType BlockId );

### 7.2 Dummy Program File

Để link (build) các code đang gọi hàm (API) của module chưa triển khai, tạo dummy source file (module name.c) của module chưa triển khai và thực hiện triển khai body của hàm (API) tại đó.

Không được mô tả body của hàm (API) vào header file và source file thuộc module (mà bạn đang định link) của bạn.

Để trống (trạng thái không có xử lý) phần body của hàm (API). Về hàm (API) có giá trị trả về, phải trả về cố định giá trị bình thường (chẳng hạn như E\_OK). Ngoài ra, về hàm (API) trả về giá trị setting thông qua giá trị trả về hay đối số (argument), thì trả về cố định là giá trị nằm trong phạm vi hợp lệ cũng được, không cần cài đặt giá trị gì cũng được.

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

Std\_ReturnType NvM\_RestorePRAMBlockDefaults( NvM\_BlockIdType BlockId ) { return E\_OK; }

# 8 Thống nhất rule update software version

Định nghĩa rule update đối với software version như sau.

### 8.1 Software Version

Định nghĩa về software version nằm trong phạm vi dự án và quản lý.

### 8.2 Source File Version

Quản lý version của các source file theo rule dưới đây và mô tả vào header comment block.

(Tham khảo phần ④ của mục "(1) ヘッダコメントブロック/Header Comment Block" thuộc 『4.3 コメントブロック/Comment Block』)

v1. 00. 00

Patch version minor version

Major Version

* Major Version

Update khi thêm chức năng mới

* Minor Version

Update khi thay đổi mà không có tính tương thích, chẳng hạn như sửa hàm do thay đổi IF v.v

Khi update major version thì set là "0".

* Patch Version

Update khi thay đổi mà có tính tương thích, chẳng hạn như sửa hàm mà không liên quan đến việc thay đổi IF v.v

Khi update major version hoặc minor version thì set là "0".

※Update version source file này theo từng file. (khác với software version.)

# 9 Đối ứng an toàn chức năng

Dưới đây định nghĩa guideline và rule của MISRA-C：2012 liên quan đến đối ứng an toàn chức năng

### 9.1 Tạo văn bản đối với định nghĩa hệ thống xử lý

Các hoạt động định nghĩa hệ thống xử lý mà output của program phụ thuộc vào đó phải được tạo thành văn bản, và phải nắm rõ. Vì thế sẽ mô tả vào trong mục này.

【MISRA-C :2012 Dir】 1.1 (Necessary)

Định nghĩa về hoạt động định nghĩa hệ thống xử lý phụ thuộc như sau.

Tiêu chuẩn của ngôn ngữ C

Đối tượng là C99.

### 9.2 Phát hiện compile error và warning

Tất cả các compile error và warning của source file không nên bị phát hiện.

Tùy vào chức năng thực hiện, có khả năng chúng sẽ biến thành bug như không đạt được mục đích v.v, do sửa các vị trí warning. Trường hợp như thế, hãy làm sai lệch và nêu rõ lý do loại trừ vào release note.

【MISRA-C:2012 Dir】 2.1 (Necessary)

### 9.3 Thực hiện check error

#### 9.3.1 Check error khi chạy

Thực hiện check error tại vị trí có khả năng phát sinh error khi chạy.

【MISRA-C:2012 Dir】 4.1 (Necessary)

Dưới đây trình bày ví dụ về vị trí cần phải thực hiện check error.

* Cân nhắc đến overflow

/\* Vùng bên trái: buffer size Vùng bên phải: độ dài data thực \*/ /\* Confirm xem data thực có lớn hơn buffer size không\*/

if ( s\_Dlt\_MBuff\_RemainSize < ( message\_length + header\_length + 1U ) ) {

/\* Xử lý abnormal \*/

}

* Bảo vệ phép chia 0

/\* data:Mẫu số Division: Thương số \*/

/\* Confirm xem mẫu số có phải là 0không \*/

if ( 0U == data ) {

/\* Trường hợp mẫu số là 0 thì không tính \*/

Division = 0;

} else {

/\* Chỉ trong trường hợp mẫu số khác 0 thì mới tính (xử lý bình thường) \*/

Division = 100 / data;

}

* Phép toán pointer phải là kết quả mong muốn Về vị trí đang thực hiện phép toán pointer, thì kiểm tra xem có thể thực hiện xử lý mong muốn khi review code không.

Thể hiện ví dụ như sau. ptr= buff; uint8 buff[10]; uint8 offset = 5; uint8\* ptr = buff;

buff[6]

buff[7]

buff[8]

buff[9]

buff[0]

buff[1]

buff[2]

buff[3]

buff[4]

buff[5]

ptr += offset; ptr += offset;

Trường hợp này, khi review code cần phải kiểm tra xem xử lý ptr chỉ vào buff[5] có đúng không.

* Check parameter của hàm Chi tiết vui lòng tham khảo mục 『9.3.3 引数のエラーチェック (Check error đối với đối số)』  Không truy cập vào pointer NULL

/\* Confirm xem address không phải NULL đúng không, khi sử dụng pointer \*/

if ( NULL == PduInfoPtr ) {

/\* Xử lý abnormal \*/

}

* Đảm bảo cho chỉ số dưới (subscript) của mảng không bị âm hoặc nằm ngoài phạm vi Ví dụ check sao cho chỉ số dưới (subscript) của mảng không bị âm.

uint8 buff[200];

/\* Confirm xem subscript có lớn hơn hoặc bằng 0 không \*/

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

/\* Xử lý bình thường \*/

buff[i] = TRUE;

}

Ví dụ check để đảm bảo chỉ số dưới (subscript) của mảng không bị nằm ngoài phạm vi.

/\* CANIF\_Q\_NUM\_OF\_TRCV : số phần tử của mảng \*/

CanIf\_TrcvInfoType s\_TrcvInfo[CANIF\_Q\_NUM\_OF\_TRCV];

/\* Confirm xem subscript có nhỏ hơn số phần tử hay không \*/

for ( i=(uint32\_least)0x0U; i < CANIF\_Q\_NUM\_OF\_TRCV; i++ ) {

/\* Xử lý bình thường \*/

s\_TrcvInfo[i].WakeupValidationCtrl = FALSE;

}

 Chi tiết không bảo lưu bộ nhớ động vui lòng tham khảo trong mục 『9.10 動的なメモリ確保禁止/Cấm bảo lưu bộ nhớ động』

#### 9.3.2 Check error cho hàm trả về thông tin error

Trường hợp hàm trả về thông tin error, có khả năng sẽ phát sinh lỗi bên trong. Do đó,

ở xử lý gọi hàm trả về thông tin error, thì thực hiện các kiểm tra thích hợp đối với giá trị output.

【MISRA-C:2012 Dir】 4.7 (Necessary)

Dưới đây trình bày ví dụ thực hiện check error thích hợp đối với giá trị trả về.

/\* Giá trị trả về của Func1 bao gồm thông tin error(= E\_NOT\_OK) \*/

Std\_ReturnType Func1 ( uint8 Num );

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

Std\_ReturnType retCode = E\_OK;

retCode = Func1( num );

/\* Check xem giá trị trả về của hàm Func1 có bao gồm thông tin error hay không \*/

if ( E\_NOT\_OK == retCode ) {

/\* Xử lý abnormal \*/

}

/\* Giá trị trả về của Func2 là pointer \*/

uint8\* Func2 ( uint8 Num );

void main( void ) { uint8 num = 0x01U; uint8\* retPtr = NULL;

retCode = Func2( num );

/\* Check xem address của giá trị trả về của hàm Func2 có phải là NULL hay không \*/ if ( NULL == retCode ) {

/\* Xử lý abnormal \*/

}

}

Dưới đây trình bày ví dụ thực hiện phán định thích hợp đối với đối số. /\* Pointer của đối số thuộc Func3 bao gồm thông tin error\*/

void Func3 ( uint8\* Status );

void main( void ) { uint8 status;

Func3( &status );

/\* Check xem pointer của đối số thuộc hàm Func3 có bao gồm thông tin error hay không \*/

if ( XXX\_ERROR\_STATUS == status ) {

/\* Xử lý abnormal \*/

}

/\* Trường hợp pointer address của đối số thuộc Func4 là NULL\*/

void Func4 ( uint8\* Ptr );

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

/\* Check xem pointer address của đối số thuộc hàm Func4 là NULL hay không \*/

if ( NULL == ptr ) {

/\* Xử lý abnormal \*/

}

#### 9.3.3 Check error đối với đối số

Thực hiện check tính hợp lệ của giá trị input khi bắt đầu các xử lý API (trừ hàm nội bộ).

【MISRA-C:2012 Dir】 4.14 (Necessary)

(1) Trường hợp confirm tính hợp lệ của giá trị input bằng giá trị biên

void Func1 ( uint8 Number) {

/\* Check xem đối số có vượt mức giá trị ngưỡng không \*/

if ( NUM\_OF\_CONFIG < Number ) {

/\* Xử lý abnormal \*/

}

(2) Trường hợp confirm xem pointer của giá trị input có phải NULL không

void Func2 ( uint8\* Ptr) {

/\* Check xem đối số không phải NULL đúng không \*/

if ( NULL == Ptr) {

/\* Xử lý abnormal \*/

}

(3) Trường hợp giá trị input là kiểu liệt kê: cần phải thực hiện check error cho trường hợp gọi hàm bằng giá trị số chưa được setting theo kiểu liệt kê. typedef enum {

BSWM\_FALSE = ( 0U ),

BSWM\_TRUE,

BSWM\_UNDEFINED

} BswM\_Q\_RuleStateType;

void Func2 ( BswM\_Q\_RuleStateType Status ) {

switch ( Status ) { case BSWM\_FALSE:

/\* Xử lý\*/

break; case BSWM\_TRUE:

/\* Xử lý\*/

break; case BSWM\_UNDEFINED: /\* Xử lý \*/ break;

default:

/\* Xử lý abnormal trong trường hợp đối số không phải là kiểu liệt kê đang được khai báo \*/ break;

}

## 9.4 Cấm sử dụng hợp ngữ (assembly language)

Không được sử dụng hợp ngữ do nó là ngôn ngữ phụ thuộc compiler (trình biên dịch).

Trường hợp bất đắc dĩ phải sử dụng hợp ngữ, phải tách biệt file sử dụng hợp ngữ với file sử dụng ngôn ngữ C, thành 1 file riêng biệt, để làm rõ đối tượng cần phải viết lại khi di chuyển. (Cấm Assembler-Chương trình dịch hợp ngữ Inline )

【MISRA-C:2012 Dir】 4.3 (Necessary)

Ngoài ra, nếu sử dụng hợp ngữ, thì mô tả lý do sử dụng hợp ngữ, compiler đang phụ thuộc vào source file dưới dạng comment.

【MISRA-C:2012 Dir】 4.2 (Recommended)

## 9.5 Cấm sử dụng kiểu cơ bản

Có khả năng kích cỡ kiểu sẽ khác nhau tùy theo hệ thống xử lý. Do đó, không sử dụng trực tiếp kiểu cơ bản (chẳng hạn như "int"). Thay vào đó, sử dụng typedef (chẳng hạn như "uint8") biểu thị có/không có ký hiệu và kích cỡ trong cái nhìn đầu tiên.

【MISRA-C:2012 Dir】 4.6 (Recommended)

Trình bày ví dụ cụ thể về kiểu bằng machine 32 bit như sau.

Bảng 2 Kiểu dữ liệu khi sử dụng machine 32 bit

|  |  |  |  |
| --- | --- | --- | --- |
| typedef | Tên kiểu | typedef | Tên kiểu |
| 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 |

※ Phù hợp với ECU sử dụng.

Ngoài ra, biến vòng lặp sử dụng kiểu đã được định nghĩa theo least để tăng hiệu quả tốc độ xử lý.

（Được định nghĩa trong SWS\_Platform)

## 9.6 Chú ý khi tạo hàm và macro hàm

Trường hợp sử dụng hàm hay macro có định dạng hàm, thì phải xem xét xem hàm hay macro định dạng hàm nào sẽ phù hợp để xử lý các vị trí tương ứng.

【MISRA-C:2012 Dir】 4.9 (Recommended)

Phải là code phù hợp với phương châm ghi dưới đây.

* Trường hợp có nhiều vị trí được gọi thì hàm thích hợp hơn macro hàm.
* Trường hợp chú trọng tốc độ thì macro hàm thích hợp hơn hàm.

※ Giá trị bằng số cụ thể thì chúng tôi sẽ xem xét riêng.

## 9.7 Chú ý khi sử dụng pointer

Khi sử dụng pointer thì các biến kiểu pointer đang sử dụng sẽ trở nên rõ ràng hơn, và có thể giải thích được tính cần thiết và tính an toàn.

Ngoài ra, cần phải xem xét xem nên khai báo pointer kiểu không hoàn chỉnh (in-complete type) hay nên khai báo pointer kiểu hoàn chỉnh (complete type) ?

【MISRA-C:2012 Dir】 4.8 (Recommended)

Trình bày ví dụ về pointer kiểu không hoàn chỉnh và pointer kiểu hoàn chỉnh như sau:

struct ImpfType; /\* Kiểu không hoàn chỉnh\*/

struct ImpfType\* ptr1; /\* Pointer chỉ kiểu hoàn chỉnh = Pointer kiểu không hoàn chỉnh \*/

uint8\* ptr2; /\* Pointer chỉ kiểu hoàn chỉnh = Pointer kiểu hoàn chỉnh \*/

Khi thay thế address cho pointer nói trên, pointer kiểu không hoàn chỉnh (ptr1 thì không thể tham chiếu đến member của address đã thay thế, còn pointer kiểu hoàn chỉnh thì có thể tham chiếu đến member của address đã thay thế (ptr2). Có thể che kiểu từ bên ngoài bằng cách sử dụng pointer không hoàn chỉnh cho xử lý chỉ thực hiện chuyển address.

## 9.8 Cấm multiple include

Trường hợp multiple include,

do sợ là kết quả không chắc chắn hoặc bị sai nên không thực hiện multiple include. Theo đó, multiple include đang bị cấm.

Trường hợp sử dụng khi xử lý memory mapping thì thực hiện multiple include cũng được.

【MISRA-C:2012 Dir】 4.10 (Necessary)

Trình bày ví dụ cấm multiple include đối với header file như sau.

#ifndef BSWM\_H

#define BSWM\_H

/\* Xử lý \*/

#endif /\* BSWM\_H \*/

Trình bày ví dụ multiple include khi xử lý memory mapping, là ví dụ ngoại lệ (exception) như sau.

#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 Cấm sử dụng standard library

Không được sử dụng standard library. Do đó, không được bao gồm header file của standard library.

【MISRA-C:2012 Dir】 4.11 (Necessary)

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

## 9.10 Cấm cấp phát bộ nhớ và object động

Tất cả object đều được đặt tĩnh, không có object nào được sinh ra chỉ trong khoảng thời gian cụ thể.

Không được cấp phát bộ nhớ động.

【MISRA-C:2012 Dir】 4.12 (Necessary)

Confirm việc cấm các nội dung dưới đây và không cấp phát bộ nhớ động khi review code.

* Sử dụng các hàm standard library dùng để thực hiện phân bổ bộ nhớ động

Ví dụ tiêu biểu: hàm calloc, malloc, realloc và free

* Không sử dụng hàm standard library, phân bổ bộ nhớ động bằng cách tự tạo. (bao gồm package của bên thứ 3)

## 9.11 Cấm thao tác đối với file

Không thực hiện thao tác đối với file.

【MISRA-C:2012 Dir】 4.13 (Recommended)

【MISRA-C:2012 Rule】 22.3 (Necessary), 22.4, 22.6 (Obligatory)

Cấm thực hiện xử lý như sau.

・Phân bổ resource (ví dụ: mở file)

・Hủy phân bổ resource (ví dụ: đóng file)

・Thao tác khác (ví dụ: đọc từ file)

― Hết ―