BOSCH

C Coding Rules for ASIL Software

Docupedia Export

Author:Pham Minh Nhat (MS/EJV51-PS)
Date:25-Jul-2024 19:24

Invented for life

Table of Contents

MANDATORY RELEASED

1 Table of contents

2 Overview

Former CM-CI1 had its own C/C++ coding guideline and CM-CI2 had its own C language coding guideline. This page is the outcome of an exercise to merge these and have one common unified coding guideline for C language at
Bosch Cross-Domain Computing Solutions Cockpit Technologies (XC-CT)

Note :

1) RBCM_SAFE_C_RULE_<Number> rules must be applied for components who have ISO 26262 requirements
2) MISRA C 2012 C Coding Rules must be applied in a project if it is having ASIL code.

References

Former CM-CI1 C/C++ Rules: https://sites.inside-share.bosch.com/sites/074985/Documents/50.Design_Rules_(DR)/03.Software_DGL_[germELL])/ELL_SW_PC_1_2_e.pdf
Former CM_CI2 C Rules: https://inside-ilm.bosch.com/irj/go/km/docs/versioning/.~system~/versions/56/32/51105815873256/C_CodingGuidelines(7).pdf
Coding rule index for the CM-CI1 and CM-CI2 rules merging activity: CodingRules-v0.5-AfterReview.xlsx

3 C Coding Rules for Software with ISO 26262 requirements at Bosch Cross-Domain Computing Solutions Cockpit Technologies (XC-CT)

Rule number in Proposed C-Coding Standards	CI1 Rule Statement	CI2 Rule Statement (if present)	Current Automation Status/ Remarks
RBCM_SAFE_C_RULE_099	Safety Critical Variable shall be identified in such a manner that they can be readily distinguished from non-safety critical variables.	If a variable changes from non-critical to safety critical because of re-use for different application, necessary change shall be made to meet this requirement.	manual / ISO26262 requirement
RBCM_SAFE_C_RULE_100	Each safety critical variable shall be protected from corruption throughout its use. It shall be stored with protective error correcting codes or in more than one memory location using diversity techniques, and compared to its redundant copy prior to use.	Protection from corruption is fundamental to the processing of safety critical functions. If critical data becomes corrupted, the result of otherwise correct critical processing can be hazardous system behavior.	manual / ISO26262 requirement
RBCM_SAFE_C_RULE_101	Related critical variables should not be grouped into data elements which utilize a common based address of the variables to result in a common mode system failure involving all elements of the structure. If they are grouped, it shall be properly justfied for safety critical functions.	Use of structures, arrays, linked lists, etc., may allow errors in the base address of the variables to result in a common mode system failure involving all elements of the structure	manual / ISO26262 requirement

RBCM_SAFE_C_RULE_102	All safety critical code shall be protected for corruption. Access to critical variables by non-critical functions shall be read only.	Safety critical code shall be protected from corruption throughout its use. As a minimum, the safety critical code shall be stored with protective error correcting codes or in more than one memory location using diversity techniques.	manual / ISO26262 requirement
RBCM_SAFE_C_RULE_103	Modifying code shall not be allowed as a part of any software executing on a safety critical system. The system shall prevent unauthorized or inadvertent access to the safety critical software and object code.		manual / ISO26262 requirement
RBCM_SAFE_C_RULE_104	Program execution including Interrupt handling shall be deterministic and controlled under all conditions. If interrupts are prioritized, completion of all interrupts or equivalent protection must be ensured prior to return to interrupted process. Improper return from interrupt must be detected and mitigated. Checks to ensure that interrupts are not blocked due to improper return from a higher priority interrupt must be provided. All interrupts must be subject to detection and correction if not completed within a predetermined time period. Positive protection of interrupted processes from resource starvation shall be provided.		manual / ISO26262 requirement
RBCM_SAFE_C_RULE_105	Software controlled safety critical sequences shall be monitored against inadvertent activation or illegal sequencing by using checks to ensure correct flow to critical modules and along critical paths.		manual / ISO26262 requirement
RBCM_SAFE_C_RULE_106	Non-safety code shall not be executed in safe task context. Function calls from safety code to non-safety code are prohibited. Whenever it is required to call non-safety functions from safety code a context switch shall be used. This rule added to restrict execution of QM code when CPU is in supervisor mode.		manual / ISO26262 requirement

4 MISRA C 2012 C Coding Rules to be followed for projects having ASIL code at Bosch Cross-Domain Computing Solutions Cockpit Technologies (XC-CT)

Rule / Directive Number	**Rule Statement**	Remarks
RBCM_MISRA_C_DIR_1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	
RBCM_MISRA_C_DIR_2.1	All source files shall compile without any compilation error	
RBCM_MISRA_C_DIR_3.1	All code shall be traceable to documented requirements	
RBCM_MISRA_C_DIR_4.1	Run-time failures shall be minimized	C12's rule gives explicit way of avoiding this: Install dynamic runtime checks for better error Examine scope of application of variables (prev Check valid objective of pointers Prevention of the loss of the most significant bi Examine validity of the array indication Possibly meaningful only before the release Equivalent rule number: RBCM_C_RECO_059.
RBCM_MISRA_C_DIR_4.2	All usage of assembly language should be documented	

RBCM_MISRA_C_DIR_4.3	Assembly language shall be encapsulated and isolated	CI1's rule statement is: "Write machine depend CI2's statement is: "Encapsulated functions: Or We keep the CI1's statement instead of the MIS
RBCM_MISRA_C_DIR_4.4	Sections of code should not be “commented out”	CI1's rule says: "Source code must be comment commented out) must be removed"
RBCM_MISRA_C_DIR_4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	
RBCM_MISRA_C_DIR_4.6	*typedefs* that indicate size and signedness should be used in place of the basic numerical types	The CI2's 15.2 gives a nice example of what mig Additionally, MISRA Directive 4.6 says this: "typedefs that indicate size and signedness sho Hence, the rephrased rule statement has both,
RBCM_MISRA_C_DIR_4.7	If a function returns error information, then that error information shall be tested	
RBCM_MISRA_C_DIR_4.8	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	
RBCM_MISRA_C_DIR_4.9	A function should be used in preference to a *function-like macro* where they are interchangeable	
RBCM_MISRA_C_DIR_4.10	Precautions shall be taken in order to prevent the contents of a *header file* being included more than once	
RBCM_MISRA_C_DIR_4.11	The validity of values passed to library functions shall be checked	CI2's original statement: "The validity of the val e.g.: Check for exceeding the limit value, zero-p
RBCM_MISRA_C_DIR_4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence	

RBCM_MISRA_C_DIR_4.14	The validity of values received from external sources shall be checked	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_1.1	The program shall contain no violations of the standard C syntax and *constraints*, and shall not exceed the implementation's translation Limits	
RBCM_MISRA_C_RULE_1.2	Language extensions should not be used	
RBCM_MISRA_C_RULE_1.3	There shall be no occurrence of undefined or critical unspecified behaviour	
RBCM_MISRA_C_RULE_2.1	A project shall not contain *unreachable code*	
RBCM_MISRA_C_RULE_2.2	There shall be no *dead code*	
RBCM_MISRA_C_RULE_2.6	A function should not contain unused label declarations	
RBCM_MISRA_C_RULE_3.1	The character sequences /* and // shall not be used within a comment	
RBCM_MISRA_C_RULE_3.2	Line-splicing shall not be used in // comments	
RBCM_MISRA_C_RULE_4.1	Octal and hexadecimal escape sequences shall be terminated	MISRA is more specific in this aspect, and expilc statement.
RBCM_MISRA_C_RULE_4.2	Trigraphs should not be used	
RBCM_MISRA_C_RULE_5.8	Identifiers that define objects or functions with external linkage shall be unique	
RBCM_MISRA_C_RULE_6.1	Bit-fields shall only be declared with an appropriate type	
RBCM_MISRA_C_RULE_6.2	Single-bit named bit fields shall not be of a signed type	
RBCM_MISRA_C_RULE_7.1	Octal constants shall not be used	

RBCM_MISRA_C_RULE_7.3	The lowercase character “l” shall not be used in a literal suffix	
RBCM_MISRA_C_RULE_7.4	A string literal shall not be *assigned* to an object unless the object’s type is “pointer to *const*-qualified *char*”	
RBCM_MISRA_C_RULE_8.1	Types shall be explicitly specified	
RBCM_MISRA_C_RULE_8.3	All declarations of an object or function shall use the same names and type qualifiers	CI2's original statement goes like this: "The number and type of parameters while call allowed. Reason: Prevention of errors due to overrun or undefine The rationale is the same as this MISRA rule.
RBCM_MISRA_C_RULE_8.4	A compatible declaration shall be visible when an object or function with external linkage is defined	
RBCM_MISRA_C_RULE_8.8	The *static* storage class specifier shall be used in all declarations of objects and functions that have internal linkage	
RBCM_MISRA_C_RULE_8.6	An identifier with external linkage shall have exactly one external definition	
RBCM_MISRA_C_RULE_8.10	An *inline function* shall be declared with the static storage class	
RBCM_MISRA_C_RULE_8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	
RBCM_MISRA_C_RULE_8.14	The *restrict* type qualifier shall not be used	
RBCM_MISRA_C_RULE_9.1	The value of an object with automatic storage duration shall not be read before it has been set	CI1's rule simply says: "All variables must be ini
RBCM_MISRA_C_RULE_9.2	The initializer for an aggregate or union shall be enclosed in braces	

RBCM_MISRA_C_RULE_9.3	Arrays shall not be partially initialized	
RBCM_MISRA_C_RULE_9.4	An element of an object shall not be initialized more than once	
RBCM_MISRA_C_RULE_9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	
RBCM_MISRA_C_RULE_10.2	Expressions of *essentially character type* shall not be used inappropriately in addition and subtraction operations	
RBCM_MISRA_C_RULE_10.3	The value of an expression shall not be assigned to an object with a narrower *essential type* or of a different *essential type category*	Original CI2's statement is: "Every object (e.g. o also specifically mentions that only narrower ty
RBCM_MISRA_C_RULE_10.4	Both operands of an operator in which the *usual arithmetic conversions* are performed shall have the same *essential type category*	
RBCM_MISRA_C_RULE_10.5	The value of an expression should not be cast to an inappropriate *essential type*	
RBCM_MISRA_C_RULE_10.8	The value of a *composite expression* shall not be cast to a different *essential type category* or a wider *essential type*	
RBCM_MISRA_C_RULE_11.1	Conversions shall not be performed between a pointer to a function and any other type	
RBCM_MISRA_C_RULE_11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type	
RBCM_MISRA_C_RULE_11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type	
RBCM_MISRA_C_RULE_11.4	A conversion should not be performed between a pointer to object and an integer type	
RBCM_MISRA_C_RULE_11.5	A conversion should not be performed from pointer to *void* into pointer to object	
RBCM_MISRA_C_RULE_11.6	A cast shall not be performed between pointer to *void* and an arithmetic type	

RBCM_MISRA_C_RULE_11.7	A cast shall not be performed between pointer to object and a noninteger arithmetic type	
RBCM_MISRA_C_RULE_11.8	A cast shall not remove any *const* or *volatile* qualification from the type pointed to by a pointer	
RBCM_MISRA_C_RULE_11.9	The macro NULL shall be the only per mitted form of integer *null pointer constant*	
RBCM_MISRA_C_RULE_12.1	The precedence of operators within expressions should be made explicit	CI1's rule 31 says: "Part expessions are to be pu CI1's recommendation 51 says: Do not assume that the operators in an express Both of these are covered by this MISRA rule.
RBCM_MISRA_C_RULE_12.2	The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the *essential type* of the left hand operand	
RBCM_MISRA_C_RULE_12.3	The comma operator should not be used	
RBCM_MISRA_C_RULE_12.4	Evaluation of *constant expressions* should not lead to unsigned integer wrap-around	
RBCM_MISRA_C_RULE_12.5	The sizeof operator shall not have an operand which is a function parameter declared as “array of type”	New rule in MISRA C:2012 Amendment 1.
RBCM_MISRA_C_RULE_13.1	*Initializer lists* shall not contain *persistent side effects*	
RBCM_MISRA_C_RULE_13.2	The value of an expression and its *persistent side effects* shall be the same under all permitted evaluation orders	
RBCM_MISRA_C_RULE_13.4	The result of an assignment operator should not be *used*	
RBCM_MISRA_C_RULE_13.5	The right hand operand of a logical && or || operator shall not contain *persistent side effects*	

RBCM_MISRA_C_RULE_13.6	The operand of the *sizeof* operator shall not contain any expression which has potential *side effects*	
RBCM_MISRA_C_RULE_14.1	A *loop counter* shall not have *essentially floating* type	
RBCM_MISRA_C_RULE_14.2	A *for* loop shall be well-formed	CI1's rule says: "No values must be assigned to The MISRA rule is more clearer. Almost always t disallows that. Hence, MISRA rule that specifica
RBCM_MISRA_C_RULE_14.4	The controlling expression of an *if* statement and the controlling expression of an *iteration-statement* shall have *essentially Boolean* type	CI1's rule 43 says: "Do not write any logical expr CI1's rule 82 says: "On data objects of the type t The MISRA rule is more clearer. By having an "e ctypes.h
RBCM_MISRA_C_RULE_15.1	The *goto* statement should not be used	This rule is more like a recommendation (also i
RBCM_MISRA_C_RULE_15.2	The *goto* statement shall jump to a label declared later in the same function	
RBCM_MISRA_C_RULE_15.3	Any label referenced by a *goto* statement shall be declared in the same block, or in any block enclosing the *goto* statement	
RBCM_MISRA_C_RULE_15.5	A function should have a single point of exit at the end	This rule is a recommendation, and we replace
RBCM_MISRA_C_RULE_15.6	The body of an *iteration-statement* or a *selection-statement* shall be a *compound-statement*	
RBCM_MISRA_C_RULE_15.7	All *if … else if* constructs shall be terminated with an *else* statement	CI2's statement: "After a construct from "if" and
RBCM_MISRA_C_RULE_16.1	All *switch* statements shall be well-formed	

RBCM_MISRA_C_RULE_16.2	A *switch label* shall only be used when the most closely-enclosing compound statement is the body of a *switch* statement	
RBCM_MISRA_C_RULE_16.3	An unconditional *break* statement shall terminate every *switch-clause*	
RBCM_MISRA_C_RULE_16.4	Every *switch* statement shall have a *default* label	
RBCM_MISRA_C_RULE_16.5	A *default* label shall appear as either the first or the last *switch label* of a *switch* statement	
RBCM_MISRA_C_RULE_16.6	Every *switch* statement shall have at least two *switch-clauses*	
RBCM_MISRA_C_RULE_16.7	A *switch-expression* shall not have *essentially Boolean type*	
RBCM_MISRA_C_RULE_17.1	The features of <stdarg.h> shall not be used	
RBCM_MISRA_C_RULE_17.3	A function shall not be declared implicitly	
RBCM_MISRA_C_RULE_17.4	All exit paths from a function with non-*void* return type shall have an explicit *return* statement with an expression	
RBCM_MISRA_C_RULE_17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	
RBCM_MISRA_C_RULE_17.6	The declaration of an array parameter shall not contain the *static* keyword between the [ ]	
RBCM_MISRA_C_RULE_17.7	The value returned by a function having non-*void* return type shall be *used*	
RBCM_MISRA_C_RULE_17.8	A function parameter should not be modified	
RBCM_MISRA_C_RULE_18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand	CI1's rule puts a blanket avoidance of pointer-a

RBCM_MISRA_C_RULE_18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array	
RBCM_MISRA_C_RULE_18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object	
RBCM_MISRA_C_RULE_18.6	The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist	CI1's rule was: "A function must never return a The MISRA rule protects the rationale better tha scope: so, a "return" from a function is just one automatic variable within a function.
RBCM_MISRA_C_RULE_18.7	Flexible array members shall not be declared	
RBCM_MISRA_C_RULE_18.8	Variable-length array types shall not be used	
RBCM_MISRA_C_RULE_19.1	An object shall not be assigned or copied to an overlapping object	
RBCM_MISRA_C_RULE_20.1	*#include* directives should only be preceded by pre-processor directives or comments	
RBCM_MISRA_C_RULE_20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a *header file* name	
RBCM_MISRA_C_RULE_20.3	The *#include* directive shall be followed by either a <filename> or "filename" sequence	
RBCM_MISRA_C_RULE_20.4	A macro shall not be defined with the same name as a keyword	
RBCM_MISRA_C_RULE_20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument	

RBCM_MISRA_C_RULE_20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	CI1's rule says: "A paramter list following the m functions, but anywhere such macro expansion
RBCM_MISRA_C_RULE_20.8	The controlling expression of a *#if* or *#elif* preprocessing directive shall evaluate to 0 or 1	
RBCM_MISRA_C_RULE_20.9	All identifiers used in the controlling expression of *#if* or *#elif* preprocessing directives shall be *#define*'d before evaluation	
RBCM_MISRA_C_RULE_20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	
RBCM_MISRA_C_RULE_20.12	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	
RBCM_MISRA_C_RULE_20.13	A line whose first token is # shall be a valid preprocessing directive	
RBCM_MISRA_C_RULE_20.14	All *#else*, *#elif* and *#endif* preprocessor directives shall reside in the same file as the *#if*, *#ifdef* or *#ifndef* directive to which they are related	
RBCM_MISRA_C_RULE_21.1	*#define* and *#undef* shall not be used on a reserved identifier or reserved macro name	
RBCM_MISRA_C_RULE_21.2	A reserved identifier or macro name shall not be declared	CI2's statement is: "Reserved words and names Since MISRA rule does not cover the discarded
RBCM_MISRA_C_RULE_21.4	The standard *header file* <setjmp.h> shall not be used	CI2's statement is: "The functions setjmp and lo
RBCM_MISRA_C_RULE_21.5	The standard *header file* <signal.h> shall not be used	
RBCM_MISRA_C_RULE_21.6	The Standard Library input/output functions shall not be used	CI2's statement: "The I/O-LIB stdio.h is not used

RBCM_MISRA_C_RULE_21.7	The *atof*, *atoi*, *atol* and *atoll* functions of <stdlib.h> shall not be used	CI2's statement: "The functions atof, atoi and a
RBCM_MISRA_C_RULE_21.9	The library functions *bsearch* and *qsort* of <stdlib.h> shall not be used	
RBCM_MISRA_C_RULE_21.10	The Standard Library time and date functions shall not be used	CI2's statement is: "The time handling function
RBCM_MISRA_C_RULE_21.11	The standard *header file* <tgmath.h> shall not be used	
RBCM_MISRA_C_RULE_21.12	The exception handling features of <fenv.h> should not be used	
RBCM_MISRA_C_RULE_21.13	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.14	The Standard Library function memcmp shall not be used to compare null terminated strings	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.15	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.16	The pointer arguments to the Standard Library function memcmp shallpoint to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type or an essentially enum type	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.17	Use of the string handling functions from <string.h> shall not resultin accesses eyond the bounds of the objects referenced by their pointer parameters	New rule inMISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.18	The size_t argument passed to any function in <string.h> shall have an appropriate value	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.19	The pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they have pointer to const-qualified type	New rule in MISRA C:2012 Amendment 1
RBCM_MISRA_C_RULE_21.20	The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function	New rule in MISRA C:2012 Amendment 1

RBCM_MISRA_C_RULE_22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	CI1's rule statement was: "You should not alloc The MISRA rule broadens the scope of this rule,
RBCM_MISRA_C_RULE_22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function	
RBCM_MISRA_C_RULE_22.3	The same file shall not be open for read and write access at the same time on different streams	
RBCM_MISRA_C_RULE_22.4	There shall be no attempt to write to a stream which has been opened as read-only	
RBCM_MISRA_C_RULE_22.5	A pointer to a FILE object shall not be dereferenced	
RBCM_MISRA_C_RULE_22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed	