MISRA-C Compliance

Version 1.00.05

2018-07-30

Application Note AN-ISC-8-1225

Author Markus Schwarz

Restrictions Customer Confidential – Vector decides

Abstract This document explains the topic of MISRA-C and HIS Metric compliance for the product MICROSAR4.
It also contains the MISRA and code metric product deviations including their justifications.

Table of Contents

1 Overview

1.1 Purpose

This document explains the topic of MISRA-C compliance for the product MICROSAR4.

It is based on the compliance concept defined in [MISRA-Compliance].

1.2 MICROSAR4

The product MICROSAR4 is developed in a product-line approach, i.e. a large set of reusable software components is developed to fulfill the functional needs of different OEMs and TIER1s for various ECUs.

These components are

> developed with a process based on AUTOMOTIVE SPICE level 3 and ISO26262.
> developed based on code guidelines and templates.
> analyzed for MISRA violations and regarding code metrics.

The product has a high configurability, so MISRA and/or code metric deviations may occur in one to many specific setups.

The product MICROSAR4 is mapped to the wording **project** from [MISRA-C:2012].

1.3 MISRA-C

The [MISRA-C:2012] guidelines define a subset of the C language to reduce the possibility to make mistakes.

2 MISRA-C Compliance

This chapter explains

- what we want to achieve regarding MISRA (chapter 2.1)
- what we do to achieve this (chapter 2.2)
- which guidelines we apply (chapter 2.3)
- how we check these guidelines (chapter 2.4)
- how we document the violations and deviations (chapter 2.5)
- our recommendations what you should do at least (chapter 2.6).

2.1 Goal

The product MICROSAR4 shall comply with [MISRA-C:2012], including [MISRA-AMD1] and [MISRA-TC1].

Deviations from these guidelines are possible, but shall be prevented whenever possible.

Hint: We switched from [MISRA-C:2004] to the 2012 version in July 2018. While most of the components are made compliant with [MISRA-C:2012], some components of the product might still comply with the 2004 standard.

2.2 Compliance Process

We check our code regularly within each development cycle to detect violations of MISRA guidelines. These checks are done both for the components and exemplary integrated product variants.

When we detect violations, we analyze the code and, if reasonable, change/extend the code to be compliant.

If code modification is not reasonable, we use deviations: We mark the violation as known (=suppress the tool message) and add a justification to the code. This justification references the deviation record. We differ between product deviations and component-specific deviations (see sub chapters).

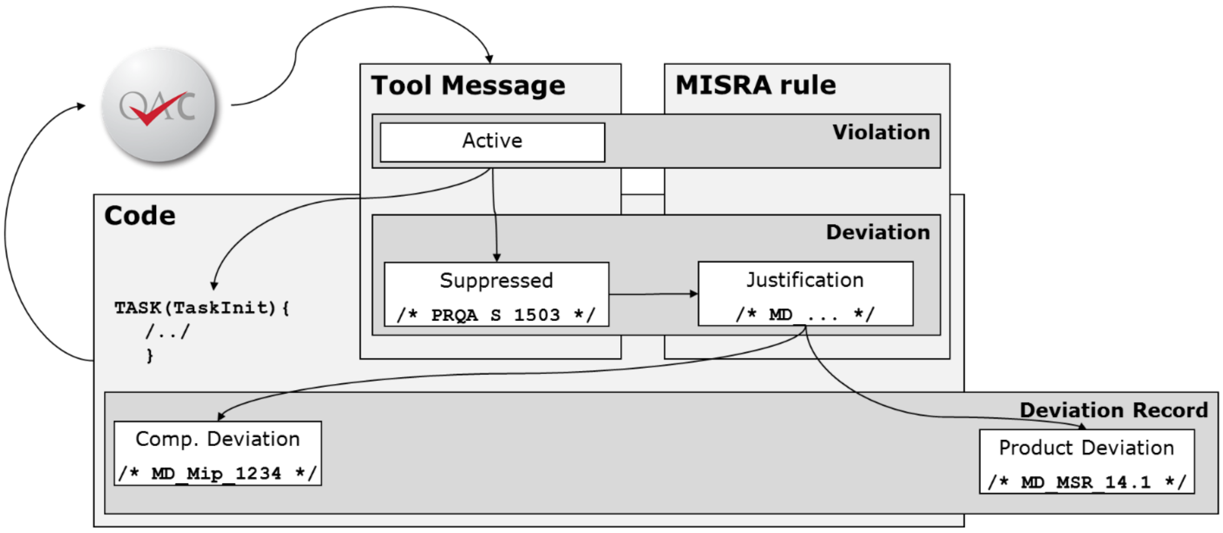

Figure 2-1 Compliance Process

Typical reasons for deviations are:

- Contradicting requirements/standards (e.g. ISO, AUTOSAR, …)
- Code quality (e.g. maintainability, efficiency (RAM/ROM/Runtime), …)
- Access to hardware.

2.2.1 Product Deviations

These are common deviations that might occur throughout many components of our product. (This is different from generally ignored rules!)

The product deviation records are listed within this document (Appendix 4.1).

They are agreed and released within the product development process.

If applicable they can be used to justify violations in the components.

2.2.2 Component-Specific Deviations

These are deviations that are component-specific.

They are documented in the source code using this template:

```
/* Justification for module-specific MISRA deviations:

MD_<CompName>_<RuleNo>
 Reason:     <Explanation of reason for deviation >
 Risk:       <Explanation of risk>
 Prevention: <Explanation how to prevent risk occurrence>
*/
```

They are reviewed and agreed within the code review.

2.3 Guideline Selection

We apply mostly guidelines of [MISRA-C:2012] and [MISRA-AMD1], although most guidelines from the Amendment deal with topics that are typically not applicable for our code.

Guidelines that are inactive are listed below:

Guideline	MISRA category	Revised category	Comment
MISRA2012-Rule2.5	Advisory	Inactive	Our product design uses a concept to provide macro abstraction. These macros are provided even if they might not be used.

Table 2-1 Guideline re-categorization plan

2.4 Guideline Enforcement

The following table shows how the applied MISRA guidelines are enforced.

Guideline	Title	Enforcement
MISRA2012-Dir2.1	All source files shall compile without any compilation errors	Build steps on component, product and delivery level.
MISRA2012-Dir3.1	All code shall be traceable to documented requirements	Review of work products, process consistency check
MISRA2012-Dir4.4	Sections of code should not be "commented out"	Code inspection
MISRA2012-Dir4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Code inspection
MISRA2012-Dir4.6	typedefs that indicate size and signedness should be used in place of the basic numerical types	Code inspection, QA-C message 5209

Guideline	Title	Enforcement
MISRA2012-Dir4.7	If a function returns error information, then that error information shall be tested	Code inspection
MISRA2012-Dir4.8	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Code inspection
MISRA2012-Dir4.9	A function should be used in preference to a function-like macro where they are interchangeable	Code inspection, QA-C message 3453
MISRA2012-Dir4.11	The validity of values passed to library functions shall be checked	N/A The C standard library shall not be used in embedded software components developed at PES. Our own libraries provide sufficient internal checking in contrast to the C standard library.
MISRA2012-Dir4.12	Dynamic memory allocation shall not be used	Code inspection
MISRA2012-Dir4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence	Code inspection
All others	-	QA-C9.2 (PRQA Framework 2.2.0, M3CM 2.3.0)

Table 2-2 Guideline enforcement plan

2.4.1 Product Configuration

The product is based on [AUTOSAR] and uses the AUTOSAR compiler and platform abstraction.

Our MISRA analysis is done with a typical platform and compiler setting.

2.4.2 Compiler Configuration

The complexity of [AUTOSAR] based software exceeds many levels defined for C90. These limits originate in early C compiler implementations with limited RAM to store definitions. We assume that all current compilers used for the product MIRCOSAR no longer have these limits.

There are many tool messages that deal with the C90/C99 compiler limits or specific behavior. They are typically related to Dir1.1.

While we focus on staying within the C90 limits on component level, the integrated product typically violates them.

The table below highlights in red where we typically violate C90 limits due to the design concept of our product. For those aspects, we use the C99 limits instead.

We further assume that the compiler can handle 127 significant characters.

Potential risks:

> A preprocessor might not correctly pre-process the software what might result in incorrect code.
> A compiler might not correctly translate the software what might result in incorrect code.

Prevention of risk:

> Each delivery is integrated and tested on the real target system.
> The compiler selection and validation has to be done by ECU supplier.

QA-C Message title	C90 Limit	C99 Limit	C90 QA-C ID	C99 QA-C ID
Nesting of parentheses exceeds <limit>	32	63	0410	0375
More than <limit> pointer, array or function declarators modifying a declaration	12	-	0609	-
Nesting of 'struct' or 'union' types exceeds <limit>	15	63	0611	0392
Size of object exceeds <limit> bytes	32767	65535	0612	0613
More than <limit> block scope identifiers defined within a block	127	511	0614	0615
Number of members in 'struct' or 'union' exceeds <limit>	127	1023	0639	0390
Number of enumeration constants exceeds <limit>	127	1023	0647	0391
Nesting of control structures (statements) exceeds <limit>	15	127	0715	0371
Number of 'case' labels exceeds <limit>	257	-	0739	-
External identifier does not differ from other identifier(s) within the specified number of significant characters.	6	31	0777	
Identifier matches other identifier(s) (e.g. '%s') in first <limit> characters	31	63	0778	
Identifier does not differ from other identifier(s) within the specified number of significant characters.	31	-	0779	
Identifier matches other macro name(s)	(31)	(63)	0785	0786
#include causes nesting to exceed <limit> levels	8	15	0810	0388
More than <limit> levels of nested conditional inclusion	8	63	0828	0372
Number of macro definitions exceeds <limit>	1024	4095	0857	0380
Number of macro parameters exceeds <limit>	31	-	0858	-
Number of arguments in macro call exceeds <limit>	31	-	0859	-
String literal exceeds <limit> characters	509	-	0875	-

Table 2-3 QA-C Messages

2.4.3 QA-C Messages Configuration

There are some QA-C messages that are deactivated in our analysis:

QA-C ID	QA-C Message Title	Rationale
0380	Number of macro definitions exceeds 4095	See chapter 2.4.2.
0639	Number of members in 'struct' or 'union' exceeds 127	See chapter 2.4.2.
0715	Nesting of control structures (statements) exceeds 15	See chapter 2.4.2.
0785	Identifier matches other macro name(s) in first 31 characters.	See chapter 2.4.2.
0810	#include causes nesting to exceed 8 levels	See chapter 2.4.2.
0828	More than 8 levels of nested conditional inclusion	See chapter 2.4.2.

0857	Number of macro definitions exceeds 1024	See chapter 2.4.2.
1055	The keyword 'inline' has been used.	For resource optimization, we use inline functions. They are abstracted and can be re-configured by the user.
1503	The function '…' is defined but is not used within this project.	To support configuration variants, we provide various functions that might not be used in each actual configuration.
1531	The object '…' is referenced in only one translation unit - but not the one in which it is defined.	Our components consist of a static and a generated part. The generated part contains the user configuration data. The code in the static part uses this configuration data. While QA-C treats the generated part as own translation unit, in our design both belong to the same component.
1532	The function '…' is only referenced in one translation unit - but not the one in which it is defined.	See 1531
2877	This loop will never be executed more than once.	Loop iterations may depend on user configuration settings, e.g. number of channels. We do not code these variants individually. The used compilers can optimize these code constructs.
3214	The macro '…' is not used and could be removed.	To support configuration variants, we provide various macros that might not be used in each actual configuration.
3432	Simple macro argument expression is not parenthesized.	The AUTOSAR compiler abstraction uses macros where parenthesized arguments would result in compiler errors.

Table 2-4 QA-C Message Configuration

2.5 Compliance Summary Report

Each component is checked for MISRA compliance. The results are stored in an internal component compliance summary named TestReport_MISRA. This report shows the MISRA compliance status and lists the deviations for that component.

2.6 Tasks for Product Users

Here are some hints what you should consider at least when using our product for MISRA-compliant software.

2.6.1 Product

The Platform_Types.h delivered with the product is adapted to the ordered platform. When you use a different Platform_Types.h (e.g. from MCAL package), please verify correct type definition according to compiler and hardware manual.

The Compiler.h delivered with the product is adapted to the ordered compiler. When you use a different Compiler.h (e.g. from MCAL package), please verify correct LOCAL_INLINE assignment.

2.6.2 Compiler Configuration

We strongly recommend validating your software project against compiler limits and configure your own static code checker with concrete limits for the used compiler.

3 Code Metric Compliance

3.1 Goal

The code for the product shall comply with HIS source code metrics [MISRA-C:2012].

3.2 Compliance Process

Like for the MISRA compliance, we analyze the metrics of each component for a set of defined configurations and correct or justify metric deviations.

3.3 Metric Selection

We differ these metric enforcement types

> Supervised: metrics are measured and deviations are justified.
> Measured: metrics are measured but deviations are not justified.

Rationale: Analysis of the HIS source code metrics in the product context has shown that only few of the metrics are helpful to measure and vote the product quality.

We use the metric thresholds defined by HIS.

HIS Metric	HIS Range	QA-C metric	Enforcement Type	Rationale, Comment
Comment Density	>0.2	COMF	Measured	> Comments shall be applied where necessary to improve the readability, not to satisfy a metric. > Annotation comments like the MISRA justifications are also counted as comments → metric target is typically fulfilled but not the intended improvement in maintainability. > We focus on the outcome of code inspection to improve also maintainability concerning good and helpful comments.
Estimated static path count	1..80	STPTH	Supervised	
Cyclomatic Complexity	1..10	STCYC	Supervised	
Number of GOTOs	0	STGTO	Measured	> Our coding guidelines allow use of GOTO in limited use-cases. > Use of GOTO is checked by MISRA rule 15.1. > Mandatory metric STBAK is applied to check for illegal use of GOTO.
Number of functions calling this function	0..5	STM29	Measured	> Our product is designed to provide functionality to an ECU project. The usage of APIs is known, predefined but not under our control. Therefor this metric depends on the concrete user software.
Number of distinct function calls	1..7	STCAL	Supervised	

HIS Metric	HIS Range	QA-C metric	Enforcement Type	Rationale, Comment
Number of function parameters	0..5	STPAR	Supervised	
Number of statements in function	1..50	STST3	Measured	> Our analysis showed that STST3 metric violations correlate with STCYC and STPATH. > We focus on STCYC and STPATH for analysis.
Maximum nesting of control structures	0..4	STMIF	Supervised	
Number of exit points	0..1	STM19	Measured	> Usage of one exit point is defined in our template pattern for C functions. > We focus on the outcome of code inspection to check for correct usage of return statements.
Language set	0..4	VOCF	Measured	> Value for very complex components has been found in HIS range (= ok), but nearly empty components have been reported as dramatically out of range (e.g. >2500). > Summary is, that analysis of metric values in the product context has not shown any significant hint on improving code maintainability or other benefits.
Number of recursions across project	0	STNRA	Measured	> Our coding guidelines forbid the use of recursions to implement functional behavior. > We develop standard software according e.g. AUTOSAR, so API usage definition is mainly out of our scope. > Our code is just a part of a “final” ECU code, so recursions caused by customer-project specific API or configuration data use cannot be detected in our selected product test use-cases. Measurement is done to be aware of potential recursions.
Number of MISRA subset violations	0	NOMV	Measured	> MISRA deviations are already justified per occurrence during the development process of each component. > The components are assembled to many different projects with different configurations resulting in different number of MISRA deviations. Due to the product-line approach and the component-local handling, creating justification

HIS Metric	HIS Range	QA-C metric	Enforcement Type	Rationale, Comment
				above the component level is not reasonable.
Number of MISRA subset violations per rule	0	NOMVRP	Measured	> See comment for NOMV
Stability Index „Si "	0..4		Measured	> Basic idea of this metric is to show that the code has less changes from release to release. This is typically true for a concrete ECU project in its given timeline. The time-boxed product-line development has features of different products developed at the same time and each release is typically "production". In addition, the code consists on a static part and a generated part. The generated part depends on the used configuration database. The reference database is updated with each time-box, too. Due to those effects creating justifications for the product is not reasonable.

Table 3-1 HIS Code Metric Compliance Matrix

4 Appendix: Product Deviation Permits

This appendix lists the approved product deviation permits.

The deviation records are identified by an ID: MD_MSR_<Rule/Msg> (MD=MISRA Deviation)

4.1 MISRA-C:2012 Justification

Deviation ID	MD_MSR_Dir1.1
Violated guideline	Dir1.1: Any implementation-defined behavior on which the output of the program depends shall be documented and understood.
Reason	Operating systems, drivers and flash boot loaders might use special compiler features such as qualifiers and pragmas to directly access hardware. Further such concepts are sometimes required for memory mapping.
Potential risks	The used language extension may not have been fully understood or used incorrectly. This can lead to wrong behavior of hardware dependent software parts.
Prevention of risks	During the code inspection hardware dependent software parts are checked. Each delivery is integrated and tested on the target system where such extensions are used. Compiler selection and validation is done by ECU supplier.
Examples	Declaration of interrupt service routine: `__interrupt void CanTxInterrupt(void);`

Table 4-1 MD_MSR_Dir1.1

Deviation ID	MD_MSR_Pragma
Violated guideline	Dir1.1 Any implementation-defined behaviour on which the output of the program depends shall be documented and understood QA-C message 3116: Unrecognized #pragma arguments.
Reason	The AUTOSAR memory mapping concept uses #pragma directives to create an optimized layout of the target platform's memory.
Potential risks	The used language extension may not have been fully understood or used incorrectly. This can lead to wrong behavior of hardware dependent software parts.
Prevention of risks	During code inspection the memory mapping sections are checked against the memory qualifier defined in the compiler abstraction. Each delivery is integrated and tested on the real target system where such extensions are used. Compiler selection and validation is done by ECU supplier.
Examples	allocate constant data to CONST segment: `#pragma memory=constseg(CONST_DATA) :far`

Table 4-2 MD_MSR_Pragma

Deviation ID	MD_MSR_FctLikeMacro
Violated guideline	Dir4.9: A function should be used in preference to a function-like macro where they are interchangeable
Reason	Function-like macros are used to fulfill the required code efficiency.
Potential risks	The code is difficult to understand or may not work as expected.
Prevention of risks	Code inspection and test of the different variants in the component test.

Examples	`#define Com_IsDirectOfTxModeTrue(Index)`
	`    (COMDIRECTOFTXMODETRUE_MASK ==`
	`    (Com_GetMaskedBitsOfTxModeTrue(Index) &`
	`    COMDIRECTOFTXMODETRUE_MASK))`

Table 4-3 MD_MSR_FctLikeMacro

Deviation ID	MD_MSR_MacroArgumentEmpty
Violated guideline	Dir1.1 Any implementation-defined behaviour on which the output of the program depends shall be documented and understood QA-C message 0850: Macro argument is empty.
Reason	AUTOSAR SWS Compiler Abstraction allows configuration of standard addressing mode by #define'ing an empty compiler abstraction qualifier in Compiler_Cfg.h, e.g. AUTOMATIC or TYPEDEF.
Potential risks	None.
Prevention of risks	None.

Examples	In Compiler_Cfg.h:
	`#define EEP_CONST`
	
	In Eep.c (implementation of component):
	`CONST(uint8, EEP_CONST) Eep_Status;`

Table 4-4 MD_MSR_MacroArgumentEmpty

Deviation ID	MD_MSR_Unreachable
Violated guideline	Rule2.1: A project shall not contain unreachable code
Reason	Since the actual usage of the product by the customer is not known, some of the provided interfaces may not be used.
Potential risks	Functions or global variables which are superfluous in a specific configuration may remain undetected which leads to a small resource overhead.
Prevention of risks	Code inspection.
Examples	-

Table 4-5 MD_MSR_Unreachable

Deviation ID	MD_MSR_DummyStmt
Violated guideline	Rule2.2: There shall be no dead code
Reason	To support different kinds of dummy statements, a dummy statement is encapsulated in a macro like MSN_DUMMY_STATEMENT(x). If no dummy statement is required, the macro is empty and only a null-statement remains.
Potential risks	None, because in case of a dummy statement it is expected that there is no side effect.
Prevention of risks	Template based usage restricted to dummy statements.
Examples	`MIP_DUMMY_STATEMENT(errorId);`

Table 4-6 MD_MSR_DummyStmt

Deviation ID	MD_MSR_IdentifierLength MD_MSR_Rule5.1 MD_MSR_Rule5.2
Violated guideline	Rule5.1: External identifiers shall be distinct Rule5.2: Identifiers declared in the same scope and name space shall be distinct
Reason	Within MICROSAR, naming rules to prefix the identifiers are allocating a significant number of characters. Many of the identifiers are generated based on their elements in the OEM-defined configuration files.
Potential risks	A compiler might not correctly pre-process the software what might result in incorrect code.
Prevention of risks	Each delivery is integrated and tested on the real target system. In addition, preprocessors and compilers used in current MICROSAR projects are not expected to suffer from this (historic) limitation. Compiler selection and validation is done by ECU supplier.
Examples	-

Table 4-7 MD_MSR_IdentifierLength

Deviation ID	MD_MSR_Rule8.7
Violated guideline	Rule8.7: Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
Reason	Since the actual usage of the software component by the customer is not known, some of the provided interfaces may not be used.
Potential risks	Functions or global variables which are superfluous in a specific configuration may remain undetected which leads to a small resource overhead. Other software parts in the application could access symbols which should only be used internally.
Prevention of risks	Code inspection.
Examples	-

Table 4-8 MD_MSR_Rule8.7

Deviation ID	MD_MSR_Rule8.13
Violated guideline	Rule8.13: A pointer should point to a const-qualified type whenever possible
Reason	Many specifications like AUTOSAR, ISO, but also OEM specifications … specify the function prototypes without const qualifier. We have to comply with these specifications.
Potential risks	Less precision in describing access rights on interface level may lead to incorrect use of the parameters.
Prevention of risks	The input specifications describe allowed parameter use and code style guide describes how to document parameter usage in the code to reduce risk of wrong usage.
Examples	-

Table 4-9 MD_MSR_Rule8.13

Deviation ID	MD_MSR_AutosarBoolean
Violated guideline	Rule10.1: Operands shall not be of an inappropriate essential type. Rule10.3: The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category. Rule10.4: Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category Rule10.5: The value of an expression should not be cast to an inappropriate essential type QAC message 4304: An expression of 'essentially Boolean' type is being cast to unsigned type. QAC message 4404: An expression of 'essentially Boolean' type is being converted to unsigned type QAC message 4558: An expression of 'essentially unsigned' type is being used as the … operand of this logical operator. QAC message 1881: The operands of this equality operator are expressions of different 'essential type' categories QAC message 1882: The 2nd and 3rd operands of this conditional operator are expressions of different 'essential type' categories
Reason	AUTOSAR defines its boolean type as unsigned char.
Potential risks	The Boolean logic does not behave as expected.
Prevention of risks	Functional tests of the component logic.
Examples	-

Table 4-10 MD_MSR_AutosarBoolean

Deviation ID	MD_MSR_VStdLibCopy
Violated guideline	Dir1.1: Any implementation-defined behaviour on which the output of the program depends shall be documented and understood QAC message 0315: Implicit conversion from a pointer to object type to a pointer to void.
Reason	The copy functions from the vstdlib have a void pointer as a function parameter.
Potential risks	No risk, because the underlying uint8 pointer type is known.
Prevention of risks	No prevention necessary.
Examples	-

Table 4-11 MD_MSR_VStdLib

Deviation ID	MD_MSR_EmptyClause
Violated guideline	Rule15.7: All if ... else if constructs shall be terminated with an else statement Rule16.4: Every switch statement shall have a default label QAC message 2013: This 'if .. else if ' construct 'else' statement is empty. QAC message 2016: This 'switch' statement 'default' clause is empty.
Reason	The logical code structure is defined by other coding rules. Based on these constraints the default clause might be required to do nothing.
Potential risks	It might be unclear if code was forgotten here.
Prevention of risks	The empty clauses contain comments indicating that they are intentionally empty.
Examples	-

Table 4-12 MD_MSR_EmptyPath

Deviation ID	MD_MSR_Union
Violated guideline	Rule19.2: The union keyword should not be used
Reason	For an efficient implementation of protocols the usage of unions may be required.
Potential risks	Union data may be misinterpreted by mixing up the different variants or by portability issues.
Prevention of risks	Code inspection and test of the different variants in the component test.
Examples	-

Table 4-13 MD_MSR_Union

Deviation ID	MD_MSR_RetVal
Violated guideline	Rule2.2: There shall be no dead code QAC message 2981: This assignment is redundant. The value of this object is never subsequently used.
Reason	For inactive DEV_ERROR_DETECT the initialization might be redundant.
Potential risks	The redundant initialization code might be confusing.
Prevention of risks	We stick to our well-known code pattern for all API functions.
Examples	-

Table 4-14 MD_MSR_RetVal

Deviation ID	MD_MSR_MemMap
Violated guideline	Rule20.1: #include directives should only be preceded by preprocessor directives or comments
Reason	The AUTOSAR memory mapping concept requires inclusion of MemMap.h multiple times in a file in order to select appropriate memory sections.
Potential risks	MemMap.h is provided by the ECU software integrator, hence many risks may occur, caused by wrong implementation of this file.
Prevention of risks	The ECU software integrator strictly has to adhere to the definitions of the AUTOSAR SWS Memory Mapping. Extensions to the file not described in the SWS may not be put into MemMap.h. This has to be verified by code inspection.

Examples	`#define ECUM_STOP_SEC_CODE`
	`#include "MemMap.h"`

Table 4-15 MD_MSR_MemMap

Deviation ID	MD_MSR_Undef
Violated guideline	Rule20.5: #undef should not be used
Reason	AUTOSAR memory mapping concept requires #undef statements to implement the #pragma selection mechanism in MemMap.h
Potential risks	MemMap.h is provided by the ECU software integrator, hence many risks may occur, caused by wrong implementation of this file.
Prevention of risks	The ECU software integrator strictly has to adhere to the definitions of the AUTOSAR SWS Memory Mapping. Extensions to the file not described in the SWS may not be put into MemMap.h. This has to be verified by code inspection.

Examples	`#ifdef START_SEC_CONST_32BIT`
	`  #pragma section MyConst32BitSection`
	`  #undef START_SEC_CONST_32BIT`
	`  #undef MEMMAP_ERROR`
	`#endif`

Table 4-16 MD_MSR_Undef

Deviation ID	MD_MSR_Rule20.10_0342
Violated guideline	Rule20.10: The # and ## preprocessor operators should not be used
Reason	Some interfaces specified by AUTOSAR can only be implemented efficiently with the glue operator. K&R compilers do not support the ISO glue operator '##'. AUTOSAR requires C90. C90 supports ## operators.
Potential risks	Readability and comprehensibility of the code might suffer. K&R compilers do not support the ISO glue operator '##'. But AUTOSAR requires C90. C90 supports ## operators.
Prevention of risks	Usage of compiler that supports the glue operator.
Examples	–

Table 4-17 MD_MSR_Rule20.10_0342

4.2 Code Metric Justification

Note
These QA-C message IDs are defined by Vector and are no standard QA-C message IDs.

Deviation ID	**MD_MSR_STPTH**
Violated metric	Estimated static path count defined by HIS shall be in range 1..80
QA-C message	6010
Reason	No separation of functionality into sub-functions due to higher voted requirements for minimized stack and runtime usage applied on the code.
Potential risks	Understandability and testability might become too complex.
Prevention of risks	Design and code review. Clearly structured and commented code.
Comments	Metric calculates the upper bound of possible (non-cyclic) paths in function's control flow. Metric addresses the TESTABILITY of a function – the higher the more test cases might be necessary.

Table 4-18 MD_MSR_STPTH

Deviation ID	**MD_MSR_STCYC**
Violated metric	Cyclomatic complexity defined by HIS shall be in range 1..10
QA-C message	6030
Reason	No separation of functionality into sub-functions due to higher voted requirements for minimized stack and runtime usage applied on the code.
Potential risks	Understandability and testability might become too complex.
Prevention of risks	Design and code review. Clearly structured and commented code.
Comments	Metric indicates potentially inadequate modularization or too much logic in one function if the value is high, the function may tend to have complexity related issues. Metric addresses a HOTSPOT in the code.

Table 4-19 MD_MSR_STCYC

Deviation ID	**MD_MSR_STCAL**
Violated metric	Number of distinct function calls defined by HIS shall be in range 0..7
QA-C message	6050
Reason	Software structure is defined by AUTOSAR standard. Standard compliance is voted higher than metric threshold. In addition, a typical approach to reduce STCAL is deeper nesting of functions, this increases call stack usage and runtime.
Potential risks	Understandability and testability might become too complex due to fan-out to many functions.
Prevention of risks	Design and code review. Clearly structured and commented code.
Comments	Metric indicates fan-out of a function, i.e. how many different functions are called. Metric addresses a HOTSPOT in the code. The fan-out depends on a mix of component-local design and on global design (AUTOSAR). Only component-local design could be adopted.

Table 4-20 MD_MSR_STCAL

Deviation ID	MD_MSR_STPAR
Violated metric	Number of function parameters defined by HIS shall be in range 0..5
QA-C message	6060
Reason	API is defined by AUTOSAR standard. Standard compliance is voted higher than metric threshold.
Potential risks	Stack usage and runtime too high for target µC. Usage of function is not easily comprehensible.
Prevention of risks	Test of resulting code on target µC. User must check stack usage in project context.
Comments	Metric counts number of declared parameters in the function argument list what have impact on runtime and call stack usage. We are aware of such environmental conditions and apply larger numbers of function parameters only, if explicitly required by standards like AUTOSAR or customer specs.

Table 4-21 MD_MSR_STPAR

Deviation ID	MD_MSR_STMIF
Violated metric	Number maximum nesting of control structures defined by HIS shall be in range 0..4
QA-C message	6080
Reason	The function covers a specific task in the overall component behavior and this task has different scenarios depending on local conditions. This results in deep nesting of control structures. If there is more common than different code, higher nesting level is accepted to keep code footprint small.
Potential risks	Code is difficult to maintain.
Prevention of risks	Design and code review. Clearly structured and commented code.
Comments	Metric addresses the MAINTAINABILITY of the code. Expectation is, that deep nesting of control structures increases the effort to understand a function.

Table 4-22 MD_MSR_STMIF

5 Additional Resources

No.	Documents
[MISRA-C:2012]	MISRA-C:2012 Guidelines for the use of the C language in critical systems (March 2013)
[MISRA-AMD1]	MISRA C:2012 Amendment 1 (April 2016)
[MISRA-TC1]	MISRA C:2012 Technical Corrigendum 1 (June 2017)
[MISRA-Compliance]	MISRA Compliance:2016 – Achieving compliance with MISRA Coding Guidelines (April 2016)
[MISRA-C:2004]	MISRA-C:2004 Guidelines for the use of the C language in critical systems (October 2004)
[HIS-Metrics]	HIS Source Code Metriken (Version 1.3.1) **Note** HIS itself does not exist any longer. Still we use the metric ranges. defined.
[AUTOSAR]	http://www.autosar.org/

Table 5-1 Additional Ressources

6 Contacts

For a full list with all Vector locations and addresses worldwide, please visit http://vector.com/contact/.