Coverity® MISRA Report

STD LIN API
v2.0.0

Enterprise	Melexis
Competence Center	SW
Coverity Connect Project	library_std-lin-api
Prepared For	Melexis LIN products
Prepared By	Grygoriy Moroz [grm]
Prepared On	15 Mar 2018 16:01

Executive Summary

This report collects results of Coverity MISRA analyses for a specific project. The results are aggregated and displayed in ways that give the reader insight into the MISRA compliance of and readiness for release of software in that project.

The intended audiences of this report are software decision-makers and developers. Because the contents may be considered sensitive, it is not intended for external release.

To review detailed code-level findings, it is recommended that developers click this link to the Coverity Connect platform (https://coverity.melexis.com:8443/reports#p10083) in order to see source code annotated with MISRA violations and associated details.

MISRA Compliance

This project has been found **consistent with MISRA compliance**:

- No Mandatory or Document rules with deviations were found.
- No Required violations were found.
- No Mandatory violations were found.
- No Advisory violations were found.

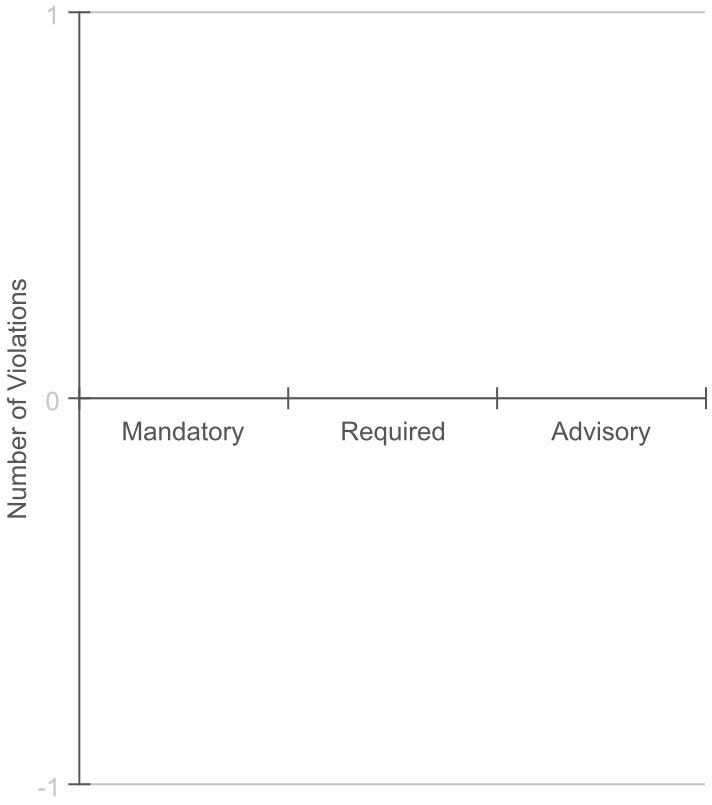
Additional Quality Measures

This table reports on issues found by Coverity analysis that were not included elsewhere in the report. Although excluded from the report, they may nonetheless indicate the presence of significant quality issues.

Type	Count
Issues Marked "False Positive" or "Intentional"	1143
Non-MISRA Issues	0
Snapshots older than 30 days	0

Violations By MISRA Category

A total of 1143 violations were found. Each violation has a MISRA category associated with its rule. The chart below shows the number of occurrences of each of the categories.



Detailed Results

Violations By Rule

Violations are divided by rule.

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Directive 1.1	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood.	Required	no		
Directive 2.1	All source files shall compile without any compilation errors.	Required	no		
Directive 3.1	All code shall be traceable to documented requirements.	Required	no		
Directive 4.1	Run-time failures shall be minimized.	Required	no		
Directive 4.2	All usage of assembly language should be documented.	Required	no		
Directive 4.3	Assembly language shall be encapsulated and isolated.	Required	yes	yes	0
Directive 4.4	Sections of code should not be "commented out".	Required	yes	yes	0
Directive 4.5	Identifiers in the same name space with overlapping visibility should be typographically unambiguous.	Required	yes	yes	0
Directive 4.6	***typedefs*** that indicate size and signedness should be used in place of the basic numerical types.	Advisory	yes	yes	0
Directive 4.7	If a function returns error information, then that error information shall be tested.	Required	yes	yes	0
Directive 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.	Advisory	yes	yes	0
Directive 4.9	A function should be used in preference to a ***function-like macro*** where they are interchangeable.	Advisory	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Directive 4.10	Precautions shall be taken in order to prevent the contents of a **header** file being included more than once.	Required	yes	yes	0
Directive 4.11	The validity of values passed to library functions shall be checked.	Required	yes	yes	0
Directive 4.12	Dynamic memory allocation shall not be used.	Required	yes	yes	0
Directive 4.13	Functions which are designed to provide operations on a resource should be called in an appropriate sequence.	Advisory	yes	yes	0
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.	Required	yes	yes	0
Rule 1.2	Language extensions should not be used.	Advisory	yes	yes	0
Rule 1.3	There shall be no occurrence of undefined or critical unspecified behaviour.	Required	no		
Rule 2.1	A project shall not contain *unreachable* code.	Required	yes	yes	0
Rule 2.2	There shall be no ***dead code***.	Required	yes	yes	0
Rule 2.3	A project should not contain unused type declarations.	Advisory	yes	yes	0
Rule 2.4	A project should not contain unused tag declarations.	Advisory	yes	yes	0
Rule 2.5	A project should not contain unused macro declarations.	Advisory	yes	yes	0
Rule 2.6	A function should not contain unused label declarations.	Advisory	yes	yes	0
Rule 2.7	There should be no unused parameters in functions.	Advisory	yes	yes	0
Rule 3.1	The character sequences /* and // shall not be used within a comment.	Required	yes	yes	0
Rule 3.2	Line-splicing shall not be used in // comments.	Required	yes	yes	0
Rule 4.1	Octal and hexadecimal escape sequences shall be terminated.	Required	yes	yes	0
Rule 4.2	Trigraphs should not be used.	Advisory	yes	yes	0
Rule 5.1	***External identifiers*** shall be distinct.	Required	yes	yes	0
Rule 5.2	Identifiers declared in the same scope and name space shall be distinct.	Required	yes	yes	0
Rule 5.3	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	Required	yes	yes	0
Rule 5.4	***Macro identifiers*** shall be distinct.	Required	yes	yes	0
Rule 5.5	Identifiers shall be distinct from macro names.	Required	yes	yes	0
Rule 5.6	A ***typedef*** name shall be a unique identifier.	Required	yes	yes	0
Rule 5.7	A tag name shall be a unique identifier.	Required	yes	yes	0
Rule 5.8	Identifiers that define objects or functions with external linkage shall be unique.	Required	yes	yes	0
Rule 5.9	Identifiers that define objects or functions with internal linkage should be unique.	Advisory	yes	yes	0
Rule 6.1	Bit-fields shall only be declared with an appropriate type.	Required	yes	yes	0
Rule 6.2	Single-bit named bit fields shall not be of a signed type.	Required	yes	yes	0
Rule 7.1	Octal constants shall not be used.	Required	yes	yes	0
Rule 7.2	A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type.	Required	yes	yes	0
Rule 7.3	The lowercase character "l" shall not be used in a literal suffix.	Required	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Rule 7.4	A string literal shall not be *assigned* to an object unless the object's type is "pointer to *const*-qualified *char*".	Required	yes	yes	0
Rule 8.1	Types shall be explicitly specified.	Required	yes	yes	0
Rule 8.2	Function types shall be in ***prototype form*** with named parameters.	Required	yes	yes	0
Rule 8.3	All declarations of an object or function shall use the same names and type qualifiers.	Required	yes	yes	0
Rule 8.4	A compatible declaration shall be visible when an object or function with external linkage is defined.	Required	yes	yes	0
Rule 8.5	An external object or function shall be declared once in one and only one file.	Required	yes	yes	0
Rule 8.6	An identifier with external linkage shall have exactly one external definition.	Required	yes	yes	0
Rule 8.7	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit.	Advisory	yes	yes	0
Rule 8.8	The **static** storage class specifier shall be used in all declarations of objects and functions that have internal linkage.	Required	yes	yes	0
Rule 8.9	An object should be defined at block scope if its identifier only appears in a single function.	Advisory	yes	yes	0
Rule 8.10	An **inline function** shall be declared with the static storage class.	Required	yes	yes	0
Rule 8.11	When an array with external linkage is declared, its size should be explicitly specified.	Advisory	yes	yes	0
Rule 8.12	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique.	Required	yes	yes	0
Rule 8.13	A pointer should point to a const-qualified type whenever possible.	Advisory	yes	yes	0
Rule 8.14	The ***restrict*** type qualifier shall not be used.	Required	yes	yes	0
Rule 9.1	The value of an object with automatic storage duration shall not be read before it has been set.	Mandatory	yes	yes	0
Rule 9.2	The initializer for an aggregate or union shall be enclosed in braces.	Required	yes	yes	0
Rule 9.3	Arrays shall not be partially initialized.	Required	yes	yes	0
Rule 9.4	An element of an object shall not be initialized more than once.	Required	yes	yes	0
Rule 9.5	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly.	Required	yes	yes	0
Rule 10.1	Operands shall not be of an inappropriate ***essential*** type.	Required	yes	yes	0
Rule 10.2	Expressions of ***essentially character type*** shall not be used inappropriately in addition and subtraction operations.	Required	yes	yes	0
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***.	Required	yes	yes	0
Rule 10.4	Both operands of an operator in which the **usual arithmetic conversions** are performed shall have the same **essential type category**.	Required	yes	yes	0
Rule 10.5	The value of an expression should not be cast to an inappropriate **essential type**.	Advisory	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Rule 10.6	The value of a ***composite expression*** **s**hall not be assigned to an object with wider e***ssential type***.	Required	yes	yes	0
Rule 10.7	If a ***composite expression*** is used as one operand of an operator in which the ***usual arithmetic conversions*** are performed then the other operand shall not have wider ***essential type***.	Required	yes	yes	0
Rule 10.8	The value of a **composite expression** shall not be cast to a different **essential type category** or a wider **essential type**.	Required	yes	yes	0
Rule 11.1	Conversions shall not be performed between a pointer to a function and any other type.	Required	yes	yes	0
Rule 11.2	Conversions shall not be performed between a pointer to an incomplete type and any other type.	Required	yes	yes	0
Rule 11.3	A cast shall not be performed between a pointer to object type and a pointer to a different object type.	Required	yes	yes	0
Rule 11.4	A conversion should not be performed between a pointer to object and an integer type.	Advisory	yes	yes	0
Rule 11.5	A conversion should not be performed from pointer to ***void*** into pointer to object.	Advisory	yes	yes	0
Rule 11.6	A cast shall not be performed between pointer to ***void*** and an arithmetic type.	Required	yes	yes	0
Rule 11.7	A conversion should not be performed between a pointer to object and a non-integer arithmetic type.	Required	yes	yes	0
Rule 11.8	A cast shall not remove any *const* or ***volatile*** qualification from the type pointed to by a pointer.	Required	yes	yes	0
Rule 11.9	The macro NULL shall be the only permitted form of integer ***null pointer constant***.	Required	yes	yes	0
Rule 12.1	The precedence of operators within expressions should be made explicit.	Advisory	yes	yes	0
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.	Required	yes	yes	0
Rule 12.3	The comma operator should not be used.	Advisory	yes	yes	0
Rule 12.4	Evaluation of ***constant expressions*** should not lead to unsigned integer wrap-around.	Advisory	yes	yes	0
Rule 13.1	***Initializer*** lists shall not contain ***persistent side effects***.	Required	yes	yes	0
Rule 13.2	The value of an expression and its ***persistent side effects*** shall be the.	Required	yes	yes	0
Rule 13.3	A full expression containing an increment (++) or decrement (--) operator should have no other potential ***side effects*** other than that caused by the increment or decrement operator.	Advisory	yes	yes	0
Rule 13.4	The result of an assignment operator should not be ***used***.	Advisory	yes	yes	0
Rule 13.5	The right hand operand of a logical && or || operator shall not contain **persistent side effects**.	Required	yes	yes	0
Rule 13.6	The operand of the ***sizeof*** operator shall not contain any expression which has potential **side effects**.	Mandatory	yes	yes	0
Rule 14.1	A ***loop counter*** shall not have ***essentially floating*** type.	Required	yes	yes	0
Rule 14.2	A ***for*** loop shall be well-formed.	Required	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Rule 14.3	Controlling expressions shall not be invariant.	Required	yes	yes	0
Rule 14.4	The controlling expression of an ***if*** statement and the controlling expression of an ***iteration-statement*** shall have ***essentially Boolean*** type.	Required	yes	yes	0
Rule 15.1	The ***goto*** statement should not be used.	Advisory	yes	yes	0
Rule 15.2	The ***goto*** statement shall jump to a label declared later in the same.	Required	yes	yes	0
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.	Required	yes	yes	0
Rule 15.4	There should be no more than one ***break*** or ***goto*** statement used to terminate any iteration statement.	Advisory	yes	yes	0
Rule 15.5	A function should have a single point of exit at the end.	Advisory	yes	yes	0
Rule 15.6	The body of an **iteration-statement** or a **selection-statement** shall be a **compound-statement**.	Required	yes	yes	0
Rule 15.7	All ***if ... else if*** constructs shall be terminated with an ***else*** statement.	Required	yes	yes	0
Rule 16.1	All ***switch*** statements shall be well-formed.	Required	yes	yes	0
Rule 16.2	A ***switch label*** shall only be used when the most closely-enclosing compound statement is the body of a **switch** statement.	Required	yes	yes	0
Rule 16.3	An unconditional ***break*** statement shall terminate every ***switch-clause***.	Required	yes	yes	0
Rule 16.4	Every ***switch*** statement shall have a ***default*** label.	Required	yes	yes	0
Rule 16.5	A **default** label shall appear as either the first or the last **switch label** of a **switch** statement.	Required	yes	yes	0
Rule 16.6	Every ***switch*** statement shall have at least two ***switch-clauses***.	Required	yes	yes	0
Rule 16.7	A ***switch-expression*** shall not have ***essentially Boolean*** type.	Required	yes	yes	0
Rule 17.1	The features of <stdarg.h> shall not be used.	Required	yes	yes	0
Rule 17.2	Functions shall not call themselves, either directly or indirectly.	Required	yes	yes	0
Rule 17.3	A function shall not be declared implicitly.	Mandatory	yes	yes	0
Rule 17.4	All exit paths from a function with non-***void*** return type shall have an explicit ***return*** statement with an expression.	Mandatory	yes	yes	0
Rule 17.5	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements.	Advisory	yes	yes	0
Rule 17.6	The declaration of an array parameter shall not contain the **static** keyword between the [ ].	Mandatory	yes	yes	0
Rule 17.7	The value returned by a function having non-**void** return type shall be **used**.	Required	yes	yes	0
Rule 17.8	A function parameter should not be modified.	Advisory	yes	yes	0
Rule 18.1	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand.	Required	yes	yes	0
Rule 18.2	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Required	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Rule 18.3	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object.	Required	yes	yes	0
Rule 18.4	The +, -, += and -= operators should not be applied to an expression of pointer type.	Advisory	yes	yes	0
Rule 18.5	Declarations should contain no more than two levels of pointer nesting.	Advisory	yes	yes	0
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.	Required	yes	yes	0
Rule 18.7	Flexible array members shall not be declared.	Required	yes	yes	0
Rule 18.8	Variable-length array types shall not be used.	Required	yes	yes	0
Rule 19.1	An object shall not be assigned or copied to an overlapping object.	Mandatory	yes	yes	0
Rule 19.2	The ***union*** keyword should not be used.	Advisory	yes	yes	0
Rule 20.1	**#include** directives should only be preceded by preprocessor directives or comments.	Advisory	yes	yes	0
Rule 20.2	The ', " or \ characters and the /* or // character sequences shall not occur in a **header** file name.	Required	yes	yes	0
Rule 20.3	The **#include** directive shall be followed by either a <filename> or "filename" sequence.	Required	yes	yes	0
Rule 20.4	A macro shall not be defined with the same name as a keyword.	Required	yes	yes	0
Rule 20.5	**#undef** should not be used.	Advisory	yes	yes	0
Rule 20.6	Tokens that look like a preprocessing directive shall not occur within a macro argument.	Required	yes	yes	0
Rule 20.7	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses.	Required	yes	yes	0
Rule 20.8	The controlling expression of a ***#if*** or ***#elif*** preprocessing directive shall evaluate to 0 or 1.	Required	yes	yes	0
Rule 20.9	All identifiers used in the controlling expression of ***#if*** or ***#elif*** *preprocessing directives shall be* ***#define'd*** *before evaluation*.	Required	yes	yes	0
Rule 20.10	The # and ## preprocessor operators should not be used.	Advisory	yes	yes	0
Rule 20.11	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator.	Required	yes	yes	0
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.	Required	yes	yes	0
Rule 20.13	A line whose first token is # shall be a valid preprocessing directive.	Required	yes	yes	0
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.	Required	yes	yes	0
Rule 21.1	***#define*** and ***#undef*** shall not be used on a reserved identifier or reserved macro name.	Required	yes	yes	0
Rule 21.2	A reserved identifier or macro name shall not be declared.	Required	yes	yes	0
Rule 21.3	The memory allocation and deallocation functions of <stdlib.h> shall not be used.	Required	yes	yes	0

Standard: MISRA C 2012

Rule ID	Description	Category	Supported	Enabled	Violations
Rule 21.4	The standard header file <setjmp.h> shall not be used.	Required	yes	yes	0
Rule 21.5	The standard header file <signal.h> shall not be used.	Required	yes	yes	0
Rule 21.6	The Standard Library input/output functions shall not be used.	Required	yes	yes	0
Rule 21.7	The ***atof***, ***atoi***, ***atol*** and ***atoll*** functions of <stdlib.h> shall not be used.	Required	yes	yes	0
Rule 21.8	The library functions ***abort***, ***exit***, ***getenv*** and ***system*** of <stdlib.h> shall not be used.	Required	yes	yes	0
Rule 21.9	The library functions *bsearch* and *qsort* of <stdlib.h> shall not be used.	Required	yes	yes	0
Rule 21.10	The Standard Library time and date functions shall not be used.	Required	yes	yes	0
Rule 21.11	The standard ***header file*** <tgmath.h> shall not be used.	Required	yes	yes	0
Rule 21.12	The exception handling features of <fenv.h> should not be used.	Advisory	yes	yes	0
Rule 22.1	All resources obtained dynamically by means of Standard Library functions shall be explicitly released.	Required	yes	yes	0
Rule 22.2	A block of memory shall only be freed if it was allocated by means of a Standard Library function.	Mandatory	yes	yes	0
Rule 22.3	The same file shall not be open for read and write access at the same time on different streams.	Required	yes	yes	0
Rule 22.4	There shall be no attempt to write to a stream which has been opened as read-only.	Mandatory	yes	yes	0
Rule 22.5	A pointer to a FILE object shall not be dereferenced.	Mandatory	yes	yes	0
Rule 22.6	The value of a pointer to a FILE shall not be used after the associated stream has been closed.	Mandatory	yes	yes	0
Total					**0**

Analysis Details

A Coverity project is a collection of one or more streams containing separately-analyzed snapshots. The latest snapshot in each stream is used when reporting results for a project. This section gives details about the streams and the analysis performed for each snapshot.

Stream	Snapshot Information				MISRA Information		
	ID	Analysis Date	Analysis Version	Target	Configuration ID	Configuration Title	Standard
library_std-lin-api-ma-13	19075	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-ma-20	19077	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-ma-21-22	19078	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-ma-J2602	19072	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-sl-13	19073	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-sl-20	19074	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-sl-21-22	19076	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012
library_std-lin-api-sl-J2602	19071	15-Mar-18	2017.07		520200	Coverity Level 7 Compliance	MISRA C 2012

Methodology

Introduction

This report aggregates the output of the Coverity Code Advisor used on a particular project (code base). Coverity Code Advisor is a static analysis tool that is capable of finding quality defects, security vulnerabilities, and test violations through the process of scanning the output of a specially-compiled code base. The information in this report is specific to MISRA vulnerabilities detected by Coverity Code Advisor.

About Static Analysis

Static analysis is the analysis of software code without executing the compiled program, for the purpose of finding logic errors or security vulnerabilities. Coverity's static analysis tools integrate with all major build systems and generate a high fidelity representation of source code to provide full code path coverage, ensuring that every line of code and execution path is analyzed. Code Advisor supports the market leading compilers for C, C++, Java, C#, Objective C, JavaScript, PHP, Python, and more.

About MISRA

MISRA is a set of software development guidelines for the C and C++ programming languages developed by MISRA (Motor Industry Software Reliability Association). Its aims are to facilitate code safety, portability and reliability in the context of embedded systems, specifically those systems programmed in ISO C and C++.

Coverity Static Analysis finds violations of MISRA guidelines for three MISRA standards: C-2004, C++-2008, and C-2012.

Rules

MISRA standards divide their compliance tests into a set of rules. Not all rules are amenable to being checked using Static Analysis. Those that are checkabale with Static Analysis are further divided into those that the specific analysis tool actually can check and those that it cannot.

Each rule has a category, one of Mandatory, Document, Required and Advisory, that indicates the extent to which the rule must be conformed, and whether exceptions called "deviations" are permitted. Deviations can be on a per-rule basis (Project deviations) or a per-violation basis (Specific deviations).

- **Mandatory** and **Document**: Deviations are not permitted.
- **Required**: Deviations are permitted, with approval and with adequate documentation.
- **Advisory**: Deviations are permitted, with less formal approval procedures.

The software team decides which rules shall have deviations and has the latitude to strengthen the rigor of rules, making particular Advisory rules Mandatory for example.

Compliance

MISRA compliance includes the development of formal procedures, processes and documentation, assets that are outside of the scope of static analysis tools. As such, those tools cannot judge whether software *is* MISRA compliant, however they can judge whether it *is not* MISRA compliant.

MISRA compliance requires, in part, that the software have no violations and no deviations for Mandatory and Document rules.