



AUTOSAR AP RELEASE 18-10 CATEGORIES MAPPED TO CODESONAR[®] 7.1



TRUSTED LEADERS OF SOFTWARE ASSURANCE AND ADVANCED CYBER-SECURITY SOLUTIONS

WWW.GRAMMATECH.COM

INTRODUCTION

AUTOSAR (AUTomotive Open System ARchitecture) is a worldwide development partnership of vehicle manufacturers, suppliers, service providers and companies from the automotive electronics, semiconductor and software industry.

For more information on AUTOSAR:

<https://www.autosar.org/about/>

The remainder of this document comprises two tables:

- A table showing the close mapping between CodeSonar warning classes and the AUTOSAR categories.
- A table showing the broad mapping between CodeSonar warning classes and the AUTOSAR categories. The broad mapping for a CodeSonar warning class includes the close mapping for the class, plus any other checks that are related to the class in a meaningful way, but not eligible for the close mapping.



AUTOSAR CLOSE MAPPING (CODESONAR V7.1)

The following table contains CodeSonar warning classes that are closely mapped to AUTOSAR categories.

Rule	Rule Name	Supported
AUTOSAR++14:A0-1-1	A project shall not contain instances of non-volatile variables being given values that are not subsequently used.	Yes
AUTOSAR++14:M0-1-1	A project shall not contain unreachable code.	Yes
AUTOSAR++14:A0-1-2	The value returned by a function having a non-void return type that is not an overloaded operator shall be used.	Yes
AUTOSAR++14:M0-1-2	A project shall not contain infeasible paths.	Yes
AUTOSAR++14:A0-1-3	Every function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used.	No
AUTOSAR++14:M0-1-3	A project shall not contain unused variables.	Yes
AUTOSAR++14:A0-1-4	There shall be no unused named parameters in non-virtual functions.	Yes
AUTOSAR++14:M0-1-4	A project shall not contain non-volatile POD variables having only one use.	Yes
AUTOSAR++14:A0-1-5	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it.	Yes
AUTOSAR++14:A0-1-6	There should be no unused type declarations.	Yes
AUTOSAR++14:M0-1-8	All functions with void return type shall have external side effect(s).	Yes
AUTOSAR++14:M0-1-9	There shall be no dead code.	Yes
AUTOSAR++14:M0-1-10	Every defined function should be called at least once.	No
AUTOSAR++14:M0-2-1	An object shall not be assigned to an overlapping object.	Yes
AUTOSAR++14:M0-3-1	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.	No
AUTOSAR++14:M0-3-2	If a function generates error information, then that error information shall be tested.	Yes
AUTOSAR++14:A0-4-1	Floating-point implementation shall comply with IEEE 754 standard.	No
AUTOSAR++14:M0-4-1	Use of scaled-integer or fixed-point arithmetic shall be documented.	No
AUTOSAR++14:A0-4-2	Type long double shall not be used.	No
AUTOSAR++14:M0-4-2	Use of floating-point arithmetic shall be documented.	No
AUTOSAR++14:A0-4-3	The implementations in the chosen compiler shall strictly comply with the C++14 Language Standard.	No
AUTOSAR++14:A0-4-4	Range, domain and pole errors shall be checked when using math functions.	Yes
AUTOSAR++14:M1-0-2	Multiple compilers shall only be used if they have a common, defined interface.	No
AUTOSAR++14:A1-1-1	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features.	Yes
AUTOSAR++14:A1-1-2	A warning level of the compilation process shall be set in compliance with project policies.	Yes
AUTOSAR++14:A1-1-3	An optimization option that disregards strict standard compliance shall not be turned on in the chosen compiler.	No
AUTOSAR++14:A1-2-1	When using a compiler toolchain (including preprocessor, compiler itself, linker, C++ standard libraries) in safety-related software, the tool confidence level (TCL) shall be determined. In case of TCL2 or TCL3, the compiler shall undergo a "Qualification of a software tool", as per ISO 26262-8.11.4.6.	No
AUTOSAR++14:A1-4-1	Code metrics and their valid boundaries shall be defined and code shall comply with defined boundaries of code metrics.	Yes
AUTOSAR++14:A1-4-3	All code should compile free of compiler warnings.	No
AUTOSAR++14:A2-3-1	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code.	No
AUTOSAR++14:A2-5-1	Trigraphs shall not be used.	Yes

AUTOSAR++14:A2-5-2	Digraphs shall not be used.	No
AUTOSAR++14:A2-7-1	The character \ shall not occur as a last character of a C++ comment.	Yes
AUTOSAR++14:M2-7-1	The character sequence /* shall not be used within a C-style comment.	Yes
AUTOSAR++14:A2-7-2	Sections of code shall not be "commented out".	Yes
AUTOSAR++14:A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation.	No
AUTOSAR++14:A2-7-5	Comments shall not document any actions or sources (e.g. tables, figures, paragraphs, etc.) that are outside of the file.	No
AUTOSAR++14:A2-8-1	A header file name should reflect the logical entity for which it provides declarations.	No
AUTOSAR++14:A2-8-2	An implementation file name should reflect the logical entity for which it provides definitions.	No
AUTOSAR++14:A2-10-1	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	Yes
AUTOSAR++14:M2-10-1	Different identifiers shall be typographically unambiguous.	Yes
AUTOSAR++14:A2-10-4	The identifier name of a non-member object with static storage duration or static function shall not be reused within a namespace.	Yes
AUTOSAR++14:A2-10-5	An identifier name of a function with static storage duration or a non-member object with external or internal linkage should not be reused.	Yes
AUTOSAR++14:A2-10-6	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope.	No
AUTOSAR++14:A2-11-1	Volatile keyword shall not be used.	No
AUTOSAR++14:A2-13-1	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used.	No
AUTOSAR++14:A2-13-2	String literals with different encoding prefixes shall not be concatenated.	No
AUTOSAR++14:M2-13-2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes
AUTOSAR++14:A2-13-3	Type wchar_t shall not be used.	No
AUTOSAR++14:M2-13-3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes
AUTOSAR++14:A2-13-4	String literals shall not be assigned to non-constant pointers.	Yes
AUTOSAR++14:M2-13-4	Literal suffixes shall be upper case.	Yes
AUTOSAR++14:A2-13-5	Hexadecimal constants should be upper case.	No
AUTOSAR++14:A2-13-6	Universal character names shall be used only inside character or string literals.	No
AUTOSAR++14:A3-1-1	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.	Yes
AUTOSAR++14:A3-1-2	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx".	No
AUTOSAR++14:M3-1-2	Functions shall not be declared at block scope.	Yes
AUTOSAR++14:A3-1-3	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp".	No
AUTOSAR++14:A3-1-4	When an array with external linkage is declared, its size shall be stated explicitly.	Yes
AUTOSAR++14:A3-1-5	A function definition shall only be placed in a class definition if (1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template.	No
AUTOSAR++14:A3-1-6	Trivial accessor and mutator functions should be inlined.	No
AUTOSAR++14:M3-2-1	All declarations of an object or function shall have compatible types.	Yes
AUTOSAR++14:M3-2-2	The One Definition Rule shall not be violated.	Yes
AUTOSAR++14:M3-2-3	A type, object or function that is used in multiple translation units shall be declared in one and only one file.	Yes
AUTOSAR++14:M3-2-4	An identifier with external linkage shall have exactly one definition.	Yes
AUTOSAR++14:A3-3-1	Objects or functions with external linkage (including members of named namespaces) shall be declared in a header file.	No
AUTOSAR++14:A3-3-2	Static and thread-local objects shall be constant-initialized.	Yes
AUTOSAR++14:M3-3-2	If a function has internal linkage then all re-declarations shall include the static storage class specifier.	No
AUTOSAR++14:M3-4-1	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.	Yes

AUTOSAR++14:A3-8-1	An object shall not be accessed outside of its lifetime.	Yes
AUTOSAR++14:A3-9-1	Fixed width integer types from <code><stdint.h></code> , indicating the size and signedness, shall be used in place of the basic numerical types.	Yes
AUTOSAR++14:M3-9-1	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.	Yes
AUTOSAR++14:M3-9-3	The underlying bit representations of floating-point values shall not be used.	Yes
AUTOSAR++14:A4-5-1	Expressions with type <code>enum</code> or <code>enum class</code> shall not be used as operands to built-in and overloaded operators other than the subscript operator <code>[]</code> , the assignment operator <code>=</code> , the equality operators <code>==</code> and <code>!=</code> , the unary <code>&</code> operator, and the relational operators <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> .	Yes
AUTOSAR++14:M4-5-1	Expressions with type <code>bool</code> shall not be used as operands to built-in operators other than the assignment operator <code>=</code> , the logical operators <code>&&</code> , <code> </code> , <code>!</code> , the equality operators <code>==</code> and <code>!=</code> , the unary <code>&</code> operator, and the conditional operator.	Yes
AUTOSAR++14:M4-5-3	Expressions with type (plain) <code>char</code> and <code>wchar_t</code> shall not be used as operands to built-in operators other than the assignment operator <code>=</code> , the equality operators <code>==</code> and <code>!=</code> , and the unary <code>&</code> operator.	Yes
AUTOSAR++14:A4-7-1	An integer expression shall not lead to data loss.	Yes
AUTOSAR++14:A4-10-1	Only <code>nullptr</code> literal shall be used as the null-pointer-constant.	No
AUTOSAR++14:M4-10-1	<code>NULL</code> shall not be used as an integer value.	Yes
AUTOSAR++14:M4-10-2	Literal zero (0) shall not be used as the null-pointer-constant.	Yes
AUTOSAR++14:A5-0-1	The value of an expression shall be the same under any order of evaluation that the standard permits.	No
AUTOSAR++14:A5-0-2	The condition of an if-statement and the condition of an iteration statement shall have type <code>bool</code> .	Yes
AUTOSAR++14:M5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions.	Yes
AUTOSAR++14:A5-0-3	The declaration of objects shall contain no more than two levels of pointer indirection.	Yes
AUTOSAR++14:M5-0-3	A <code>cvalue</code> expression shall not be implicitly converted to a different underlying type.	Yes
AUTOSAR++14:A5-0-4	Pointer arithmetic shall not be used with pointers to non-final classes.	Yes
AUTOSAR++14:M5-0-4	An implicit integral conversion shall not change the signedness of the underlying type.	Yes
AUTOSAR++14:M5-0-5	There shall be no implicit floating-integral conversions.	Yes
AUTOSAR++14:M5-0-6	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.	Yes
AUTOSAR++14:M5-0-7	There shall be no explicit floating-integral conversions of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-8	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-10	If the bitwise operators <code>~</code> and <code><<</code> are applied to an operand with an underlying type of unsigned <code>char</code> or unsigned <code>short</code> , the result shall be immediately cast to the underlying type of the operand.	Yes
AUTOSAR++14:M5-0-11	The plain <code>char</code> type shall only be used for the storage and use of character values.	Yes
AUTOSAR++14:M5-0-12	Signed <code>char</code> and unsigned <code>char</code> type shall only be used for the storage and use of numeric values.	Yes
AUTOSAR++14:M5-0-14	The first operand of a conditional-operator shall have type <code>bool</code> .	Yes
AUTOSAR++14:M5-0-15	Array indexing shall be the only form of pointer arithmetic.	Yes
AUTOSAR++14:M5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.	Yes
AUTOSAR++14:M5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Yes
AUTOSAR++14:M5-0-18	<code>></code> , <code>>=</code> , <code><</code> , <code><=</code> shall not be applied to objects of pointer type, except where they point to the same array.	Yes
AUTOSAR++14:M5-0-20	Non-constant operands to a binary bitwise operator shall have the same underlying type.	Yes
AUTOSAR++14:M5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type.	Yes
AUTOSAR++14:A5-1-1	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead.	No
AUTOSAR++14:A5-1-2	Variables shall not be implicitly captured in a lambda expression.	Yes
AUTOSAR++14:A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression.	Yes

AUTOSAR++14:A5-1-4	A lambda expression object shall not outlive any of its reference-captured objects.	No
AUTOSAR++14:A5-1-6	Return type of a non-void return type lambda expression should be explicitly specified.	Yes
AUTOSAR++14:A5-1-7	A lambda shall not be an operand to decltype or typeid.	No
AUTOSAR++14:A5-1-8	Lambda expressions should not be defined inside another lambda expression.	No
AUTOSAR++14:A5-1-9	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression.	No
AUTOSAR++14:A5-2-1	dynamic_cast should not be used.	No
AUTOSAR++14:A5-2-2	Traditional C-style casts shall not be used.	Yes
AUTOSAR++14:M5-2-2	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.	Yes
AUTOSAR++14:A5-2-3	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.	Yes
AUTOSAR++14:M5-2-3	Casts from a base class to a derived class should not be performed on polymorphic types.	No
AUTOSAR++14:A5-2-4	reinterpret_cast shall not be used.	Yes
AUTOSAR++14:A5-2-5	An array or container shall not be accessed beyond its range.	Yes
AUTOSAR++14:A5-2-6	The operands of a logical && or shall be parenthesized if the operands contain binary operators.	No
AUTOSAR++14:M5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.	Yes
AUTOSAR++14:M5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes
AUTOSAR++14:M5-2-9	A cast shall not convert a pointer type to an integral type.	Yes
AUTOSAR++14:M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression.	Yes
AUTOSAR++14:M5-2-11	The comma operator, && operator and the operator shall not be overloaded.	Yes
AUTOSAR++14:M5-2-12	An identifier with array type passed as a function argument shall not decay to a pointer.	Yes
AUTOSAR++14:A5-3-1	Evaluation of the operand to the typeid operator shall not contain side effects.	No
AUTOSAR++14:M5-3-1	Each operand of the ! operator, the logical && or the logical operators shall have type bool.	Yes
AUTOSAR++14:A5-3-2	Null pointers shall not be dereferenced.	Yes
AUTOSAR++14:M5-3-2	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	Yes
AUTOSAR++14:A5-3-3	Pointers to incomplete class types shall not be deleted.	No
AUTOSAR++14:M5-3-3	The unary & operator shall not be overloaded.	Yes
AUTOSAR++14:M5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects.	Yes
AUTOSAR++14:A5-5-1	A pointer to member shall not access non-existent class members.	No
AUTOSAR++14:A5-6-1	The right hand operand of the integer division or remainder operators shall not be equal to zero.	Yes
AUTOSAR++14:M5-8-1	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes
AUTOSAR++14:A5-10-1	A pointer to member virtual function shall only be tested for equality with null-pointer-constant.	No
AUTOSAR++14:M5-14-1	The right hand operand of a logical &&, operators shall not contain side effects.	Yes
AUTOSAR++14:A5-16-1	The ternary conditional operator shall not be used as a sub-expression.	No
AUTOSAR++14:M5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.	No
AUTOSAR++14:M5-18-1	The comma operator shall not be used.	Yes
AUTOSAR++14:M5-19-1	Evaluation of constant unsigned integer expressions shall not lead to wrap-around.	No
AUTOSAR++14:A6-2-1	Move and copy assignment operators shall either move or respectively copy base classes and data members of a class, without any side effects.	No
AUTOSAR++14:M6-2-1	Assignment operators shall not be used in sub-expressions.	Yes
AUTOSAR++14:A6-2-2	Expression statements shall not be explicit calls to constructors of temporary objects only.	No
AUTOSAR++14:M6-2-2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.	Yes
AUTOSAR++14:M6-2-3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.	No

AUTOSAR++14:M6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Yes
AUTOSAR++14:A6-4-1	A switch statement shall have at least two case-clauses, distinct from the default label.	Yes
AUTOSAR++14:M6-4-1	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	Yes
AUTOSAR++14:M6-4-2	All if ... else if constructs shall be terminated with an else clause.	Yes
AUTOSAR++14:M6-4-3	A switch statement shall be a well-formed switch statement.	Yes
AUTOSAR++14:M6-4-4	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	Yes
AUTOSAR++14:M6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause.	Yes
AUTOSAR++14:M6-4-6	The final clause of a switch statement shall be the default-clause.	Yes
AUTOSAR++14:M6-4-7	The condition of a switch statement shall not have bool type.	Yes
AUTOSAR++14:A6-5-1	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used.	No
AUTOSAR++14:A6-5-2	A for loop shall contain a single loop-counter which shall not have floating-point type.	Yes
AUTOSAR++14:M6-5-2	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.	No
AUTOSAR++14:A6-5-3	Do statements should not be used.	No
AUTOSAR++14:M6-5-3	The loop-counter shall not be modified within condition or statement.	Yes
AUTOSAR++14:A6-5-4	For-init-statement and expression should not perform actions other than loop-counter initialization and modification.	Yes
AUTOSAR++14:M6-5-4	The loop-counter shall be modified by one of: --, ++, - = n, or + = n; where n remains constant for the duration of the loop.	Yes
AUTOSAR++14:M6-5-5	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.	Yes
AUTOSAR++14:M6-5-6	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.	No
AUTOSAR++14:A6-6-1	The goto statement shall not be used.	Yes
AUTOSAR++14:M6-6-1	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.	Yes
AUTOSAR++14:M6-6-2	The goto statement shall jump to a label declared later in the same function body.	Yes
AUTOSAR++14:M6-6-3	The continue statement shall only be used within a well-formed for loop.	Yes
AUTOSAR++14:A7-1-1	Constexpr or const specifiers shall be used for immutable data declaration.	Yes
AUTOSAR++14:A7-1-2	The constexpr specifier shall be used for values that can be determined at compile time.	No
AUTOSAR++14:M7-1-2	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.	Yes
AUTOSAR++14:A7-1-3	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name.	No
AUTOSAR++14:A7-1-4	The register keyword shall not be used.	No
AUTOSAR++14:A7-1-5	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.	No
AUTOSAR++14:A7-1-6	The typedef specifier shall not be used.	No
AUTOSAR++14:A7-1-7	Each expression statement and identifier declaration shall be placed on a separate line.	Yes
AUTOSAR++14:A7-1-8	A non-type specifier shall be placed before a type specifier in a declaration.	No
AUTOSAR++14:A7-1-9	A class, structure, or enumeration shall not be declared in the definition of its type.	No
AUTOSAR++14:A7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	Yes
AUTOSAR++14:A7-2-2	Enumeration underlying base type shall be explicitly defined.	No
AUTOSAR++14:A7-2-3	Enumerations shall be declared as scoped enum classes.	No
AUTOSAR++14:A7-2-4	In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized.	Yes

AUTOSAR++14:A7-2-5	Enumerations should be used to represent sets of related named constants.	No
AUTOSAR++14:A7-3-1	All overloads of a function shall be visible from where it is called.	Yes
AUTOSAR++14:M7-3-1	The global namespace shall only contain main, namespace declarations and extern "C" declarations.	Yes
AUTOSAR++14:M7-3-2	The identifier main shall not be used for a function other than the global function main.	No
AUTOSAR++14:M7-3-3	There shall be no unnamed namespaces in header files.	Yes
AUTOSAR++14:M7-3-4	Using-directives shall not be used.	Yes
AUTOSAR++14:M7-3-6	Using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes
AUTOSAR++14:A7-4-1	The asm declaration shall not be used.	Yes
AUTOSAR++14:M7-4-1	All usage of assembler shall be documented.	No
AUTOSAR++14:M7-4-2	Assembler instructions shall only be introduced using the asm declaration.	Yes
AUTOSAR++14:M7-4-3	Assembly language shall be encapsulated and isolated.	Yes
AUTOSAR++14:A7-5-1	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	No
AUTOSAR++14:M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes
AUTOSAR++14:A7-5-2	Functions shall not call themselves, either directly or indirectly.	Yes
AUTOSAR++14:M7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Yes
AUTOSAR++14:A7-6-1	Functions declared with the [[noreturn]] attribute shall not return.	Yes
AUTOSAR++14:M8-0-1	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.	No
AUTOSAR++14:A8-2-1	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.	No
AUTOSAR++14:M8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Yes
AUTOSAR++14:A8-4-1	Functions shall not be defined using the ellipsis notation.	Yes
AUTOSAR++14:A8-4-2	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	Yes
AUTOSAR++14:M8-4-2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	Yes
AUTOSAR++14:A8-4-3	Common ways of passing parameters should be used.	No
AUTOSAR++14:A8-4-4	Multiple output values from a function should be returned as a struct or tuple.	No
AUTOSAR++14:M8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Yes
AUTOSAR++14:A8-4-5	"consume" parameters declared as X && shall always be moved from.	No
AUTOSAR++14:A8-4-6	"forward" parameters declared as T && shall always be forwarded.	No
AUTOSAR++14:A8-4-7	"in" parameters for "cheap to copy" types shall be passed by value.	No
AUTOSAR++14:A8-4-8	Output parameters shall not be used.	No
AUTOSAR++14:A8-4-9	"in-out" parameters declared as T & shall be modified.	No
AUTOSAR++14:A8-4-10	A parameter shall be passed by reference if it can't be NULL.	No
AUTOSAR++14:A8-4-11	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics.	No
AUTOSAR++14:A8-4-12	A std::unique_ptr shall be passed to a function as: (1) a copy to express the function assumes ownership (2) an lvalue reference to express that the function replaces the managed object.	No
AUTOSAR++14:A8-4-13	A std::shared_ptr shall be passed to a function as: (1) a copy to express the function shares ownership (2) an lvalue reference to express that the function replaces the managed object (3) a const lvalue reference to express that the function retains a reference count.	No
AUTOSAR++14:A8-4-14	Interfaces shall be precisely and strongly typed.	No
AUTOSAR++14:A8-5-0	All memory shall be initialized before it is read.	Yes

AUTOSAR++14:A8-5-1	In an initialization list, the order of initialization shall be following: (1) virtual base classes in depth and left to right order of the inheritance graph, (2) direct base classes in left to right order of inheritance list, (3) non-static data members in the order they were declared in the class definition.	Yes
AUTOSAR++14:A8-5-2	Braced-initialization {}, without equals sign, shall be used for variable initialization.	No
AUTOSAR++14:M8-5-2	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.	Yes
AUTOSAR++14:A8-5-3	A variable of type auto shall not be initialized using {} or ={} braced-initialization.	No
AUTOSAR++14:A8-5-4	If a class has a user-declared constructor that takes a parameter of type std::initializer_list, then it shall be the only constructor apart from special member function constructors.	No
AUTOSAR++14:A9-3-1	Member functions shall not return non-const "raw" pointers or references to private or protected data owned by the class.	No
AUTOSAR++14:M9-3-1	Const member functions shall not return non-const pointers or references to class-data.	No
AUTOSAR++14:M9-3-3	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.	Yes
AUTOSAR++14:A9-5-1	Unions shall not be used.	Yes
AUTOSAR++14:A9-6-1	Data types used for interfacing with hardware or conforming to communication protocols shall be trivial, standard-layout and only contain members of types with defined sizes.	No
AUTOSAR++14:M9-6-1	When the absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented.	No
AUTOSAR++14:A9-6-2	Bit-fields shall be used only when interfacing to hardware or conforming to communication protocols.	No
AUTOSAR++14:M9-6-4	Named bit-fields with signed integer type shall have a length of more than one bit.	Yes
AUTOSAR++14:A10-0-1	Public inheritance shall be used to implement "is-a" relationship.	No
AUTOSAR++14:A10-0-2	Membership or non-public inheritance shall be used to implement "has-a" relationship.	No
AUTOSAR++14:A10-1-1	Class shall not be derived from more than one base class which is not an interface class.	No
AUTOSAR++14:M10-1-1	Classes should not be derived from virtual bases.	Yes
AUTOSAR++14:M10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy.	Yes
AUTOSAR++14:M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.	Yes
AUTOSAR++14:A10-2-1	Non-virtual public or protected member functions shall not be redefined in derived classes.	No
AUTOSAR++14:M10-2-1	All accessible entity names within a multiple inheritance hierarchy should be unique.	No
AUTOSAR++14:A10-3-1	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final.	No
AUTOSAR++14:A10-3-2	Each overriding virtual function shall be declared with the override or final specifier.	No
AUTOSAR++14:A10-3-3	Virtual functions shall not be introduced in a final class.	No
AUTOSAR++14:M10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.	No
AUTOSAR++14:A10-3-5	A user-defined assignment operator shall not be virtual.	No
AUTOSAR++14:A10-4-1	Hierarchies should be based on interface classes.	No
AUTOSAR++14:A11-0-1	A non-POD type should be defined as class.	No
AUTOSAR++14:M11-0-1	Member data in non-POD class types shall be private.	No
AUTOSAR++14:A11-0-2	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class.	No
AUTOSAR++14:A11-3-1	Friend declarations shall not be used.	No
AUTOSAR++14:A12-0-1	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.	No
AUTOSAR++14:A12-0-2	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	No
AUTOSAR++14:A12-1-1	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	No
AUTOSAR++14:M12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor.	Yes

AUTOSAR++14:A12-1-2	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	No
AUTOSAR++14:A12-1-3	If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.	No
AUTOSAR++14:A12-1-4	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	No
AUTOSAR++14:A12-1-5	Common class initialization for non-constant members shall be done by a delegating constructor.	No
AUTOSAR++14:A12-1-6	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors.	No
AUTOSAR++14:A12-4-1	Destructor of a base class shall be public virtual, public override or protected non-virtual.	Yes
AUTOSAR++14:A12-4-2	If a public destructor of a class is non-virtual, then the class should be declared final.	No
AUTOSAR++14:A12-6-1	All class data members that are initialized by the constructor shall be initialized using member initializers.	No
AUTOSAR++14:A12-7-1	If the behavior of a user-defined special member function is identical to implicitly defined special member function, then it shall be defined "=default" or be left undefined.	No
AUTOSAR++14:A12-8-1	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any side effects.	Yes
AUTOSAR++14:A12-8-2	User-defined copy and move assignment operators should use user-defined no-throw swap function.	No
AUTOSAR++14:A12-8-3	Moved-from object shall not be read-accessed.	No
AUTOSAR++14:A12-8-4	Move constructor shall not initialize its class members and base classes using copy semantics.	No
AUTOSAR++14:A12-8-5	A copy assignment and a move assignment operators shall handle self-assignment.	No
AUTOSAR++14:A12-8-6	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.	Yes
AUTOSAR++14:A12-8-7	Assignment operators should be declared with the ref-qualifier &.	No
AUTOSAR++14:A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters.	No
AUTOSAR++14:A13-1-3	User defined literals operators shall only perform conversion of passed parameters.	No
AUTOSAR++14:A13-2-1	An assignment operator shall return a reference to "this".	No
AUTOSAR++14:A13-2-2	A binary arithmetic operator and a bitwise operator shall return a "prvalue".	No
AUTOSAR++14:A13-2-3	A relational operator shall return a boolean value.	Yes
AUTOSAR++14:A13-3-1	A function that contains "forwarding reference" as its argument shall not be overloaded.	No
AUTOSAR++14:A13-5-1	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented.	No
AUTOSAR++14:A13-5-2	All user-defined conversion operators shall be defined explicit.	No
AUTOSAR++14:A13-5-3	User-defined conversion operators should not be used.	No
AUTOSAR++14:A13-5-4	If two opposite operators are defined, one shall be defined in terms of the other.	No
AUTOSAR++14:A13-5-5	Comparison operators shall be non-member functions with identical parameter types and noexcept.	No
AUTOSAR++14:A13-6-1	Digit sequences separators ' shall only be used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits.	No
AUTOSAR++14:A14-1-1	A template should check if a specific template argument is suitable for this template.	No
AUTOSAR++14:A14-5-1	A template constructor shall not participate in overload resolution for a single argument of the enclosing class type.	No
AUTOSAR++14:A14-5-2	Class members that are not dependent on template class parameters should be defined in a separate base class.	No
AUTOSAR++14:A14-5-3	A non-member generic operator shall only be declared in a namespace that does not contain class (struct) type, enum type or union type declarations.	No
AUTOSAR++14:M14-5-3	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.	No
AUTOSAR++14:M14-6-1	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.	No
AUTOSAR++14:A14-7-1	A type used as a template argument shall provide all members that are used by the template.	No

AUTOSAR++14:A14-7-2	Template specialization shall be declared in the same file (1) as the primary template (2) as a user-defined type, for which the specialization is declared.	No
AUTOSAR++14:A14-8-2	Explicit specializations of function templates shall not be used.	No
AUTOSAR++14:A15-0-1	A function shall not exit with an exception if it is able to complete its task.	No
AUTOSAR++14:A15-0-2	At least the basic guarantee for exception safety shall be provided for all operations. In addition, each function may offer either the strong guarantee or the nothrow guarantee.	No
AUTOSAR++14:A15-0-3	Exception safety guarantee of a called function shall be considered.	No
AUTOSAR++14:M15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement.	No
AUTOSAR++14:A15-0-4	Unchecked exceptions shall be used to represent errors from which the caller cannot reasonably be expected to recover.	No
AUTOSAR++14:A15-0-5	Checked exceptions shall be used to represent errors from which the caller can reasonably be expected to recover.	No
AUTOSAR++14:A15-0-6	An analysis shall be performed to analyze the failure modes of exception handling. In particular, the following failure modes shall be analyzed: (a) worst time execution time not existing or cannot be determined, (b) stack not correctly unwound, (c) exception not thrown, other exception thrown, wrong catch activated, (d) memory not available while exception handling.	No
AUTOSAR++14:A15-0-7	Exception handling mechanism shall guarantee a deterministic worst-case time execution time.	No
AUTOSAR++14:A15-0-8	A worst-case execution time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions processing.	No
AUTOSAR++14:A15-1-1	Only instances of types derived from std::exception should be thrown.	No
AUTOSAR++14:M15-1-1	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.	No
AUTOSAR++14:A15-1-2	An exception object shall not be a pointer.	No
AUTOSAR++14:M15-1-2	NULL shall not be thrown explicitly.	No
AUTOSAR++14:A15-1-3	All thrown exceptions should be unique.	No
AUTOSAR++14:M15-1-3	An empty throw (throw;) shall only be used in the compound statement of a catch handler.	No
AUTOSAR++14:A15-1-4	If a function exits with an exception, then before a throw, the function shall place all objects/resources that the function constructed in valid states or it shall delete them.	No
AUTOSAR++14:A15-1-5	Exceptions shall not be thrown across execution boundaries.	No
AUTOSAR++14:A15-2-1	Constructors that are not noexcept shall not be invoked before program startup.	No
AUTOSAR++14:A15-2-2	If a constructor is not noexcept and the constructor cannot finish object initialization, then it shall deallocate the object's resources and it shall throw an exception.	No
AUTOSAR++14:M15-3-1	Exceptions shall be raised only after start-up and before termination.	No
AUTOSAR++14:A15-3-2	If a function throws an exception, it shall be handled when meaningful actions can be taken, otherwise it shall be propagated.	No
AUTOSAR++14:A15-3-3	Main function and a task main function shall catch at least: base class exceptions from all third-party libraries used, std::exception and all otherwise unhandled exceptions.	No
AUTOSAR++14:M15-3-3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.	No
AUTOSAR++14:A15-3-4	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines.	No
AUTOSAR++14:M15-3-4	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.	No
AUTOSAR++14:A15-3-5	A class type exception shall be caught by reference or const reference.	Yes
AUTOSAR++14:M15-3-6	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.	Yes
AUTOSAR++14:M15-3-7	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.	Yes
AUTOSAR++14:A15-4-1	Dynamic exception-specification shall not be used.	No
AUTOSAR++14:A15-4-2	If a function is declared to be noexcept, noexcept(true) or noexcept(), then it shall not exit with an exception.	No
AUTOSAR++14:A15-4-3	The noexcept specification of a function shall either be identical across all translation units, or identical or more restrictive between a virtual member function and an override.	No

AUTOSAR++14:A15-4-4	A declaration of non-throwing function shall contain noexcept specification.	No
AUTOSAR++14:A15-4-5	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its overriders.	No
AUTOSAR++14:A15-5-1	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate.	No
AUTOSAR++14:A15-5-2	Program shall not be abruptly terminated. In particular, an implicit or explicit invocation of <code>std::abort()</code> , <code>std::quick_exit()</code> , <code>std::_Exit()</code> , <code>std::terminate()</code> shall not be done.	Yes
AUTOSAR++14:A15-5-3	The <code>std::terminate()</code> function shall not be called implicitly.	No
AUTOSAR++14:A16-0-1	The pre-processor shall only be used for unconditional and conditional file inclusion and include guards, and using the following directives: (1) <code>#ifndef</code> , (2) <code>#ifdef</code> , (3) <code>#if</code> , (4) <code>#if defined</code> , (5) <code>#elif</code> , (6) <code>#else</code> , (7) <code>#define</code> , (8) <code>#endif</code> , (9) <code>#include</code> .	Yes
AUTOSAR++14:M16-0-1	<code>#include</code> directives in a file shall only be preceded by other pre-processor directives or comments.	Yes
AUTOSAR++14:M16-0-2	Macros shall only be <code>#define'd</code> or <code>#undef'd</code> in the global namespace.	No
AUTOSAR++14:M16-0-5	Arguments to a function-like macro shall not contain tokens that look like pre-processing directives.	Yes
AUTOSAR++14:M16-0-6	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of <code>#</code> or <code>##</code> .	Yes
AUTOSAR++14:M16-0-7	Undefined macro identifiers shall not be used in <code>#if</code> or <code>#elif</code> pre-processor directives, except as operands to the defined operator.	Yes
AUTOSAR++14:M16-0-8	If the <code>#</code> token appears as the first token on a line, then it shall be immediately followed by a pre-processing token.	Yes
AUTOSAR++14:M16-1-1	The defined pre-processor operator shall only be used in one of the two standard forms.	No
AUTOSAR++14:M16-1-2	All <code>#else</code> , <code>#elif</code> and <code>#endif</code> pre-processor directives shall reside in the same file as the <code>#if</code> or <code>#ifdef</code> directive to which they are related.	Yes
AUTOSAR++14:A16-2-1	The <code>'</code> , <code>"</code> , <code>/*</code> , <code>//</code> , <code>\</code> characters shall not occur in a header file name or in <code>#include</code> directive.	Yes
AUTOSAR++14:A16-2-2	There shall be no unused include directives.	No
AUTOSAR++14:A16-2-3	An include directive shall be added explicitly for every symbol used in a file.	No
AUTOSAR++14:M16-2-3	Include guards shall be provided.	No
AUTOSAR++14:M16-3-1	There shall be at most one occurrence of the <code>#</code> or <code>##</code> operators in a single macro definition.	Yes
AUTOSAR++14:M16-3-2	The <code>#</code> and <code>##</code> operators should not be used.	Yes
AUTOSAR++14:A16-6-1	<code>#error</code> directive shall not be used.	No
AUTOSAR++14:A16-7-1	The <code>#pragma</code> directive shall not be used.	No
AUTOSAR++14:A17-0-1	Reserved identifiers, macros and functions in the C++ standard library shall not be defined, redefined or undefined.	Yes
AUTOSAR++14:A17-0-2	All project's code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.	No
AUTOSAR++14:M17-0-2	The names of standard library macros and objects shall not be reused.	Yes
AUTOSAR++14:M17-0-3	The names of standard library functions shall not be overridden.	Yes
AUTOSAR++14:M17-0-5	The <code>setjmp</code> macro and the <code>longjmp</code> function shall not be used.	Yes
AUTOSAR++14:A17-1-1	Use of the C Standard Library shall be encapsulated and isolated.	No
AUTOSAR++14:A17-6-1	Non-standard entities shall not be added to standard namespaces.	No
AUTOSAR++14:A18-0-1	The C library facilities shall only be accessed through C++ library headers.	Yes
AUTOSAR++14:A18-0-2	The error state of a conversion from string to a numeric value shall be checked.	Yes
AUTOSAR++14:A18-0-3	The library (<code>locale.h</code>) and the <code>setlocale</code> function shall not be used.	No
AUTOSAR++14:M18-0-3	The library functions <code>abort</code> , <code>exit</code> , <code>getenv</code> and <code>system</code> from library shall not be used.	Yes
AUTOSAR++14:M18-0-4	The time handling functions of library shall not be used.	Yes
AUTOSAR++14:M18-0-5	The unbounded functions of library shall not be used.	Yes
AUTOSAR++14:A18-1-1	C-style arrays shall not be used.	No
AUTOSAR++14:A18-1-2	The <code>std::vector</code> specialization shall not be used.	No

AUTOSAR++14:A18-1-3	The std::auto_ptr type shall not be used.	No
AUTOSAR++14:A18-1-4	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type.	No
AUTOSAR++14:A18-1-6	All std::hash specializations for user-defined types shall have a noexcept function call operator.	No
AUTOSAR++14:M18-2-1	The macro offsetof shall not be used.	Yes
AUTOSAR++14:A18-5-1	Functions malloc, calloc, realloc and free shall not be used.	Yes
AUTOSAR++14:A18-5-2	Non-placement new or delete expressions shall not be used.	No
AUTOSAR++14:A18-5-3	The form of the delete expression shall match the form of the new expression used to allocate the memory.	Yes
AUTOSAR++14:A18-5-4	If a project has sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined.	No
AUTOSAR++14:A18-5-5	Memory management functions shall ensure the following: (a) deterministic behavior resulting with the existence of worst-case execution time, (b) avoiding memory fragmentation, (c) avoid running out of memory, (d) avoiding mismatched allocations or deallocations, (e) no dependence on non-deterministic calls to kernel.	Yes
AUTOSAR++14:A18-5-6	An analysis shall be performed to analyze the failure modes of dynamic memory management. In particular, the following failure modes shall be analyzed: (a) non-deterministic behavior resulting with nonexistence of worst-case execution time, (b) memory fragmentation, (c) running out of memory, (d) mismatched allocations and deallocations, (e) dependence on non-deterministic calls to kernel.	No
AUTOSAR++14:A18-5-7	If non-realtime implementation of dynamic memory management functions is used in the project, then memory shall only be allocated and deallocated during non-realtime program phases.	Yes
AUTOSAR++14:A18-5-8	Objects that do not outlive a function shall have automatic storage duration.	No
AUTOSAR++14:A18-5-9	Custom implementations of dynamic memory allocation and deallocation functions shall meet the semantic requirements specified in the corresponding "Required behaviour" clause from the C++ Standard.	No
AUTOSAR++14:A18-5-10	Placement new shall be used only with properly aligned pointers to sufficient storage capacity.	No
AUTOSAR++14:A18-5-11	"operator new" and "operator delete" shall be defined together.	No
AUTOSAR++14:M18-7-1	The signal handling facilities of shall not be used.	Yes
AUTOSAR++14:A18-9-1	The std::bind shall not be used.	No
AUTOSAR++14:A18-9-2	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding reference.	No
AUTOSAR++14:A18-9-3	The std::move shall not be used on objects declared const or const&.	No
AUTOSAR++14:A18-9-4	An argument to std::forward shall not be subsequently used.	No
AUTOSAR++14:M19-3-1	The error indicator errno shall not be used.	No
AUTOSAR++14:A20-8-1	An already-owned pointer value shall not be stored in an unrelated smart pointer.	No
AUTOSAR++14:A20-8-2	A std::unique_ptr shall be used to represent exclusive ownership.	No
AUTOSAR++14:A20-8-3	A std::shared_ptr shall be used to represent shared ownership.	No
AUTOSAR++14:A20-8-4	A std::weak_ptr shall be used over std::shared_ptr if ownership sharing is not required.	No
AUTOSAR++14:A20-8-5	std::make_unique shall be used to construct objects owned by std::unique_ptr.	No
AUTOSAR++14:A20-8-6	std::make_shared shall be used to construct objects owned by std::shared_ptr.	No
AUTOSAR++14:A20-8-7	A std::weak_ptr shall be used to represent temporary shared ownership.	No
AUTOSAR++14:A21-8-1	Arguments to character-handling functions shall be representable as an unsigned char.	Yes
AUTOSAR++14:A23-0-1	An iterator shall not be implicitly converted to const_iterator.	No
AUTOSAR++14:A23-0-2	Elements of a container shall only be accessed via valid references, iterators, and pointers.	No
AUTOSAR++14:A25-1-1	Non-static data members or captured values of predicate function objects that are state related to this object's identity shall not be copied.	No
AUTOSAR++14:A25-4-1	Ordering predicates used with associative containers and STL sorting and related algorithms shall adhere to a strict weak ordering relation.	No
AUTOSAR++14:A26-5-1	Pseudorandom numbers shall not be generated using std::rand().	No
AUTOSAR++14:A26-5-2	Random number engines shall not be default-initialized.	No

AUTOSAR++14:A27-0-1	Inputs from independent components shall be validated.	Yes
AUTOSAR++14:M27-0-1	The stream input/output library shall not be used.	Yes
AUTOSAR++14:A27-0-2	A C-style string shall guarantee sufficient space for data and the null terminator.	Yes
AUTOSAR++14:A27-0-3	Alternate input and output operations on a file stream shall not be used without an intervening flush or positioning call.	Yes
AUTOSAR++14:A27-0-4	C-style strings shall not be used.	No



AUTOSAR BROAD MAPPING (CODESONAR V7.1)

The following table contains CodeSonar warning classes that are broadly mapped to AUTOSAR categories.

Rule	Rule Name	Supported
AUTOSAR++14:A0-1-1	A project shall not contain instances of non-volatile variables being given values that are not subsequently used.	Yes
AUTOSAR++14:M0-1-1	A project shall not contain unreachable code.	Yes
AUTOSAR++14:A0-1-2	The value returned by a function having a non-void return type that is not an overloaded operator shall be used.	Yes
AUTOSAR++14:M0-1-2	A project shall not contain infeasible paths.	Yes
AUTOSAR++14:A0-1-3	Every function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used.	No
AUTOSAR++14:M0-1-3	A project shall not contain unused variables.	Yes
AUTOSAR++14:A0-1-4	There shall be no unused named parameters in non-virtual functions.	Yes
AUTOSAR++14:M0-1-4	A project shall not contain non-volatile POD variables having only one use.	Yes
AUTOSAR++14:A0-1-5	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it.	Yes
AUTOSAR++14:A0-1-6	There should be no unused type declarations.	Yes
AUTOSAR++14:M0-1-8	All functions with void return type shall have external side effect(s).	Yes
AUTOSAR++14:M0-1-9	There shall be no dead code.	Yes
AUTOSAR++14:M0-1-10	Every defined function should be called at least once.	Yes
AUTOSAR++14:M0-2-1	An object shall not be assigned to an overlapping object.	Yes
AUTOSAR++14:M0-3-1	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.	No
AUTOSAR++14:M0-3-2	If a function generates error information, then that error information shall be tested.	Yes
AUTOSAR++14:A0-4-1	Floating-point implementation shall comply with IEEE 754 standard.	No
AUTOSAR++14:M0-4-1	Use of scaled-integer or fixed-point arithmetic shall be documented.	No
AUTOSAR++14:A0-4-2	Type long double shall not be used.	No
AUTOSAR++14:M0-4-2	Use of floating-point arithmetic shall be documented.	No
AUTOSAR++14:A0-4-3	The implementations in the chosen compiler shall strictly comply with the C++14 Language Standard.	No
AUTOSAR++14:A0-4-4	Range, domain and pole errors shall be checked when using math functions.	Yes
AUTOSAR++14:M1-0-2	Multiple compilers shall only be used if they have a common, defined interface.	No
AUTOSAR++14:A1-1-1	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features.	Yes
AUTOSAR++14:A1-1-2	A warning level of the compilation process shall be set in compliance with project policies.	Yes
AUTOSAR++14:A1-1-3	An optimization option that disregards strict standard compliance shall not be turned on in the chosen compiler.	No
AUTOSAR++14:A1-2-1	When using a compiler toolchain (including preprocessor, compiler itself, linker, C++ standard libraries) in safety-related software, the tool confidence level (TCL) shall be determined. In case of TCL2 or TCL3, the compiler shall undergo a "Qualification of a software tool", as per ISO 26262-8.11.4.6.	No
AUTOSAR++14:A1-4-1	Code metrics and their valid boundaries shall be defined and code shall comply with defined boundaries of code metrics.	Yes
AUTOSAR++14:A1-4-3	All code should compile free of compiler warnings.	No
AUTOSAR++14:A2-3-1	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code.	No
AUTOSAR++14:A2-5-1	Trigraphs shall not be used.	Yes

AUTOSAR++14:A2-5-2	Digraphs shall not be used.	No
AUTOSAR++14:A2-7-1	The character \ shall not occur as a last character of a C++ comment.	Yes
AUTOSAR++14:M2-7-1	The character sequence /* shall not be used within a C-style comment.	Yes
AUTOSAR++14:A2-7-2	Sections of code shall not be "commented out".	Yes
AUTOSAR++14:A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation.	No
AUTOSAR++14:A2-7-5	Comments shall not document any actions or sources (e.g. tables, figures, paragraphs, etc.) that are outside of the file.	No
AUTOSAR++14:A2-8-1	A header file name should reflect the logical entity for which it provides declarations.	No
AUTOSAR++14:A2-8-2	An implementation file name should reflect the logical entity for which it provides definitions.	No
AUTOSAR++14:A2-10-1	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope.	Yes
AUTOSAR++14:M2-10-1	Different identifiers shall be typographically unambiguous.	Yes
AUTOSAR++14:A2-10-4	The identifier name of a non-member object with static storage duration or static function shall not be reused within a namespace.	Yes
AUTOSAR++14:A2-10-5	An identifier name of a function with static storage duration or a non-member object with external or internal linkage should not be reused.	Yes
AUTOSAR++14:A2-10-6	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope.	No
AUTOSAR++14:A2-11-1	Volatile keyword shall not be used.	No
AUTOSAR++14:A2-13-1	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used.	No
AUTOSAR++14:A2-13-2	String literals with different encoding prefixes shall not be concatenated.	No
AUTOSAR++14:M2-13-2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes
AUTOSAR++14:A2-13-3	Type wchar_t shall not be used.	No
AUTOSAR++14:M2-13-3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes
AUTOSAR++14:A2-13-4	String literals shall not be assigned to non-constant pointers.	Yes
AUTOSAR++14:M2-13-4	Literal suffixes shall be upper case.	Yes
AUTOSAR++14:A2-13-5	Hexadecimal constants should be upper case.	No
AUTOSAR++14:A2-13-6	Universal character names shall be used only inside character or string literals.	No
AUTOSAR++14:A3-1-1	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.	Yes
AUTOSAR++14:A3-1-2	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx".	No
AUTOSAR++14:M3-1-2	Functions shall not be declared at block scope.	Yes
AUTOSAR++14:A3-1-3	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp".	No
AUTOSAR++14:A3-1-4	When an array with external linkage is declared, its size shall be stated explicitly.	Yes
AUTOSAR++14:A3-1-5	A function definition shall only be placed in a class definition if (1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template.	No
AUTOSAR++14:A3-1-6	Trivial accessor and mutator functions should be inlined.	No
AUTOSAR++14:M3-2-1	All declarations of an object or function shall have compatible types.	Yes
AUTOSAR++14:M3-2-2	The One Definition Rule shall not be violated.	Yes
AUTOSAR++14:M3-2-3	A type, object or function that is used in multiple translation units shall be declared in one and only one file.	Yes
AUTOSAR++14:M3-2-4	An identifier with external linkage shall have exactly one definition.	Yes
AUTOSAR++14:A3-3-1	Objects or functions with external linkage (including members of named namespaces) shall be declared in a header file.	No
AUTOSAR++14:A3-3-2	Static and thread-local objects shall be constant-initialized.	Yes
AUTOSAR++14:M3-3-2	If a function has internal linkage then all re-declarations shall include the static storage class specifier.	No
AUTOSAR++14:M3-4-1	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.	Yes

AUTOSAR++14:A3-8-1	An object shall not be accessed outside of its lifetime.	Yes
AUTOSAR++14:A3-9-1	Fixed width integer types from <code><stdint.h></code> , indicating the size and signedness, shall be used in place of the basic numerical types.	Yes
AUTOSAR++14:M3-9-1	The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations.	Yes
AUTOSAR++14:M3-9-3	The underlying bit representations of floating-point values shall not be used.	Yes
AUTOSAR++14:A4-5-1	Expressions with type <code>enum</code> or <code>enum class</code> shall not be used as operands to built-in and overloaded operators other than the subscript operator <code>[]</code> , the assignment operator <code>=</code> , the equality operators <code>==</code> and <code>!=</code> , the unary <code>&</code> operator, and the relational operators <code><</code> , <code><=</code> , <code>></code> , <code>>=</code> .	Yes
AUTOSAR++14:M4-5-1	Expressions with type <code>bool</code> shall not be used as operands to built-in operators other than the assignment operator <code>=</code> , the logical operators <code>&&</code> , <code> </code> , <code>!</code> , the equality operators <code>==</code> and <code>!=</code> , the unary <code>&</code> operator, and the conditional operator.	Yes
AUTOSAR++14:M4-5-3	Expressions with type (plain) <code>char</code> and <code>wchar_t</code> shall not be used as operands to built-in operators other than the assignment operator <code>=</code> , the equality operators <code>==</code> and <code>!=</code> , and the unary <code>&</code> operator.	Yes
AUTOSAR++14:A4-7-1	An integer expression shall not lead to data loss.	Yes
AUTOSAR++14:A4-10-1	Only <code>nullptr</code> literal shall be used as the null-pointer-constant.	No
AUTOSAR++14:M4-10-1	<code>NULL</code> shall not be used as an integer value.	Yes
AUTOSAR++14:M4-10-2	Literal zero (0) shall not be used as the null-pointer-constant.	Yes
AUTOSAR++14:A5-0-1	The value of an expression shall be the same under any order of evaluation that the standard permits.	No
AUTOSAR++14:A5-0-2	The condition of an if-statement and the condition of an iteration statement shall have type <code>bool</code> .	Yes
AUTOSAR++14:M5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions.	Yes
AUTOSAR++14:A5-0-3	The declaration of objects shall contain no more than two levels of pointer indirection.	Yes
AUTOSAR++14:M5-0-3	A <code>cvalue</code> expression shall not be implicitly converted to a different underlying type.	Yes
AUTOSAR++14:A5-0-4	Pointer arithmetic shall not be used with pointers to non-final classes.	Yes
AUTOSAR++14:M5-0-4	An implicit integral conversion shall not change the signedness of the underlying type.	Yes
AUTOSAR++14:M5-0-5	There shall be no implicit floating-integral conversions.	Yes
AUTOSAR++14:M5-0-6	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.	Yes
AUTOSAR++14:M5-0-7	There shall be no explicit floating-integral conversions of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-8	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a <code>cvalue</code> expression.	Yes
AUTOSAR++14:M5-0-10	If the bitwise operators <code>~</code> and <code><<</code> are applied to an operand with an underlying type of unsigned <code>char</code> or unsigned <code>short</code> , the result shall be immediately cast to the underlying type of the operand.	Yes
AUTOSAR++14:M5-0-11	The plain <code>char</code> type shall only be used for the storage and use of character values.	Yes
AUTOSAR++14:M5-0-12	Signed <code>char</code> and unsigned <code>char</code> type shall only be used for the storage and use of numeric values.	Yes
AUTOSAR++14:M5-0-14	The first operand of a conditional-operator shall have type <code>bool</code> .	Yes
AUTOSAR++14:M5-0-15	Array indexing shall be the only form of pointer arithmetic.	Yes
AUTOSAR++14:M5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.	Yes
AUTOSAR++14:M5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array.	Yes
AUTOSAR++14:M5-0-18	<code>></code> , <code>>=</code> , <code><</code> , <code><=</code> shall not be applied to objects of pointer type, except where they point to the same array.	Yes
AUTOSAR++14:M5-0-20	Non-constant operands to a binary bitwise operator shall have the same underlying type.	Yes
AUTOSAR++14:M5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type.	Yes
AUTOSAR++14:A5-1-1	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead.	No
AUTOSAR++14:A5-1-2	Variables shall not be implicitly captured in a lambda expression.	Yes
AUTOSAR++14:A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression.	Yes

AUTOSAR++14:A5-1-4	A lambda expression object shall not outlive any of its reference-captured objects.	No
AUTOSAR++14:A5-1-6	Return type of a non-void return type lambda expression should be explicitly specified.	Yes
AUTOSAR++14:A5-1-7	A lambda shall not be an operand to decltype or typeid.	No
AUTOSAR++14:A5-1-8	Lambda expressions should not be defined inside another lambda expression.	No
AUTOSAR++14:A5-1-9	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression.	No
AUTOSAR++14:A5-2-1	dynamic_cast should not be used.	No
AUTOSAR++14:A5-2-2	Traditional C-style casts shall not be used.	Yes
AUTOSAR++14:M5-2-2	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.	Yes
AUTOSAR++14:A5-2-3	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.	Yes
AUTOSAR++14:M5-2-3	Casts from a base class to a derived class should not be performed on polymorphic types.	No
AUTOSAR++14:A5-2-4	reinterpret_cast shall not be used.	Yes
AUTOSAR++14:A5-2-5	An array or container shall not be accessed beyond its range.	Yes
AUTOSAR++14:A5-2-6	The operands of a logical && or shall be parenthesized if the operands contain binary operators.	No
AUTOSAR++14:M5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.	Yes
AUTOSAR++14:M5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes
AUTOSAR++14:M5-2-9	A cast shall not convert a pointer type to an integral type.	Yes
AUTOSAR++14:M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression.	Yes
AUTOSAR++14:M5-2-11	The comma operator, && operator and the operator shall not be overloaded.	Yes
AUTOSAR++14:M5-2-12	An identifier with array type passed as a function argument shall not decay to a pointer.	Yes
AUTOSAR++14:A5-3-1	Evaluation of the operand to the typeid operator shall not contain side effects.	No
AUTOSAR++14:M5-3-1	Each operand of the ! operator, the logical && or the logical operators shall have type bool.	Yes
AUTOSAR++14:A5-3-2	Null pointers shall not be dereferenced.	Yes
AUTOSAR++14:M5-3-2	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.	Yes
AUTOSAR++14:A5-3-3	Pointers to incomplete class types shall not be deleted.	No
AUTOSAR++14:M5-3-3	The unary & operator shall not be overloaded.	Yes
AUTOSAR++14:M5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects.	Yes
AUTOSAR++14:A5-5-1	A pointer to member shall not access non-existent class members.	No
AUTOSAR++14:A5-6-1	The right hand operand of the integer division or remainder operators shall not be equal to zero.	Yes
AUTOSAR++14:M5-8-1	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes
AUTOSAR++14:A5-10-1	A pointer to member virtual function shall only be tested for equality with null-pointer-constant.	No
AUTOSAR++14:M5-14-1	The right hand operand of a logical &&, operators shall not contain side effects.	Yes
AUTOSAR++14:A5-16-1	The ternary conditional operator shall not be used as a sub-expression.	No
AUTOSAR++14:M5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.	No
AUTOSAR++14:M5-18-1	The comma operator shall not be used.	Yes
AUTOSAR++14:M5-19-1	Evaluation of constant unsigned integer expressions shall not lead to wrap-around.	No
AUTOSAR++14:A6-2-1	Move and copy assignment operators shall either move or respectively copy base classes and data members of a class, without any side effects.	No
AUTOSAR++14:M6-2-1	Assignment operators shall not be used in sub-expressions.	Yes
AUTOSAR++14:A6-2-2	Expression statements shall not be explicit calls to constructors of temporary objects only.	No
AUTOSAR++14:M6-2-2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.	Yes
AUTOSAR++14:M6-2-3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character.	No

AUTOSAR++14:M6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.	Yes
AUTOSAR++14:A6-4-1	A switch statement shall have at least two case-clauses, distinct from the default label.	Yes
AUTOSAR++14:M6-4-1	An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement.	Yes
AUTOSAR++14:M6-4-2	All if ... else if constructs shall be terminated with an else clause.	Yes
AUTOSAR++14:M6-4-3	A switch statement shall be a well-formed switch statement.	Yes
AUTOSAR++14:M6-4-4	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.	Yes
AUTOSAR++14:M6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause.	Yes
AUTOSAR++14:M6-4-6	The final clause of a switch statement shall be the default-clause.	Yes
AUTOSAR++14:M6-4-7	The condition of a switch statement shall not have bool type.	Yes
AUTOSAR++14:A6-5-1	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used.	No
AUTOSAR++14:A6-5-2	A for loop shall contain a single loop-counter which shall not have floating-point type.	Yes
AUTOSAR++14:M6-5-2	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.	No
AUTOSAR++14:A6-5-3	Do statements should not be used.	No
AUTOSAR++14:M6-5-3	The loop-counter shall not be modified within condition or statement.	Yes
AUTOSAR++14:A6-5-4	For-init-statement and expression should not perform actions other than loop-counter initialization and modification.	Yes
AUTOSAR++14:M6-5-4	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop.	Yes
AUTOSAR++14:M6-5-5	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.	Yes
AUTOSAR++14:M6-5-6	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.	No
AUTOSAR++14:A6-6-1	The goto statement shall not be used.	Yes
AUTOSAR++14:M6-6-1	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.	Yes
AUTOSAR++14:M6-6-2	The goto statement shall jump to a label declared later in the same function body.	Yes
AUTOSAR++14:M6-6-3	The continue statement shall only be used within a well-formed for loop.	Yes
AUTOSAR++14:A7-1-1	Constexpr or const specifiers shall be used for immutable data declaration.	Yes
AUTOSAR++14:A7-1-2	The constexpr specifier shall be used for values that can be determined at compile time.	No
AUTOSAR++14:M7-1-2	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified.	Yes
AUTOSAR++14:A7-1-3	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name.	No
AUTOSAR++14:A7-1-4	The register keyword shall not be used.	No
AUTOSAR++14:A7-1-5	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax.	No
AUTOSAR++14:A7-1-6	The typedef specifier shall not be used.	No
AUTOSAR++14:A7-1-7	Each expression statement and identifier declaration shall be placed on a separate line.	Yes
AUTOSAR++14:A7-1-8	A non-type specifier shall be placed before a type specifier in a declaration.	No
AUTOSAR++14:A7-1-9	A class, structure, or enumeration shall not be declared in the definition of its type.	No
AUTOSAR++14:A7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.	Yes
AUTOSAR++14:A7-2-2	Enumeration underlying base type shall be explicitly defined.	No
AUTOSAR++14:A7-2-3	Enumerations shall be declared as scoped enum classes.	No
AUTOSAR++14:A7-2-4	In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized.	Yes

AUTOSAR++14:A7-2-5	Enumerations should be used to represent sets of related named constants.	No
AUTOSAR++14:A7-3-1	All overloads of a function shall be visible from where it is called.	Yes
AUTOSAR++14:M7-3-1	The global namespace shall only contain main, namespace declarations and extern "C" declarations.	Yes
AUTOSAR++14:M7-3-2	The identifier main shall not be used for a function other than the global function main.	No
AUTOSAR++14:M7-3-3	There shall be no unnamed namespaces in header files.	Yes
AUTOSAR++14:M7-3-4	Using-directives shall not be used.	Yes
AUTOSAR++14:M7-3-6	Using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes
AUTOSAR++14:A7-4-1	The asm declaration shall not be used.	Yes
AUTOSAR++14:M7-4-1	All usage of assembler shall be documented.	No
AUTOSAR++14:M7-4-2	Assembler instructions shall only be introduced using the asm declaration.	Yes
AUTOSAR++14:M7-4-3	Assembly language shall be encapsulated and isolated.	Yes
AUTOSAR++14:A7-5-1	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	No
AUTOSAR++14:M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes
AUTOSAR++14:A7-5-2	Functions shall not call themselves, either directly or indirectly.	Yes
AUTOSAR++14:M7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Yes
AUTOSAR++14:A7-6-1	Functions declared with the [[noreturn]] attribute shall not return.	Yes
AUTOSAR++14:M8-0-1	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.	No
AUTOSAR++14:A8-2-1	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.	No
AUTOSAR++14:M8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Yes
AUTOSAR++14:A8-4-1	Functions shall not be defined using the ellipsis notation.	Yes
AUTOSAR++14:A8-4-2	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.	Yes
AUTOSAR++14:M8-4-2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	Yes
AUTOSAR++14:A8-4-3	Common ways of passing parameters should be used.	No
AUTOSAR++14:A8-4-4	Multiple output values from a function should be returned as a struct or tuple.	No
AUTOSAR++14:M8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Yes
AUTOSAR++14:A8-4-5	"consume" parameters declared as X && shall always be moved from.	No
AUTOSAR++14:A8-4-6	"forward" parameters declared as T && shall always be forwarded.	No
AUTOSAR++14:A8-4-7	"in" parameters for "cheap to copy" types shall be passed by value.	No
AUTOSAR++14:A8-4-8	Output parameters shall not be used.	No
AUTOSAR++14:A8-4-9	"in-out" parameters declared as T & shall be modified.	No
AUTOSAR++14:A8-4-10	A parameter shall be passed by reference if it can't be NULL.	No
AUTOSAR++14:A8-4-11	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics.	No
AUTOSAR++14:A8-4-12	A std::unique_ptr shall be passed to a function as: (1) a copy to express the function assumes ownership (2) an lvalue reference to express that the function replaces the managed object.	No
AUTOSAR++14:A8-4-13	A std::shared_ptr shall be passed to a function as: (1) a copy to express the function shares ownership (2) an lvalue reference to express that the function replaces the managed object (3) a const lvalue reference to express that the function retains a reference count.	No
AUTOSAR++14:A8-4-14	Interfaces shall be precisely and strongly typed.	No
AUTOSAR++14:A8-5-0	All memory shall be initialized before it is read.	Yes

AUTOSAR++14:A8-5-1	In an initialization list, the order of initialization shall be following: (1) virtual base classes in depth and left to right order of the inheritance graph, (2) direct base classes in left to right order of inheritance list, (3) non-static data members in the order they were declared in the class definition.	Yes
AUTOSAR++14:A8-5-2	Braced-initialization {}, without equals sign, shall be used for variable initialization.	No
AUTOSAR++14:M8-5-2	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.	Yes
AUTOSAR++14:A8-5-3	A variable of type auto shall not be initialized using {} or ={} braced-initialization.	No
AUTOSAR++14:A8-5-4	If a class has a user-declared constructor that takes a parameter of type std::initializer_list, then it shall be the only constructor apart from special member function constructors.	No
AUTOSAR++14:A9-3-1	Member functions shall not return non-const "raw" pointers or references to private or protected data owned by the class.	No
AUTOSAR++14:M9-3-1	Const member functions shall not return non-const pointers or references to class-data.	No
AUTOSAR++14:M9-3-3	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.	Yes
AUTOSAR++14:A9-5-1	Unions shall not be used.	Yes
AUTOSAR++14:A9-6-1	Data types used for interfacing with hardware or conforming to communication protocols shall be trivial, standard-layout and only contain members of types with defined sizes.	No
AUTOSAR++14:M9-6-1	When the absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented.	No
AUTOSAR++14:A9-6-2	Bit-fields shall be used only when interfacing to hardware or conforming to communication protocols.	No
AUTOSAR++14:M9-6-4	Named bit-fields with signed integer type shall have a length of more than one bit.	Yes
AUTOSAR++14:A10-0-1	Public inheritance shall be used to implement "is-a" relationship.	No
AUTOSAR++14:A10-0-2	Membership or non-public inheritance shall be used to implement "has-a" relationship.	No
AUTOSAR++14:A10-1-1	Class shall not be derived from more than one base class which is not an interface class.	No
AUTOSAR++14:M10-1-1	Classes should not be derived from virtual bases.	Yes
AUTOSAR++14:M10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy.	Yes
AUTOSAR++14:M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.	Yes
AUTOSAR++14:A10-2-1	Non-virtual public or protected member functions shall not be redefined in derived classes.	No
AUTOSAR++14:M10-2-1	All accessible entity names within a multiple inheritance hierarchy should be unique.	No
AUTOSAR++14:A10-3-1	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final.	No
AUTOSAR++14:A10-3-2	Each overriding virtual function shall be declared with the override or final specifier.	No
AUTOSAR++14:A10-3-3	Virtual functions shall not be introduced in a final class.	No
AUTOSAR++14:M10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.	No
AUTOSAR++14:A10-3-5	A user-defined assignment operator shall not be virtual.	No
AUTOSAR++14:A10-4-1	Hierarchies should be based on interface classes.	No
AUTOSAR++14:A11-0-1	A non-POD type should be defined as class.	No
AUTOSAR++14:M11-0-1	Member data in non-POD class types shall be private.	No
AUTOSAR++14:A11-0-2	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class.	No
AUTOSAR++14:A11-3-1	Friend declarations shall not be used.	No
AUTOSAR++14:A12-0-1	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.	No
AUTOSAR++14:A12-0-2	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	No
AUTOSAR++14:A12-1-1	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	No
AUTOSAR++14:M12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor.	Yes

AUTOSAR++14:A12-1-2	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	No
AUTOSAR++14:A12-1-3	If all user-defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.	No
AUTOSAR++14:A12-1-4	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	No
AUTOSAR++14:A12-1-5	Common class initialization for non-constant members shall be done by a delegating constructor.	No
AUTOSAR++14:A12-1-6	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors.	No
AUTOSAR++14:A12-4-1	Destructor of a base class shall be public virtual, public override or protected non-virtual.	Yes
AUTOSAR++14:A12-4-2	If a public destructor of a class is non-virtual, then the class should be declared final.	No
AUTOSAR++14:A12-6-1	All class data members that are initialized by the constructor shall be initialized using member initializers.	No
AUTOSAR++14:A12-7-1	If the behavior of a user-defined special member function is identical to implicitly defined special member function, then it shall be defined "=default" or be left undefined.	No
AUTOSAR++14:A12-8-1	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any side effects.	Yes
AUTOSAR++14:A12-8-2	User-defined copy and move assignment operators should use user-defined no-throw swap function.	No
AUTOSAR++14:A12-8-3	Moved-from object shall not be read-accessed.	No
AUTOSAR++14:A12-8-4	Move constructor shall not initialize its class members and base classes using copy semantics.	No
AUTOSAR++14:A12-8-5	A copy assignment and a move assignment operators shall handle self-assignment.	No
AUTOSAR++14:A12-8-6	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.	Yes
AUTOSAR++14:A12-8-7	Assignment operators should be declared with the ref-qualifier &.	No
AUTOSAR++14:A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters.	No
AUTOSAR++14:A13-1-3	User defined literals operators shall only perform conversion of passed parameters.	No
AUTOSAR++14:A13-2-1	An assignment operator shall return a reference to "this".	No
AUTOSAR++14:A13-2-2	A binary arithmetic operator and a bitwise operator shall return a "prvalue".	No
AUTOSAR++14:A13-2-3	A relational operator shall return a boolean value.	Yes
AUTOSAR++14:A13-3-1	A function that contains "forwarding reference" as its argument shall not be overloaded.	No
AUTOSAR++14:A13-5-1	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented.	No
AUTOSAR++14:A13-5-2	All user-defined conversion operators shall be defined explicit.	No
AUTOSAR++14:A13-5-3	User-defined conversion operators should not be used.	No
AUTOSAR++14:A13-5-4	If two opposite operators are defined, one shall be defined in terms of the other.	No
AUTOSAR++14:A13-5-5	Comparison operators shall be non-member functions with identical parameter types and noexcept.	No
AUTOSAR++14:A13-6-1	Digit sequences separators ' shall only be used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits.	No
AUTOSAR++14:A14-1-1	A template should check if a specific template argument is suitable for this template.	No
AUTOSAR++14:A14-5-1	A template constructor shall not participate in overload resolution for a single argument of the enclosing class type.	No
AUTOSAR++14:A14-5-2	Class members that are not dependent on template class parameters should be defined in a separate base class.	No
AUTOSAR++14:A14-5-3	A non-member generic operator shall only be declared in a namespace that does not contain class (struct) type, enum type or union type declarations.	No
AUTOSAR++14:M14-5-3	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.	No
AUTOSAR++14:M14-6-1	In a class template with a dependent base, any name that may be found in that dependent base shall be referred to using a qualified-id or this->.	No
AUTOSAR++14:A14-7-1	A type used as a template argument shall provide all members that are used by the template.	No

AUTOSAR++14:A14-7-2	Template specialization shall be declared in the same file (1) as the primary template (2) as a user-defined type, for which the specialization is declared.	No
AUTOSAR++14:A14-8-2	Explicit specializations of function templates shall not be used.	No
AUTOSAR++14:A15-0-1	A function shall not exit with an exception if it is able to complete its task.	No
AUTOSAR++14:A15-0-2	At least the basic guarantee for exception safety shall be provided for all operations. In addition, each function may offer either the strong guarantee or the nothrow guarantee.	No
AUTOSAR++14:A15-0-3	Exception safety guarantee of a called function shall be considered.	No
AUTOSAR++14:M15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement.	No
AUTOSAR++14:A15-0-4	Unchecked exceptions shall be used to represent errors from which the caller cannot reasonably be expected to recover.	No
AUTOSAR++14:A15-0-5	Checked exceptions shall be used to represent errors from which the caller can reasonably be expected to recover.	No
AUTOSAR++14:A15-0-6	An analysis shall be performed to analyze the failure modes of exception handling. In particular, the following failure modes shall be analyzed: (a) worst time execution time not existing or cannot be determined, (b) stack not correctly unwound, (c) exception not thrown, other exception thrown, wrong catch activated, (d) memory not available while exception handling.	No
AUTOSAR++14:A15-0-7	Exception handling mechanism shall guarantee a deterministic worst-case time execution time.	No
AUTOSAR++14:A15-0-8	A worst-case execution time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions processing.	No
AUTOSAR++14:A15-1-1	Only instances of types derived from std::exception should be thrown.	No
AUTOSAR++14:M15-1-1	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.	No
AUTOSAR++14:A15-1-2	An exception object shall not be a pointer.	No
AUTOSAR++14:M15-1-2	NULL shall not be thrown explicitly.	No
AUTOSAR++14:A15-1-3	All thrown exceptions should be unique.	No
AUTOSAR++14:M15-1-3	An empty throw (throw;) shall only be used in the compound statement of a catch handler.	No
AUTOSAR++14:A15-1-4	If a function exits with an exception, then before a throw, the function shall place all objects/resources that the function constructed in valid states or it shall delete them.	No
AUTOSAR++14:A15-1-5	Exceptions shall not be thrown across execution boundaries.	No
AUTOSAR++14:A15-2-1	Constructors that are not noexcept shall not be invoked before program startup.	No
AUTOSAR++14:A15-2-2	If a constructor is not noexcept and the constructor cannot finish object initialization, then it shall deallocate the object's resources and it shall throw an exception.	No
AUTOSAR++14:M15-3-1	Exceptions shall be raised only after start-up and before termination.	No
AUTOSAR++14:A15-3-2	If a function throws an exception, it shall be handled when meaningful actions can be taken, otherwise it shall be propagated.	No
AUTOSAR++14:A15-3-3	Main function and a task main function shall catch at least: base class exceptions from all third-party libraries used, std::exception and all otherwise unhandled exceptions.	No
AUTOSAR++14:M15-3-3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.	No
AUTOSAR++14:A15-3-4	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines.	No
AUTOSAR++14:M15-3-4	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point.	No
AUTOSAR++14:A15-3-5	A class type exception shall be caught by reference or const reference.	Yes
AUTOSAR++14:M15-3-6	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.	Yes
AUTOSAR++14:M15-3-7	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.	Yes
AUTOSAR++14:A15-4-1	Dynamic exception-specification shall not be used.	No
AUTOSAR++14:A15-4-2	If a function is declared to be noexcept, noexcept(true) or noexcept(), then it shall not exit with an exception.	No
AUTOSAR++14:A15-4-3	The noexcept specification of a function shall either be identical across all translation units, or identical or more restrictive between a virtual member function and an override.	No

AUTOSAR++14:A15-4-4	A declaration of non-throwing function shall contain noexcept specification.	No
AUTOSAR++14:A15-4-5	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its overriders.	No
AUTOSAR++14:A15-5-1	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate.	No
AUTOSAR++14:A15-5-2	Program shall not be abruptly terminated. In particular, an implicit or explicit invocation of <code>std::abort()</code> , <code>std::quick_exit()</code> , <code>std::_Exit()</code> , <code>std::terminate()</code> shall not be done.	Yes
AUTOSAR++14:A15-5-3	The <code>std::terminate()</code> function shall not be called implicitly.	No
AUTOSAR++14:A16-0-1	The pre-processor shall only be used for unconditional and conditional file inclusion and include guards, and using the following directives: (1) <code>#ifndef</code> , (2) <code>#ifdef</code> , (3) <code>#if</code> , (4) <code>#if defined</code> , (5) <code>#elif</code> , (6) <code>#else</code> , (7) <code>#define</code> , (8) <code>#endif</code> , (9) <code>#include</code> .	Yes
AUTOSAR++14:M16-0-1	<code>#include</code> directives in a file shall only be preceded by other pre-processor directives or comments.	Yes
AUTOSAR++14:M16-0-2	Macros shall only be <code>#define'd</code> or <code>#undef'd</code> in the global namespace.	No
AUTOSAR++14:M16-0-5	Arguments to a function-like macro shall not contain tokens that look like pre-processing directives.	Yes
AUTOSAR++14:M16-0-6	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of <code>#</code> or <code>##</code> .	Yes
AUTOSAR++14:M16-0-7	Undefined macro identifiers shall not be used in <code>#if</code> or <code>#elif</code> pre-processor directives, except as operands to the defined operator.	Yes
AUTOSAR++14:M16-0-8	If the <code>#</code> token appears as the first token on a line, then it shall be immediately followed by a pre-processing token.	Yes
AUTOSAR++14:M16-1-1	The defined pre-processor operator shall only be used in one of the two standard forms.	No
AUTOSAR++14:M16-1-2	All <code>#else</code> , <code>#elif</code> and <code>#endif</code> pre-processor directives shall reside in the same file as the <code>#if</code> or <code>#ifdef</code> directive to which they are related.	Yes
AUTOSAR++14:A16-2-1	The <code>'</code> , <code>"</code> , <code>/*</code> , <code>//</code> , <code>\</code> characters shall not occur in a header file name or in <code>#include</code> directive.	Yes
AUTOSAR++14:A16-2-2	There shall be no unused include directives.	No
AUTOSAR++14:A16-2-3	An include directive shall be added explicitly for every symbol used in a file.	No
AUTOSAR++14:M16-2-3	Include guards shall be provided.	No
AUTOSAR++14:M16-3-1	There shall be at most one occurrence of the <code>#</code> or <code>##</code> operators in a single macro definition.	Yes
AUTOSAR++14:M16-3-2	The <code>#</code> and <code>##</code> operators should not be used.	Yes
AUTOSAR++14:A16-6-1	<code>#error</code> directive shall not be used.	No
AUTOSAR++14:A16-7-1	The <code>#pragma</code> directive shall not be used.	No
AUTOSAR++14:A17-0-1	Reserved identifiers, macros and functions in the C++ standard library shall not be defined, redefined or undefined.	Yes
AUTOSAR++14:A17-0-2	All project's code including used libraries (including standard and user-defined libraries) and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines.	No
AUTOSAR++14:M17-0-2	The names of standard library macros and objects shall not be reused.	Yes
AUTOSAR++14:M17-0-3	The names of standard library functions shall not be overridden.	Yes
AUTOSAR++14:M17-0-5	The <code>setjmp</code> macro and the <code>longjmp</code> function shall not be used.	Yes
AUTOSAR++14:A17-1-1	Use of the C Standard Library shall be encapsulated and isolated.	No
AUTOSAR++14:A17-6-1	Non-standard entities shall not be added to standard namespaces.	No
AUTOSAR++14:A18-0-1	The C library facilities shall only be accessed through C++ library headers.	Yes
AUTOSAR++14:A18-0-2	The error state of a conversion from string to a numeric value shall be checked.	Yes
AUTOSAR++14:A18-0-3	The library (<code>locale.h</code>) and the <code>setlocale</code> function shall not be used.	No
AUTOSAR++14:M18-0-3	The library functions <code>abort</code> , <code>exit</code> , <code>getenv</code> and <code>system</code> from library shall not be used.	Yes
AUTOSAR++14:M18-0-4	The time handling functions of library shall not be used.	Yes
AUTOSAR++14:M18-0-5	The unbounded functions of library shall not be used.	Yes
AUTOSAR++14:A18-1-1	C-style arrays shall not be used.	No
AUTOSAR++14:A18-1-2	The <code>std::vector</code> specialization shall not be used.	No

AUTOSAR++14:A18-1-3	The <code>std::auto_ptr</code> type shall not be used.	No
AUTOSAR++14:A18-1-4	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type.	No
AUTOSAR++14:A18-1-6	All <code>std::hash</code> specializations for user-defined types shall have a <code>noexcept</code> function call operator.	No
AUTOSAR++14:M18-2-1	The macro <code>offsetof</code> shall not be used.	Yes
AUTOSAR++14:A18-5-1	Functions <code>malloc</code> , <code>calloc</code> , <code>realloc</code> and <code>free</code> shall not be used.	Yes
AUTOSAR++14:A18-5-2	Non-placement <code>new</code> or <code>delete</code> expressions shall not be used.	No
AUTOSAR++14:A18-5-3	The form of the <code>delete</code> expression shall match the form of the <code>new</code> expression used to allocate the memory.	Yes
AUTOSAR++14:A18-5-4	If a project has sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined.	No
AUTOSAR++14:A18-5-5	Memory management functions shall ensure the following: (a) deterministic behavior resulting with the existence of worst-case execution time, (b) avoiding memory fragmentation, (c) avoid running out of memory, (d) avoiding mismatched allocations or deallocations, (e) no dependence on non-deterministic calls to kernel.	Yes
AUTOSAR++14:A18-5-6	An analysis shall be performed to analyze the failure modes of dynamic memory management. In particular, the following failure modes shall be analyzed: (a) non-deterministic behavior resulting with nonexistence of worst-case execution time, (b) memory fragmentation, (c) running out of memory, (d) mismatched allocations and deallocations, (e) dependence on non-deterministic calls to kernel.	No
AUTOSAR++14:A18-5-7	If non-realtime implementation of dynamic memory management functions is used in the project, then memory shall only be allocated and deallocated during non-realtime program phases.	Yes
AUTOSAR++14:A18-5-8	Objects that do not outlive a function shall have automatic storage duration.	No
AUTOSAR++14:A18-5-9	Custom implementations of dynamic memory allocation and deallocation functions shall meet the semantic requirements specified in the corresponding "Required behaviour" clause from the C++ Standard.	No
AUTOSAR++14:A18-5-10	Placement <code>new</code> shall be used only with properly aligned pointers to sufficient storage capacity.	No
AUTOSAR++14:A18-5-11	"operator <code>new</code> " and "operator <code>delete</code> " shall be defined together.	No
AUTOSAR++14:M18-7-1	The signal handling facilities shall not be used.	Yes
AUTOSAR++14:A18-9-1	The <code>std::bind</code> shall not be used.	No
AUTOSAR++14:A18-9-2	Forwarding values to other functions shall be done via: (1) <code>std::move</code> if the value is an rvalue reference, (2) <code>std::forward</code> if the value is forwarding reference.	No
AUTOSAR++14:A18-9-3	The <code>std::move</code> shall not be used on objects declared <code>const</code> or <code>const&</code> .	No
AUTOSAR++14:A18-9-4	An argument to <code>std::forward</code> shall not be subsequently used.	No
AUTOSAR++14:M19-3-1	The error indicator <code>errno</code> shall not be used.	No
AUTOSAR++14:A20-8-1	An already-owned pointer value shall not be stored in an unrelated smart pointer.	No
AUTOSAR++14:A20-8-2	A <code>std::unique_ptr</code> shall be used to represent exclusive ownership.	No
AUTOSAR++14:A20-8-3	A <code>std::shared_ptr</code> shall be used to represent shared ownership.	No
AUTOSAR++14:A20-8-4	A <code>std::weak_ptr</code> shall be used over <code>std::shared_ptr</code> if ownership sharing is not required.	No
AUTOSAR++14:A20-8-5	<code>std::make_unique</code> shall be used to construct objects owned by <code>std::unique_ptr</code> .	No
AUTOSAR++14:A20-8-6	<code>std::make_shared</code> shall be used to construct objects owned by <code>std::shared_ptr</code> .	No
AUTOSAR++14:A20-8-7	A <code>std::weak_ptr</code> shall be used to represent temporary shared ownership.	No
AUTOSAR++14:A21-8-1	Arguments to character-handling functions shall be representable as an unsigned char.	Yes
AUTOSAR++14:A23-0-1	An iterator shall not be implicitly converted to <code>const_iterator</code> .	No
AUTOSAR++14:A23-0-2	Elements of a container shall only be accessed via valid references, iterators, and pointers.	No
AUTOSAR++14:A25-1-1	Non-static data members or captured values of predicate function objects that are state related to this object's identity shall not be copied.	No
AUTOSAR++14:A25-4-1	Ordering predicates used with associative containers and STL sorting and related algorithms shall adhere to a strict weak ordering relation.	No
AUTOSAR++14:A26-5-1	Pseudorandom numbers shall not be generated using <code>std::rand()</code> .	No
AUTOSAR++14:A26-5-2	Random number engines shall not be default-initialized.	No

AUTOSAR++14:A27-0-1	Inputs from independent components shall be validated.	Yes
AUTOSAR++14:M27-0-1	The stream input/output library shall not be used.	Yes
AUTOSAR++14:A27-0-2	A C-style string shall guarantee sufficient space for data and the null terminator.	Yes
AUTOSAR++14:A27-0-3	Alternate input and output operations on a file stream shall not be used without an intervening flush or positioning call.	Yes
AUTOSAR++14:A27-0-4	C-style strings shall not be used.	No

GammaTech is a leading global provider of application testing (AST) solutions used by the world's most security conscious organizations to detect, measure, analyze and resolve vulnerabilities for software they develop or use. The company is also a trusted cybersecurity and artificial intelligence research partner for the nation's civil, defense, and intelligence agencies.

CodeSonar and CodeSentry are registered trademarks of GammaTech, Inc.
© GammaTech, Inc. All rights reserved.

