

C++

Checks

Check ID	Check Name	Supported	Automation	Category	Severity
A0-1-1	A project shall not contain instances of non-volatile variables being given values that are not subsequently used	Yes	Automated	Required	
A0-1-2	The value returned by a function shall be used	Yes	Automated	Required	
A0-1-3	Every function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used	Yes	Automated	Required	
A0-1-4	There shall be no unused named parameters in non-virtual functions	Yes	Automated	Required	
A0-1-5	There shall be no unused named parameters in the set of parameters for a virtual function and all the	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	functions that override it				
A0-1-6	There should be no unused type declarations	Yes	Automated	Advisory	
A0-4-1	Floating-point implementation shall comply with IEEE 754 standard	No	Non-automated	Required	
A0-4-2	Type long double shall not be used	Yes	Automated	Required	
A0-4-3	The implementations in the chosen compiler shall strictly comply with the C++14 Language Standard	No	Automated	Required	
A0-4-4	Range, domain and pole errors shall be checked when using math functions	No	Non-automated	Required	
A1-1-1	All code shall conform to ISO/IEC 14882:2014 - Programming Language C++ and shall not use deprecated features	No	Automated	Required	
A1-1-2	A warning level of the compilation	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	process shall be set in compliance with project policies				
A1-1-3	An optimization option that disregards strict standard compliance shall not be turned on in the chosen compiler	No	Non-automated	Required	
A1-2-1	When using a compiler toolchain, in safety-related software, the tool confidence level (TCL) shall be determined	No	Non-automated	Required	
A1-4-1	Code metrics and their valid boundaries shall be defined and code shall comply with defined boundaries of code metrics	No	Non-automated	Required	
A1-4-3	All code should compile free of compiler warnings	Yes	Automated	Advisory	
A2-3-1	Only those characters specified in	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the C++ Language Standard basic source character set shall be used in the source code				
A2-5-1	2-3-1 Trigraphs shall not be used	Yes	Automated	Required	
A2-5-2	Digraphs shall not be used	Yes	Automated	Required	
A2-7-1	The character \ shall not occur as a last character of a C++ comment	Yes	Automated	Required	
A2-7-2	Sections of code shall not be "commented out"	Yes	Non-automated	Required	
A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation	Yes	Automated	Required	
A2-7-5	Comments shall not document any actions or sources (e.g. tables, figures, paragraphs, etc.) that are	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	outside of the file				
A2-8-1	A header file name should reflect the logical entity for which it provides declarations	Yes	Non-automated	Required	
A2-8-2	An implementation file name should reflect the logical entity for which it provides definitions	Yes	Non-automated	Advisory	
A2-10-1	Shadowed Identifiers	Yes	Automated	Required	
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	Automated	Required	
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	Automated	Advisory	
A2-10-6	A class or	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope				
A2-11-1	Volatile keyword shall not be used	Yes	Automated	Required	
A2-13-1	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used	Yes	Automated	Required	
A2-13-2	Concatenating String Literals of Different Encodings	Yes	Automated	Required	
A2-13-3	Type wchar_t shall not be used	Yes	Automated	Required	
A2-13-4	String literals shall not be assigned to non-constant pointers	Yes	Automated	Required	
A2-13-5	Hexadecimal constants should be upper case	Yes	Automated	Advisory	
A2-13-6	Universal character names shall be used only inside character or string literals	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A3-1-1	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule	Yes	Automated	Required	
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"	Yes	Automated	Required	
A3-1-3	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes	Automated	Advisory	
A3-1-4	When an array with external linkage is declared, its size shall be stated explicitly	Yes	Automated	Required	
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	Yes	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	a member function template (3) it is a member function of a class template				
A3-1-6	Trivial accessor and mutator functions should be inlined.	Yes	Automated	Advisory	
A3-3-1	Objects or functions with external linkage (including members of named namespaces) shall be declared in a header file	Yes	Automated	Required	
A3-3-2	Static and thread-local objects shall be constant-initialized	Yes	Automated	Required	
A3-8-1	An object shall not be accessed outside of its lifetime	No	Automated	Required	
A3-9-1	Fixed Width Integers	Yes	Automated	Required	
A4-5-1	Expressions with type enum or enum class shall not be used as operands to built-in and	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	overloaded operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=				
A4-7-1	An integer expression shall not lead to data loss.	Yes	Automated	Required	
A4-10-1	Only nullptr literal shall be used as the null-pointer-constant	Yes	Automated	Required	
A5-0-1	The value of an expression shall be the same under any order of evaluation that the standard permits	Yes	Automated	Required	
A5-0-2	Condition of if statement shall be bool	Yes	Automated	Required	
A5-0-3	No more than 2 levels of pointer indirection	Yes	Automated	Required	
A5-0-4	Pointer arithmetic shall not be used	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	with pointers to non-final classes				
A5-1-1	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead	Yes	Automated	Required	
A5-1-2	Variables shall not be implicitly captured in a lambda expression	Yes	Automated	Required	
A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression	Yes	Automated	Required	
A5-1-4	A lambda expression object shall not outlive any of its reference-captured objects	Yes	Automated	Required	
A5-1-6	Specify Lambda Return Type	Yes	Automated	Advisory	
A5-1-7	A lambda shall not be an operand to decltype or typeid	Yes	Automated	Required	
A5-1-8	Lambda expressions should not be	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
A5-1-9	defined inside another lambda expression Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression	Yes	Automated	Advisory	
A5-2-1	dynamic_cast should not be used	Yes	Automated	Advisory	
A5-2-2	Traditional C-style casts shall not be used	Yes	Automated	Required	
A5-2-3	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes	Automated	Required	
A5-2-4	reinterpret_cast shall not be used	Yes	Automated	Required	
A5-2-5A	An array or container shall not be accessed beyond its range (Part A)	Yes	Automated	Required	
A5-2-5B	An array or container shall not be accessed	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	beyond its range Part B				
A5-2-6	Operands of Logical Boolean Operators	Yes	Automated	Required	
A5-3-1	Evaluation of the operand to the typeid operator shall not contain side effects.	Yes	Non-automated	Required	
A5-3-2	Before dereferencing a pointer, compare it with NULL	Yes	Partially Automated	Required	
A5-3-3	Deleting Pointers to Incomplete Class Types	Yes	Automated	Required	
A5-5-1	A pointer to member shall not access non-existent class members	Yes	Automated	Required	
A5-6-1A	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-6-1B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-10-1	A pointer to member virtual	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	function shall only be tested for equality with null-pointer-constant				
A5-16-1	The ternary conditional operator shall not be used as a sub-expression	Yes	Automated	Required	
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	Yes	Automated	Required	
A6-2-2	Explicit Calls to Constructors of Temporary Objects	Yes	Automated	Required	
A6-4-1	A switch statement shall have at least two case-clauses, distinct from the default label	Yes	Automated	Required	
A6-5-1	A for-loop that loops through all elements of the container and does not use its loop-counter shall	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A6-5-2	not be used A for loop shall contain a single loop-counter which shall not have floating-point type	Yes	Automated	Required	
A6-5-3	Do statements should not be used	Yes	Automated	Advisory	
A6-5-4	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes	Automated	Advisory	
A6-6-1	The goto statement shall not be used.	Yes	Automated	Required	
A7-1-1	Constexpr or const specifiers shall be used for immutable data declaration	Yes	Automated	Required	
A7-1-2	The constexpr specifier shall be used for values that can be determined at compile time	Yes	Automated	Required	
A7-1-3	CV-qualifiers shall be placed on the right hand side of	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the type that is a typedef or a using name				
A7-1-4	The register keyword shall not be used	Yes	Automated	Required	
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	Yes	Automated	Required	
A7-1-6	The typedef specifier shall not be used	Yes	Automated	Required	
A7-1-7	Each expression statement and identifier declaration	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be placed on a separate line				
A7-1-8	A non-type specifier shall be placed before a type specifier in a declaration.	Yes	Automated	Required	
A7-1-9	A class, structure, or enumeration shall not be declared in the definition of its type	Yes	Automated	Required	
A7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes	Automated	Required	
A7-2-2	Enumeration underlying base type shall be explicitly defined	Yes	Automated	Required	
A7-2-3	Enumerations shall be declared as scoped enum classes	Yes	Automated	Required	
A7-2-4	In an enumeration, either (1) none, (2) the first or (3) all enumerators	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A7-2-5	shall be initialized Enumerations should be used to represent sets of related named constants	No	Non-automated	Advisory	
A7-3-1	Overloaded Function Not Visible From Where it is Called	Yes	Automated	Required	
A7-4-1	The asm declaration shall not be used.	Yes	Automated	Required	
A7-5-1	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	Yes	Automated	Required	
A7-5-2	Functions shall not call themselves, either directly or indirectly.	Yes	Automated	Required	
A7-6-1	Functions declared with the [[noreturn]] attribute shall not return	Yes	Automated	Required	
A8-2-1	When declaring function templates, the trailing return type syntax	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be used if the return type depends on the type of parameters.				
A8-4-1	Functions shall not be defined using the ellipsis notation.	Yes	Automated	Required	
A8-4-2	Always return a value in non-void functions	Yes	Automated	Required	
A8-4-3	Common ways of passing parameters should be used.	Yes	Automated	Required	
A8-4-4	Multiple output values from a function should be returned as a struct or tuple.	Yes	Automated	Advisory	
A8-4-5	"consume" parameters declared as X && shall always be moved from.	Yes	Automated	Required	
A8-4-6	"forward" parameters declared as T && shall always be forwarded.	Yes	Automated	Required	
A8-4-7	"in" parameters for "cheap to copy" types shall be passed by	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	value.				
A8-4-8	Output parameters shall not be used.	Yes	Automated	Required	
A8-4-9	"in-out" parameters declared as T & shall be modified.	Yes	Automated	Required	
A8-4-10	A parameter shall be passed by reference if it can't be NULL	Yes	Automated	Required	
A8-4-11	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics	Yes	Automated	Required	
A8-4-12	Invalid Use of std::unique_ptr	Yes	Automated	Required	
A8-4-13	Invalid Use of std::shared_ptr	Yes	Automated	Required	
A8-4-14	Interfaces shall be precisely and strongly typed	No	Non-automated	Required	
A8-5-0	Uninitialized Memory Read	Yes	Automated	Required	
A8-5-1	Incorrect Order of Initialization	Yes	Automated	Required	
A8-5-2	Initializing Variables Without Using Braced-Initialization	Yes	Automated	Required	
A8-5-3	Auto Variable	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A8-5-4	Class Constructor with Parameter Type std::initializer_list	Yes	Automated	Advisory	
A9-3-1	Member functions shall not return non-const raw pointers or references to private or protected data owned by the class	Yes	Automated	Required	
A9-5-1	Unions Shall not be Used	Yes	Automated	Required	
A9-6-1	Data types used for interfacing	Yes	Partially Automated	Required	
A9-6-2	Bit-fields shall be used only when interfacing to hardware or conforming to communication protocols	No	Non-automated	Required	
A10-0-1	Public Inheritance not Used in a "is-a" Relationship	Yes	Non-automated	Required	
A10-0-2	Membership or non-public inheritance shall be used to implement "has-a" relationship	No	Non-automated	Required	
A10-1-1	Multiple Base Classes	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A10-2-1	Non-virtual public or protected member functions shall not be redefined in derived classes	Yes	Automated	Required	
A10-3-1	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final	Yes	Automated	Required	
A10-3-2	Use Override	Yes	Automated	Required	
A10-3-3	Virtual functions shall not be introduced in a final class	Yes	Automated	Required	
A10-3-5	User-defined assignment operator shall not be virtual	Yes	Automated	Required	
A10-4-1	Hierarchies should be based on interface classes	Yes	Non-automated	Advisory	
A11-0-1	A non-POD type should be defined as class	Yes	Automated	Advisory	
A11-0-2	A type defined as struct shall: (1) provide only public	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
A11-3-1	Friend declarations shall not be used.	Yes	Automated	Required	
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.	Yes	Automated	Required	
A12-0-2	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A12-1-1	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	Yes	Automated	Required	
A12-1-2	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	Yes	Automated	Required	
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.	Yes	Automated	Required	
A12-1-4	All constructors that are callable with a single argument of fundamental type shall be declared	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A12-1-5	explicit. Common class initialization for non-constant members shall be done by a delegating constructor.	Yes	Partially Automated	Required	
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	Automated	Required	
A12-4-1	Destructor of a base class shall be public virtual, public override or protected non-virtual	Yes	Automated	Required	
A12-4-2	If a public destructor of a class is non-virtual, then the class should be declared final.	Yes	Automated	Advisory	
A12-6-1	All class data members that are initialized by the constructor shall be initialized using member	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A12-7-1	initializers. 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.	Yes	Automated	Required	
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	Automated	Required	
A12-8-2	User-defined copy and move assignment operators should use user-defined no-throw swap function.	Yes	Automated	Advisory	
A12-8-3	Moved-from object shall not be read-accessed.	Yes	Partially Automated	Advisory	
A12-8-4	Move constructor shall not initialize its class members	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	and base classes using copy semantics.				
A12-8-5	A copy assignment and a move assignment operators shall handle self-assignment.	Yes	Automated	Required	
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	Automated	Required	
A12-8-7	Assignment operators should be declared with the ref-qualifier &.	Yes	Automated	Advisory	
A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters	Yes	Automated	Required	
A13-1-3	User defined literals	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	operators shall only perform conversion of passed parameters				
A13-2-1	An assignment operator shall return a reference to "this"	Yes	Automated	Required	
A13-2-2	A binary arithmetic operator and a bitwise operator shall return a "prvalue"	Yes	Automated	Required	
A13-2-3	A relational operator shall return a boolean value	Yes	Automated	Required	
A13-3-1	A function that contains "forwarding reference" as its argument shall not be overloaded	Yes	Automated	Required	
A13-5-1	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes	Automated	Required	
A13-5-2	All user-defined conversion operators shall be defined explicit	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A13-5-3	User-defined conversion operators should not be used	Yes	Automated	Advisory	
A13-5-4	If two opposite operators are defined, one shall be defined in terms of the other	Yes	Automated	Required	
A13-5-5	Comparison operators shall be non-member functions with identical parameter types and noexcept	Yes	Automated	Required	
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	Yes	Automated	Required	
A14-1-1	A template should check if a specific template argument is suitable for this template	Yes	Non-automated	Advisory	
A14-5-1	A template constructor	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not participate in overload resolution for a single argument of the enclosing class type				
A14-5-2	Class members that are not dependent on template class parameters should be defined in a separate base class	Yes	Partially Automated	Advisory	
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	Yes	Automated	Advisory	
A14-7-1	A type used as a template argument shall provide all members that are used by the template	Yes	Automated	Required	
A14-7-2	Template specialization shall be declared in the same file as the primary template	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A14-8-2	Explicit specializations of function templates shall not be used	Yes	Automated	Required	
A15-0-1	A function shall not exit with an exception if it is able to complete its task	No	Non-automated	Required	
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	Partially Automated	Required	
A15-0-3	Exception safety guarantee of a called function shall be considered	No	Non-automated	Required	
A15-0-4	Unchecked exceptions shall be used to represent errors from which the caller cannot reasonably be expected to recover.	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A15-0-5	Checked exceptions shall be used to represent errors from which the caller can reasonably be expected to recover	No	Non-automated	Required	
A15-0-6	An analysis shall be performed to analyze the failure modes of exception handling	No	Non-automated	Required	
A15-0-7	Exception handling mechanism shall guarantee a deterministic worst-case time execution time	No	Partially Automated	Required	
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	Non-automated	Required	
A15-1-1	Only instances of types derived from <code>std::exception</code>	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should be thrown				
A15-1-2	An exception object shall not be a pointer	Yes	Automated	Required	
A15-1-3	All thrown exceptions should be unique	Yes	Automated	Advisory	
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	Yes	Partially Automated	Required	
A15-1-5	Exceptions thrown across execution boundaries	Yes	Non-automated	Required	
A15-2-1	Constructors that are not noexcept shall not be invoked before program startup	Yes	Automated	Required	
A15-2-2	If a constructor is not noexcept and the constructor cannot finish object initialization, then it shall	Yes	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	deallocate the object's resources and it shall throw an exception				
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	Non-automated	Required	
A15-3-3	Unhandled Exceptions on Main Function	Yes	Partially Automated	Required	
A15-3-4	Catch-all (ellipsis and <code>std::exception</code>) 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	Yes	Non-automated	Required	
A15-3-5	A class type exception shall	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	be caught by reference or const reference				
A15-4-1	Dynamic exception-specification shall not be used	Yes	Automated	Required	
A15-4-2	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes	Automated	Required	
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	Yes	Automated	Required	
A15-4-4	A declaration of non-throwing function shall contain noexcept specification	Yes	Automated	Required	
A15-4-5	Checked	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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.				
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	Yes	Automated	Required	
A15-5-2	Program shall not be abruptly terminated	Yes	Automated	Required	
A15-5-3	The <code>std::terminate()</code> function shall not be called implicitly	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A16-0-1	Incorrect Use of Pre-processor	Yes	Automated	Required	
A16-2-1	Header File Name	Yes	Automated	Required	
A16-2-2	There shall be no unused include directives (slow)	Yes	Automated	Required	
A16-2-3	An include directive shall be added explicitly for every symbol used in a file	Yes	Non-automated	Required	
A16-6-1	#error directive shall not be used	Yes	Automated	Required	
A16-7-1	The #pragma directive shall not be used	Yes	Automated	Required	
A17-0-1	Reserved Builtin Macros	Yes	Automated	Required	
A17-0-2	All project's code including used libraries and any third-party user code shall conform to the AUTOSAR C++14 Coding Guidelines	No	Non-automated	Required	
A17-1-1	Use of the C Standard Library shall be encapsulated and isolated	No	Non-automated	Required	
A17-6-1	Non-standard entities shall not be added	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A18-0-1	to standard namespaces The C library facilities shall only be accessed through C++ library headers	Yes	Automated	Required	
A18-0-2	The error state of a conversion from string to a numeric value shall be checked	Yes	Automated	Required	
A18-0-3	Library <locale> (locale.h)	Yes	Automated	Required	
A18-1-1	C-style Array	Yes	Automated	Required	
A18-1-2	The std::vector<bool> specialization shall not be used	Yes	Automated	Required	
A18-1-3	The std::auto_ptr type shall not be used	Yes	Automated	Required	
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	Yes	Automated	Required	
A18-1-6	All std::hash specializations for user-defined types	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall have a noexcept function call operator				
A18-5-1	Functions malloc, calloc, realloc and free shall not be used	Yes	Automated	Required	
A18-5-2	Non-placement new or delete expressions shall not be used	Yes	Partially Automated	Required	
A18-5-3	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes	Automated	Required	
A18-5-4	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined	Yes	Automated	Required	
A18-5-5	Memory management functions shall ensure the following	No	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A18-5-6	An analysis shall be performed to analyze the failure modes of dynamic memory management	No	Non-automated	Required	
A18-5-7	Dynamic Memory Usage on Realtime Phase	Yes	Non-automated	Required	
A18-5-8	Objects that do not outlive a function shall have automatic storage duration	Yes	Partially Automated	Required	
A18-5-9	New Method Throwing an Exception	Yes	Automated	Required	
A18-5-10	Placement new shall be used only with properly aligned pointers to sufficient storage capacity	No	Automated	Required	
A18-5-11	operator "new" and operator "delete" shall be defined together	Yes	Automated	Required	
A18-9-1	The std::bind shall not be used	Yes	Automated	Required	
A18-9-2	Forwarding values to other functions shall be done via:	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	(1) <code>std::move</code> if the value is an rvalue reference, (2) <code>std::forward</code> if the value is forwarding reference				
A18-9-3	The <code>std::move</code> shall not be used on objects declared <code>const</code> or <code>const&</code>	Yes	Automated	Required	
A18-9-4	An argument to <code>std::forward</code> shall not be subsequently used	Yes	Automated	Required	
A20-8-1	An already-owned pointer value shall not be stored in an unrelated smart pointer	Yes	Automated	Required	
A20-8-2	A <code>std::unique_ptr</code> shall be used to represent exclusive ownership	Yes	Automated	Required	
A20-8-3	A <code>std::shared_ptr</code> shall be used to represent shared ownership	Yes	Automated	Required	
A20-8-4	A <code>std::weak_ptr</code> shall be used over <code>std::shared_ptr</code>	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A20-8-5	if ownership sharing is not required std::make_unique shall be used to construct objects owned by std::unique_ptr	Yes	Automated	Required	
A20-8-6	std::make_shared shall be used to construct objects owned by std::shared_ptr	Yes	Automated	Required	
A20-8-7	Cyclic Structure of std::shared_ptr	Yes	Non-automated	Required	
A21-8-1	Arguments to character-handling functions shall be representable as an unsigned char	Yes	Automated	Required	
A23-0-1	An iterator shall not be implicitly converted to const_iterator	Yes	Automated	Required	
A23-0-2	Elements of a container shall only be accessed via valid references, iterators, and pointers	No	Automated	Required	
A25-1-1	Predicate	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A25-4-1	Function Objects Copied Incorrectly Ordering predicates used with associative containers and STL sorting and related algorithms shall adhere to a strict weak ordering relation	No	Non-automated	Required	
A26-5-1	Pseudorandom numbers shall not be generated using <code>std::rand()</code>	Yes	Automated	Required	
A26-5-2	Random number engines shall not be default-initialized	Yes	Automated	Required	
A27-0-1	Inputs from independent components shall be validated	Yes	Non-automated	Required	
A27-0-2	A C-style string shall guarantee sufficient space for data and the null terminator	No	Automated	Advisory	
A27-0-3	Alternate input and output operations on	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	a file stream shall not be used without an intervening flush or positioning call				
A27-0-4	C-style strings shall not be used	Yes	Automated	Required	
AC_00	No Control Code Characters	Yes			
AC_01	No Direct or Indirect Recursion	Yes			
AC_HIS_02	Number of Paths (PATH)	Yes			
AC_HIS_04	Cyclomatic Complexity (v(G))	Yes			
AC_HIS_05	Calling Functions (CALLING)	Yes			
AC_HIS_06	Called Functions (CALLS)	Yes			
AC_HIS_07	Function Parameters (PARAM)	Yes			
AC_HIS_08	Number of Statements (STMT)	Yes			
AC_HIS_09	Number of call levels (LEVEL)	Yes			
AC_HIS_10	Number of return points (RETURN)	Yes			
AC_HIS_11	Language scope (VOCF)	Yes			
AC_HIS_12	Recursion (AP_CG_CYCL	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	E)				
AC_HIS_13	Statements Changed (SCHG)	Yes			
AC_HIS_14	Statements Deleted (SDEL)	Yes			
AC_HIS_15	New Statements (SNEW)	Yes			
AC_HIS_16	Stability Index (S)	Yes			
ARR30-C	Do not form or use out-of-bounds pointers or array subscripts	No			High
ARR32-C	Ensure size arguments for variable length arrays are in a valid range	No			High
ARR38-C	Guarantee that library functions do not form invalid pointers	No			High
CON32-C	Prevent data races when accessing bit-fields from multiple threads	No			Medium
CON34-C	Declare objects shared between threads with appropriate storage durations	No			Medium
CON35-C	Avoid deadlock by locking in a	No			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	predefined order				
CON43-C	Do not allow data races in multithreaded code	No			Medium
CON50-CPP	Do not destroy a mutex while it is locked	Yes			Medium
CON51-CPP	Ensure actively held locks are released on exceptional conditions	Yes			Low
CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes			Medium
CON53-CPP	Avoid deadlock by locking in a predefined order	No			Low
CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes			Medium
CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes			Low
CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	thread				
CPP_A000	Assembler instructions shall only be introduced using the asm declaration.	Yes			
CPP_A001	Assembly language shall be encapsulated and isolated.	Yes			
CPP_A002	Assignment Operator Return This	Yes			
CPP_A003	Assignment Operator Self Assignment	Yes			
CPP_A004	Parameter of assignment operator is a constant reference	Yes			
CPP_A005	Move and copy assignment operators shall either move or respectively copy base classes and data members of a class, without any side effects	Yes			
CPP_A006	The asm declaration shall not be used.	Yes			
CPP_A008	Assembly Language Code Usage not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_A009	Documented User-defined copy and move assignment operators should use user-defined no-throw swap function.	Yes			
CPP_A010	Move constructor shall not initialize its class members and base classes using copy semantics.	Yes			
CPP_A011	A copy assignment and a move assignment operators shall handle self-assignment.	Yes			
CPP_A012	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.	Yes			
CPP_A013	Assignment operators should be declared with	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	the ref-qualifier &.				
CPP_A014	The semantic equivalence between a binary operator and its assignment operator form shall be preserved	Yes			
CPP_A015	An assignment operator shall return a reference to "this"	Yes			
CPP_A016	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->	Yes			
CPP_A017	A template should check if a specific template argument is suitable for this template	Yes			
CPP_AO000	Assignment operators shall not be used in sub-expressions	Yes			
CPP_B000	Bool, Unsigned, or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_B001	Signed Bit-fields (Fuzzy parser) Bit-fields shall only be declared with an appropriate type	Yes			
CPP_B002	Enum Bit-fields	Yes			
CPP_B003	The underlying bit representation s of floating-point values shall not be used	Yes			
CPP_B004	Named signed bit-field length	Yes			
CPP_B005	Signed single-bit named bit-fields	Yes			
CPP_B006	Bit-field Length	Yes			
CPP_C000	Commented Out Code	Yes			
CPP_C001	Line-Splicing in // Comments	Yes			
CPP_C002	No Nested Comments	Yes			
CPP_C003	C99 / Comments	Yes			
CPP_C004	Parameter of copy constructor is a constant reference	Yes			
CPP_C005	Members in function-try-blocks in constructors or destructors	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_C006	Explicitly call all immediate and virtual base classes	Yes			
CPP_C007	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter	Yes			
CPP_C008	Side Effects in Copy Constructors	Yes			
CPP_C009	Explicit Constructors	Yes			
CPP_C010	Incomplete constructor initialization list	Yes			
CPP_C011	An object's dynamic type shall not be used from the body of its constructor or destructor	Yes			
CPP_C012	Virtual Function Call In Constructor	Yes			
CPP_C013	The statement forming the body of a switch, while, do ... while or for statement shall be a compound	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	statement				
CPP_C014	Dangling Else	Yes			
CPP_C015	A for loop shall contain a single loop-counter which shall not have floating-point type	Yes			
CPP_C016	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			
CPP_C017	The body of an iteration-statement or a selection-statement shall be a compound-statement	Yes			
CPP_C018	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes			
CPP_C019	A loop-	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	control-variable other than the loop-counter shall not be modified within condition or expression				
CPP_C020	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=	Yes			
CPP_C021	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes			
CPP_C022	The loop-counter shall not be modified within condition or statement	Yes			
CPP_C023	The goto statement shall jump to a label declared later in the same function body	Yes			
CPP_C024	No Continue Statements	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_C025	Goto Statements	Yes			
CPP_C026	There should be no more than one break or goto statement used to terminate any iteration statement	Yes			
CPP_C027	Non-private Member Data	Yes			
CPP_C028	A null statement shall only occur on a line by itself	Yes			
CPP_C029	Single exit point at end	Yes			
CPP_C030	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes			
CPP_C031	Switch Has Default	Yes			
CPP_C032	Every switch statement shall have at least two switch-clauses	Yes			
CPP_C033	An unconditional throw or break statement shall terminate every non-	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	empty switch-clause				
CPP_C034	Unreachable Code	Yes			
CPP_C035	No Backslash at End of Comment	Yes			
CPP_C036	If statements shall not have assignments in the conditions	Yes			
CPP_C037	Documentation 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	Yes			
CPP_C038		Yes			
CPP_C039	A switch statement shall have at least two case-clauses, distinct from the default label	Yes			
CPP_C040	A loop-control-variable other than the loop-counter which	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	is modified in statement shall have type bool				
CPP_C041	Do statements should not be used	Yes			
CPP_C042	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes			
CPP_C043	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.	Yes			
CPP_C044	Continue Statement Used in a not Well-formed For Loop	Yes			
CPP_C045	Write constructor member initializers in the canonical	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	order				
CPP_C046	All switch statements shall be well-formed	Yes			
CPP_C047	All if and else if constructs shall be terminated with an else clause	Yes			
CPP_C048	Transferring Control to a Try or Catch Block Using Goto or Switch Statement	Yes	Non-automated	Required	
CPP_C049	Class Constructor with Parameter Type <code>std::initializer_list</code>	Yes			
CPP_C050	A for-loop that loops through all elements of the container and does not use its loop-counter shall not be used	Yes			
CPP_C051	Constructors that are not noexcept shall not be invoked before program startup	Yes			
CPP_C052	If a constructor is not noexcept and the constructor	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	cannot finish object initialization, then it shall deallocate the object's resources and it shall throw an exception				
CPP_C053	Explicit Calls to Constructors of Temporary Objects	Yes			
CPP_C054	When a "deep copy" constructor is not implemented, comments in the class header shall describe this fact	Yes			
CPP_C055	Constructors that can be used with one argument should be declared explicit.	Yes			
CPP_C056	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any side effects	Yes			
CPP_CF000	The condition of a switch	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	statement shall not have bool type				
CPP_CF001	All cases in a switch statement shall have a break or it shall be well commented	Yes			
CPP_CF002	Switch statements should have a default case	Yes			
CPP_CF003	Switch label unstructured	Yes			
CPP_CF004	The std::terminate() function shall not be called implicitly	Yes			
CPP_CF005	Program shall not be abruptly terminated	Yes			
CPP_CF006	Simple Control Flow	Yes			
CPP_CF007	Loops with Fixed Limits	Yes			
CPP_CF008	A for loop shall be well-formed	Yes			
CPP_CF009	The validity of values received from external sources shall be checked	Yes			
CPP_CF010	Legacy for statements should be simple	Yes			
CPP_CF011	Goto Into or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_CM000	Between Blocks Comments shall precede code being commented and shall align with code they represent	Yes			
CPP_CM001	Each function shall end with a comment	Yes			
CPP_CM002	Timing delays shall be preceded by comments explaining the delay	Yes			
CPP_CM003	Class headers shall include a short description for every member function declaration and a comment for every data member declared	Yes			
CPP_CM004	All usage of assembly language should be documented	Yes			
CPP_CT_BUGP RONE_ASSERT _SIDE_EFFECT	Assert Side Effect	Yes			High
CPP_CT_BUGP RONE_BRANC H_CLONE	Branch Clone	Yes			High
CPP_CT_BUGP	Copy	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
RONE_COPY_CONSTRUCTO R_INIT	Constructor Init				
CPP_CT_BUGP RONE_INFINIT E_LOOP	Infinte Loop	Yes			High
CPP_CT_BUGP RONE_MACRO _REPEATED_SIDE_EFFECTS	Macro Side Effects	Yes			High
CPP_CT_BUGP RONE_NOT_NULL_TERMINATED_RESULT	Missing Null Terminator	Yes			High
CPP_CT_BUGP RONE_REDUNDANT_BRANCH_CONDITION	Redundant Condition	Yes			High
CPP_CT_CPPC OREGUIDELINES ES_AVOID_NON_CONST_GLOBALS	Non-const Globals	Yes			
CPP_CT_MISC _CONST_CORRECTNESS	Missing Const for Locals and Parameters	Yes			
CPP_CT_MOD ERNIZE_USE_DEFAULT_MEMBER_INIT	Default Member Init	Yes			
CPP_CT_MOD ERNIZE_USE_EQUALS_DEFAULT_FUNCTION	Default Member Function	Yes			
CPP_CT_MOD ERNIZE_USE_EQUALS_DELETE	Delete Member Function	Yes			
CPP_CT_MOD ERNIZE_USE_NULLPTR	Null Pointer Keyword	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_CT_READ ABILITY_DELETE_NULL_POINTER	Delete Null Pointer	Yes			High
CPP_CT_READ ABILITY_MAKE_MEMBER_FUNCTIONS_CONSTANT	Non-const Member Functions	Yes			
CPP_CT_READ ABILITY_NON_CONSTANT_PARAMETER	Non-const Parameters	Yes			
CPP_CT_READ ABILITY_REDUNDANT_CASTING	Redundant Cast	Yes			High
CPP_D000	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes			
CPP_D001	Do not delete a polymorphic object without a virtual destructor	Yes			
CPP_D002	Single Declarations	Yes			
CPP_D003	Array Size Missing	Yes			
CPP_D004	Unsigned Integer Literal Without Suffix	Yes			
CPP_D005	A base class shall only be declared virtual if it is used in a diamond	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D006	hierarchy Class Derived From Virtual Bases	Yes			
CPP_D007	A compatible declaration shall be visible when an object or function with external linkage is defined	Yes			
CPP_D008	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter	Yes			
CPP_D009	Declaration and Using-Declaration Clash	Yes			
CPP_D010	= construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes			
CPP_D011	Use the static keyword for internal linkage	Yes			
CPP_D012	An external object or function shall be declared in	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	one and only one file				
CPP_D013	An identifier with external linkage shall have exactly one definition	Yes			
CPP_D015	Externals shall have the same type in the declaration and definition	Yes			
CPP_D017	Non-static Inline Functions	Yes			
CPP_D018	Literal suffixes shall be upper case	Yes			
CPP_D019	The comma operator, && operator and the operator shall not be overloaded	Yes			
CPP_D020	Lowercase L Suffix	Yes			
CPP_D021	Narrow and wide string literals shall not be concatenated	Yes			
CPP_D022	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit	Yes			
CPP_D023	Single-Function Global Objects	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D026	The register keyword shall not be used	Yes			
CPP_D027	The unary & operator shall not be overloaded	Yes			
CPP_D028	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes			
CPP_D029	Destructor Set Data Ptr to 0	Yes			
CPP_D030	Exceptions in Destructors	Yes			
CPP_D031	Non-Virtual Destructors in Base Classes	Yes			
CPP_D032	Virtual Function Call In Destructor	Yes			
CPP_D033	A function shall not be declared implicitly	Yes			
CPP_D034	Datamembers should be declared private	Yes			
CPP_D035	Destructor of a base class shall be public virtual, public override or protected non-virtual	Yes			
CPP_D036	Volatile	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	keyword shall not be used				
CPP_D037	Functions shall not be declared at block scope	Yes			
CPP_D038	When an array with external linkage is declared, its size shall be stated explicitly	Yes			
CPP_D039	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	Yes			
CPP_D040	All declarations of an object or function shall have compatible types	Yes			
CPP_D041	The One Definition Rule	Yes			
CPP_D042	If a function has internal linkage then all redeclarations shall include the static	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	storage class specifier				
CPP_D043	Static and thread-local objects shall be constant-initialized	Yes			
CPP_D044	Declarations at Lowest Scope	Yes			
CPP_D045	A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes			
CPP_D046	Constexpr or const specifiers shall be used for immutable data declaration	Yes			
CPP_D047	The constexpr specifier shall be used for values that can be determined at compile time	Yes			
CPP_D048	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
CPP_D049	A class, structure, or enumeration shall not be declared in the definition of its type	Yes			
CPP_D050	Enumerations shall be declared as scoped enum classes	Yes			
CPP_D051	A non-type specifier shall be placed before a type specifier in a declaration.	Yes			
CPP_D052	Use the same identifier in definition and declaration of functions.	Yes			
CPP_D053	Multiple Base	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D054	Classes Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final	Yes			
CPP_D056	User-defined assignment operator shall not be virtual	Yes			
CPP_D057	Hierarchies should be based on interface classes	Yes			
CPP_D058	A non-POD type should be defined as class	Yes			
CPP_D059	Friend declarations shall not be used.	Yes			
CPP_D060	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	member functions shall be declared as well.				
CPP_D061	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	Yes			
CPP_D062	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	Yes			
CPP_D063	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.	Yes			
CPP_D064	All constructors that are callable with a single	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	argument of fundamental type shall be declared explicit.				
CPP_D065	Common class initialization for non-constant members shall be done by a delegating constructor.	Yes			
CPP_D066	If a public destructor of a class is non-virtual, then the class should be declared final.	Yes			
CPP_D067	All class data members that are initialized by the constructor shall be initialized using member initializers.	Yes			
CPP_D068	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.	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D069	Member Data in Non-POD Class not Private	Yes			
CPP_D070	Template specialization shall be declared in the same file as the primary template	Yes			
CPP_D071	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	Yes			
CPP_D072	Non-standard entities shall not be added to standard namespaces	Yes			
CPP_D073	There shall be one variable declaration per line	Yes			
CPP_D074	An external	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	variable shall not be set to a value where it is being declared				
CPP_D075	Incorrect Order of Initialization	Yes			
CPP_D076	If a class requires a virtual destructor but has nothing to undo from a constructor, an empty implementation should be provided.	Yes			
CPP_D077	There shall be no tentative definitions in a header file	Yes			
CPP_D078	There should be no external declarations in a source file	Yes			
CPP_DD000	The defines, typedefs, structures, externals, globals, statics, external prototypes, and local prototypes shall be grouped by category.	Yes			
CPP_DD001	Use of global functions and variables shall	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	be limited				
CPP_DD002	Globals should not be used in macros	Yes			
CPP_DD003	There shall be a function prototype for all functions	Yes			
CPP_DD004	Prototypes for static functions shall include the static storage class	Yes			
CPP_DD005	Any defined constants that are used as argument or return variables shall be placed in an include file	Yes			
CPP_DD006	Initializer lists shall be written in the order in which they are declared	Yes			
CPP_DD007	The private keyword should be used in class definitions	Yes			
CPP_DD008	Nesting template class definitions should be avoided.	Yes			
CPP_DD009	Assignment operators should check for self-assignment	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_DD010	The use of friend classes should be avoided	Yes			
CPP_DD011	If the subscript operator (operator[]) is overloaded, both the const and non-const versions should be defined.	Yes			
CPP_DD012	Layering techniques, where applicable, should be used instead of private inheritance.	Yes			
CPP_DD013	Public Inheritance not Used in a "is-a" Relationship	Yes			
CPP_DD014	Use the same parameter names and type qualifiers for all declarations and definitions	Yes			
CPP_DD015	Overload allocation and deallocation functions as a pair in the same scope	Yes			
CPP_DD016	Do not write syntactically ambiguous declarations	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_DD017	Avoid cycles during initialization of static objects	Yes			
CPP_DD018	Obey the one-definition rule	Yes			
CPP_DD019	Arrays shall not be partially initialized	Yes			
CPP_DD020	An element of an object shall not be initialized more than once	Yes			
CPP_DD021	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes			
CPP_DD022	Make sure that objects are initialized before they are used	Yes			
CPP_DD023	Use the same form in corresponding uses of new and delete	Yes			
CPP_DD024	Postpone variable definitions as long as possible	Yes			
CPP_DD025	Avoid hiding inherited names	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_DD026	Never redefine an inherited non-virtual function	Yes			
CPP_DD027	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes			
CPP_DD028	A pointer should point to a const-qualified type whenever possible	Yes			
CPP_DD029	Variable-length arrays shall not be used	Yes			
CPP_DD030	Storage Class Specifiers Not At Beginning	Yes			
CPP_DD032	Names from a dependent base shall be prefixed with this->, qualified with a using declaration, or qualified with the base name	Yes			
CPP_DD033	A variable declared in an inner scope shall not hide a variable declared in an outer scope	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E000	Exception by Value	Yes			
CPP_E001	There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes			
CPP_E002	Catch-All Statement Before Last	Yes			
CPP_E003	Catch Const References	Yes			
CPP_E004	Destructors Not Throw Exceptions	Yes			
CPP_E005	Empty Throw	Yes			
CPP_E006	Order of Catch Blocks with Derived Classes	Yes			
CPP_E007	An exception object shall not be a pointer	Yes			
CPP_E008	Exceptions shall be raised only after start-up and before termination of the program	Yes			
CPP_E009	Exceptions thrown shall be the type indicated by the function	Yes			
CPP_E010	Inconsistent Exception-Specification	Yes			
CPP_E011	No "errno"	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	allowed				
CPP_E012	NULL Throw	Yes			
CPP_E013	Throw exceptions by value, not by pointer	Yes			
CPP_E014	The assignment-expression of a throw statement shall not itself cause an exception to be thrown	Yes			
CPP_E015	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator	Yes			
CPP_E016	Character Operators	Yes			
CPP_E017	Code Slicing Should Not Occur	Yes			
CPP_E018	Expressions with type enum or enum class	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be used as operands to built-in and overloaded operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=				
CPP_E019	Avoid Trigraphs	Yes			
CPP_E020	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes			
CPP_E021	Octal and Hexadecimal Sequences	Yes			
CPP_E022	Nonstandard Escape Sequences	Yes			
CPP_E023	Expression uses operand of side-effect more than once	Yes			
CPP_E024	Signed	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	operands to modulus or division operator				
CPP_E025	Floating Equality Test	Yes			
CPP_E026	Minimization of run-time failures shall be ensured by the use of static analysis tools	Yes			
CPP_E027	Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used	Yes			
CPP_E028	Hexadecimal constants should be upper case	Yes			
CPP_E029	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes			
CPP_E030	Concatenating String Literals of Different Encodings	Yes	Automated	Required	
CPP_E031	Traditional C-style casts shall not be used	Yes			
CPP_E032	Infeasible Paths	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E033	Do not rely on the value of a moved-from object	Yes			
CPP_E034	Limited dependence should be placed on C++ operator precedence rules in expressions	Yes			
CPP_E035	Parameter list (possibly empty) shall be included in every lambda expression	Yes			
CPP_E036	Specify Lambda Return Type	Yes			
CPP_E037	Lambda expressions should not be defined inside another lambda expression	Yes			
CPP_E038	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression	Yes			
CPP_E039	A lambda shall not be an operand to decltype or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E040	typeid dynamic_cast should not be used	Yes			
CPP_E041	reinterpret_cast shall not be used	Yes			
CPP_E042	Operands of Logical Boolean Operators	Yes			
CPP_E043	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes			
CPP_E044	Each operand of the ! operator, the logical && or the logical operators shall have type bool	Yes			
CPP_E045	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes			
CPP_E046	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	hand operand.				
CPP_E047	The ternary conditional operator shall not be used as a sub-expression	Yes			
CPP_E048	Each expression statement and identifier declaration shall be placed on a separate line	Yes			
CPP_E049	The comma operator shall not be used.	Yes			
CPP_E050A	Evaluation of the operand to the typeid operator shall not contain side effects	Yes			
CPP_E050B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes			
CPP_E051	Unary Minus Operator Applied to an Expression with an Unsigned Type	Yes			
CPP_E052	The right-hand operand of a logical && or operator should not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	contain persistent side effects				
CPP_E055	Exception Object	Yes			
CPP_E056	A lambda expression object shall not outlive any of its reference-captured objects	Yes			
CPP_E057	Only instances of types derived from <code>std::exception</code> should be thrown	Yes			
CPP_E059	All thrown exceptions should be unique	Yes			
CPP_E060	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	Yes			
CPP_E061	Dynamic exception-specification shall not be used	Yes			
CPP_E062	A class type	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	exception shall be caught by reference or const reference				
CPP_E063	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	Yes			
CPP_E064	Unhandled Exceptions on Main Function	Yes			
CPP_E065	Condition of if statement shall be bool	Yes			
CPP_E066	Const Should be placed on the left-hand side of the comparison	Yes			
CPP_E067	Floats shall not be tested for direct equality	Yes			
CPP_E068	Provide a valid	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	ordering predicate				
CPP_E069	Assignment in SubExpressions	Yes			
CPP_E070	Boolean operators	Yes			
CPP_E071	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations	Yes			
CPP_E072	Int to Float Conversion	Yes			
CPP_E073	An implicit integral conversion shall not change the signedness of the underlying type	Yes			
CPP_E074	Operands shall not be of an inappropriate essential type	Yes			
CPP_E075	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes			
CPP_E076	The value of an expression	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be assigned to an object with a narrower essential type or of a different essential type category				
CPP_E077	The value of a composite expression shall not be assigned to an object with wider essential type	Yes			
CPP_E078	The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes			
CPP_E079	Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes			
CPP_E080	A conversion shall not be performed between a pointer to object type and a pointer to a different	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E081	object type A conversion should not be performed between a pointer to object and an integer type	Yes			
CPP_E082	Initializer lists shall not contain persistent side effects	Yes			
CPP_E083	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes			
CPP_E084	The macro NULL shall be the only permitted form of integer null pointer constant	Yes			
CPP_E085	The result of an assignment operator should not be used	Yes			
CPP_E086	A loop counter shall not have essentially floating type	Yes			
CPP_E087	Minimize casting	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E088	Parentheses should be used to make the meaning of an expression appropriately explicit	Yes			
CPP_E089	An unsigned arithmetic operation with constant operands should not wrap	Yes			
CPP_E092	The operands of a logical && or shall be primary-expressions	Yes			
CPP_E093	The controlling expression of a for statement shall not contain any objects of floating type	Yes			
CPP_EH000	Program shall not be abruptly terminated	Yes			
CPP_EH001	The <code>std::terminate()</code> function shall not be called implicitly	Yes			
CPP_EH002	Library objects shall not generate error messages directly	Yes			
CPP_EH003	Destructors should not throw	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_EH004	exceptions Exceptions should be caught only by reference	Yes			
CPP_EH005	A declaration of non-throwing function shall contain noexcept specification	Yes			
CPP_EH006	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes			
CPP_EH007	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	Yes			
CPP_EH008	Exceptions thrown across execution boundaries	Yes			
CPP_EH009	New Method Throwing an Exception	Yes			
CPP_EH010	Use Assertion Statements	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_EH011	Catch exceptions by lvalue reference	Yes			
CPP_F000	All prototype parameters must have an identifier.	Yes			
CPP_F001	Templates Not Instantiated	Yes			
CPP_F002	Const Member Function Returning Non-Const Pointer or Reference	Yes			
CPP_F003	Unused Functions	Yes			
CPP_F005	Declare functions at file scope	Yes			
CPP_F006	A Function identifier shall either be used to call the function or it shall be preceded by &	Yes			
CPP_F007	Functions must not return objects by value.	Yes			
CPP_F008	Functions shall not be defined using the ellipsis notation	Yes			
CPP_F009	Use Named Parameters and Prototype Form	Yes			
CPP_F010	Functions shall not be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	declared implicitly				
CPP_F011	Inline functions defined in the class body	Yes			
CPP_F012	The identifier main shall not be used for a function other than the global function main	Yes			
CPP_F013	Method Returning Non-const Handle to Class Data	Yes			
CPP_F014	Member Functions That Should Be Static Or Const	Yes			
CPP_F015	Missing parameter name in function declarations	Yes			
CPP_F016	variable numbers of arguments shall not be used.	Yes			
CPP_F017	Overloaded function templates shall not be explicitly specialized	Yes			
CPP_F018	Parameters in an overriding virtual function shall either use the same default arguments as	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	the function they override, or else shall not specify any default arguments.				
CPP_F019	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			
CPP_F020	Inconsistent Parameter Names	Yes			
CPP_F021	The features of <stdarg.h> shall not be used	Yes			
CPP_F022	Objects should not be passed by reference	Yes			
CPP_F023	A function parameter should not be modified	Yes			
CPP_F024	The value returned by a function shall be used	Yes			
CPP_F025	All functions with void return type shall have external side effect(s)	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_F026	Every function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used	Yes			
CPP_F027	There shall be no unused named parameters in non-virtual functions	Yes			
CPP_F028	There shall be no unused named parameters in the set of parameters for a virtual function and all the functions that override it	Yes			
CPP_F029	operator "new" and operator "delete" shall be defined together	Yes			
CPP_F030	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	versions shall be defined				
CPP_F031	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes			
CPP_F032	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	Yes			
CPP_F033	Always return a value in non-void functions	Yes			
CPP_F034	Trivial accessor and mutator functions should be inlined.	Yes			
CPP_F035	Non-virtual public or protected member functions shall not be redefined in derived classes	Yes			
CPP_F036	Use Override	Yes			
CPP_F037	Time Handling Functions of <ctime>	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_F038_A	Check Parameters and Return Values - Ignored Return Values	Yes			
CPP_F039	A function that contains "forwarding reference" as its argument shall not be overloaded	Yes			
CPP_F040	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes			
CPP_F041	Member functions shall not return non-const raw pointers or references to private or protected data owned by the class	Yes			
CPP_F042	If two opposite operators are defined, one shall be defined in terms of the other	Yes			
CPP_F043	Comparison operators shall be non-member	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	functions with identical parameter types and noexcept				
CPP_F044	Overloaded Function Not Visible From Where it is Called	Yes			
CPP_F045	Virtual functions shall not be introduced in a final class	Yes			
CPP_F046	Predicate Function Objects Copied Incorrectly	Yes			
CPP_F047	A template constructor shall not participate in overload resolution for a single argument of the enclosing class type	Yes			
CPP_F048	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_F049	Explicit specializations of function templates shall not be used	Yes			
CPP_F050	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	Yes			
CPP_F051	A function should be inlined only if it has one or two lines of code	Yes			
CPP_F052	The function gets() should not be used	Yes			
CPP_F053	Every function shall have an explicitly declared return type.	Yes			
CPP_F054	Boolean functions shall explicitly return true or false	Yes			
CPP_F055	The default parameter list, when redeclaring or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	overriding methods, should be kept constant				
CPP_F056	Each function shall contain a prologue	Yes			
CPP_F057	Function prologue shall be in header or source	Yes			
CPP_F058	Function prologue shall contain certain specific information	Yes			
CPP_F059	Variable-length argument lists should not be used	Yes			
CPP_F060	A method that does not change the visible properties of a class shall be declared const	Yes			
CPP_F061	The type of the return and all method arguments (even type void) shall be specified when defining a method	Yes			
CPP_F062	When overloading standardized operators (e.g., <code>a += b</code> , <code>a-=b</code> etc.), the	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	resulting behavior should remain consistent with the expected outcome of the operator.				
CPP_F063	Member function arguments should not share the same name as class state variables	Yes			
CPP_F064	Member functions should always be declared const unless they modify state variables	Yes			
CPP_F065	Any parameter not modified by a method should be passed to the method as a const.	Yes			
CPP_F066	Tail-Call Optimization	Yes			
CPP_F067	Functions declared with the [[noreturn]] attribute shall not return	Yes			
CPP_F068	The function argument corresponding to a parameter declared to have an array	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	type shall have an appropriate number of elements				
CPP_F069	A signal handler must be a plain old function	Yes			
CPP_F070	Consider alternatives to virtual functions	Yes			
CPP_F071	Special member functions shall be provided appropriately	Yes			
CPP_F072	A function should be used in preference to a function-like macro where they are interchangeable	Yes			
CPP_F073	If a function returns error information, then that error information shall be tested	Yes			
CPP_F074	Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_H000	The #include directive shall be followed by either a <filename> or "filename" sequence	Yes			
CPP_H001	The backslash character should not occur in a header file name	Yes			
CPP_H002	The ', ", /* or // characters shall not occur in a header file name	Yes			
CPP_H003	Definitions in Header Files	Yes			
CPP_H004	There shall be no unnamed namespaces in header files.	Yes			
CPP_H005	Objects or functions with external linkage shall be declared in a header file	Yes			
CPP_H006	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule	Yes			
CPP_H007	Unnecessary #Includes	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_H008	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes			
CPP_H009	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx"	Yes			
CPP_H010	Header File Name	Yes			
CPP_H011	Absolute path names shall not be used for header files	Yes			
CPP_H012	All references to header files shall be listed one per line	Yes			
CPP_H013	Names of private header files should not be identical to names of library header files	Yes			
CPP_H014	All public header files shall be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	capable of being included by a C++ file as well as a C file				
CPP_H016	If prototypes, typedefs, macros, structure definitions, or enums are needed in multiple modules, they shall be placed in header files	Yes			
CPP_H017	C++ version of the header file should be used	Yes			
CPP_H018	When including C Standard Library header files, C++ Standard Library header files without a '.h' appended should be used	Yes			
CPP_H019	Forward referencing should be used, when appropriate, over direct inclusion when documenting dependencies within a header file.	Yes			
CPP_H020	The standard header file	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	<tgmath.h> shall not be used				
CPP_H021	The standard header file <setjmp.h> shall not be used	Yes			
CPP_I000	A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes			
CPP_I001	Different identifiers shall be typographically unambiguous	Yes			
CPP_I002	External identifiers shall be distinct	Yes			
CPP_I003	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes			
CPP_I004	Global Namespace Declarations	Yes			
CPP_I005	Identifier Reuse	Yes			
CPP_I006	Same Identifier and Macro Name	Yes			
CPP_I007	Identifiers declared in the same scope and name	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_I008	space shall be distinct Identifiers that define objects or functions with internal linkage should be unique	Yes			
CPP_I009	Indistinct Macro Identifiers	Yes			
CPP_I010	The identifier name of a non-member object or function with static storage duration should not be reused	Yes			
CPP_I012	Static name reuse	Yes			
CPP_I013	A tag name shall be a unique identifier	Yes			
CPP_I014	A typedef name shall be a unique identifier.	Yes			
CPP_I015	Identifier Reuse in Multiple C Name Spaces	Yes			
CPP_I016	Reserved Identifiers or Macros	Yes			
CPP_I017	Shadowed Identifiers	Yes			
CPP_I018	A class or enumeration name shall not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	be hidden by a variable, function or enumerator declaration in the same scope				
CPP_I019	The identifier name of a non-member object with static storage duration or static function shall not be reused within a namespace	Yes			
CPP_I020	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			
CPP_I021	Universal character names shall be used only inside character or string literals	Yes			
CPP_I022	Similar Entity Names within Multiple Inheritance	Yes			
CPP_I023	Uppercase 'O' shall not be used as an	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	identifier				
CPP_I024	Lowercase 'l' shall not be used as an identifier	Yes			
CPP_I025	The using namespace directive should be used only at the method or function scope.	Yes			
CPP_L000	Calls to COTS library functions that might throw an exception must be enclosed in a try block.	Yes			
CPP_L001	The C library shall not be used	Yes			
CPP_L002	The signal handling facilities of <csignal> shall not be used	Yes			
CPP_L003	The stream input/output library <cstdio> shall not be used	Yes			
CPP_L004	<cstdlib> Library Functions	Yes			
CPP_L005	Avoid atof, atoi, atol, and atoll from <cstdlib> or <stdlib.h>	Yes			
CPP_L006	Unbounded	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Functions of <cstring>				
CPP_L007	Avoid using the library <ctime>	Yes			
CPP_L008	No "errno" allowed	Yes			
CPP_L009	No offsetof allowed	Yes			
CPP_L010	The setjmp macro and the longjmp function shall not be used	Yes			
CPP_L011	Signal.h should not be used	Yes			
CPP_L012	Standard Library Function Names	Yes			
CPP_L013	Including <stdio.h>	Yes			
CPP_L014	Library stdlib.h - avoid: abort, exit, getenv and system	Yes			
CPP_L015	Guarantee that library functions do not overflow	Yes			
CPP_L016	The library <time.h> shall not be used	Yes			
CPP_L017	Inputs from independent components shall be validated	Yes			
CPP_L018	Ensure your random number	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	generator is properly seeded				
CPP_L019	Random number engines shall not be default-initialized	Yes			
CPP_L020	Do not unlock or destroy another POSIX thread's mutex	Yes			
CPP_L021	An iterator shall not be implicitly converted to const_iterator	Yes			
CPP_L022	An argument to std::forward shall not be subsequently used	Yes			
CPP_L023	The std::move shall not be used on objects declared const or const&	Yes			
CPP_L024	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	Yes			
CPP_L025	The std::bind shall not be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_L026	used Alternate input and output operations on a file stream shall not be used without an intervening flush or positioning call	Yes			
CPP_L027	All std::hash specializations for user-defined types shall have a noexcept function call operator	Yes			
CPP_L028	The std::auto_ptr type shall not be used	Yes			
CPP_L029	Library <locale> (locale.h)	Yes			
CPP_L030	Avoid deadlock with POSIX threads by locking in predefined order	Yes			
CPP_L031	Evaluation of the operand to the typeid operator shall not contain side effects.	Yes			
CPP_L032	Bounds-checking Interfaces	Yes			
CPP_L033	Reserved Builtin Macros	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_L034	Use of the iostream library is preferred over stdio.h	Yes			
CPP_L035	Atomic Operations with Inconsistent Order	Yes			
CPP_L036	Character Function Misuse	Yes			
CPP_L037	Incompatible Pointers	Yes			
CPP_L038	There shall be no attempt to write to a stream which has been opened as read-only	Yes			
CPP_L039	Use of Closed FILE Pointers (Partial)	Yes			
CPP_L040	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF	Yes			
CPP_L041	Out of Bounds with string.h	Yes			
CPP_L042	EOF Macro Misuse	Yes			
CPP_L043	Thread Data Misuse	Yes			
CPP_L044	C Standard Library I/O	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Functions				
CPP_L045	Standard Header signal.h	Yes			
CPP_L046	Result of std::remove, std::remove_if, std::unique, and empty shall be used	Yes			
CPP_L047	Predicates shall not have persistent side effects	Yes			
CPP_M000	Dynamic Memory Allocation	Yes			
CPP_M001	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes			
CPP_M002	Non-placement new or delete expressions shall not be used	Yes			
CPP_M003	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_M004	Moved-from object shall not be read-accessed.	Yes			
CPP_M005	Uninitialized Memory Read	Yes			
CPP_M006	Functions malloc, calloc, realloc and free shall not be used	Yes			
CPP_M007	When reading strings a maximum field width should be specified	Yes			
CPP_M008	Dynamically allocated memory shall be set to some value prior to its use as an rvalue or in a test	Yes			
CPP_M009	Memory that has been freed shall not be referenced	Yes			
CPP_M010	The new[] and delete[] operators shall be used for the allocation and deallocation of memory resources	Yes			
CPP_M011	The delete[] operator shall be used to deallocate arrays that have been	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_M012	allocated with the new[] operator The delete[] operator shall be called in the destructor for all member pointers in an object that are pointing to memory that was dynamically allocated by that object	Yes			
CPP_M013	Users shall provide a copy constructor, destructor and assignment operator for a class that uses dynamic memory allocation	Yes			
CPP_M014	The operator new should be called with the nothrow option.	Yes			
CPP_M015	When overloading the new[] operator, a corresponding delete[] operator should be provided.	Yes			
CPP_M016	Overloaded new operator	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	should not hide the global new operator				
CPP_M017	All local allocations made in a method, other than the destructor, should be deallocated prior to exiting the method.	Yes			
CPP_M018	Dynamic Memory Usage on Realtime Phase	Yes			
CPP_M019	No Dynamic Memory Allocation	Yes			
CPP_M020	Properly pair allocation and deallocation functions	Yes			
CPP_M021	Declare objects shared between POSIX threads with appropriate storage durations	Yes			
CPP_M022	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes			
CPP_M023	A block of	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	memory shall only be freed if it was allocated by means of a Standard Library function				
CPP_M024	An object shall not be accessed outside of its lifetime	Yes			
CPP_N000	Naming Convention: Classes	Yes			
CPP_N001	Naming Convention: Enumerator	Yes			
CPP_N002	Naming Convention: Enums	Yes			
CPP_N003	Naming Convention: Files	Yes			
CPP_N004	Naming Convention: Functions	Yes			
CPP_N005	Naming Convention: Macros	Yes			
CPP_N006	Naming Convention: Namespaces	Yes			
CPP_N007	Naming Convention: Parameters	Yes			
CPP_N008	Naming Convention: Structs	Yes			
CPP_N009	Naming Convention:	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_N010	Typedefs Naming Convention:	Yes			
CPP_N011	Unions Naming Convention:	Yes			
CPP_N012	Variables Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes			
CPP_N013	Header File Names Naming Convention:	Yes			
CPP_N014	Implementation File Names Naming Convention:	Yes			
CPP_N015	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes			
CPP_N016	User defined suffixes of the user defined literal operators shall start with underscore followed by	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_N017	one or more letters 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	Yes			
CPP_N018	All macros shall be fully capitalized	Yes			
CPP_N019	Function and variable names shall not be fully capitalized	Yes			
CPP_P000	No more than 2 levels of pointer indirection	Yes			
CPP_P001	Hide Implementation of Pointers Not Dereferenced	Yes			
CPP_P002	Pointer initialization must use 0, not NULL.	Yes			
CPP_P003	Pointer function parameters must be tested for equality to 0 before accessing the	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	data being pointed to				
CPP_P004	Pointers Must Be Initialized	Yes			
CPP_P005	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes			
CPP_P006	std::make_unique shall be used to construct objects owned by std::unique_ptr	Yes			
CPP_P007	A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required	Yes			
CPP_P008	Do Not Use #define	Yes			
CPP_P010	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes			
CPP_P011	Ifndef Wrappers or Pragma Once	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_P012	File Include Matching Header	Yes			
CPP_P013	Function-like macros shall not be defined	Yes			
CPP_P014_A	Restrict Pointer Usage - Multiple Dereferences	Yes			
CPP_P014_B	Restrict Pointer Usage - Other	Yes			
CPP_P015	Inactive Code	Yes			
CPP_P016	Size of Array Parameter	Yes			
CPP_P017	#include directives in a file shall only be preceded by other preprocessor directives or comments	Yes			
CPP_P018	Keyword Macros	Yes			
CPP_P019	Macros in Blocks	Yes			
CPP_P020	Invalid Macro Usage	Yes			
CPP_P021	Before dereferencing a pointer, compare it with NULL	Yes			
CPP_P022	The pre-processor shall only be used for file inclusion and include guards	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_P023	Reserved or Standard Library Identifiers as Macros	Yes			
CPP_P024	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			
CPP_P025	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator	No			
CPP_P026	Preprocessor #undef	Yes			
CPP_P027	Dereference of FILE Pointer	Yes			
CPP_P028	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_P029	Unused Macros	Yes			
CPP_P030	Invalid Use of std::shared_ptr	Yes			
CPP_P031	Invalid Use of std::unique_ptr	Yes			
CPP_P032	Cyclic Structure of std::shared_ptr	Yes			
CPP_P033	For pointer declarations, the asterisk shall be placed with the variable	Yes			
CPP_P034	Invalid Header Name	Yes			
CPP_P035	std::make_shar ed shall be used to construct objects owned by std::shared_ptr	Yes			
CPP_P036	A std::shared_ptr shall be used to represent shared ownership	Yes			
CPP_P037	A std::unique_ptr shall be used to represent exclusive ownership	Yes			
CPP_P038	An already- owned pointer value shall not be stored in an unrelated smart pointer	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_P039	String literals shall not be assigned to non-constant pointers	Yes			
CPP_P040	Only nullptr literal shall be used as the null-pointer-constant	Yes			
CPP_P041	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes			
CPP_P042	Pointer arithmetic shall not be used with pointers to non-final classes	Yes			
CPP_P043	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes			
CPP_P044	Deleting Pointers to Incomplete Class Types	Yes			
CPP_P045	Array indexing over pointer arithmetic	Yes			
CPP_P046	A pointer pointing to an element of an	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	array of objects shall not be passed to a smart pointer of single object type				
CPP_P047	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes			
CPP_P048	A pointer to member virtual function shall only be tested for equality with null-pointer-constant	Yes			
CPP_P049	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes			
CPP_P050	Literal zero (0) shall not be used as the null-pointer-constant.	Yes			
CPP_P051	Pointer to Integer Cast	Yes			
CPP_P052	A parameter	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be passed by reference if it can't be NULL				
CPP_P053	A pointer to member shall not access non-existent class members	Yes			
CPP_P054	References should be used instead of pointers when possible.	Yes			
CPP_P055	For pointer declarations, the placement of the * shall be consistent	Yes			
CPP_P056	Pointer functions shall return a valid pointer on success and a zero pointer on failure	Yes			
CPP_P057	A pointer to dynamic memory that is declared and allocated locally should be declared as an auto_ptr.	Yes			
CPP_P058	Store newed objects in smart pointers in standalone statements	Yes			
CPP_P059	A pointer to a virtual base class shall only	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	be cast to a pointer to a derived class by means of dynamic_cast				
CPP_P060	Prefer pass-by-reference-to-const to pass by value	Yes			
CPP_P061	Shared Pointer Capture	Yes			
CPP_P062	The +, -, += and -= operators should not be applied to an expression of pointer type	Yes			
CPP_P063	Declarations should contain no more than two levels of pointer nesting	Yes			
CPP_P064	Casts shall not be performed between a pointer to function and any other type	Yes			
CPP_P065	Pointer arithmetic shall not form an invalid pointer	Yes			
CPP_P066	The built-in relational operators >, >=, < and <= shall not be applied to objects of pointer type, except where	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	they point to the same array				
CPP_P067	Pointers shall not be implicitly compared to NULL	Yes			
CPP_PR001	Include guards shall be provided	Yes			
CPP_PR002	Constants defined by #define shall be explicitly declared with uppercase suffixes	Yes			
CPP_PR003	Macros shall not be used to change language syntax	Yes			
CPP_PR004	Limit Preprocessor Usage	Yes			
CPP_PR005	#include directives should only be preceded by preprocessor directives or comments	Yes			
CPP_PR006	There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes			
CPP_PR007	The defined preprocessor operator shall	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	only be used in one of the two standard forms				
CPP_PR021	The names of standard library macros and objects shall not be reused	Yes			
CPP_PR030	The #pragma directive shall not be used	Yes			
CPP_PR031	#error directive shall not be used	Yes			
CPP_PR032	The # and ## operators should not be used	Yes			
CPP_PR033	The macro offsetof shall not be used	Yes			
CPP_PR034	There shall be no unused include directives (slow)	Yes			
CPP_PR036	Invalid Preprocessor Directives	Yes			
CPP_PR037	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes			
CPP_PR038	Missing	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Parentheses in Macros				
CPP_PR039	Function-like Macro Containing Preprocessing Directives	Yes			
CPP_PR040	#include Directives Not Grouped Together	Yes			
CPP_PR041	Incorrect Use of Pre-processor	Yes			
CPP_PR042	The argument of an integer constant macro shall have an appropriate form	Yes			
CPP_PR043	An initializer using chained designators shall not contain initializers without designators	Yes			
CPP_PR044	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	Yes			
CPP_PR045	All identifiers used in the controlling expression of #if or #elif	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	preprocessing directives shall be #define'd before evaluation				
CPP_PR046	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes			
CPP_PR047	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes			
CPP_S000	Unions	Yes			
CPP_S001	Flexible Array Members	Yes			
CPP_S002	Incorrect Initializer Lists	Yes			
CPP_S003	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3)	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	not be a base of another struct or class, (4) not inherit from another struct or class				
CPP_S004	Unions Shall not be Used	Yes			
CPP_S005	Invalid Memory Compare	Yes			
CPP_S006	Unions	Yes			
CPP_S007	A union member shall not be read unless it has been previously set	Yes			
CPP_SA_DANGLING_POINTER	Dangling Pointer	Yes			High
CPP_SA_DEAD_STORES	Dead Stores	Yes			
CPP_SA_DIV_ZERO	Division by Zero	Yes			High
CPP_SA_LEAKS	Memory Leak	Yes			High
CPP_SA_NULL_PTR	Null Pointer Dereference	Yes			High
CPP_SA_STACK_ADDRESS_ESCAPE	Stack Address Escape	Yes			High
CPP_SA_UNDEFINED_CALL	Undefined Call	Yes			High
CPP_SA_UNINITIALIZED	Uninitialized Value	Yes			High
CPP_SA_VIRTUAL_CALLS	Virtual Call	Yes			High
CPP_ST001	Not more than one space should precede a ";"	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	with the exception of the null statement				
CPP_ST002	Equal signs should be aligned when they occur in a series of assignment operators	Yes			
CPP_ST003	Placement of braces for functions shall adhere to one of the following formats and shall be consistent	Yes			
CPP_ST004	Code between the beginning and ending braces of a function shall start with one level of indentation	Yes			
CPP_ST005	Enum lists should not contain a trailing comma	Yes			
CPP_ST006	No line of code should extend beyond column 80	Yes			
CPP_ST007	Declarations shall not be made within an individual block but shall be placed at	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	the function level or at the module level.				
CPP_ST008	Blank lines should be used to separate distinct algorithmic parts	Yes			
CPP_ST009	Parentheses should be used in lengthy logical and arithmetic expressions for clarity.	Yes			
CPP_ST010	Items should be logically grouped	Yes			
CPP_ST011	Inline functions should be used instead of macros	Yes			
CPP_ST012	Names that differ in case only or that look similar should not be used.	Yes			
CPP_ST013	Switch statements should be used instead of deeply nested else-ifs when testing a variable for multiple values	Yes			
CPP_ST014	No line of code should extend beyond 80 characters	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_ST015	Incrementing and decrementing control variables in loops	Yes			
CPP_ST016	Calls to free should have an if test around them if it is uncertain that the pointer has been properly allocated.	Yes			
CPP_ST017	White space shall not be used in the following places	Yes			
CPP_ST018	Continuation lines shall be indented at least one level from the line being continued	Yes			
CPP_ST019	Statements under case labels shall be indented one level	Yes			
CPP_ST020	For the if-else, while, do, and for control structure, the statement(s) shall be indented one level	Yes			
CPP_ST021	Placement of braces for constructs	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be consistent within a module				
CPP_ST022	Boolean expressions involving non-boolean values should always use an explicit test for equality or non-equality.	Yes			
CPP_ST023	At least one blank line shall be placed before comments	Yes			
CPP_ST024	Functions shall have at least one blank line between them	Yes			
CPP_ST025	Each area of declarations shall have at least one blank line before and after it	Yes			
CPP_ST026	Class naming conventions	Yes			
CPP_ST027	Naming conventions for class data members vs. member function internal data	Yes			
CPP_ST028	Data type naming conventions	Yes			
CPP_ST029	Immutable data naming	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_ST030	conventions Class design should include the following format	Yes			
CPP_ST031	Separate lines should be used for each member declaration	Yes			
CPP_ST032	Indentation shall be at least three spaces, and consistent across modules	Yes			
CPP_ST033	Short Functions	Yes			
CPP_T000	Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes			
CPP_T001	Arguments to character-handling functions shall be representable as an unsigned char	Yes			
CPP_T002	The <code>std::vector<bool></code> specialization shall not be used	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_T003	There should be no unused type declarations	Yes			
CPP_T004	Type long double shall not be used	Yes			
CPP_T005	Type wchar_t shall not be used	Yes			
CPP_T006	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			
CPP_T007	A cvalue expression shall not be implicitly converted to a different underlying type	Yes			
CPP_T008	An implicit integral conversion shall not change the signedness of the underlying type	Yes			
CPP_T009	There shall be no implicit floating-	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_T010	integral conversions An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes			
CPP_T011	There shall be no explicit floating-integral conversions of a cvalue expression	Yes			
CPP_T012	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes			
CPP_T013	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes			
CPP_T014	If the bitwise operators ~ and << are	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand				
CPP_T015	The plain char type shall only be used for the storage and use of character values	Yes			
CPP_T016	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes			
CPP_T017	The first operand of a conditional-operator shall have type bool	Yes			
CPP_T018	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes			
CPP_T019	C-style Array	Yes			
CPP_T020	Casts from a base class to a	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	derived class should not be performed on polymorphic types				
CPP_T021	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes			
CPP_T022	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes			
CPP_T023	Array to Pointer Decay	Yes			
CPP_T024	NULL shall not be used as an integer value	Yes			
CPP_T025	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name	Yes			
CPP_T026	The typedef specifier shall not be used	Yes			
CPP_T027	An expression with enum underlying type shall only have values corresponding	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	to the enumerators of the enumeration				
CPP_T028	Enumeration underlying base type shall be explicitly defined	Yes			
CPP_T029	In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized	Yes			
CPP_T030	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.	Yes			
CPP_T031	Common ways of passing parameters should be used.	Yes			
CPP_T032	Multiple output values from a function should be returned as a struct or tuple.	Yes			
CPP_T033	"consume" parameters declared as X	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_T034	&& shall always be moved from. "forward" parameters declared as T && shall always be forwarded.	Yes			
CPP_T035	"in" parameters for "cheap to copy" types shall be passed by value.	Yes			
CPP_T036	Output parameters shall not be used.	Yes			
CPP_T037	"in-out" parameters declared as T & shall be modified.	Yes			
CPP_T038	Fixed Width Integers	Yes			
CPP_T039	Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes			
CPP_T040	User defined literals operators shall only perform conversion of passed parameters	Yes			
CPP_T041	A binary	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	arithmetic operator and a bitwise operator shall return a "prvalue"				
CPP_T042	A relational operator shall return a boolean value	Yes			
CPP_T043	If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes			
CPP_T044	Undocumented Use of Floating-point Arithmetic	Yes			
CPP_T045	Undocumented Use of Scaled-integer or Fixed-point Arithmetic	Yes			
CPP_T046	Assigning Object to an Overlapping Object (Partial)	Yes			
CPP_T047	Data types used for interfacing	Yes			
CPP_T048	All user-defined conversion operators shall be defined explicit	Yes			
CPP_T049	User-defined conversion	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	operators should not be used				
CPP_T050	Types shall be explicitly specified	Yes			
CPP_T051	C-style strings shall not be used	Yes			
CPP_T052	String-to-Number Conversion Handling	Yes			
CPP_T053	A type used as a template argument shall provide all members that are used by the template	Yes			
CPP_T054A	An array or container shall not be accessed beyond its range (Part A)	Yes			
CPP_T054B	An array or container shall not be accessed beyond its range Part B	Yes			
CPP_T055	A value should not be retrieved from a structure or union with a different type than with which it was stored	Yes			
CPP_T056	Explicit type	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	casting shall be used when performing calculations with a mix of signed and unsigned values.				
CPP_T057	Actual arguments shall be explicitly type cast to the appropriate type	Yes			
CPP_T058	Simple integers shall be used to test and set booleans	Yes			
CPP_T059	Width-sensitive types should be typedef'd and placed in a header file	Yes			
CPP_T060	Converting a pointer to integer or integer to pointer	Yes			
CPP_T061	All Checks/ Language Specific/C and C++/Types/Use Const whenever possible	Yes			
CPP_T062	Small Integer Constant Macros	Yes			
CPP_T063	Implicit Casts	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	of Operations				
CPP_T064	Pointer to Variable-length Array	Yes			
CPP_T068	The operands of bitwise operators and shift operators shall be appropriate	Yes			
CPP_T069	Integral promotion and the usual arithmetic conversions shall not change the signedness or the type category of an operand	Yes			
CPP_T070	Assignment between numeric types shall be appropriate	Yes			
CPP_U000	Digraphs shall not be used	Yes			
CPP_U001	Discarded return values.	Yes			
CPP_U002	Inline Functions have more than X LOC	Yes			
CPP_U003	Unused Parameters in Non-virtual Functions	Yes			
CPP_U004	Unused Static Globals	Yes			
CPP_U005	Unused Tag Declarations	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_U006	Unused Type Declarations	Yes			
CPP_U007	Unused Labels	Yes			
CPP_U008	Unnecessary Friends	Yes			
CPP_U009	Special Member Functions	Yes			
CPP_U010	Unused Entities	Yes			
CPP_U012	Language extensions should not be used (Partial)	Yes			
CPP_V000	Magic Numbers	Yes			
CPP_V001	One Variable per Line	Yes			
CPP_V002	Reference Symbols Spacing, (& *)	Yes			
CPP_V003	Declare each variable in a separate declaration statement	Yes			
CPP_V004	A project shall not contain non-volatile POD variables having only one use.	Yes			
CPP_V005	Types or externals declared at the function level.	Yes			
CPP_V006	A variable which is not modified shall be const qualified	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_V007	Unused Local Variables	Yes			
CPP_V009	Using-directives shall not be used.	Yes			
CPP_V010	Variables should be commented	Yes			
CPP_V011	All variables shall have a defined value before they are used	Yes			
CPP_V012	Explicit Virtual	Yes			
CPP_V013	Virtual Function Redefinition	Yes			
CPP_V014	Pure Virtual Overriding Non-pure	Yes			
CPP_V015	There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it	Yes			
CPP_V016	Virtual Call in Constructor/Destructor	Yes			
CPP_V017	A project shall not contain instances of non-volatile variables being given values that are not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	subsequently used				
CPP_V018	Auto Variable	Yes			
CPP_V019	Initializing Variables Without Using Braced-Initialization	Yes			
CPP_V020	Class members that are not dependent on template class parameters should be defined in a separate base class	Yes			
CPP_V021	Variables should not be altered more than once in an expression	Yes			
CPP_V022	Variables shall not be implicitly captured in a lambda expression	Yes			
CPP_V023	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead	Yes			
CPP_V024	Variables of type char shall be explicitly qualified as signed or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	unsigned when used to store numbers				
CPP_V025	Every variable shall be declared with a specific type	Yes			
CPP_V026	Local variables shall be initialized when declared	Yes			
CPP_V027	Globals in header files shall be ifdef'd	Yes			
CPP_V028	Constants should be declared as const values as opposed to #define directives.	Yes			
CPP_V029	The const_cast operator should be used exclusively for altering the constness attribute of a variable.	Yes			
CPP_V030	The dynamic_cast operator should be used exclusively for casting within an inheritance hierarchy.	Yes			
CPP_V031	The static_cast operator should be used for routine cast operations not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	provided by const_cast and dynamic_cast.				
CPP_V032	Use of the reinterpret_cast operator should be avoided	Yes			
CPP_V033	Typedef'd variables in a class shall be placed in an include file	Yes			
CPP_V034	STL containers (vector, list, map, etc.) should be used instead of C- style arrays whenever possible.	Yes			
CPP_V035	Objects that do not outlive a function shall have automatic storage duration	Yes			
CPP_V036	Static data member initialization should be placed in the class .cpp file	Yes			
CPP_V037	Initializer lists should be used to initialize member variables over direct assignment.	Yes			
CPP_V038	The concept of information	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	hiding should be implemented.				
CPP_V039	Within an object, most instance variables should be accessed directly. Methods should be used to set variables whose values are determined by an algorithm.	Yes			
CPP_V040	Mutexes with One Condition Variable	Yes			
CPP_V042	An object shall not be accessed outside of its lifetime	No			
CPP_VF000	Every class that contains virtual functions shall provide a virtual destructor	Yes			
CPP_VF001	Access levels should not be mixed (public, protected, private) when overriding virtual functions.	Yes			
CPP_WARN_ABSOLUTE_VAL	Absolute Value Proper Usage	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
UE					
CPP_WARN_A	Abstract	Yes			
BSTRACT_FIN	Classes Should				
AL_CLASS	Not Be Final or Sealed				
CPP_WARN_A	No Useless Init	Yes			
BSTRACT_VBA	for Abstract				
SE_INIT	Virtual Base				
CPP_WARN_A	Do Not Take	Yes			
DDRESS_OF_P	the Address of				
ACKED_MEMB	Packed				
ER	Members				
CPP_WARN_A	Do Not Take	Yes			High
DDRESS_OF_T	the Address of				
EMPORARY	Temporary Objects				
CPP_WARN_AI	IBM AIX	Yes			
X_COMPAT	Compatibility with Byte Alignment				
CPP_WARN_A	Match Byte	Yes			
LIGN_MISMAT	Alignment of				
CH	Arguments				
CPP_WARN_A	Do Not Use	Yes			
LLOCA	Certain Allocation Functions				
CPP_WARN_A	Correct Usage	Yes			
LLOCA_WITH_	of Second				
ALIGN_ALIGN	Argument of				
OF	Certain Allocation Functions				
CPP_WARN_AL	Always Inline	Yes			
WAYS_INLINE_	Coroutine				
COROUTINE	Functions				
CPP_WARN_A	Ambiguous	Yes			
MBIGUOUS_D	Delete				
ELETE					
CPP_WARN_A	Ambiguous	Yes			
MBIGUOUS_EL	Ellipsis				
LIPSIS					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_A MBIGUOUS_M ACRO	Ambiguous Macro	Yes			
CPP_WARN_A MBIGUOUS_M EMBER_TEMPL ATE	Ambiguous Member Template	Yes			
CPP_WARN_A MBIGUOUS_R EVERSED_OPE RATOR	Ambiguous Reversed Operator	Yes			
CPP_WARN_A NALYZER_INC OMPATIBLE_P LUGIN	Analyzer Incompatible Plugin	Yes			
CPP_WARN_A NON_ENUM_E NUM_CONVER SION	Anon-Enum Enum Conversion	Yes			
CPP_WARN_A NONYMOUS_P ACK_PARENS	Anonymous Pack Parentheses	Yes			
CPP_WARN_A RC_BRIDGE_C ASTS_DISALL OWED_IN_NO NARC	ARC (Automatic Reference Counting) Bridge Casts Disallowed in Non-ARC	Yes			
CPP_WARN_A RC_MAYBE_RE PEATED_USE_ OF_WEAK	ARC Maybe Repeated Use of Weak	Yes			
CPP_WARN_A RC_RETAIN_C YCLES	ARC Retain Cycles	Yes			
CPP_WARN_A RC_UNSAFE_R ETAINED_ASSI GN	ARC Unsafe Retained Assign	Yes			
CPP_WARN_A RGUMENT_OU	Argument Outside Range	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
TSIDE_RANGE					
CPP_WARN_A	Argument	Yes			
RGUMENT_UN	Undefined				
DEFINED_BEH	Behaviour				
AVIOUR					
CPP_WARN_A	Array Bounds	Yes			High
RRAY_BOUND					
S					
CPP_WARN_A	Array Bounds	Yes			
RRAY_BOUND	Pointer				
S_POINTER_A	Arithmetic				
RITHMETIC					
CPP_WARN_A	Array	Yes			
RRAY_PARAME	Parameter				
TER					
CPP_WARN_A	Assembly	Yes			
SM_OPERAND	Operand				
_WIDTHS	Widths				
CPP_WARN_A	Assign Enum	Yes			
SSIGN_ENUM					
CPP_WARN_A	Discarded Side	Yes			
SSUME	Effects to				
	__assume				
	Function				
CPP_WARN_AT	Atomic Access	Yes			
OMIC_ACCESS					
CPP_WARN_AT	Atomic	Yes			
OMIC_ALIGNM	Alignment				
ENT					
CPP_WARN_AT	Atomic	Yes			
OMIC_IMPLICIT	Implicitly				
T_SEQ_CST	Sequentially-				
	Consistent				
CPP_WARN_AT	Atomic	Yes			
OMIC_MEMOR	Memory				
Y_ORDERING	Ordering				
CPP_WARN_A	Auto Disable	Yes			
UTO_DISABLE	Virtual Pointer				
_VPTR_SANITI	Sanitizer				
ZER					
CPP_WARN_A	Auto Storage	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
UTO_STORAG E_CLASS	Class				
CPP_WARN_AV AILABILITY	Availability Attribute	Yes			High
CPP_WARN_A VR_RTLIB_LIN KING_QUIRKS	AVR RTLIB (Real-Time Library) Linking Quirks	Yes			
CPP_WARN_B ACKEND_PLU GIN	Backend Plugin	Yes			
CPP_WARN_B ACKSLASH_NE WLINE_ESCAP E	Backslash Newline Escape	Yes			High
CPP_WARN_B AD_FUNCTION _CAST	Do Not Cast from Function Call of One Type to Another	Yes			
CPP_WARN_BI ND_TO_TEMP ORARY_COPY	Bind to Temporary Copy	Yes			
CPP_WARN_BI NDING_IN_CO NDITION	Binding in Condition	Yes			
CPP_WARN_BI T_INT_EXTEN SION	Bit Int Extension	Yes			
CPP_WARN_BI TFIELD_CONS TANT_CONVER SION	Bitfield Constant Conversion	Yes			High
CPP_WARN_BI TFIELD_ENUM _CONVERSION	Bitfield Enum Conversion	Yes			
CPP_WARN_BI TFIELD_WIDTH	Do Not Exceed Bit-Field Width	Yes			High
CPP_WARN_BI TWISE_CONDI TIONAL_PARE	Bitwise Conditional Parentheses	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NTHESES					
CPP_WARN_BITWISE_INSTEAD_OF_LOGICAL	Bitwise Instead of Logical	Yes			
CPP_WARN_PARENTHESES	Bitwise Operator Parentheses	Yes			
CPP_WARN_BOOL_CONVERSION	Bool Conversion	Yes			High
CPP_WARN_BOOL_OPERATION	Bool Operation	Yes			
CPP_WARN_BRACED_SCALAR_INIT	Braced Scalar Init	Yes			
CPP_WARN_BRANCH_PROTECTION	Branch Protection	Yes			
CPP_WARN_BUILTIN_ASSUME_ALIGNED	Builtin Assume Aligned Alignment	Yes			
CPP_WARN_BUILTIN_MACRO_REDEFINED	Builtin Macro Redefined	Yes			High
CPP_WARN_BUILTIN_MEMCPY_CHECK_SIZE	Builtin Malloc Check Size	Yes			High
CPP_WARN_BUILTIN_REQUIRES_HEADER	Builtin Requires Header	Yes			
CPP_WARN_C2X_EXTENSIONS	C2X Extensions	Yes			
CPP_WARN_C11_EXTENSIONS	C11 Extensions	Yes			
CPP_WARN_C99_COMPATIBILITY	C99 Compatibility	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_C99_DESIGNATOR	C99 Designator	Yes			
CPP_WARN_C99_EXTENSIONS	C99 Extensions	Yes			
CPP_WARN_ALL_TO_PURE_VIRTUAL_FROM_CONSTRUCTOR_OR_DESTRUCTOR	Call to Pure Virtual from Constructor or Destructor	Yes			High
CPP_WARN_CALLED_ONCE_PARAMETER	Called once Parameter	Yes			
CPP_WARN_CAST_ALIGN	Cast Align	Yes			
CPP_WARN_CAST_CALLING_CONVENTION	Cast Calling Convention	Yes			
CPP_WARN_CAST_FUNCTION_TYPE	Cast Function Type	Yes			
CPP_WARN_CAST_QUALIFIERS	Cast Qualifiers	Yes			
CPP_WARN_CAST_QUALIFIERS_UNRELATED	Cast Qualifiers Unrelated	Yes			High
CPP_WARN_CHAR_SUBSCRIPTS	Char Subscripts	Yes			
CPP_WARN_CLANG_CL_PRECOMPILED_HEADERS	Clang-CL Precompiled Headers	Yes			
CPP_WARN_CLASS_CONVERSION	Class Conversion	Yes			High
CPP_WARN_CLASS_VARIADIC_ARGS	Class Variadic Arguments	Yes			
CPP_WARN_CMSE_Cortex-M_SUPPORT	CMSE (Cortex-M) Support	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
EAK	Security Extension) Union Leak				
CPP_WARN_C OMMA	Comma Operator Misuse	Yes			
CPP_WARN_C OMMENT	Comment Misuse	Yes			
CPP_WARN_C OMPARE_DIST INCT_POINTER _TYPES	Compare Distinct Pointer Types	Yes			High
CPP_WARN_C OMPLEX_COM PONENT_INIT	Complex Component Init	Yes			
CPP_WARN_C OMPOUND_TO KEN_SPLIT_BY _MACRO	Compound Token Split by Macro	Yes			High
CPP_WARN_C OMPOUND_TO KEN_SPLIT_BY _SPACE	Compound Token Split by Space	Yes			
CPP_WARN_C ONDITIONAL_ TYPE_MISMAT CH	Conditional Type Mismatch	Yes			High
CPP_WARN_C ONDITIONAL_ UNINITIALIZED	Conditional Uninitialized	Yes			
CPP_WARN_C ONSTANT_CO NVERSION	Constant Conversion	Yes			High
CPP_WARN_C ONSTANT_EVA LUATED	Constant Evaluated	Yes			
CPP_WARN_C ONSTANT_LO GICAL_OPERA ND	Constant Logical Operand	Yes			High
CPP_WARN_C	Constexpr Not	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ONSTEXPR_N OT_CONST	Const				
CPP_WARN_C ONSUMED	Consumable Attribute	Yes			
CPP_WARN_C ONVERSION	Type Conversion	Yes			
CPP_WARN_C OROUTINE	Coroutine Return Type	Yes			
CPP_WARN_C OROUTINE_MI	Coroutine Missing	Yes			
SSING_UNHA NDLED_EXCEP TION	Unhandled Exception				
CPP_WARN_C OVERED_SWIT CH_DEFAULT	Covered Switch Default	Yes			
CPP_WARN_C PP_COMPAT	C++ Compatibility	Yes			
CPP_WARN_C PP2B_EXTENS IONS	C++2B Extensions	Yes			
CPP_WARN_C PP11_COMPAT	C++11 Compatibility	Yes			
CPP_WARN_C PP11_COMPAT _DEPRECATED	C++11 Compatibility Deprecated	Yes			High
_WRITABLE_S TRINGS	Writable Strings				
CPP_WARN_C PP11_COMPAT _RESERVED_U	C++11 Compatibility Reserved User	Yes			
SER_DEFINED _LITERAL	Defined Literal				
CPP_WARN_C PP11_EXTENS IONS	C++11 Extensions	Yes			
CPP_WARN_C PP11_EXTRA_ SEMI	C++11 Extra Semicolon	Yes			
CPP_WARN_C PP11_INLINE_ NAMESPACE	C++11 Inline Namespace	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NAMESPACE					
CPP_WARN_C PP11_LONG_L	C++11 Long Long	Yes			
CPP_WARN_C PP11_NARROWING	C++11 Narrowing	Yes			
CPP_WARN_C PP14_ATTRIBUTE_EXTENSIONS	C++14 Attribute Extensions	Yes			
CPP_WARN_C PP14_BINARY_LITERAL	C++14 Binary Literal	Yes			
CPP_WARN_C PP14_EXTENSIONS	C++14 Extensions	Yes			
CPP_WARN_C PP17_ATTRIBUTE_EXTENSIONS	C++17 Attribute Extensions	Yes			
CPP_WARN_C PP17_COMPATIBILITY_MANGLING	C++17 Compatibility Mangling	Yes			
CPP_WARN_C PP17_EXTENSIONS	C++17 Extensions	Yes			
CPP_WARN_C PP20_ATTRIBUTE_EXTENSIONS	C++20 Attribute Extensions	Yes			
CPP_WARN_C PP20_COMPATIBILITY	C++20 Compatibility	Yes			
CPP_WARN_C PP20_DESIGNATOR	C++20 Designator	Yes			
CPP_WARN_C PP20_EXTENSIONS	C++20 Extensions	Yes			
CPP_WARN_C PP98_COMPATIBILITY	C++98 Compatibility	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
T					
CPP_WARN_C	C++98	Yes			
PP98_COMPAT	Compatibility				
_BIND_TO_TE	Bind to				
MPORARY_CO	Temporary				
PY	Copy				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT	Compatibility				
_EXTRA_SEMI	Extra				
	Semicolon				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT	Compatibility				
_LOCAL_TYPE	Local Type				
_TEMPLATE_A	Template Args				
RGS					
CPP_WARN_C	C++98	Yes			
PP98_COMPAT	Compatibility				
_PEDANTIC	Pedantic				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT	Compatibility				
_UNNAMED_T	Unnamed Type				
YPE_TEMPLAT	Template Args				
E_ARGS					
CPP_WARN_C	C++98 C++11	Yes			
PP98_CPP11_	Compatibility				
COMPAT_BINA	Binary Literal				
RY_LITERAL					
CPP_WARN_CT	CTAD (Class	Yes			
AD_MAYBE_U	Template				
NSUPPORTED	Argument				
	Deduction)				
	Maybe				
	Unsupported				
CPP_WARN_C	C++ Attribute	Yes			
XX_ATTRIBUT	Extension				
E_EXTENSION					
CPP_WARN_D	Dangling	Yes			High
ANGLING	Pointers				
CPP_WARN_D	Dangling Else	Yes			
ANGLING_ELS					
E					
CPP_WARN_D	Dangling Field	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
ANGLING_FIEL D					
CPP_WARN_D	Dangling	Yes			High
ANGLING_GSL	Pointers Found by Guidelines Support Library				
CPP_WARN_D	Dangling	Yes			High
ANGLING_INIT IALIZER_LIST	Initializer List				
CPP_WARN_D	Darwin SDK	Yes			
ARWIN_SDK_S ETTINGS	Settings				
CPP_WARN_D	Date and Time	Yes			
ATE_TIME	Macros				
CPP_WARN_D	Dealloc in	Yes			
EALLOC_IN_C ATEGORY	Category				
CPP_WARN_D	Debug	Yes			
EBUG_COMPR	Compression				
SSION_UNAV AILABLE	Unavailable				
CPP_WARN_D	Declaration	Yes			
ECLARATION_ AFTER_STATE MENT	After Statement				
CPP_WARN_D	Defaulted	Yes			High
EFAULTED_FU NCTION_DELE TED	Function Deleted				
CPP_WARN_D	Delegating	Yes			High
ELEGATING_C TOR_CYCLES	Constructor Cycles				
CPP_WARN_D	Delete	Yes			High
ELETE_ABSTR ACT_NON_VIR TUAL_DTOR	Abstract Non- Virtual Destructor				
CPP_WARN_D	Delete	Yes			High
ELETE_INCOM PLETE	Incomplete				
CPP_WARN_D	Delete Non-	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ELETE_NON_A BSTRACT_NO N_VIRTUAL_D TOR	Abstract Non-Virtual Destructor				
CPP_WARN_D EPRECATED	Deprecated	Yes			
CPP_WARN_D EPRECATED_A	Deprecated Altivec	Yes			
LTIVEC_SRC_C OMPAT	Instruction Set Source Compatibility				
CPP_WARN_D EPRECATED_A	Deprecated Anon-Enum,	Yes			
NON_ENUM_E NUM_CONVER SION	Enum Conversion				
CPP_WARN_D EPRECATED_A	Deprecated Array Compare	Yes			
RRAY_COMPA RE					
CPP_WARN_D EPRECATED_A	Deprecated Attributes	Yes			
TTRIBUTES					
CPP_WARN_D EPRECATED_B	Deprecated Builtins	Yes			High
UULTINS					
CPP_WARN_D EPRECATED_C	Deprecated Comma	Yes			
OMMA_SUBSC RIPT	Subscript				
CPP_WARN_D EPRECATED_C	Deprecated Copy	Yes			
OPY					
CPP_WARN_D EPRECATED_C	Deprecated Copy with	Yes			
OPY_WITH_DT OR	Destructor				
CPP_WARN_D EPRECATED_C	Deprecated Copy with User	Yes			
OPY_WITH_US ER_PROVIDED _COPY	Provided Copy				

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_D EPRECATED_C OPY_WITH_US ER_PROVIDED _DTOR	Deprecated Copy with User Provided Destructor	Yes			
CPP_WARN_D EPRECATED_C OROUTINE	Deprecated Coroutine	Yes			
CPP_WARN_D EPRECATED_D ECLARATIONS	Deprecated Declarations	Yes			High
CPP_WARN_D EPRECATED_D YNAMIC_EXCE PTION_SPEC	Deprecated Dynamic Exception Spec	Yes			
CPP_WARN_D EPRECATED_E XPERIMENTAL _COROUTINE	Deprecated Experimental Coroutine	Yes			
CPP_WARN_D EPRECATED_I MPLEMENTATI ONS	Deprecated Implementatio ns	Yes			
CPP_WARN_D EPRECATED_I NCREMENT_B OOL	Deprecated Increment Bool OOL	Yes			High
CPP_WARN_D EPRECATED_N ON_PROTOTY PE	Deprecated Non-Prototype	Yes			High
CPP_WARN_D EPRECATED_R EGISTER	Deprecated Register	Yes			High
CPP_WARN_D EPRECATED_S TATIC_ANALYZ ER_FLAG	Deprecated Static Analyzer Flag	Yes			
CPP_WARN_D EPRECATED_T HIS_CAPTURE	Deprecated This Capture	Yes			
CPP_WARN_D	Deprecated	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
EPRECATED_T YPE	Type				
CPP_WARN_D EPRECATED_V OLATILE	Deprecated Volatile	Yes			High
CPP_WARN_DI SABLED_MAC RO_EXPANSIO N	Disabled Macro Expansion	Yes			
CPP_WARN_DI VISION_BY_ZE RO	Division by Zero	Yes			High
CPP_WARN_D LL_ATTRIBUTE _ON_REDECLA RATION	DLL Attribute on Re- _ON_REDECLA Declaration	Yes			
CPP_WARN_D LLEXPORT_EX PLICIT_INSTA NTIATION_DE CL	DLLexport Explicit Instantiation Decl	Yes			
CPP_WARN_D LLIMPORT_ST ATIC_FIELD_D EF	DLLimport Static Field Def ATIC_FIELD_D EF	Yes			
CPP_WARN_D OCUMENTATIO N	Documentation Warnings	Yes			
CPP_WARN_D OCUMENTATIO N_DEPRECATE D_SYNC	Documentation Deprecated Sync	Yes			
CPP_WARN_D OCUMENTATIO N_HTML	Documentation Html	Yes			
CPP_WARN_D OCUMENTATIO N_PEDANTIC	Documentation Pedantic	Yes			
CPP_WARN_D OCUMENTATIO N_UNKNOWN_ COMMAND	Documentation Unknown Command	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_DOLLAR_IN_IDENTIFIER_EXTENSION	Dollar in Identifier Extension	Yes			
CPP_WARN_DOUBLE_PROMOTION	Double Promotion	Yes			
CPP_WARN_DESTRUCTOR_NAME	Destructor Name	Yes			
CPP_WARN_DESTRUCTOR_TYPEDEF	Destructor Typedef	Yes			
CPP_WARN_DUPLICATE_DECL_SPECIFIER	Duplicate Decl Specifier	Yes			
CPP_WARN_DUPLICATE_ENUM	Duplicate Enum	Yes			
CPP_WARN_DUPLICATE_METHOD_ARG	Duplicate Method Arg	Yes			
CPP_WARN_DUPLICATE_METHOD_MATCH	Duplicate Method Match	Yes			
CPP_WARN_DUPLICATE_PROTOCOL	Duplicate Protocol	Yes			
CPP_WARN_DYNAMIC_CLASS_MEMORY_ACCESS	Dynamic Class Memory Access	Yes			High
CPP_WARN_DYNAMIC_EXCEPTION_SPEC	Dynamic Exception Spec	Yes			
CPP_WARN_ELABORATED_ENUM_BASE	Elaborated Enum Base	Yes			
CPP_WARN_ELABORATED_ENUM_CLASS	Elaborated Enum Class	Yes			
CPP_WARN_EMBEDDED_DIRECTIVE	Embedded Directive	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ECTIVE					
CPP_WARN_E	Control Loop	Yes			
MPTY_BODY	Shall Not Have Empty Body				
CPP_WARN_E	Decomposition	Yes			
MPTY_DECOM	Group Shall				
POSITION	Not Be Empty				
CPP_WARN_E	No Empty	Yes			
MPTY_INIT_ST	Initialization				
MT	Statements				
CPP_WARN_E	Empty	Yes			
MPTY_TRANSL	Translation				
ATION_UNIT	Unit				
CPP_WARN_E	Enum Compare	Yes			High
NUM_COMPAR					
E					
CPP_WARN_E	Enum Compare	Yes			
NUM_COMPAR	Conditional				
E_CONDITION					
AL					
CPP_WARN_E	Enum Compare	Yes			High
NUM_COMPAR	Switch				
E_SWITCH					
CPP_WARN_E	Enum	Yes			
NUM_CONVER	Conversion				
SION					
CPP_WARN_E	Enum Enum	Yes			
NUM_ENUM_C	Conversion				
ONVERSION					
CPP_WARN_E	Enum Float	Yes			
NUM_FLOAT_C	Conversion				
ONVERSION					
CPP_WARN_E	Enum Too	Yes			
NUM_TOO_LA	Large				
RGE					
CPP_WARN_EX	Exceptions	Yes			High
CEPTIONS					
CPP_WARN_EX	Excess	Yes			
CESS_INITIALI	Initializers				
ZERS					
CPP_WARN_E	Exit Time	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
XIT_TIME_DESTRUCTORS	Destructors				
CPP_WARN_EXPANSION_TO_DEFINED	Expansion to Defined	Yes			
CPP_WARN_EXPORT_UNNAMED	Export Unnamed	Yes			
CPP_WARN_EXPORT_USING_DIRECTIVE	Export Using Directive	Yes			
CPP_WARN_EXTERN_C_COMPAT	Extern C Compatibility	Yes			High
CPP_WARN_EXTERN_INITIALIZER	Extern Initializer	Yes			High
CPP_WARN_EXTRA_WARNINGS	Extra Warnings	Yes			
CPP_WARN_EXTRA_QUALIFICATION	Extra Qualification	Yes			High
CPP_WARN_EXTRA_SEMICOLON	Extra Semicolon	Yes			
CPP_WARN_EXTRA_SEMICOLON_IN_EMPTY_EXPRESSION_STATEMENT	Extra Semicolon in Empty Expression Statement	Yes			
CPP_WARN_EXTRA_TOKENS	Extra Tokens	Yes			
CPP_WARN_FINAL_DESTRUCTOR_NON_FINAL_CLASS	Final Destructor Non-Final Class	Yes			
CPP_WARN_FINAL_MACROS_SHOULD_NOT_BE_REDEFINED	Final Macros Should Not Be Redefined	Yes			
CPP_WARN_FIXED_ENUM_EXTENSION	Fixed Enum Extension	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_FIXED_POINT_OVERFLOW	Fixed Point Overflow	Yes			High
CPP_WARN_FLAG_ENUM	Flag Enum	Yes			High
CPP_WARN_FLEXIBLE_ARRAY_EXTENSIONS	Flexible Array Extensions	Yes			
CPP_WARN_FLOAT_CONVERSION	Float Conversion	Yes			
CPP_WARN_FLOAT_EQUAL	Float Equal	Yes			
CPP_WARN_FLOAT_OVERFLOW_CONVERSION	Float Overflow Conversion	Yes			
CPP_WARN_FLOAT_ZERO_CONVERSION	Float Zero Conversion	Yes			
CPP_WARN_FOR_LOOP_ANALYSIS	For Loop Analysis	Yes			
CPP_WARN_FORMAT_STRING	Format String	Yes			High
CPP_WARN_FORMAT_EXTRA_ARGS	Format Extra Args	Yes			High
CPP_WARN_FORMAT_INSUFFICIENT_ARGS	Format Insufficient Args	Yes			High
CPP_WARN_FORMAT_INVALID_SPECIFIER	Format Invalid Specifier	Yes			High
CPP_WARN_FORMAT_NON_ISO	Format Non-ISO	Yes			
CPP_WARN_FORMAT_NON_LITERAL	Format Non-Literal	Yes			
CPP_WARN_FORMAT	Format	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
FORMAT_PEDANTIC	Pedantic				
CPP_WARN_FORMAT_SECURITY	Format Security	Yes			High
CPP_WARN_FORMAT_TYPE_CONFUSION	Format Type Confusion	Yes			
CPP_WARN_FORMAT_ZERO_LENGTH	Format Zero Length	Yes			High
CPP_WARN_FORTIFY_SOURCE	Fortify Source	Yes			High
CPP_WARN_FOUR_CHAR_CONSTANTS	Four Char Constants	Yes			
CPP_WARN_FRAME_ADDRESS	Frame Address	Yes			
CPP_WARN_FREE_NON_HEAP_OBJECT	Free Non-Heap Object	Yes			High
CPP_WARN_FUNCTION_MULTIVERSION	Function Multi-Version	Yes			
CPP_WARN_FUSE_LD_PATH	Fuse LD Path	Yes			
CPP_WARN_GCC_COMPAT	GCC Compatibility	Yes			
CPP_WARN_GLOBAL_CONSTRUCTORS	Global Constructors	Yes			
CPP_WARN_GLOBAL_SEL	GlobalSel (Global Instruction Selection) Framework	Yes			
CPP_WARN_GNU_ALIGNOF_EXPRESSION	GNU Alignof Expression	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_G NU_ANONYM OUS_STRUCT	GNU Anonymous Struct	Yes			
CPP_WARN_G NU_ARRAY_M EMBER_PAREN _INIT	GNU Array Member Parentheses Init	Yes			
CPP_WARN_G NU_AUTO_TY PE	GNU Auto Type PE	Yes			
CPP_WARN_G NU_BINARY_LI TERAL	GNU Binary Literal	Yes			
CPP_WARN_G NU_CASE_RA NGE	GNU Case Range	Yes			
CPP_WARN_G NU_COMPLEX _INTEGER	GNU Complex Integer	Yes			
CPP_WARN_G NU_COMPOU ND_LITERAL_I NITIALIZER	GNU Compound Literal Initializer	Yes			
CPP_WARN_G NU_CONDITIO NAL_OMITTED _OPERAND	GNU Conditional Omitted Operand	Yes			
CPP_WARN_G NU_DESIGNAT OR	GNU Designator	Yes			
CPP_WARN_G NU_EMPTY_IN ITIALIZER	GNU Empty Initializer	Yes			
CPP_WARN_G NU_EMPTY_S TRUCT	GNU Empty Struct	Yes			
CPP_WARN_G NU_FLEXIBLE_ ARRAY_INITIAL IZER	GNU Flexible Array Initializer	Yes			
CPP_WARN_G	GNU Flexible	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NU_FLEXIBLE_	Array Union				
ARRAY_UNION_	Member				
_MEMBER					
CPP_WARN_G	GNU Folding	Yes			
NU_FOLDING_	Constant				
CONSTANT					
CPP_WARN_G	GNU Imaginary	Yes			
NU_IMAGINAR	Constant				
Y_CONSTANT					
CPP_WARN_G	GNU Include	Yes			
NU_INCLUDE_	Next				
NEXT					
CPP_WARN_G	GNU Inline Cpp	Yes			
NU_INLINE_CP	Without Extern				
P_WITHOUT_E					
XTERN					
CPP_WARN_G	GNU Label as	Yes			
NU_LABEL_AS	Value				
_VALUE					
CPP_WARN_G	GNU Line	Yes			
NU_LINE_MAR	Marker				
KER					
CPP_WARN_G	GNU Null	Yes			
NU_NULL_POI	Pointer				
NTER_ARITHM	Arithmetic				
ETIC					
CPP_WARN_G	GNU Pointer	Yes			
NU_POINTER_	Arithmetic				
ARITH					
CPP_WARN_G	GNU Re-	Yes			
NU_REDECLA	Declared Enum				
RED_ENUM					
CPP_WARN_G	GNU	Yes			
NU_STATEME	Statement				
NT_EXPRESSI	Expression				
ON					
CPP_WARN_G	GNU	Yes			
NU_STATEME	Statement				
NT_EXPRESSI	Expression				
ON_FROM_MA	from Macro				
CRO_EXPANSI	Expansion				
ON					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_G NU_STATIC_FL OAT_INIT	GNU Static Float Init	Yes			
CPP_WARN_G NU_STRING_LI TERAL_OPERA TOR_TEMPLAT E	GNU String Literal Operator Template	Yes			
CPP_WARN_G NU_UNION_C AST	GNU Union Cast	Yes			
CPP_WARN_G NU_VARIABLE _SIZED_TYPE_ NOT_AT_END	GNU Variable Sized Type Not at End	Yes			
CPP_WARN_G NU_ZERO_LIN E_DIRECTIVE	GNU Zero Line Directive	Yes			
CPP_WARN_G NU_ZERO_VAR IADIC_MACRO _ARGUMENTS	GNU Zero Variadic Macro Arguments	Yes			
CPP_WARN_H EADER_GUAR D	Header Guard	Yes			High
CPP_WARN_H EADER_HYGIE NE	Header Hygiene	Yes			
CPP_WARN_ID IOMATIC_PARE NTHESSES	Idiomatic Parentheses	Yes			
CPP_WARN_IG NORED_ATTRI BUTES	Ignored Attributes	Yes			High
CPP_WARN_IG NORED_AVAIL ABILITY_WITH OUT_SDK_SET TINGS	Ignored Availability Without Sdk Settings	Yes			
CPP_WARN_IG NORED_OPTIM IZATION_ARG	Ignored Optimization Argument	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_IG	Ignored	Yes			
NORED_PRAG	Pragma				
MA_INTRINSIC	Intrinsic				
CPP_WARN_IG	Ignored	Yes			
NORED_PRAG	Pragmas				
MAS					
CPP_WARN_IG	Ignored	Yes			High
NORED_REFER	Reference				
ENCE_QUALIFI	Qualifiers				
ERS					
CPP_WARN_IM	Implicit Const	Yes			High
PLICIT_CONST	Int Float				
_INT_FLOAT_C	Conversion				
ONVERSION					
CPP_WARN_IM	Implicit	Yes			High
PLICIT_CONVE	Conversion				
RSION_FLOATI	Floating Point				
NG_POINT_TO	to Bool				
_BOOL					
CPP_WARN_IM	Implicit	Yes			High
PLICIT_EXCEP	Exception				
TION_SPEC_M	Spec Mismatch				
ISMATCH					
CPP_WARN_IM	Implicit	Yes			
PLICIT_FALLT	Fallthrough				
HROUGH					
CPP_WARN_IM	Implicit	Yes			
PLICIT_FALLT	Fallthrough Per				
HROUGH_PER	Function				
_FUNCTION					
CPP_WARN_IM	Implicit Fixed	Yes			High
PLICIT_FIXED_	Point				
POINT_CONVE	Conversion				
RSION					
CPP_WARN_IM	Implicit Float	Yes			
PLICIT_FLOAT_	Conversion				
CONVERSION					
CPP_WARN_IM	Implicit	Yes			
PLICIT_FUNCT	Function				
ION_DECLARA	Declaration				
TION					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_IMPLICIT_INT	Implicit Int	Yes			
CPP_WARN_IMPLICIT_INT_CONVERSION	Implicit Int Conversion	Yes			
CPP_WARN_IMPLICIT_INT_FLOAT_CONVERSION	Implicit Int Float Conversion	Yes			
CPP_WARN_IMPLICIT_RETAIN_SELF	Implicit Retain Self	Yes			
CPP_WARN_IMPLICITLY_UNSIGNED_LITERAL	Implicitly Unsigned Literal	Yes			High
CPP_WARN_IMPORT_PREPROCESSOR_DIRECTIVE_PEDANTIC	Import Preprocessor Directive Pedantic	Yes			
CPP_WARN_INACCESSIBLE_BASE	Inaccessible Base	Yes			High
CPP_WARN_INCLUDE_NEXT_ABSOLUTE_PATH	Include Next Absolute Path	Yes			
CPP_WARN_INCLUDE_NEXT_OUTSIDE_HEADER	Include Next Outside Header	Yes			
CPP_WARN_INCOMPATIBLE_EXCEPTION_SPEC	Incompatible Exception Spec	Yes			High
CPP_WARN_INCOMPATIBLE_FUNCTION_POINTER_TYPES	Incompatible Function Pointer Types	Yes			
CPP_WARN_INCOMPATIBLE_LIBRARY	Incompatible Library	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
LIBRARY_REDECLARATION	Redeclaration				
CPP_WARN_INCOMPATIBLE_MS_STRUCT	Incompatible Microsoft Struct	Yes			
CPP_WARN_INCOMPATIBLE_POINTER_TYPES	Incompatible Pointer Types	Yes			High
CPP_WARN_INCOMPATIBLE_POINTER_TYPES_DISCARDS_QUALIFIERS	Incompatible Pointer Types Discards Qualifiers	Yes			High
CPP_WARN_INCOMPATIBLE_SYSROOT	Incompatible Sysroot	Yes			
CPP_WARN_INCOMPLETE_IMPLEMENTATION	Incomplete Implementation	Yes			
CPP_WARN_INCOMPLETE_SETJMP_DECLARATION	Incomplete Setjmp Declaration	Yes			
CPP_WARN_INCONSISTENT_DLLIMPORT	Inconsistent DLLimport	Yes			
CPP_WARN_INCONSISTENT_MISSING_DESTRUCTOR_OVERRIDE	Inconsistent Missing Destructor Override	Yes			
CPP_WARN_INCONSISTENT_MISSING_OVERRIDE	Inconsistent Missing Override	Yes			High
CPP_WARN_INCREMENT_BOOL	Increment Bool	Yes			High
CPP_WARN_INFINITE_RECURSION	Infinite Recursion	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_IN_INITIALIZER_OVERRIDES	Initializer Overrides	Yes			
CPP_WARN_IN_INJECTED_CLASS_NAME	Injected Class Name	Yes			High
CPP_WARN_IN_INLINE_ASM	Inline Assembly	Yes			
CPP_WARN_IN_INLINE_NAMESPACES	Inline Namespace	Yes			High
CPP_WARN_IN_INLINE_NEW_DELETE	Inline New Delete	Yes			
CPP_WARN_IN_INSTANTIATION_AFTER_SPECIALIZATION	Instantiation After Specialization	Yes			High
CPP_WARN_IN_INT_CONVERSION	Int Conversion	Yes			
CPP_WARN_IN_INT_IN_BOOL_CONTEXT	Int in Bool Context	Yes			
CPP_WARN_IN_INT_TO_POINTER_CAST	Int to Pointer Cast	Yes			High
CPP_WARN_IN_INT_TO_VOID_POINTER_CAST	Int to Void Pointer Cast	Yes			High
CPP_WARN_IN_INTEGER_OVERFLOW	Integer Overflow	Yes			High
CPP_WARN_IN_INTERRUPT_SERVICE_ROUTINE	Interrupt Service Routine	Yes			
CPP_WARN_IN_INVALID_COMMAND_LINE	Invalid Command Line	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
AND_LINE_ARGUMENT	Argument				
CPP_WARN_INVALID_CONST_EXPR	Invalid Constexpr	Yes			
CPP_WARN_INVALID_IBOUTLET_COLLECTION	Invalid IOutletCollection (Interface Builder Outlet Collection)	Yes			
CPP_WARN_INVALID_INITIALIZER_FROM_SYSTEM_HEADER	Invalid Initializer from System Header	Yes			
CPP_WARN_INVALID_IOS_DEPLOYMENT_TARGET	Invalid iOS Deployment Target	Yes			
CPP_WARN_INVALID_NO_BUILTIN_NAMES	Invalid No Builtin Names	Yes			High
CPP_WARN_INVALID_NORETURN_ATTRIBUTE	Invalid Noreturn Attribute	Yes			High
CPP_WARN_INVALID_OFFSET_OF	Invalid Offsetof	Yes			High
CPP_WARN_INVALID_OR_NONEXISTENT_DIRECTORY	Invalid or Nonexistent Directory	Yes			
CPP_WARN_INVALID_PARTIAL_SPECIALIZATION	Invalid Partial Specialization	Yes			
CPP_WARN_INVALID_PP_TOKEN	Invalid Preprocessor Token	Yes			High
CPP_WARN_INVALID_SOURCE_ENCODING	Invalid Source Encoding	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_IN VALID_TOKEN _PASTE	Invalid Token Paste	Yes			
CPP_WARN_IN VALID_UTF8	Invalid UTF-8	Yes			
CPP_WARN_J UMP_SEH_FIN ALLY	Jump SEH (Structured Exception Handling) Finally	Yes			
CPP_WARN_K EYWORD_COM PAT	Keyword Compatibility	Yes			
CPP_WARN_K EYWORD_MAC RO	Keyword Macro	Yes			
CPP_WARN_K NR_PROMOTE D_PARAMETER	K&R Promoted Parameter	Yes			
CPP_WARN_L ANGUAGE_EX TENSION_TOK EN	Language Extension Token	Yes			
CPP_WARN_L ARGE_BY_VAL UE_COPY	Large by Value Copy	Yes			High
CPP_WARN_LI NKER_WARNIN GS	Linker Warnings	Yes			
CPP_WARN_LI TERAL_CONVE RSION	Literal Conversion	Yes			High
CPP_WARN_LI TERAL_RANGE	Literal Range	Yes			High
CPP_WARN_L OCAL_TYPE_T EMPLATE_ARG S	Local Type Template Args	Yes			
CPP_WARN_L OGICAL_NOT_ PARENTHESES	Logical Not Parentheses	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_L LOGICAL_OP_P ARENTHESSES	Logical Operator Parentheses	Yes			
CPP_WARN_L ONG_LONG	Long Long	Yes			
CPP_WARN_M ACRO_REDEFI NED	Macro Redefined	Yes			High
CPP_WARN_M AIN	Main Function Conventions	Yes			
CPP_WARN_M AIN_RETURN_ TYPE	Main Return Type	Yes			High
CPP_WARN_M ALFORMED_W ARNING_CHEC K	Malformed Warning Check	Yes			
CPP_WARN_M ANY_BRACES_ AROUND_SCA LAR_INIT	Many Braces Around Scalar Init	Yes			High
CPP_WARN_M AX_TOKENS	Max Tokens	Yes			
CPP_WARN_M AX_UNSIGNED _ZERO	Max Unsigned Zero	Yes			High
CPP_WARN_M EMSET_TRANS POSED_ARGS	Memset Transposed Args	Yes			High
CPP_WARN_M EMSIZE_COMP ARISON	Memsize Comparison	Yes			High
CPP_WARN_MI CROSOFT_AB STRACT	Microsoft Abstract	Yes			
CPP_WARN_MI CROSOFT_AN ON_TAG	Microsoft Anonymous Tag	Yes			
CPP_WARN_MI CROSOFT_CA ST	Microsoft Cast	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_MI_CROSOFT_CH_ARIZE	Microsoft Charizing Operator	Yes			
CPP_WARN_MI_CROSOFT_COMMENT_PASTE	Microsoft Comment Paste	Yes			
CPP_WARN_MI_CROSOFT_CONST_INIT	Microsoft Const Init	Yes			
CPP_WARN_MI_CROSOFT_CPP_MACRO	Microsoft C++ Macro	Yes			
CPP_WARN_MI_CROSOFT_DEFAULT_ARG_REDEFINITION	Microsoft Default Arg Redefinition	Yes			
CPP_WARN_MI_CROSOFT_DIRECTIVE_SECTION	Microsoft Drectve Section	Yes			
CPP_WARN_MI_CROSOFT_END_OF_FILE	Microsoft End of File	Yes			
CPP_WARN_MI_CROSOFT_ENUM_FORWARD_REFERENCE	Microsoft Enum Forward Reference	Yes			
CPP_WARN_MI_CROSOFT_ENUM_VALUE	Microsoft Enum Value	Yes			
CPP_WARN_MI_CROSOFT_EXCEPTION_SPEC	Microsoft Exception Spec	Yes			
CPP_WARN_MI_CROSOFT_EXISTS	Microsoft Exists	Yes			
CPP_WARN_MI_CROSOFT_EXPLICIT_CONSTRUCTOR_CALL	Microsoft Explicit Constructor Call	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_MI_CROSOFT_EXT_RA_QUALIFICATION	Microsoft Extra Qualification	Yes			
CPP_WARN_MI_CROSOFT_FIX_ENUM_ENUM	Microsoft Fixed Enum	Yes			
CPP_WARN_MI_CROSOFT_FLEXIBLE_ARRAY	Microsoft Flexible Array	Yes			
CPP_WARN_MI_CROSOFT_GOTO	Microsoft Goto	Yes			
CPP_WARN_MI_CROSOFT_INACCESSIBLE_BASE	Microsoft Inaccessible Base	Yes			
CPP_WARN_MI_CROSOFT_INCLUDE	Microsoft Include	Yes			
CPP_WARN_MI_CROSOFT_MUTABLE_REFERENCE	Microsoft Mutable Reference	Yes			
CPP_WARN_MI_CROSOFT_PURE_DEFINITION	Microsoft Pure Definition	Yes			
CPP_WARN_MI_CROSOFT_REDECLARE_STATIC	Microsoft Redeclare Static	Yes			
CPP_WARN_MI_CROSOFT_SEALED	Microsoft Sealed	Yes			
CPP_WARN_MI_CROSOFT_STATIC_ASSERT	Microsoft Static Assert	Yes			
CPP_WARN_MI_CROSOFT_TEMPLATE	Microsoft Template	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_MI_CROSOFT_TEMPLATE_SHADOW_DOW	Microsoft Template Shadow DOW	Yes			
CPP_WARN_MI_CROSOFT_UNION_MEMBER_REFERENCE	Microsoft Union Member Reference	Yes			
CPP_WARN_MI_CROSOFT_UNQUALIFIED_FRIEND_IEND	Microsoft Unqualified Friend IEND	Yes			
CPP_WARN_MI_CROSOFT_USING_DECLING_DECL	Microsoft Using DeclING_DECL	Yes			
CPP_WARN_MI_CROSOFT_VOID_PSEUDO_D_PSEUDO_DT_OR	Microsoft Void Pseudo D_PSEUDO_DT OR	Yes			
CPP_WARN_MI_SEXPECT___builtin_expect()	Misuse of __builtin_expect()	Yes			
CPP_WARN_MI_SLEADING_INDENTATION	Misleading Indentation	Yes			
CPP_WARN_MI_SMATCHED_NEW_DELETE	Mismatched New Delete	Yes			High
CPP_WARN_MI_SMATCHED_TAGS	Mismatched Tags	Yes			
CPP_WARN_MI_SSING_BRACES	Missing Braces	Yes			
CPP_WARN_MI_SSING_CONST_INIT	Missing Constinit	Yes			
CPP_WARN_MI_SSING_DECLARATIONS	Missing Declarations	Yes			High
CPP_WARN_MI_Missing	Missing	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
SSING_EXCEPTION_SPEC	Exception Spec				
CPP_WARN_MI_MISSING_FIELD_INITIALIZERS	Missing Field Initializers	Yes			
SSING_METHOD_RETURN_TYPE	Method Return Type	Yes			
CPP_WARN_MI_MISSING_NORETURN_ATTRIBUTE	Missing Noreturn Attribute	Yes			
SSING_PROTO_TYPE_FOR_CALLING_CONVENTION	Prototype for Calling Convention	Yes			
CPP_WARN_MI_MISSING_PROTO_TYPES	Missing Prototypes	Yes			
SSING_SELECTOR_NAME_TOR_NAME	Selector Name	Yes			
CPP_WARN_MI_MISSING_SYSROOT	Missing Sysroot	Yes			
SSING_VARIABLE_DECLARATIONS	Variable Declarations	Yes			
CPP_WARN_MI_MISSPELLED_ASSUMPTION	Misspelled Assumption	Yes			
CPP_WARN_MODULE_CONFLICT	Module Conflict	Yes			
CPP_WARN_MODULE_FILE_CONFIG_MISMATCH	Module File Config Mismatch	Yes			
CPP_WARN_MODULE_FILE_EXTENSION	Module File Extension	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
XTENSION					
CPP_WARN_M	Module Import	Yes			
MODULE_IMPORT	in Extern C				
T_IN_EXTERN_C					
CPP_WARN_M	Modules	Yes			
MODULES_AMBI	Ambiguous				
UNAMBIGUOUS_INTER	Internal				
NAL_LINKAGE	Linkage				
CPP_WARN_M	Modules	Yes			
MODULES_IMPO	Import Nested				
RT_NESTED_R	Redundant				
REDUNDANT					
CPP_WARN_M	MSVC Not	Yes			
SVC_NOT_FOUND	Found				
UND					
CPP_WARN_M	Multiple	Yes			High
ULTICHAR	Characters in				
	Character				
	Literal				
CPP_WARN_M	Multiple Move	Yes			High
MULTIPLE_MOV	Virtual Base				
E_VBASE					
CPP_WARN_N	Nested Anon-	Yes			
NESTED_ANON_	Types				
TYPES					
CPP_WARN_N	New Returns	Yes			High
NEW_RETURNS	Null				
_NULL					
CPP_WARN_N	Newline EOF	Yes			
NEWLINE_EOF	(End of File)				
CPP_WARN_N	Noderef	Yes			High
NO_DEREF	Attribute				
CPP_WARN_N	Non-C Typedef	Yes			High
NON_C_TYPEDEF	for Linkage				
F_FOR_LINKAGE					
GE					
CPP_WARN_N	Non-Literal	Yes			High
NON_LITERAL_	Null				
NULL_CONVERSION	Conversion				
RSION					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_N ON_POD_VAR ARGS	Non-POD (Plain Old Data) Variadic Arguments	Yes			High
CPP_WARN_N ON_POWER_O F_TWO_ALIGN MENT	Non-Power of Two Alignment	Yes			High
CPP_WARN_N ON_VIRTUAL_ DTOR	Non-Virtual Destructor	Yes			
CPP_WARN_N ONNULL	Null as an Argument	Yes			High
CPP_WARN_N ONPORTABLE_ INCLUDE_PAT H	Non-Portable Include Path	Yes			High
CPP_WARN_N ONPORTABLE_ SYSTEM_INCL UDE_PATH	Non-Portable System Include Path	Yes			
CPP_WARN_N ONPORTABLE_ VECTOR_INITI ALIZATION	Non-Portable Vector Initialization	Yes			
CPP_WARN_N ONTRIVIAL_M EMACCESS	Nontrivial Memory Access	Yes			High
CPP_WARN_N ULL_ARITHME TIC	Null Arithmetic	Yes			High
CPP_WARN_N ULL_CHARACT ER	Null Character	Yes			High
CPP_WARN_N ULL_CONVERS ION	Null Conversion	Yes			High
CPP_WARN_N ULL_DEREFER ENCE	Null Dereference	Yes			High
CPP_WARN_N	Null Pointer	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ULL_POINTER_ARITHMETIC	Arithmetic				
CPP_WARN_NULL_POINTER	Null Pointer	Yes			
ULL_POINTER_SUBTRACTION	Subtraction				
CPP_WARN_OBJC_C_BOOL_CONSTANT_CONVERSION	Objective-C Bool Constant Conversion	Yes			
CPP_WARN_OBJC_C_CIRCULAR_CONTAINER	Objective-C Circular Container	Yes			
CPP_WARN_OBJC_C_MULTIPLE_METHOD_NAMES	Objective-C Multiple Method Names	Yes			
CPP_WARN_OBJC_C_READONLY_WITH_SETTER_PROPERTY	Objective-C Readonly with Setter Property	Yes			
CPP_WARN_OBJC_C_SIGNED_CHAR_BOOL_IMPLICIT_FLOAT_CONVERSION	Objective-C Signed Char Bool Implicit Float Conversion	Yes			
CPP_WARN_OBJC_C_SIGNED_CHAR_BOOL_IMPLICIT_INT_CONVERSION	Objective-C Signed Char Bool Implicit Int Conversion	Yes			
CPP_WARN_ONE_DEFINITION_RULE	One Definition Rule	Yes			High
CPP_WARN_OLD_STYLE_CAST	Old Style Cast	Yes			
CPP_WARN_OPENMP_51_EXTENSIONS	OpenMP 51 Extensions	Yes			
CPP_WARN_OPENMP_CLAUSE	OpenMP Clauses	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
SES					
CPP_WARN_O PENMP_LOOP_ FORM	OpenMP Loop Form	Yes			
CPP_WARN_O PENMP_MAPPI NG	OpenMP Mapping	Yes			
CPP_WARN_O PENMP_TARGE T	OpenMP Target	Yes			
CPP_WARN_O PTION_IGNOR ED	Option Ignored	Yes			
CPP_WARN_O RDERED_COM PARE_FUNCTI ON_POINTERS	Ordered Compare Function Pointers	Yes			High
CPP_WARN_O UT_OF_LINE_D ECLARATION	Out of Line Declaration	Yes			
CPP_WARN_O UT_OF_SCOPE _FUNCTION	Out of Scope Function	Yes			High
CPP_WARN_O VER_ALIGNED	Over Aligned	Yes			
CPP_WARN_O VERLENGTH_S TRINGS	Long String Literals	Yes			
CPP_WARN_O VERLOADED_S HIFT_OP_PARE NTHESES	Overloaded Shift Operator Parentheses	Yes			High
CPP_WARN_O VERLOADED_V IRTUAL	Overloaded Virtual	Yes			
CPP_WARN_O VERRIDE_MOD ULE	Override Module	Yes			
CPP_WARN_O VERRIDING_T_ OPTION	Overriding Slash T Option	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_PACKED_ATTRIBUTE	Packed Attribute	Yes			
CPP_WARN_IMPLICIT_PADDING	Implicit Padding	Yes			
CPP_WARN_PARENTHESES	Parentheses	Yes			
CPP_WARN_PARENTHESES_EQUALITY	Parentheses Equality	Yes			High
CPP_WARN_PASS_FAILED	Pass Failed	Yes			
CPP_WARN_PCH	PCH	Yes			
CH_DATE_TIME	(Precompiled Header) Date Time				
CPP_WARN_PEDANTIC	Pedantic	Yes			
CPP_WARN_PEDANTIC_CORE_FEATURES	Pedantic Core Features	Yes			
CPP_WARN_PESSIMIZING_MOVE	Pessimizing Move	Yes			
CPP_WARN_POINTER_ARITHMETIC	Pointer Arithmetic	Yes			High
CPP_WARN_POINTER_BOOL_CONVERSION	Pointer Bool Conversion	Yes			High
CPP_WARN_POINTER_COMPARE	Pointer Compare	Yes			High
CPP_WARN_POINTER_INTEGER_COMPARE	Pointer Integer Compare	Yes			High
CPP_WARN_POINTER_SIGN	Pointer Sign	Yes			High
CPP_WARN_POINTER_TO_ENUM_CAST	Pointer to Enum Cast	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_P OINTER_TO_IN T_CAST	Pointer to Int Cast	Yes			High
CPP_WARN_P OINTER_TYPE _MISMATCH	Pointer Type Mismatch	Yes			High
CPP_WARN_P OISON_SYSTE M_DIRECTORI ES	Poison System Directories	Yes			
CPP_WARN_P OTENTIALLY_E VALUATED_EX PRESSION	Potentially Evaluated Expression	Yes			High
CPP_WARN_P RAGMA_CLAN G_ATTRIBUTE	Pragma Clang Attribute	Yes			
CPP_WARN_P RAGMA_MESS AGES	Preprocessor #Pragma Messages	Yes			
CPP_WARN_P RAGMA_ONCE _OUTSIDE_HE ADER	Pragma once Outside Header	Yes			High
CPP_WARN_P RAGMA_PACK	Pragma Pack	Yes			
CPP_WARN_P RAGMA_PACK _SUSPICIOUS _INCLUDE	Pragma Pack Suspicious Include	Yes			
CPP_WARN_P RAGMA_SYST EM_HEADER_ OUTSIDE_HEA DER	Pragma System Header Outside Header	Yes			
CPP_WARN_P RAGMAS	Pragmas	Yes			
CPP_WARN_P RE_C2X_COM PAT	Pre C2X Compatibility	Yes			
CPP_WARN_P	Pre C++2B	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
RE_CPP2B_CO	Compatibility				
MPAT					
CPP_WARN_P	Pre C++14	Yes			
RE_CPP14_CO	Compatibility				
MPAT					
CPP_WARN_P	Pre C++17	Yes			
RE_CPP17_CO	Compatibility				
MPAT					
CPP_WARN_P	Pre C++17	Yes			
RE_CPP17_CO	Compatibility				
MPAT_PEDANTIC	Pedantic				
IC					
CPP_WARN_P	Pre C++20	Yes			
RE_CPP20_CO	Compatibility				
MPAT					
CPP_WARN_P	Pre C++20	Yes			
RE_CPP20_CO	Compatibility				
MPAT_PEDANTIC	Pedantic				
IC					
CPP_WARN_P	Pre OpenMP 5.1	Yes			
RE_OPENMP_51_COMPAT	Compatibility				
CPP_WARN_P	Predefined	Yes			
REDEFINED_IDENTIFIER_OUTSIDE_FUNCTION	Identifier Outside Function				
CPP_WARN_P	Private Extern	Yes			
PRIVATE_EXTERN					
CPP_WARN_P	Private Header	Yes			
PRIVATE_HEADER					
CPP_WARN_P	Profile	Yes			
ROFILE_INSTRUMENTED_CODE_MISSING	Instrumented Code Missing				
CPP_WARN_P	Profile	Yes			
ROFILE_INSTRUMENTED_CODE_OUT_OF_DATE	Instrumented Code Out of Date				
CPP_WARN_P	Profile	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ROFILE_INSTR _UNPROFILED	Instrumented Code Unprofiled				
CPP_WARN_P SABI	PSABI (Processor- Specific Application Binary Interface)	Yes			
CPP_WARN_Q UALIFIED_VOI D_RETURN_TY PE	Qualified Void Return Type	Yes			High
CPP_WARN_R ANGE_LOOP_B IND_REFEREN CE	Range Loop Bind Reference	Yes			
CPP_WARN_R ANGE_LOOP_C ONSTRUCT	Range Loop Construct	Yes			
CPP_WARN_R EDECLARED_C LASS_MEMBE R	Re-Declared Class Member	Yes			High
CPP_WARN_R EDUNDANT_C ONSTEVAL_IF	Redundant Consteval If	Yes			High
CPP_WARN_R EDUNDANT_M OVE	Redundant Move	Yes			
CPP_WARN_R EDUNDANT_P ARENS	Redundant Parentheses	Yes			
CPP_WARN_R EGISTER	Register Keyword	Yes			
CPP_WARN_R EINTERPRET_B ASE_CLASS	Reinterpret Base Class	Yes			High
CPP_WARN_R EORDER_CTO R	Reorder Constructor	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_R EORDER_INIT_ LIST	Reorder Initializer List	Yes			High
CPP_WARN_R ESERVED_IDE NTIFIER	Reserved Identifier	Yes			
CPP_WARN_R ESERVED_MA CRO_IDENTIFI ER	Reserved Macro Identifier	Yes			
CPP_WARN_R ESERVED_USE R_DEFINED_LI TERAL	Reserved User Defined Literal	Yes			
CPP_WARN_R ESTRICT_EXPA NSION	Restrict Expansion	Yes			
CPP_WARN_R ETAINED_LAN GUAGE_LINKA GE	Retained Language Linkage	Yes			
CPP_WARN_R ETURN_STACK _ADDRESS	Return Stack Address	Yes			High
CPP_WARN_R ETURN_TYPE	Return Type	Yes			High
CPP_WARN_R ETURN_TYPE_ C_LINKAGE	Return Type C Linkage	Yes			High
CPP_WARN_R EWRITE_NOT_ BOOL	Rewrite Not Bool	Yes			
CPP_WARN_R TTI	Run-Time Type Information	Yes			
CPP_WARN_S ARIF_FORMAT _UNSTABLE	SARIF Format Unstable	Yes			
CPP_WARN_SE CTION	Section Attributes	Yes			
CPP_WARN_SE LF_ASSIGN	Self Assign	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_SELF_ASSIGN_FIELD_ELD	Self Assign Field	Yes			High
CPP_WARN_SELF_ASSIGN_OVERLOADED	Self Assign Overloaded	Yes			
CPP_WARN_SELF_MOVE	Self Move	Yes			
CPP_WARN_SENTEINEL_ATTRIBUTE	Sentinel Attribute	Yes			
CPP_WARN_SERIALIZED_DIAGNOSTICS	Serialized Diagnostics	Yes			
CPP_WARN_SHADOWING_IDENTIFIERS	Shadowing Identifiers	Yes			
CPP_WARN_SHADOWING_FIELD	Shadowing Field	Yes			
CPP_WARN_SHADOWING_FIELD_IN_CONSTRUCTOR	Shadowing Field in Constructor	Yes			
CPP_WARN_SHADOWING_FIELD_IN_CONSTRUCTOR_MODIFIED	Shadowing Field in Constructor Modified	Yes			
CPP_WARN_SHADOWING_UNCAPTURED_LOCAL	Shadowing Uncaptured Local	Yes			
CPP_WARN_SHIFT_COUNT_NEGATIVE	Shift Count Negative	Yes			High
CPP_WARN_SHIFT_COUNT_OVERFLOW	Shift Count Overflow	Yes			High
CPP_WARN_SHIFT_NEGATIVE_VALUE	Shift Negative Value	Yes			High
CPP_WARN_SHIFT_OPERATOR_PARENTHESES	Shift Operator Parentheses	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
NTHESES					
CPP_WARN_S SHIFT_OVERFLOW	Shift Overflow	Yes			High
CPP_WARN_S SHIFT_SIGN_OVERFLOW	Shift Sign Overflow	Yes			
CPP_WARN_S SHORTEN_64_T O_32	Shorten Integer Type Width	Yes			
CPP_WARN_SI GN_COMPARE	Sign Compare	Yes			
CPP_WARN_SI GN_CONVERSION	Sign Conversion	Yes			
CPP_WARN_SI GNED_ENUM_BITFIELD	Signed Enum Bitfield	Yes			
CPP_WARN_SI GNED_UNSIG_NED_WCHAR	Signed Unsigned Wchar	Yes			
CPP_WARN_SI NGLE_BIT_BIT FIELD_CONST	Single Bit Bitfield Constant	Yes			High
CPP_WARN_SI ZEOF_ARRAY_ARGUMENT	Sizeof Array Argument	Yes			High
CPP_WARN_SI ZEOF_ARRAY_DECAY	Sizeof Array Decay	Yes			High
CPP_WARN_SI ZEOF_ARRAY_DIV	Sizeof Array Division	Yes			High
CPP_WARN_SI ZEOF_POINTER_DIV	Sizeof Pointer Division	Yes			High
CPP_WARN_SI ZEOF_POINTER_MEMORY_ACCESS	Sizeof Pointer Memory Access	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
S					
CPP_WARN_SLASH_U_FILENAME	Slash U Filename	Yes			
CPP_WARN_SLASH_U_ASM_GOTO	SLH (Speculative Load Hardening) Assembly Goto	Yes			
CPP_WARN_SOMETIMES_UNINITIALIZED	Sometimes Uninitialized	Yes			
CPP_WARN_SOURCE_USES_OPENMP	Source Uses OpenMP	Yes			
CPP_WARN_SPIR_COMPAT	SPIR (Sampler Initializer) Compatibility	Yes			
CPP_WARN_STACK_EXHAUSTED	Stack Exhausted	Yes			
CPP_WARN_STACK_PROTECTOR	Stack Protector	Yes			
CPP_WARN_STATIC_FLOAT_INIT	Static Float Init	Yes			
CPP_WARN_STATIC_INLINE	Static in Inline	Yes			
CPP_WARN_STATIC_INLINE_EXPLICIT_INSTANTIATION	Static Inline Explicit Instantiation	Yes			High
CPP_WARN_STATIC_LOCAL_INLINE	Static Local in Inline	Yes			High
CPP_WARN_STATIC_SELF_INIT	Static Self Init	Yes			High
T					
CPP_WARN_LIBSTDCPP	LibStdC++	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
TDLIBCXX_NO_T_FOUND	Headers Not Found				
CPP_WARN_S	Strict	Yes			
TRICT_POTENTIALLY_DIRECT_SELECTOR	Potentially Direct Selector				
CPP_WARN_S	Strict	Yes			
TRICT_PROTO_TYPES	Prototypes				
CPP_WARN_S	Strict Selector	Yes			
TRICT_SELECT_OR_MATCH	Match				
CPP_WARN_S	String	Yes			High
TRING_COMPARE	Compare				
CPP_WARN_S	String	Yes			
TRING_CONCATENATION	Concatenation				
CPP_WARN_S	String	Yes			
TRING_CONVERSION	Conversion				
CPP_WARN_S	String Plus	Yes			High
TRING_PLUS_CHAR	Char				
CPP_WARN_S	String Plus Int	Yes			High
TRING_PLUS_INT					
CPP_WARN_S	Strlcpy Strlcat	Yes			High
TRLCPY_STRLCAT_SIZE	Size				
CPP_WARN_S	Strncat Size	Yes			High
TRNCAT_SIZE					
CPP_WARN_S	Suggest	Yes			
UGGEST_DESTROY_OVERRIDE	Destructor Override				
CPP_WARN_S	Suggest	Yes			
UGGEST_OVERRIDE	Override				
CPP_WARN_S	Super Class	Yes			
UPER_CLASS_METHOD	Method				

Check ID	Check Name	Supported	Automation	Category	Severity
METHOD_MISMATCH	Mismatch				
CPP_WARN_SUSPICIOUS_BZERO	Suspicious Argument for Bzero Function	Yes			
CPP_WARN_SWITCH_WITCH	Switch Statements	Yes			High
CPP_WARN_SWITCH_BOOL_WITCH_BOOL	Switch Bool	Yes			High
CPP_WARN_SWITCH_ENUM_WITCH_ENUM	Switch Enum	Yes			
CPP_WARN_SYNC_FETCH_AND_NAND_SEMANTICS_CHANGED	Sync Fetch And Nand Semantics Changed	Yes			
CPP_WARN_TARGET_CLONES_MIXED_SPECIFIERS	Target Clones Mixed Specifiers	Yes			
CPP_WARN_TAUTOLOGICAL_BITWISE_COMPARE	Tautological Bitwise Compare	Yes			
CPP_WARN_TAUTOLOGICAL_COMPARE	Tautological Compare	Yes			
CPP_WARN_TAUTOLOGICAL_CONSTANT_COMPARE	Tautological Constant Compare	Yes			High
CPP_WARN_TAUTOLOGICAL_CONSTANT_OUT_OF_RANGE_COMPARE	Tautological Constant Out of Range Compare	Yes			High
CPP_WARN_TAUTOLOGICAL_OVERLAP_COMPARE	Tautological Overlap Compare	Yes			
CPP_WARN_TAUTOLOGICAL	Tautological	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
UTOLOGICAL_POINTER_COMPARE	Pointer Compare				
CPP_WARN_TAUTOLOGICAL_TYPE_LIMIT_COMPARE	Tautological Type Limit Compare	Yes			
UTOLOGICAL_TYPE_LIMIT_COMPARE	Type Limit Compare				
CPP_WARN_TAUTOLOGICAL_UNDEFINED_COMPARE	Tautological Undefined Compare	Yes			High
UTOLOGICAL_UNDEFINED_COMPARE	Undefined Compare				
CPP_WARN_TAUTOLOGICAL_UNSIGNED_CHAR_ZERO_COMPARE	Tautological Unsigned Char Zero Compare	Yes			
UTOLOGICAL_UNSIGNED_CHAR_ZERO_COMPARE	Unsigned Char Zero Compare				
CPP_WARN_TAUTOLOGICAL_UNSIGNED_ENUM_ZERO_COMPARE	Tautological Unsigned Enum Zero Compare	Yes			
UTOLOGICAL_UNSIGNED_ENUM_ZERO_COMPARE	Unsigned Enum Zero Compare				
CPP_WARN_TAUTOLOGICAL_UNSIGNED_ZERO_COMPARE	Tautological Unsigned Zero Compare	Yes			
UTOLOGICAL_UNSIGNED_ZERO_COMPARE	Unsigned Zero Compare				
CPP_WARN_TAUTOLOGICAL_VALUE_RANGE_COMPARE	Tautological Value Range Compare	Yes			
UTOLOGICAL_VALUE_RANGE_COMPARE	Value Range Compare				
CPP_WARN_TCB_ENFORCEMENT	TCB (Trusted Computing Base) Enforcement	Yes			
CPP_WARN_TENTATIVE_DEFINITION_INCOMPLETE_TYPE	Tentative Definition Incomplete Type	Yes			High
CPP_WARN_THREAD_SAFETY_ANALYSIS	Thread Safety Analysis	Yes			
CPP_WARN_THREAD_SAFETY_ATTRIBUTES	Thread Safety Attributes	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
Y_ATTRIBUTES					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFETY	Beta				
Y_BETA					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFETY	Negative				
Y_NEGATIVE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFETY	Precise				
Y_PRECISE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFETY	Reference				
Y_REFERENCE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFETY	Verbose				
Y_VERBOSE					
CPP_WARN_TR	Trigraphs	Yes			High
IGRAPHS					
CPP_WARN_TY	Type Safety	Yes			High
PE_SAFETY					
CPP_WARN_TY	Typedef	Yes			High
PEDEF_REDEFINITION	Redefinition				
CPP_WARN_TY	Typename	Yes			High
PENAME_MISSING	Missing				
CPP_WARN_U	Unable to	Yes			
NABLE_TO_OPEN_STATS_FILE	Open Stats File				
CPP_WARN_U	Unaligned	Yes			
NALIGNED_ACCESS	Access				
CPP_WARN_U	Unaligned	Yes			
NALIGNED_QUALIFIER_IMPLICIT_CAST	Qualifier				
CPP_WARN_U	Undefined	Yes			
NDEF	Macros				
CPP_WARN_U	Undefined	Yes			
NDEF_PREFIX	Macros of a				

Check ID	Check Name	Supported	Automation	Category	Severity
	Certain Prefix				
CPP_WARN_U NDEFINED_BO OL_CONVERSI ON	Undefined Bool Conversion	Yes			High
CPP_WARN_U NDEFINED_FU NC_TEMPLATE	Undefined Function Template	Yes			
CPP_WARN_U NDEFINED_INL INE	Undefined Inline	Yes			High
CPP_WARN_U NDEFINED_INT ERNAL	Undefined Internal	Yes			High
CPP_WARN_U NDEFINED_INT ERNAL_TYPE	Undefined Internal Type	Yes			
CPP_WARN_U NDEFINED_REI NTERPRET_CA ST	Undefined Reinterpret Cast	Yes			
CPP_WARN_U NDEFINED_VA R_TEMPLATE	Undefined Var Template	Yes			High
CPP_WARN_U NDERALIGNED _EXCEPTION_ OBJECT	Underaligned Exception Object	Yes			
CPP_WARN_U NEVALUATED_ EXPRESSION	Unevaluated Expression	Yes			High
CPP_WARN_U NGUARDED_A VAILABILITY	Unguarded Availability	Yes			
CPP_WARN_U NGUARDED_A VAILABILITY_N EW	Unguarded Availability New	Yes			High
CPP_WARN_U NICODE	Unicode Escape Sequences	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_U NICODE_HOM OGLYPH	Unicode Homoglyph	Yes			High
CPP_WARN_U NICODE_WHIT ESPACE	Unicode Whitespace	Yes			High
CPP_WARN_U NICODE_ZERO _WIDTH	Unicode Zero Width	Yes			High
CPP_WARN_U NINITIALIZED	Uninitialized	Yes			
CPP_WARN_U NINITIALIZED_ CONST_REFER ENCE	Uninitialized Const Reference	Yes			
CPP_WARN_U NKNOWN_AR GUMENT	Unknown Argument	Yes			
CPP_WARN_U NKNOWN_ASS UMPTION	Unknown Assumption	Yes			
CPP_WARN_U NKNOWN_ATT RIBUTES	Unknown Attributes	Yes			
CPP_WARN_U NKNOWN_DIR ECTIVES	Unknown Directives	Yes			High
CPP_WARN_U NKNOWN_ESC APE_SEQUEN CE	Unknown Escape Sequence	Yes			High
CPP_WARN_U NKNOWN_PRA GMAS	Unknown Pragmas	Yes			
CPP_WARN_U NKNOWN_SAN ITIZERS	Unknown Sanitizers	Yes			
CPP_WARN_U NKNOWN_WA RNING_OPTIO N	Unknown Warning Option	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_U NNAMED_TYP E_TEMPLATE_ ARGS	Unnamed Type Template Args	Yes			
CPP_WARN_U NNEEDED_INT ERNAL_DECLA RATION	Unneeded Internal Declaration	Yes			
CPP_WARN_U NNEEDED_ME MBER_FUNCTI ON	Unneeded Member Function	Yes			
CPP_WARN_U NQUALIFIED_S TD_CAST_CAL L	Unqualified Standard Cast Call	Yes			High
CPP_WARN_U NREACHABLE_ CODE	Unreachable Code	Yes			
CPP_WARN_U NREACHABLE_ CODE_BREAK	Unreachable Code Break	Yes			
CPP_WARN_U NREACHABLE_ CODE_FALLTH ROUGH	Unreachable Code Fallthrough	Yes			
CPP_WARN_U NREACHABLE_ CODE_GENERI C_ASSOC	Unreachable Code Generic Assoc	Yes			
CPP_WARN_U NREACHABLE_ CODE_LOOP_I NCREMENT	Unreachable Code Loop Increment	Yes			
CPP_WARN_U NREACHABLE_ CODE_RETUR N	Unreachable Code Return	Yes			
CPP_WARN_U NSAFE_BUFFER R_USAGE	Unsafe Buffer Usage	Yes			
CPP_WARN_U	Unsequenced	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
NSEQUENCED	Modifications				
CPP_WARN_U NSUPPORTED	Unsupported ABI	Yes			
_ABI	(Application Binary Interface)				
CPP_WARN_U NSUPPORTED	Unsupported Absolute Value	Yes			
_ABS	Argument				
CPP_WARN_U NSUPPORTED	Unsupported Availability	Yes			High
_AVAILABILITY	Guard				
_GUARD					
CPP_WARN_U NSUPPORTED	Unsupported Compact	Yes			
_CB	Branches				
CPP_WARN_U NSUPPORTED	Unsupported DLL Base	Yes			
_DLL_BASE_C	Class Template				
LASS_TEMPLA					
TE					
CPP_WARN_U NSUPPORTED	Unsupported Floating Point	Yes			
_FLOATING_P	Option				
OINT_OPT					
CPP_WARN_U NSUPPORTED	Unsupported Friend	Yes			High
_FRIEND					
CPP_WARN_U NSUPPORTED	Unsupported GPOpt	Yes			
_GPOPT	(Gaussian Process Optimization)				
CPP_WARN_U NSUPPORTED	Unsupported Nan Argument	Yes			
_NAN					
CPP_WARN_U NSUPPORTED	Unsupported Target Option	Yes			
_TARGET_OPT					
CPP_WARN_U NSUPPORTED	Unsupported Visibility	Yes			
_VISIBILITY					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_U NUSABLE_PAR TIAL_SPECIALI ZATION	Unusable Partial Specialization	Yes			
CPP_WARN_U NUSED_BUT_S ET_PARAMETE R	Unused but Set Parameter	Yes			
CPP_WARN_U NUSED_BUT_S ET_VARIABLE	Unused but Set Variable	Yes			
CPP_WARN_U NUSED_COMM AND_LINE_AR GUMENT	Unused Command Line Argument	Yes			
CPP_WARN_U NUSED_COMP ARISON	Unused Comparison	Yes			High
CPP_WARN_U NUSED_CONS T_VARIABLE	Unused Const Variable	Yes			
CPP_WARN_U NUSED_EXCEP TION_PARAME TER	Unused Exception Parameter	Yes			
CPP_WARN_U NUSED_FUNC TION	Unused Function	Yes			
CPP_WARN_U NUSED_GETTE R_RETURN_VA LUE	Unused Getter Return Value	Yes			
CPP_WARN_U NUSED_LABEL	Unused Label	Yes			
CPP_WARN_U NUSED_LAMB DA_CAPTURE	Unused Lambda Capture	Yes			
CPP_WARN_U NUSED_LOCA L_TYPEDEF	Unused Local Typedef	Yes			
CPP_WARN_U	Unused	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NUSED_MACROS	Macros				
OS					
CPP_WARN_UNUSED	Unused	Yes			
NUSED_MEMBER_FUNCTION	Member Function				
CPP_WARN_UNUSED	Unused	Yes			
NUSED_PARAMETER	Parameter				
METER					
CPP_WARN_UNUSED	Unused Private	Yes			
NUSED_PRIVATE_FIELD	Field				
TE_FIELD					
CPP_WARN_UNUSED	Unused	Yes			
NUSED_PROPERTY_IVAR	Property IVar (Instance Variable)				
CPP_WARN_UNUSED	Unused Result	Yes			High
NUSED_RESULT					
CPP_WARN_UNUSED	Unused	Yes			
NUSED_TEMPLATE	Template				
LATE					
CPP_WARN_UNUSED	Unused Value	Yes			High
NUSED_VALUE					
CPP_WARN_UNUSED	Unused	Yes			
NUSED_VARIABLE	Variable				
BLE					
CPP_WARN_UNUSED	Unused	Yes			High
NUSED_VOLATILE_LVALUE	Volatile Lvalue				
ILE_LVALUE					
CPP_WARN_UNUSED	Used but	Yes			
SED_BUT_MARKED	Marked				
RKED_UNUSED	Unused				
D					
CPP_WARN_UNUSED	User Defined	Yes			High
SER_DEFINED_LITERALS	Literals				
_LITERALS					
CPP_WARN_UNUSED	User Defined	Yes			
SER_DEFINED_WARNINGS	Warnings				
WARNINGS					
CPP_WARN_VARIADIC	Variadic	Yes			High
RARGS	Arguments				

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_VARIADIC_MACROS	Variadic Macros	Yes			
CPP_WARN_VECTOR_ELEMENT_SIZE	Vector Element Size	Yes			
CPP_WARN_VECTOR_CONVERSION	Vector Conversion	Yes			
CPP_WARN_VEXING_PARSE_OCCURRENCES	Vexing Parse Occurrences	Yes			High
CPP_WARN_VISIBILITY_DECLARATIONS	Visibility of Declarations	Yes			High
CPP_WARN_VLA_LENGTH_ARRAY	VLA (Variable Length Array)	Yes			
CPP_WARN_VLA_LENGTH_ARRAY_EXTENSION	VLA (Variable Length Array) Extension	Yes			
CPP_WARN_VOID_POINTER_TO_ENUM_CAST	Void Pointer to Enum Cast	Yes			High
CPP_WARN_VOID_POINTER_TO_INT_CAST	Void Pointer to Int Cast	Yes			High
CPP_WARN_VOID_POINTER_DEREFERENCE	Void Pointer Dereference	Yes			High
CPP_WARN_PREPROCESSOR_WARNINGS	Preprocessor #Warnings	Yes			
CPP_WARN_WASM_EXCEPTION_SPEC	Wasm Exception Spec	Yes			
CPP_WARN_WEAK_VTABLES	Weak VTables (Virtual Tables)	Yes			
CPP_WARN_WRITABLE_STRINGS	Writable Strings	Yes			High
CPP_WARN_XOR_USED_AS_POWER	Xor Used as Power	Yes			High
CPP_WARN_ZERO_AS_NULL	Zero as Null	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
ERO_AS_NULL_POINTER_CONSTANT	Pointer Constant				
CPP_WARN_ZERO_LENGTH_ARRAY	Zero Length Array	Yes			
CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes			High
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes			High
CTR52-CPP	Guarantee that library functions do not overflow	Yes			High
CTR53-CPP	Use valid iterator ranges	Yes			High
CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes			Medium
CTR55-CPP	Do not use an additive operator on an iterator if the result would overflow	Yes			High
CTR56-CPP	Do not use pointer arithmetic on polymorphic objects	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CTR57-CPP	Provide a valid ordering predicate	Yes			Low
CTR58-CPP	Predicate function objects should not be mutable	Yes			Low
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	Yes			
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('Command Injection')	Yes			
CWE-119A	Improper Restriction of Operations within the Bounds of a Memory Buffer(Part A: Read)	Yes			
CWE-119B	Improper Restriction of Operations within the Bounds of a Memory Buffer(Part B: Write)	Yes			
CWE-125	Out-of-bounds Read	Yes			
CWE-190	Integer Overflow or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Wraparound				
CWE-416	Use After Free	No			
CWE-476	NULL Pointer Dereference	Yes			
CWE-787	Out-of-bounds Write	Yes			
CWE-798	Use of Hard-coded Credentials (Partial)	Yes			
DCL31-C	Declare identifiers before using them	Yes			Low
DCL36-C	Do not declare an identifier with conflicting linkage classifications	Yes			Medium
DCL38-C	Use the correct syntax when declaring a flexible array member	Yes			Low
DCL39-C	Avoid information leakage when passing a structure across a trust boundary	No			Low
DCL40-C	Do not create incompatible declarations of the same function or object	Yes			Low
DCL50-CPP	Do not define a C-style variadic function	Yes			High
DCL51-CPP	Do not declare	No			

Check ID	Check Name	Supported	Automation	Category	Severity
	or define a reserved identifier				
DCL52-CPP	Never qualify a reference type with const or volatile	Yes			Low
DCL53-CPP	Do not write syntactically ambiguous declarations	Yes			Low
DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	Yes			Low
DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary	No			Low
DCL56-CPP	Avoid cycles during initialization of static objects	Yes			Low
DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes			Low
DCL58-CPP	Do not modify the standard namespaces	Yes			High
DCL59-CPP	Do not define an unnamed namespace in a header file	Yes			Medium
DCL60-CPP	Obey the one-definition rule	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
EFFECTIVECPP_1	1. View C++ as a federation of languages	No			
EFFECTIVECPP_02	2. Do Not Use #define	Yes			
EFFECTIVECPP_03	3. Use Const whenever possible	Yes			
EFFECTIVECPP_04	4. Make sure that objects are initialized before they are used	Yes			
EFFECTIVECPP_5	5. Know what functions C++ silently writes and calls	No			
EFFECTIVECPP_6	6. Explicitly disallow the use of compiler-generated functions you do not want	No			
EFFECTIVECPP_07	7. Non-Virtual Destructors in Base Classes	Yes			
EFFECTIVECPP_08	8. Exceptions in Destructors	Yes			
EFFECTIVECPP_09	9. Virtual Call in Constructor/Destructor	Yes			
EFFECTIVECPP_10	10. Assignment Operator Return This	Yes			
EFFECTIVECPP_11	11. Assignment Operator Self Assignment	Yes			
EFFECTIVECPP_12	12. Copy all parts of an object	No			

Check ID	Check Name	Supported	Automation	Category	Severity
EFFECTIVECPP_13	13. Use objects to manage resources	No			
EFFECTIVECPP_14	14. Think carefully about copying behavior in resource-managing classes	No			
EFFECTIVECPP_15	15. Provide access to raw resources in resource-managing classes	No			
EFFECTIVECPP_16	16. Use the same form in corresponding uses of new and delete	Yes			
EFFECTIVECPP_17	17. Store newed objects in smart pointers in standalone statements	Yes			
EFFECTIVECPP_18	18. Make interfaces easy to use correctly and hard to use incorrectly	No			
EFFECTIVECPP_19	19. Treat class design as type design	No			
EFFECTIVECPP_20	20. Prefer pass-by-reference-to-const to pass by value	Yes			
EFFECTIVECPP_21	21. Dont try to	No			

Check ID	Check Name	Supported	Automation	Category	Severity
_21	return a reference when you must return an object				
EFFECTIVECPP_22	22. Datamembers should be declared private	Yes			
EFFECTIVECPP_23	23. Prefer non-member non-friend functions to member functions	No			
EFFECTIVECPP_24	24. Declare non-member functions when type conversions should apply to all parameters	No			
EFFECTIVECPP_25	25. Consider support for a non-throwing swap	No			
EFFECTIVECPP_26	26. Postpone variable definitions as long as possible	Yes			
EFFECTIVECPP_27	27. Minimize casting	Yes			
EFFECTIVECPP_28	28. Avoid returning "handles" to object internals	No			
EFFECTIVECPP_29	29. Strive for exception-safe code	No			

Check ID	Check Name	Supported	Automation	Category	Severity
EFFECTIVECPP_30	30. Understand the ins and outs of inlining	No			
EFFECTIVECPP_31	31. Minimize compilation dependencies between files	No			
EFFECTIVECPP_32	32. Make sure public inheritance models "is-a"	No			
EFFECTIVECPP_33	33. Avoid hiding inherited names	Yes			
EFFECTIVECPP_34	34. Differentiate between inheritance of interface and inheritance of implementation	No			
EFFECTIVECPP_35	35. Consider alternatives to virtual functions	Yes			
EFFECTIVECPP_36	36. Never redefine an inherited non-virtual function	Yes			
EFFECTIVECPP_37	37. Never redefine a (virtual) functions inherited default parameter value	No			
EFFECTIVECPP_38	38. Model "has-a" or "is-	No			

Check ID	Check Name	Supported	Automation	Category	Severity
	implemented-in-terms-of" through composition				
EFFECTIVECPP_39	39. Use private inheritance judiciously	No			
EFFECTIVECPP_40	40. Use multiple inheritance judiciously	No			
EFFECTIVECPP_41	41. Understand implicit interfaces and compile-time polymorphism	No			
EFFECTIVECPP_42	42. Understand the two meanings of typename	No			
EFFECTIVECPP_43	43. Know how to access names in templated base classes	No			
EFFECTIVECPP_44	44. Factor parameter-independent code out of templates	No			
EFFECTIVECPP_45	45. Use member function templates to accept "all compatible types"	No			
EFFECTIVECPP_46	46. Define non-member functions inside	No			

Check ID	Check Name	Supported	Automation	Category	Severity
	templates when type conversions are desired				
EFFECTIVECPP _47	47. Use traits classes for information about types	No			
EFFECTIVECPP _48	48. Be aware of template metaprogramm ing	No			
EFFECTIVECPP _49	49. Understand the behavior of the new- handler	No			
EFFECTIVECPP _50	50. Understand when it makes sense to replace new and delete	No			
EFFECTIVECPP _51	51. Adhere to convention when writing new and delete	No			
EFFECTIVECPP _52	52. Write placement delete if you write placement new	No			
EFFECTIVECPP _53	53. Pay attention to compiler warnings	No			
EFFECTIVECPP _54	54. Familiarize yourself with the standard library, including TR1	No			
EFFECTIVECPP _55	55. Familiarize	No			

Check ID	Check Name	Supported	Automation	Category	Severity
_55	yourself with Boost				
ERR32-C	Do not rely on indeterminate values of errno	No			Low
ERR33-C	Detect and handle standard library errors	Yes			High
ERR34-C	Detect errors when converting a string to a number	Yes			Medium
ERR50-CPP	Do not abruptly terminate the program	Yes			Low
ERR51-CPP	Handle all exceptions	Yes			Low
ERR52-CPP	Do not use setjmp() or longjmp()	Yes			Low
ERR53-CPP	Do not reference base classes or class data members in a constructor or destructor	Yes			Low
ERR54-CPP	function-try-block handler Catch handlers should order their parameter types from most derived to least derived	Yes			Medium
ERR55-CPP	Honor exception	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	specifications				
ERR56-CPP	Guarantee exception safety	No			
ERR57-CPP	Do not leak resources when handling exceptions	Yes			Low
ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes			Low
ERR59-CPP	Do not throw an exception across execution boundaries	Yes			High
ERR60-CPP	Exception objects must be nothrow copy constructible	Yes			Low
ERR61-CPP	Catch exceptions by lvalue reference	Yes			Low
ERR62-CPP	Detect errors when converting a string to a number	Yes			Medium
EXP35-C	Do not modify objects with temporary lifetime	No			Low
EXP40-C	Do not modify constant objects	No			Low
EXP42-C	Do not compare padding data	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
EXP43-C	Avoid undefined behavior when using restrict-qualified pointers	No			Medium
EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes			Medium
EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes			Low
EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes			Low
EXP53-CPP	Do not read uninitialized memory	Yes			High
EXP54-CPP	Do not access an object outside of its lifetime	Yes			High
EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	Yes			Medium
EXP56-CPP	Do not call a function with a mismatched language linkage	No			Low
EXP57-CPP	Do not cast or delete pointers to incomplete classes	Yes			Medium
EXP58-CPP	Pass an object	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	of the correct type to va_start				
EXP59-CPP	Use offsetof() on valid types and members	Yes			Medium
EXP60-CPP	Do not pass a nonstandard-layout type object across execution boundaries	No			
EXP61-CPP	A lambda object must not outlive any of its reference captured objects	Yes			High
EXP62-CPP	Do not access the bits of an object representation that are not part of the object's value representation	Yes			High
EXP63-CPP	Do not rely on the value of a moved-from object	Yes			Medium
FIO32-C	Do not perform operations on devices that are only appropriate for files	No			Medium
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	No			High

Check ID	Check Name	Supported	Automation	Category	Severity
FIO38-C	Do not copy a FILE object	Yes			Low
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes			Low
FIO51-CPP	Close files when they are no longer needed	Yes			Medium
FLP30-C	Do not use floating-point variables as loop counters	Yes			Low
FLP32-C	Prevent or detect domain and range errors in math functions	No			Medium
FLP34-C	Ensure that floating-point conversions are within range of the new type	No			Low
FLP36-C	Preserve precision when converting integral values to floating-point type	No			Low
FLP37-C	Do not use object representations to compare floating-point values	Yes			Low
HIS_01	1. Comment	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Density (COMF)				
HIS_02	2. Number of Paths (PATH)	Yes			
HIS_03	3. Number of Goto Statements (GOTO)	Yes			
HIS_04	4. Cyclomatic Complexity (v(G))	Yes			
HIS_05	5. Calling Functions (CALLING)	Yes			
HIS_06	6. Called Functions (CALLS)	Yes			
HIS_07	7. Function Parameters (PARAM)	Yes			
HIS_08	8. Number of Statements (STMT)	Yes			
HIS_09	9. Number of call levels (LEVEL)	Yes			
HIS_10	10. Number of return points (RETURN)	Yes			
HIS_11	11. Language scope (VOCF)	Yes			
HIS_12	12. Recursion (AP_CG_CYCLE)	Yes			
HIS_13	13. Statements Changed (SCHG)	Yes			
HIS_14	14. Statements Deleted (SDEL)	Yes			
HIS_15	15. New Statements	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	(SNEW)				
HIS_16	16. Stability Index (S)	Yes			
HIS_17	17. MISRA-HIS Violations (NOMV)	Yes			
HIS_18	18. MISRA-HIS Violations per Rule (NOMVPR)	Yes			
INT32-C	Ensure that operations on signed integers do not result in overflow	No			High
INT34-C	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand	No			Low
INT35-C	Use correct integer precisions	No			Low
INT36-C	Converting a pointer to integer or integer to pointer	Yes			Low
INT50-CPP	Do not cast to an out-of-range enumeration value	Yes			Medium
M0-1-1	A project shall not contain unreachable	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	code				
M0-1-2	A project shall not contain infeasible paths	Yes	Automated	Required	
M0-1-3	A project shall not contain unused variables	Yes	Automated	Required	
M0-1-4	A project shall not contain non-volatile POD variables having only one use.	Yes	Automated	Required	
M0-1-8	All functions with void return type shall have external side effect(s)	Yes	Automated	Required	
M0-1-9	There shall be no dead code	No	Automated	Required	
M0-1-10	Every defined function shall be called at least once.	Yes	Automated	Advisory	
M0-2-1	Assigning Object to an Overlapping Object (Partial)	Yes	Automated	Required	
M0-3-1	Minimization of run-time failures shall be ensured by the use of static analysis tools	Yes	Non-automated	Required	
M0-3-2	If a function generates error information,	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	then that error information shall be tested				
M0-4-1	Undocumented Use of Scaled-integer or Fixed-point Arithmetic	Yes	Non-automated	Required	
M0-4-2	Undocumented Use of Floating-point Arithmetic	Yes	Non-automated	Required	
M1-0-2	Multiple compilers shall only be used if they have a common, defined interface	No	Non-automated	Required	
M2-7-1	The character sequence /* shall not be used within a C-style comment.	Yes	Automated	Required	
M2-10-1	Different identifiers shall be typographically unambiguous	Yes	Automated	Required	
M2-13-2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes	Automated	Required	
M2-13-3	A "U" suffix shall be applied to all	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	octal or hexadecimal integer literals of unsigned type.				
M2-13-4	Literal suffixes shall be upper case	Yes	Automated	Required	
M3-1-2	Functions shall not be declared at block scope	Yes	Automated	Required	
M3-2-1	All declarations of an object or function shall have compatible types	Yes	Automated	Required	
M3-2-2	The One Definition Rule	Yes	Automated	Required	
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	Automated	Required	
M3-2-4	An identifier with external linkage shall have exactly one definition	Yes	Automated	Required	
M3-3-2	If a function has internal linkage then all redeclarations shall include the static storage class	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	specifier				
M3-4-1	Declarations at Lowest Scope	Yes	Automated	Required	
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	Automated	Required	
M3-9-3	The underlying bit representations of floating-point values shall not be used	Yes	Automated	Required	
M4-5-1	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M4-5-3	Character Operators	Yes	Automated	Required	
M4-10-1	NULL shall not be used as an integer value	Yes	Automated	Required	
M4-10-2	Literal zero (0) shall not be used as the null-pointer-constant.	Yes	Automated	Required	
M5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions	Yes	Automated	Advisory	
M5-0-3	A cvalue expression shall not be implicitly converted to a different underlying type	Yes	Automated	Required	
M5-0-4	An implicit integral conversion shall not change the signedness of the underlying type	Yes	Automated	Required	
M5-0-5	There shall be no implicit floating-integral conversions	Yes	Automated	Required	
M5-0-6	An implicit integral or floating-point	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	conversion shall not reduce the size of the underlying type				
M5-0-7	There shall be no explicit floating-integral conversions of a cvalue expression	Yes	Automated	Required	
M5-0-8	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes	Automated	Required	
M5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes	Automated	Required	
M5-0-10	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	short, the result shall be immediately cast to the underlying type of the operand				
M5-0-11	The plain char type shall only be used for the storage and use of character values	Yes	Automated	Required	
M5-0-12	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes	Automated	Required	
M5-0-14	The first operand of a conditional-operator shall have type bool	Yes	Automated	Required	
M5-0-15	Array indexing over pointer arithmetic	Yes	Automated	Required	
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	Automated	Required	
M5-0-17	Subtraction between	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-0-18	pointers shall only be applied to pointers that address elements of the same array >, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes	Automated	Required	
M5-0-20	Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes	Automated	Required	
M5-0-21	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes	Automated	Required	
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	Automated	Required	
M5-2-3	Casts from a base class to a derived class should not be performed on polymorphic	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-2-6	types A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes	Automated	Required	
M5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes	Automated	Required	
M5-2-9	Pointer to Integer Cast	Yes	Automated	Required	
M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes	Automated	Required	
M5-2-11	The comma operator, && operator and the operator shall not be overloaded	Yes	Automated	Required	
M5-2-12	Array to Pointer Decay	Yes	Automated	Required	
M5-3-1	Each operand of the ! operator, the logical && or the logical operators shall have type bool	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-3-2	Unary Minus Operator Applied to an Expression with an Unsigned Type	Yes	Automated	Required	
M5-3-3	The unary & operator shall not be overloaded	Yes	Automated	Required	
M5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes	Automated	Required	
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	Partially Automated	Required	
M5-14-1	The right hand operand of a logical &&, operators shall not contain side effects	Yes	Automated	Required	
M5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved	Yes	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-18-1	The comma operator shall not be used.	Yes	Automated	Required	
M5-19-1	Evaluation of constant unsigned integer expressions shall not lead to wrap-around	No	Automated	Required	
M6-2-1	Assignment operators shall not be used in sub-expressions	Yes	Automated	Required	
M6-2-2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality	Yes	Partially Automated	Required	
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	Yes	Automated	Required	
M6-3-1	The statement	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	forming the body of a switch, while, do ... while or for statement shall be a compound statement				
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	Automated	Required	
M6-4-2	All if and else if constructs shall be terminated with an else clause	Yes	Automated	Required	
M6-4-3	A switch statement shall be a well-formed switch statement	Yes	Automated	Required	
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	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes	Automated	Required	
M6-4-6	The final clause of a switch statement shall be the default-clause	Yes	Automated	Required	
M6-4-7	The condition of a switch statement shall not have bool type	Yes	Automated	Required	
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 >=	Yes	Automated	Required	
M6-5-3	The loop-counter shall not be modified within condition or statement	Yes	Automated	Required	
M6-5-4	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	constant for the duration of the loop				
M6-5-5	A loop-control-variable other than the loop-counter shall not be modified within condition or expression	Yes	Automated	Required	
M6-5-6	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes	Automated	Required	
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	Automated	Required	
M6-6-2	The goto statement shall jump to a label declared later in the same function body	Yes	Automated	Required	
M6-6-3	Continue Statement Used in a not Well-formed For Loop	Yes	Automated	Required	
M7-1-2	A pointer or	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified				
M7-3-1	Global Namespace Declarations	Yes	Automated	Required	
M7-3-2	The identifier main shall not be used for a function other than the global function main	Yes	Automated	Required	
M7-3-3	There shall be no unnamed namespaces in header files.	Yes	Automated	Required	
M7-3-4	Using-directives shall not be used.	Yes	Automated	Required	
M7-3-6	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes	Automated	Required	
M7-4-1	Assembly Language Code Usage	Yes	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M7-4-2	not Documented Assembler instructions shall only be introduced using the asm declaration.	Yes	Automated	Required	
M7-4-3	Assembly language shall be encapsulated and isolated.	Yes	Automated	Required	
M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes	Non-automated	Required	
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	Non-automated	Required	
M8-0-1	Single Declarations	Yes	Automated	Required	
M8-3-1	Parameters in an overriding virtual function shall either use the same	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	default arguments as the function they override, or else shall not specify any default arguments.				
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	Automated	Required	
M8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Yes	Automated	Required	
M8-5-2	Incorrect Initializer Lists	Yes	Automated	Required	
M9-3-1	Const Member Function Returning Non-Const Pointer or Reference	Yes	Automated	Required	
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	Automated	Required	
M9-6-1	When the	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented				
M9-6-4	Bit-field Length	Yes	Automated	Required	
M10-1-1	Class Derived From Virtual Bases	Yes	Automated	Advisory	
M10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes	Automated	Required	
M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes	Automated	Required	
M10-2-1	Similar Entity Names within Multiple Inheritance	Yes	Automated	Advisory	
M10-3-3	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M11-0-1	Member Data in Non-POD Class not Private	Yes	Automated	Required	
M12-1-1	An object's dynamic type shall not be used from the body of its constructor or destructor	Yes	Automated	Required	
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	Yes	Automated	Required	
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->	Yes	Automated	Required	
M15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M15-1-1	Exception Object	Yes	Automated	Required	
M15-1-2	NULL Throw	Yes	Automated	Required	
M15-1-3	Empty Throw	Yes	Automated	Required	
M15-3-1	Exceptions shall be raised only after start-up and before termination of the program	Yes	Automated	Required	
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	Yes	Automated	Required	
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	Yes	Automated	Required	
M15-3-6	Order of Catch Blocks with Derived Classes	Yes	Automated	Required	
M15-3-7	Where multiple handlers are provided in a	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last				
M16-0-1	#include Directives Not Grouped Together	Yes	Automated	Required	
M16-0-2	Macros shall only be #define'd or #undef'd in the global namespace.	Yes	Automated	Required	
M16-0-5	Function-like Macro Containing Preprocessing Directives	Yes	Automated	Required	
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 # or ##	Yes	Automated	Required	
M16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	directives, except as operands to the defined operator				
M16-0-8	Invalid Preprocessor Directives	Yes	Automated	Required	
M16-1-1	The defined preprocessor operator shall only be used in one of the two standard forms	Yes	Automated	Required	
M16-1-2	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes	Automated	Required	
M16-2-3	Include guards shall be provided	Yes	Automated	Required	
M16-3-1	There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes	Automated	Required	
M16-3-2	The # and ## operators should not be used	Yes	Automated	Advisory	
M17-0-2	The names of standard library macros	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	and objects shall not be reused				
M17-0-3	Standard Library Function Names	Yes	Automated	Required	
M17-0-5	The setjmp macro and the longjmp function shall not be used	Yes	Automated	Required	
M18-0-3	<cstdlib> Library Functions	Yes	Automated	Required	
M18-0-4	Time Handling Functions of <ctime>	Yes	Automated	Required	
M18-0-5	Unbounded Functions of <cstring>	Yes	Automated	Required	
M18-2-1	The macro offsetof shall not be used	Yes	Automated	Required	
M18-7-1	The signal handling facilities of <csignal> shall not be used	Yes	Automated	Required	
M19-3-1	The error indicator errno shall not be used	Yes	Automated	Required	
M27-0-1	The stream input/output library <stdio> shall not be used	Yes	Automated	Required	
MEM30-C	Do not access freed memory	No			High
MEM36-C	Do not modify the alignment	No			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	of objects by calling realloc()				
MEM50-CPP	Do not access freed memory	No			High
MEM51-CPP	Properly deallocate dynamically allocated resources	Yes			High
MEM52-CPP	Detect and handle memory allocation errors	Yes			High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No			High
MEM54-CPP	Provide placement new with properly aligned pointers to sufficient storage capacity	Yes			High
MEM55-CPP	Honor replacement dynamic storage management requirements	No			
MEM56-CPP	Do not store an already-owned pointer value in an unrelated smart pointer	Yes			High
MEM57-CPP	Avoid using	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	default operator new for over-aligned types				
METRIC_00	Program Unit Call Count	Yes			
METRIC_01	Program Unit Callby Count	Yes			
METRIC_02	Program Unit Comment to Code Ratio	Yes			
METRIC_03	Program Unit Cyclomatic Complexity	Yes			
METRIC_04	Program Unit Max Length	Yes			
METRIC_05	Program Unit Max Nesting Depth	Yes			
METRIC_06	Program Unit Parameters Count	Yes			
METRIC_07	Program Unit Path Count	Yes			
METRIC_08	Program Unit Statement Count	Yes			
METRIC_09	Coupling Between Object Classes	Yes			
METRIC_11	Depth of Inheritance Tree	Yes			
METRIC_12	Lack of Cohesion in Methods	Yes			
METRIC_13	Maintainability Index	Yes			
MISRA04_2.2	2.2 C99 / Comments	Yes		Required	
MISRA04_5.6	5.6 Identifier	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Reuse in Multiple C Name Spaces				
MISRA04_5.7	5.7 Identifier Reuse	Yes		Advisory	
MISRA04_8.7	8.7 Objects shall be local if only accessed from one function	Yes		Required	
MISRA04_8.12	8.12 Array Size Missing	Yes		Required	
MISRA04_14.5	14.5 No Continue Statements	Yes		Required	
MISRA04_16.4	16.4 Inconsistent Parameter Names	Yes		Required	
MISRA04_18.4	18.4 Unions	Yes		Required	
MISRA04_19.6	19.6 Preprocessor #undef	Yes		Required	
MISRA04_19.9	19.9 Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA04_19.10	19.10 In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	operand of # or ##				
MISRA04_20.1	20.1 Reserved or Standard Library Identifiers as Macros	Yes		Required	
MISRA04_20.4	20.4 Dynamic Memory Allocation	Yes		Required	
MISRA04_20.9	20.9 Including <stdio.h>	Yes		Required	
MISRA08_0-1-1	0-1-1 A project shall not contain unreachable code	Yes		Required	
MISRA08_0-1-2	0-1-2 Infeasible Paths	Yes		Required	
MISRA08_0-1-3	0-1-3 A project shall not contain unused variables	Yes		Required	
MISRA08_0-1-4	0-1-4 A project shall not contain non- volatile POD variables having only one use.	Yes		Required	
MISRA08_0-1-5	0-1-5 Unused Type Declarations	Yes		Required	
MISRA08_0-1-6	0-1-6 A project shall not contain instances of non-volatile variables being given values that are never	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	subsequently used				
MISRA08_0-1-7	0-1-7 The value returned by a function having a non-void return type that is not an overloaded operator shall always be used	Yes		Required	
MISRA08_0-1-8	0-1-8 All functions with void return type shall have external side effect(s)	Yes		Required	
MISRA08_0-1-9	0-1-9 There shall be no dead code	No		Required	
MISRA08_0-1-10	0-1-10 All defined functions called	Yes		Required	
MISRA08_0-1-11	0-1-11 Unused Parameters in Non-virtual Functions	Yes		Required	
MISRA08_0-1-12	0-1-12 There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it	Yes		Required	
MISRA08_0-2-	0-2-1 An	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	object shall not be assigned to an overlapping object				
MISRA08_0-3-1	0-3-1 Minimization of run-time failures shall be ensured by the use of static analysis tools	Yes		Document	
MISRA08_0-3-2	0-3-2 If a function generates error information, then that error information shall be tested	No		Required	
MISRA08_0-4-1	0-4-1 Use of scaled-integer or fixed-point arithmetic shall be documented	No		Document	
MISRA08_0-4-2	0-4-2 Use of floating-point arithmetic shall be documented	No		Document	
MISRA08_0-4-3	0-4-3 Floating-point implementations shall comply with a defined floating-point standard	No		Document	
MISRA08_1-0-1	1-0-1 All code shall conform to ISO/IEC	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	14882:2003 "The C++ Standard Incorporating Technical Corrigendum 1"				
MISRA08_1-0-2	1-0-2 Multiple compilers shall only be used if they have a common, defined interface	No		Document	
MISRA08_1-0-3	1-0-3 The implementation of integer division in the chosen compiler shall be determined and documented	No		Document	
MISRA08_2-2-1	2-2-1 The character set and the corresponding encoding shall be documented	No		Document	
MISRA08_2-3-1	2-3-1 Trigraphs shall not be used	Yes		Required	
MISRA08_2-5-1	2-5-1 Digraphs shall not be used	Yes		Advisory	
MISRA08_2-7-1	2-7-1 The character sequence /* shall not be used within a C-style	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	comment.				
MISRA08_2-7-2	2-7-2 Sections of code shall not be "commented out" using C-style comments	Yes		Required	
MISRA08_2-7-3	2-7-3 Sections of code should not be "commented out" using C++ comments	Yes		Advisory	
MISRA08_2-10-1	2-10-1 Different identifiers shall be typographically unambiguous	Yes		Required	
MISRA08_2-10-2	2-10-2 Shadowed Identifiers	Yes		Required	
MISRA08_2-10-3	2-10-3 A typedef name shall be a unique identifier	Yes		Required	
MISRA08_2-10-4	2-10-4 A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes		Required	
MISRA08_2-10-5	2-10-5 The identifier name of a non-member object or function with static	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	storage duration should not be reused				
MISRA08_2-10-6	2-10-6 If an identifier refers to a type, it shall not also refer to an object or a function in the same scope	No		Required	
MISRA08_2-13-1	2-13-1 Nonstandard Escape Sequences	Yes		Required	
MISRA08_2-13-2	2-13-2 Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes		Required	
MISRA08_2-13-3	2-13-3 A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes		Required	
MISRA08_2-13-4	2-13-4 Literal suffixes shall be upper case	Yes		Required	
MISRA08_2-13-5	2-13-5 Narrow and wide string literals shall not be concatenated	Yes		Required	
MISRA08_3-1-	3-1-1 It shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	be possible to include any header file in multiple translation units without violating the One Definition Rule				
MISRA08_3-1-2	3-1-2 Functions shall not be declared at block scope	Yes		Required	
MISRA08_3-2-1	3-2-1 All declarations of an object or function shall have compatible types	Yes		Required	
MISRA08_3-2-2	3-2-2 The One Definition Rule	Yes		Required	
MISRA08_3-2-3	3-2-3 A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes		Required	
MISRA08_3-2-4	3-2-4 An identifier with external linkage shall have exactly one definition	Yes		Required	
MISRA08_3-3-1	3-3-1 Objects or functions with external	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	linkage shall be declared in a header file				
MISRA08_3-3-2	3-3-2 If a function has internal linkage then all redeclarations shall include the static storage class specifier	Yes		Required	
MISRA08_3-4-1	3-4-1 Declarations at Lowest Scope	Yes		Required	
MISRA08_3-9-1	3-9-1 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		Required	
MISRA08_3-9-2	3-9-2 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA08_3-9-3	3-9-3 The underlying bit representations of floating-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	point values shall not be used				
MISRA08_4-5-1	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator	Yes		Required	
MISRA08_4-5-2	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	operators <, <=, >, >=				
MISRA08_4-5-3	4-5-3 Character Operators	Yes		Required	
MISRA08_4-10-1	4-10-1 NULL shall not be used as an integer value	Yes		Required	
MISRA08_4-10-2	4-10-2 Literal zero (0) shall not be used as the null-pointer-constant.	Yes		Required	
MISRA08_5-0-1	5-0-1 The value of an expression shall be the same under any order of evaluation that the standard permits	Yes		Required	
MISRA08_5-0-2	5-0-2 Limited dependence should be placed on C++ operator precedence rules in expressions	Yes		Advisory	
MISRA08_5-0-3	5-0-3 A cvalue expression shall not be implicitly converted to a different underlying type	Yes		Required	
MISRA08_5-0-4	5-0-4 An implicit	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	integral conversion shall not change the signedness of the underlying type				
MISRA08_5-0-5	5-0-5 There shall be no implicit floating-integral conversions	Yes		Required	
MISRA08_5-0-6	5-0-6 An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes		Required	
MISRA08_5-0-7	5-0-7 There shall be no explicit floating-integral conversions of a cvalue expression	Yes		Required	
MISRA08_5-0-8	5-0-8 An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_5-0-9	5-0-9 An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes		Required	
MISRA08_5-0-10	5-0-10 If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand	Yes		Required	
MISRA08_5-0-11	5-0-11 The plain char type shall only be used for the storage and use of character values	Yes		Required	
MISRA08_5-0-12	5-0-12 Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_5-0-13	5-0-13 The condition of an if-statement and the condition of an iteration-statement shall have type bool	No		Required	
MISRA08_5-0-14	5-0-14 The first operand of a conditional-operator shall have type bool	Yes		Required	
MISRA08_5-0-15	5-0-15 Array indexing shall be the only form of pointer arithmetic	Yes		Required	
MISRA08_5-0-16	5-0-16 A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	No		Required	
MISRA08_5-0-17	5-0-17 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA08_5-0-18	5-0-18 >, >=, <, <= shall not	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	be applied to objects of pointer type, except where they point to the same array				
MISRA08_5-0-19	5-0-19 No more than 2 levels of pointer indirection	Yes		Required	
MISRA08_5-0-20	5-0-20 Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes		Required	
MISRA08_5-0-21	5-0-21 Bitwise operators shall only be applied to operands of unsigned underlying type	Yes		Required	
MISRA08_5-2-1	5-2-1 Each operand of a logical && or shall be a postfix expression	No		Required	
MISRA08_5-2-2	5-2-2 A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast	No		Required	
MISRA08_5-2-3	5-2-3 Casts	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
3	from a base class to a derived class should not be performed on polymorphic types				
MISRA08_5-2-4	5-2-4 C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used	No		Required	
MISRA08_5-2-5	5-2-5 A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes		Required	
MISRA08_5-2-6	5-2-6 A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes		Required	
MISRA08_5-2-7	5-2-7 An object with pointer type shall not be converted to an unrelated pointer type,	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	either directly or indirectly				
MISRA08_5-2-8	5-2-8 An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes		Required	
MISRA08_5-2-9	5-2-9 Pointer to Integer Cast	Yes		Advisory	
MISRA08_5-2-10	5-2-10 The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes		Advisory	
MISRA08_5-2-11	5-2-11 The comma operator, && operator and the operator shall not be overloaded	Yes		Required	
MISRA08_5-2-12	5-2-12 Array to Pointer Decay	Yes		Required	
MISRA08_5-3-1	5-3-1 Each operand of the ! operator, the logical && or the logical operators shall have type bool	Yes		Required	
MISRA08_5-3-2	5-3-2 The unary minus operator shall not be applied	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	to an expression whose underlying type is unsigned				
MISRA08_5-3-3	5-3-3 The unary & operator shall not be overloaded	Yes		Required	
MISRA08_5-3-4	5-3-4 Evaluation of the operand to the sizeof operator shall not contain side effects	Yes		Required	
MISRA08_5-8-1	5-8-1 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		Required	
MISRA08_5-14-1	5-14-1 The right hand operand of a logical && or operator shall not contain side effects	Yes		Required	
MISRA08_5-17-1	5-17-1 The semantic equivalence between a binary operator and its	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	assignment operator form shall be preserved				
MISRA08_5-18-1	5-18-1 The comma operator shall not be used	Yes		Required	
MISRA08_5-19-1	5-19-1 Evaluation of constant unsigned integer expressions should not lead to wrap-around	No		Advisory	
MISRA08_6-2-1	6-2-1 Assignment operators shall not be used in sub-expressions	Yes		Required	
MISRA08_6-2-2	6-2-2 Floating-point expressions shall not be directly or indirectly tested for equality or inequality	Yes		Required	
MISRA08_6-2-3	6-2-3 Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the first character following the null statement is a white-space character				
MISRA08_6-3-1	6-3-1 The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes		Required	
MISRA08_6-4-1	6-4-1 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		Required	
MISRA08_6-4-2	6-4-2 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA08_6-4-3	6-4-3 A switch statement shall be a well-formed switch statement	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_6-4-4	6-4-4 A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes		Required	
MISRA08_6-4-5	6-4-5 An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes		Required	
MISRA08_6-4-6	6-4-6 The final clause of a switch statement shall be the default-clause	Yes		Required	
MISRA08_6-4-7	6-4-7 The condition of a switch statement shall not have bool type	Yes		Required	
MISRA08_6-4-8	6-4-8 Every switch statement shall have at least one case clause	Yes		Required	
MISRA08_6-5-1	6-5-1 A for loop shall contain a single loop-counter which	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not have floating-point type				
MISRA08_6-5-2	6-5-2 If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=	Yes		Required	
MISRA08_6-5-3	6-5-3 The loop-counter shall not be modified within condition or statement	Yes		Required	
MISRA08_6-5-4	6-5-4 The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes		Required	
MISRA08_6-5-5	6-5-5 A loop-control-variable other than the loop-counter shall not be modified within condition or expression	Yes		Required	
MISRA08_6-5-6	6-5-6 A loop-control-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	variable other than the loop-counter which is modified in statement shall have type bool				
MISRA08_6-6-1	6-6-1 Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes		Required	
MISRA08_6-6-2	6-6-2 The goto statement shall jump to a label declared later in the same function body	Yes		Required	
MISRA08_6-6-3	6-6-3 The continue statement shall only be used within a well-formed for loop	No		Required	
MISRA08_6-6-4	6-6-4 For any iteration statement there shall be no more than one break or goto statement used for loop termination	Yes		Required	
MISRA08_6-6-5	6-6-5 A function shall have a single	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	point of exit at the end of the function				
MISRA08_7-1-1	7-1-1 A variable which is not modified shall be const qualified	Yes		Required	
MISRA08_7-1-2	7-1-2 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		Required	
MISRA08_7-2-1	7-2-1 An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes		Required	
MISRA08_7-3-1	7-3-1 Global Namespace Declarations	Yes		Required	
MISRA08_7-3-2	7-3-2 The identifier main shall not be used for a function other than the global function main	Yes		Required	
MISRA08_7-3-3	7-3-3 There	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
3	shall be no unnamed namespaces in header files.				
MISRA08_7-3-4	7-3-4 Using-directives shall not be used.	Yes		Required	
MISRA08_7-3-5	7-3-5 Declaration and Using-Declaration Clash	Yes		Required	
MISRA08_7-3-6	7-3-6 using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes		Required	
MISRA08_7-4-1	7-4-1 All usage of assembler shall be documented	No		Document	
MISRA08_7-4-2	7-4-2 Assembler instructions shall only be introduced using the asm declaration.	Yes		Required	
MISRA08_7-4-3	7-4-3 Assembly language shall be encapsulated and isolated.	Yes		Required	
MISRA08_7-5-1	7-5-1 A	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.				
MISRA08_7-5-2	7-5-2 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		Required	
MISRA08_7-5-3	7-5-3 A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference	No		Required	
MISRA08_7-5-4	7-5-4 Functions should not call themselves, either directly or indirectly.	Yes		Advisory	
MISRA08_8-0-1	8-0-1 Single Declarations	Yes		Required	
MISRA08_8-3-1	8-3-1	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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.				
MISRA08_8-4-1	8-4-1 Functions shall not be defined using the ellipsis notation	Yes		Required	
MISRA08_8-4-2	8-4-2 Use the same identifier in definition and declaration of functions.	Yes		Required	
MISRA08_8-4-3	8-4-3 Always return a value in non-void functions	Yes		Required	
MISRA08_8-4-4	8-4-4 A function identifier shall either be used to call the function or it shall be preceded by &	Yes		Required	
MISRA08_8-5-1	8-5-1 All variables shall have a defined value before they are used	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_8-5-2	8-5-2 Incorrect Initializer Lists	Yes		Required	
MISRA08_8-5-3	8-5-3 The = construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes		Required	
MISRA08_9-3-1	9-3-1 Const Member Function Returning Non-Const Pointer or Reference	Yes		Required	
MISRA08_9-3-2	9-3-2: Method Returning Non-const Handle to Class Data	Yes		Required	
MISRA08_9-3-3	9-3-3 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		Required	
MISRA08_9-5-1	9-5-1 Unions	Yes		Required	
MISRA08_9-6-1	9-6-1 When the absolute positioning of bits representing a bit-field is required, then	No		Document	

Check ID	Check Name	Supported	Automation	Category	Severity
	the behavior and packing of bit-fields shall be documented				
MISRA08_9-6-4	9-6-4 Named signed bit-field length	Yes		Required	
MISRA08_10-1-1	10-1-1 Classes should not be derived from virtual bases	Yes		Advisory	
MISRA08_10-1-2	10-1-2 A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes		Required	
MISRA08_10-1-3	10-1-3 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA08_10-2-1	10-2-1 All accessible entity names within a multiple inheritance hierarchy should be unique	No		Advisory	
MISRA08_10-3-1	10-3-1 Virtual Function Redefinition	Yes		Required	
MISRA08_10-3-2	10-3-2 Each overriding virtual function	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be declared with the virtual keyword.				
MISRA08_10-3-3	10-3-3 Pure Virtual Overriding Non-pure	Yes		Required	
MISRA08_11-0-1	11-0-1 Non-private Member Data	Yes		Required	
MISRA08_12-1-1	12-1-1 An object's dynamic type shall not be used from the body of its constructor or destructor	Yes		Required	
MISRA08_12-1-2	12-1-2 Explicitly call all immediate and virtual base classes	Yes		Advisory	
MISRA08_12-1-3	12-1-3 All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes		Required	
MISRA08_12-8-1	12-8-1 Side Effects in Copy Constructors	Yes		Required	
MISRA08_12-8-2	12-8-2 The copy assignment operator shall be declared	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	protected or private in an abstract class				
MISRA08_14-5-1	14-5-1 A non-generic function shall only be declared in a namespace that is not an associated namespace	No		Required	
MISRA08_14-5-2	14-5-2 A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter	Yes		Required	
MISRA08_14-5-3	14-5-3 A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter	Yes		Required	
MISRA08_14-6-1	14-6-1 In a class template with a dependent base, any name that may be found in that dependent	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	base shall be referred to using a qualified-id or this->				
MISRA08_14-6-2	14-6-2 The function or operator chosen by overload resolution shall resolve to a function declared previously in the translation unit	No		Required	
MISRA08_14-7-1	14-7-1 Templates Not Instantiated	Yes		Required	
MISRA08_14-7-2	14-7-2 For any given template specialization, an explicit instantiation of the template with the template-arguments used in the specialization shall not render the program ill-formed	No		Required	
MISRA08_14-7-3	14-7-3 All partial and explicit specializations for a template shall be declared in the same file as	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the declaration of their primary template				
MISRA08_14-8-1	Overloaded function templates shall not be explicitly specialized	Yes		Required	
MISRA08_14-8-2	The viable function set for a function call should either contain no function specializations, or only contain function specializations	No		Advisory	
MISRA08_15-0-1	Exceptions shall only be used for error handling	No		Document	
MISRA08_15-0-2	An exception object should not have pointer type	Yes		Advisory	
MISRA08_15-0-3	Control shall not be transferred into a try or catch block using a goto or a switch statement	No		Required	
MISRA08_15-1-1	The assignment-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	expression of a throw statement shall not itself cause an exception to be thrown				
MISRA08_15-1-2	15-1-2 NULL Throw	Yes		Required	
MISRA08_15-1-3	15-1-3 Empty Throw	Yes		Required	
MISRA08_15-3-1	15-3-1 Exceptions shall be raised only after start-up and before termination of the program	Yes		Required	
MISRA08_15-3-2	15-3-2 There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes		Advisory	
MISRA08_15-3-3	15-3-3 Members in function-try-blocks in constructors or destructors	Yes		Required	
MISRA08_15-3-4	15-3-4 Each exception explicitly thrown in the code shall have a handler of a compatible type in all call	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	paths that could lead to that point				
MISRA08_15-3-6	15-3-6 Order of Catch Blocks with Derived Classes	Yes		Required	
MISRA08_15-3-7	15-3-7 Catch-All Statement Before Last	Yes		Required	
MISRA08_15-4-1	15-4-1 Inconsistent Exception-Specification	Yes		Required	
MISRA08_15-5-1	15-5-1 A class destructor shall not exit with an exception	Yes		Required	
MISRA08_15-5-2	15-5-2 Exceptions thrown shall be the type indicated by the function	Yes		Required	
MISRA08_15-5-3	15-5-3 The terminate() function shall not be called implicitly	No		Required	
MISRA08_16-0-1	16-0-1 #include directives in a file shall only be preceded by other preprocessor directives or comments	Yes		Required	
MISRA08_16-0-2	16-0-2 Macros shall only be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	#define'd or #undef'd in the global namespace				
MISRA08_16-0-3	Preprocessor #undef	Yes		Required	
MISRA08_16-0-4	Function-like macros shall not be defined	Yes		Required	
MISRA08_16-0-5	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA08_16-0-6	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 # or ##	Yes		Required	
MISRA08_16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the defined operator				
MISRA08_16-0-8	16-0-8 Invalid Preprocessor Directives	Yes		Required	
MISRA08_16-1-1	16-1-1 The defined preprocessor operator shall only be used in one of the two standard forms	Yes		Required	
MISRA08_16-1-2	16-1-2 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes		Required	
MISRA08_16-2-1	16-2-1 The pre-processor shall only be used for file inclusion and include guards	Yes		Required	
MISRA08_16-2-2	16-2-2 Invalid Macro Usage	Yes		Required	
MISRA08_16-2-3	16-2-3 Include guards shall be provided	Yes		Required	
MISRA08_16-2-4	16-2-4 The ', ", /* or // characters shall not occur in a header file name	Yes		Required	
MISRA08_16-2-5	16-2-5 The	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
-5	backslash character should not occur in a header file name				
MISRA08_16-2-6	16-2-6 The #include directive shall be followed by either a <filename> or "filename" sequence	Yes		Required	
MISRA08_16-3-1	16-3-1 There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes		Required	
MISRA08_16-3-2	16-3-2 The # and ## operators should not be used	Yes		Advisory	
MISRA08_16-6-1	16-6-1 All uses of the #pragma directive shall be documented	No		Document	
MISRA08_17-0-1	17-0-1 Reserved or Standard Library Identifiers as Macros	Yes		Required	
MISRA08_17-0-2	17-0-2 The names of standard library macros	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	and objects shall not be reused				
MISRA08_17-0-3	17-0-3 Standard Library Function Names	Yes		Required	
MISRA08_17-0-4	17-0-4 All library code shall conform to MISRA C++	No		Document	
MISRA08_17-0-5	17-0-5 The setjmp macro and the longjmp function shall not be used	Yes		Required	
MISRA08_18-0-1	18-0-1 The C library shall not be used	Yes		Required	
MISRA08_18-0-2	18-0-2 The library functions atof, atoi and atol from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-3	18-0-3 The library functions abort, exit, getenv and system from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-4	18-0-4 The time handling functions of library <ctime> shall not be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_18-0-5	used 18-0-5 Unbounded Functions of <cstring>	Yes		Required	
MISRA08_18-2-1	18-2-1 The macro offsetof shall not be used.	Yes		Required	
MISRA08_18-4-1	18-4-1 Dynamic Memory Allocation	Yes		Required	
MISRA08_18-7-1	18-7-1 The signal handling facilities of <csignal> shall not be used	Yes		Required	
MISRA08_19-3-1	19-3-1 The error indicator "errno" shall not be used.	Yes		Required	
MISRA08_27-0-1	27-0-1 The stream input/output library <cstdio> shall not be used	Yes		Required	
MISRA12_1.2	1.2 Language extensions should not be used (Partial)	Yes		Advisory	
MISRA12_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA12_2.4	2.4 Unused Tag Declarations	Yes		Advisory	
MISRA12_2.5	2.5 Unused Macros	Yes		Advisory	
MISRA12_2.7	2.7 There	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should be no unused parameters in functions				
MISRA12_3.1	3.1 The character sequences /* and // shall not be used within a comment	Yes		Required	
MISRA12_3.2	3.2 Line-Splicing in // Comments	Yes		Required	
MISRA12_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA12_5.2	5.2 Identifiers declared in the same scope and name space shall be distinct	Yes		Required	
MISRA12_5.4	5.4 Indistinct Macro Identifiers	Yes		Required	
MISRA12_5.5	5.5 Same Identifier and Macro Name	Yes		Required	
MISRA12_5.8	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes		Required	
MISRA12_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA12_6.2	6.2 Signed single-bit named bit-fields	Yes		Required	
MISRA12_7.2	7.2 Unsigned Integer Literal Without Suffix	Yes		Required	
MISRA12_7.3	7.3 Lowercase L Suffix	Yes		Required	
MISRA12_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes		Required	
MISRA12_8.2	8.2 Use Named Parameters and Prototype Form	Yes		Required	
MISRA12_8.9	8.9 Objects shall be local if only accessed from one function	Yes		Advisory	
MISRA12_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA12_8.13	8.13 A pointer should point to a const-qualified type whenever possible	Yes		Advisory	
MISRA12_9.1	9.1 The value	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	of an object with automatic storage duration shall not be read before it has been set				
MISRA12_9.2	9.2 The initializer for an aggregate or union shall be enclosed in braces	Yes		Required	
MISRA12_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA12_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA12_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA12_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA12_10.2	10.2 Expressions of essentially character type shall not be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	used inappropriately in addition and subtraction operations				
MISRA12_10.3	10.3 The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category	Yes		Required	
MISRA12_10.4	10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes		Required	
MISRA12_10.6	10.6 The value of a composite expression shall not be assigned to an object with wider essential type	Yes		Required	
MISRA12_10.7	10.7 Implicit Casts of Operations	Yes		Required	
MISRA12_10.8	10.8 The value of a composite expression	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be cast to a different essential type category or a wider essential type				
MISRA12_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA12_11.2	11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes		Required	
MISRA12_11.3	11.3 A cast shall not be performed between a pointer to object type and a pointer to a different object type	Yes		Required	
MISRA12_11.4	11.4 A conversion should not be performed between a pointer to object and an integer type	Yes		Advisory	
MISRA12_11.5	11.5 A	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	conversion should not be performed from pointer to void into pointer to object				
MISRA12_11.8	11.8 A cast shall not remove any const or volatile qualification from the type pointed to by a pointer	Yes		Required	
MISRA12_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA12_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA12_12.4	12.4 Evaluation of constant expressions should not lead to unsigned integer wrap-around	Yes		Advisory	
MISRA12_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA12_13.3	13.3 A full expression	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator				
MISRA12_13.4	13.4 The result of an assignment operator should not be used	Yes		Advisory	
MISRA12_13.5	13.5 The right hand operand of a logical && or operator shall not contain persistent side effects	Yes		Required	
MISRA12_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA12_14.2	14.2 A for loop shall be well-formed	Yes		Required	
MISRA12_14.4	14.4 The controlling expression of an if statement and the controlling expression of an iteration-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	statement shall have essentially Boolean type				
MISRA12_15.2	15.2 The goto statement shall jump to a label declared later in the same function	Yes		Required	
MISRA12_15.3	15.3 Goto Into or Between Blocks	Yes		Required	
MISRA12_16.5	16.5 A default label shall appear as either the first or the last switch label of a switch statement	Yes		Required	
MISRA12_16.7	16.7 Switch Boolean	Yes		Required	
MISRA12_17.5	17.5 The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	Yes		Required	
MISRA12_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	Yes		Advisory	
MISRA12_18.5	18.5	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Declarations should contain no more than two levels of pointer nesting				
MISRA12_18.7	18.7 Flexible Array Members	Yes		Required	
MISRA12_18.8	18.8 Variable-length arrays shall not be used	Yes		Required	
MISRA12_19.1	19.1 An object shall not be assigned or copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA12_19.2	19.2 Unions	Yes		Advisory	
MISRA12_20.2	20.2 Invalid Header Name	Yes		Required	
MISRA12_20.4	20.4 Keyword Macros	Yes		Required	
MISRA12_20.5	20.5 Preprocessor #undef	Yes		Advisory	
MISRA12_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Required	
MISRA12_20.7	20.7 Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	Yes		Required	
MISRA12_20.8	20.8 The	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1				
MISRA12_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation	Yes		Required	
MISRA12_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes		Required	
MISRA12_21.2	21.2 Reserved Identifiers or Macros	Yes		Required	
MISRA12_21.5	21.5 Standard Header signal.h	Yes		Required	
MISRA12_21.6	21.6 C Standard Library I/O	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_22.1	Functions 22.1 All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes		Required	
MISRA12_22.2	22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function	Yes		Required	
MISRA12_22.4	22.4 There shall be no attempt to write to a stream which has been opened as read-only	Yes		Mandatory	
MISRA12_22.5	22.5 Dereference of FILE Pointer	Yes		Mandatory	
MISRA12_22.6	22.6 The value of a pointer to a FILE shall not be used after the associated stream has been closed (Partial)	Yes		Mandatory	
MISRA12_DIR_4.2	Directive 4.2 All usage of assembly	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	language should be documented				
MISRA12_DIR_4.7	Directive 4.7 If a function returns error information, then that error information shall be tested	Yes		Required	
MISRA12_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA12_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they are interchangeable	Yes		Advisory	
MISRA12_DIR_4.12	Directive 4.12 Dynamic Memory Allocation	Yes		Required	
MISRA12_DIR_4.13	Directive 4.13 Functions which are designed to provide operations on a resource	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should be called in an appropriate sequence (Partial)				
MISRA23_0.0.1	0.0.1 A function shall not contain unreachable statements	Yes		Required	
MISRA23_0.0.2	0.0.2 Controlling expressions should not be invariant	No		Required	
MISRA23_0.1.2	0.1.2 The value returned by a function shall be used	Yes		Required	
MISRA23_0.2.1	0.2.1 Variables with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.2	0.2.2 A named function parameter shall be used at least once	Yes		Required	
MISRA23_0.2.3	0.2.3 Types with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.4	0.2.4 Functions with limited visibility should be used at least once	Yes		Advisory	
MISRA23_1.2	1.2 Language extensions	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should not be used (Partial)				
MISRA23_1.4	1.4 Bounds-checking Interfaces	Yes		Required	
MISRA23_1.5	1.5 Storage Class Specifiers Not At Beginning	Yes		Required	
MISRA23_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA23_2.4	2.4 Unused Tag Declarations	Yes		Advisory	
MISRA23_2.5	2.5 Unused Macros	Yes		Advisory	
MISRA23_2.7	2.7 A function should not contain unused parameters	Yes		Advisory	
MISRA23_2.8	2.8 Unused Objects	Yes		Advisory	
MISRA23_3.1	3.1 The character sequences /* and // shall not be used within a comment	Yes		Required	
MISRA23_3.2	3.2 Line-Splicing in // Comments	Yes		Required	
MISRA23_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA23_4.1.1	4.1.1 A program shall conform to ISO/IEC 14882:2017	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	(C++17)				
MISRA23_4.1.2	4.1.2 Deprecated features should not be used	No		Advisory	
MISRA23_4.1.3	4.1.3 There shall be no occurrence of undefined or critical unspecified behaviour	No		Required	
MISRA23_5.0.1	5.0.1 Trigraph-like sequences should not be used	Yes		Required	
MISRA23_5.2	5.2 Identifiers declared in the same scope and name space shall be distinct	Yes		Required	
MISRA23_5.4	5.4 Indistinct Macro Identifiers	Yes		Required	
MISRA23_5.5	5.5 Same Identifier and Macro Name	Yes		Required	
MISRA23_5.7.1	5.7.1 The character sequence /* shall not be used within a C-style comment	Yes		Required	
MISRA23_5.7.2	5.7.2 Sections of code should not be "commented out"	Yes		Advisory	
MISRA23_5.7.3	5.7.3 Line-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
3	Splicing in // Comments				
MISRA23_5.8	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes		Required	
MISRA23_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	
MISRA23_5.10.1	5.10.1 User-defined identifiers shall have an appropriate form	Yes		Required	
MISRA23_5.13.1	5.13.1 Within character literals and non raw-string literals, \ shall only be used to form a defined escape sequence or universal character name	Yes		Required	
MISRA23_5.13.2	5.13.2 Octal escape sequences, hexadecimal escape sequences and universal character	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	names shall be terminated				
MISRA23_5.13.3	5.13.3 Octal constants shall not be used	Yes		Required	
MISRA23_5.13.4	5.13.4 Unsigned integer literals shall be appropriately suffixed	Yes		Required	
MISRA23_5.13.5	5.13.5 The lowercase form of L shall not be used as the first character in a literal suffix	Yes		Required	
MISRA23_5.13.6	5.13.6 An integer-literal of type long long shall not use a single L or l in any suffix	Yes		Required	
MISRA23_5.13.7	5.13.7 String literals with different encoding prefixes shall not be concatenated	Yes		Required	
MISRA23_6.0.1	6.0.1 Block scope declarations shall not be visually ambiguous	Yes		Required	
MISRA23_6.0.2	6.0.2 When an array with external linkage is	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	declared, its size should be explicitly specified				
MISRA23_6.0.3	6.0.3 Global Namespace Declarations	Yes		Advisory	
MISRA23_6.0.4	6.0.4 The identifier main shall not be used for a function other than the global function main	Yes		Required	
MISRA23_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA23_6.2	6.2 Signed single-bit named bit-fields	Yes		Required	
MISRA23_6.2.1	6.2.1 The one-definition rule shall not be violated	Yes		Required	
MISRA23_6.2.2	6.2.2 All declarations of a variable or function shall have the same type	Yes		Required	
MISRA23_6.2.3	6.2.3 The source code used to implement an entity shall appear only once	Yes		Required	
MISRA23_6.2.4	6.2.4 A header file shall not	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	contain definitions of functions or objects that are non-inline and have external linkage				
MISRA23_6.3	6.3 A bit field shall not be declared as a member of a union	Yes		Required	
MISRA23_6.4.1	6.4.1 A variable declared in an inner scope shall not hide a variable declared in an outer scope	Yes		Required	
MISRA23_6.4.2	6.4.2 Derived classes shall not conceal functions that are inherited from their bases	Yes		Required	
MISRA23_6.4.3	6.4.3 Names from a dependent base shall be prefixed with this->, qualified with a using declaration, or qualified with the base name	Yes		Required	
MISRA23_6.5.1	6.5.1 A function or object with	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	external linkage should be introduced in a header file				
MISRA23_6.5.2	6.5.2 Internal linkage should be specified appropriately	Yes		Advisory	
MISRA23_6.7.1	6.7.1 Local variables shall not have static storage duration	Yes		Required	
MISRA23_6.7.2	6.7.2 Global variables shall not be used	Yes		Required	
MISRA23_6.8.1	6.8.1 An object shall not be accessed outside of its lifetime	Yes		Required	
MISRA23_6.8.2	6.8.2 A function must not return a reference or a pointer to a local variable with automatic storage duration	Yes		Mandatory	
MISRA23_6.8.3	6.8.3 An assignment operator shall not assign the address of an object with automatic storage duration to an object with a greater lifetime	Yes		Required	
MISRA23_6.8.4	6.8.4 Member	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
4	functions returning references to their object should be ref-qualified appropriately				
MISRA23_6.9.1	6.9.1 The same type aliases shall be used in all declarations of the same entity	Yes		Required	
MISRA23_6.9.2	6.9.2 The names of the standard signed integer types and standard unsigned integer types should not be used	Yes		Advisory	
MISRA23_7.0.1	7.0.1 There shall be no conversion from type bool	Yes		Required	
MISRA23_7.0.2	7.0.2 There shall be no conversion to type bool	Yes		Required	
MISRA23_7.0.3	7.0.3 The numerical value of a character shall not be used	Yes		Required	
MISRA23_7.0.4	7.0.4 The operands of bitwise operators and shift operators	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be appropriate				
MISRA23_7.0.5	7.0.5 Integral promotion and the usual arithmetic conversions shall not change the signedness or the type category of an operand	Yes		Required	
MISRA23_7.0.6	7.0.6 Assignment between numeric types shall be appropriate	Yes		Required	
MISRA23_7.2	7.2 Unsigned Integer Literal Without Suffix	Yes		Required	
MISRA23_7.3	7.3 Lowercase L Suffix	Yes		Required	
MISRA23_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes		Required	
MISRA23_7.5	7.5 The argument of an integer constant macro shall have an appropriate form	Yes		Mandatory	
MISRA23_7.6	7.6 Small	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	Integer Constant Macros				
MISRA23_7.11.1	7.11.1 nullptr shall be the only form of the null-pointer-constant	Yes		Required	
MISRA23_7.11.2	7.11.2 Array to Pointer Decay	Yes		Required	
MISRA23_7.11.3	7.11.3 A conversion from function type to pointer-to-function type shall only occur in appropriate contexts	Yes		Required	
MISRA23_8.0.1	8.0.1 Parentheses should be used to make the meaning of an expression appropriately explicit	Yes		Required	
MISRA23_8.1.1	8.1.1 A non-transient lambda shall not implicitly capture this	Yes		Required	
MISRA23_8.1.2	8.1.2 Variables should be captured explicitly in a non-transient lambda	Yes		Advisory	
MISRA23_8.2	8.2 Use Named Parameters	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	and Prototype Form				
MISRA23_8.2.1	8.2.1 A virtual base class shall only be cast to a derived class by means of dynamic_cast	Yes		Required	
MISRA23_8.2.2	8.2.2 C-style casts and functional notation casts shall not be used	Yes		Required	
MISRA23_8.2.3	8.2.3 A cast shall not remove any const or volatile qualification from the type accessed via a pointer or by reference	Yes		Required	
MISRA23_8.2.4	8.2.4 Casts shall not be performed between a pointer to function and any other type	Yes		Required	
MISRA23_8.2.5	8.2.5 reinterpret_cast shall not be used	Yes		Required	
MISRA23_8.2.6	8.2.6 An object with integral, enumerated, or pointer to void type shall not be cast to a	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	pointer type				
MISRA23_8.2.7	8.2.7 Pointer to Integer Cast	Yes		Advisory	
MISRA23_8.2.8	8.2.8 An object of pointer type shall not be cast to an integral type other than std::uintptr_t or std::intptr_t	Yes		Required	
MISRA23_8.2.9	8.2.9 The operand to typeid shall not be an expression of polymorphic class type	Yes		Required	
MISRA23_8.2.10	8.2.10 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA23_8.2.11	8.2.11 An argument passed via ellipsis shall have an appropriate type	Yes		Required	
MISRA23_8.3.1	8.3.1 The built-in unary - operator should not be applied to an expression of unsigned type	Yes		Advisory	
MISRA23_8.3.2	8.3.2 The built-in unary + operator should not be	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	used				
MISRA23_8.7.1	8.7.1 Pointer arithmetic shall not form an invalid pointer	Yes		Required	
MISRA23_8.7.2	8.7.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA23_8.9	8.9 An object should be declared at block scope if its identifier only appears in a single function	Yes		Advisory	
MISRA23_8.9.1	8.9.1 The built-in relational operators >, >=, < and <= shall not be applied to objects of pointer type, except where they point to the same array	Yes		Required	
MISRA23_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA23_8.13	8.13 A pointer should point to a const-qualified type whenever possible	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_8.14.1	8.14.1 The right-hand operand of a logical && or operator should not contain persistent side effects	Yes		Advisory	
MISRA23_8.15	8.15 All declarations of an object with an explicit alignment specification shall specify the same alignment	Yes		Required	
MISRA23_8.16	8.16 The alignment specification of zero should not appear in an object declaration	Yes		Advisory	
MISRA23_8.17	8.17 At most one explicit alignment specifier should appear in an object declaration	Yes		Advisory	
MISRA23_8.18.1	8.18.1 An object or subobject must not be copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA23_8.18.2	8.18.2 The result of an assignment	No		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	operator should not be used				
MISRA23_8.19.1	8.19.1 The comma operator shall not be used.	Yes		Advisory	
MISRA23_8.20.1	8.20.1 An unsigned arithmetic operation with constant operands should not wrap	Yes		Advisory	
MISRA23_9.1	9.1 The value of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
MISRA23_9.2	9.2 The initializer for an aggregate or union shall be enclosed in braces	Yes		Required	
MISRA23_9.2.1	9.2.1 An explicit type conversion shall not be an expression statement	Yes		Required	
MISRA23_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA23_9.3.1	9.3.1 The body of an iteration-statement or a	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	selection-statement shall be a compound-statement				
MISRA23_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA23_9.4.1	9.4.1 All if ... else if constructs shall be terminated with an else statement	Yes		Required	
MISRA23_9.4.2	9.4.2 The structure of a switch statement shall be appropriate	Yes		Required	
MISRA23_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA23_9.5.1	9.5.1 Legacy for statements should be simple	Yes		Advisory	
MISRA23_9.5.2	9.5.2 A for-range-initializer shall contain at most one function call	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_9.6	9.6 An initializer using chained designators shall not contain initializers without designators	Yes		Required	
MISRA23_9.6.1	9.6.1 The goto statement should not be used	Yes		Advisory	
MISRA23_9.6.2	9.6.2 A goto statement shall reference a label in a surrounding block	Yes		Required	
MISRA23_9.6.3	9.6.3 The goto statement shall jump to a label declared later in the function body	Yes		Required	
MISRA23_9.6.4	9.6.4 A function declared with the [[noreturn]] attribute shall not return	Yes		Required	
MISRA23_9.6.5	9.6.5 A function with non-void return type shall return a value on all paths	Yes		Required	
MISRA23_9.7	9.7 Atomic objects shall be	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	appropriately initialized before being accessed				
MISRA23_10.0.1	10.0.1 A declaration should not declare more than one variable or member variable	Yes		Advisory	
MISRA23_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA23_10.1.1	10.1.1 The target type of a pointer or lvalue reference parameter should be const-qualified appropriately	Yes		Advisory	
MISRA23_10.1.2	10.1.2 The volatile qualifier shall be used appropriately	Yes		Required	
MISRA23_10.2	10.2 Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations	Yes		Required	
MISRA23_10.2.1	10.2.1 An	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	enumeration shall be defined with an explicit underlying type				
MISRA23_10.2.2	10.2.2 Unscoped enumerations should not be declared	Yes		Advisory	
MISRA23_10.2.3	10.2.3 The numeric value of an unscoped enumeration with no fixed underlying type shall not be used	Yes		Required	
MISRA23_10.3	10.3 The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category	Yes		Required	
MISRA23_10.3.1	10.3.1 There should be no unnamed namespaces in header files	Yes		Advisory	
MISRA23_10.4	10.4 Both operands of an operator in which the usual	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	arithmetic conversions are performed shall have the same essential type category				
MISRA23_10.4.1	10.4.1 The asm declaration shall not be used	Yes		Required	
MISRA23_10.6	10.6 The value of a composite expression shall not be assigned to an object with wider essential type	Yes		Required	
MISRA23_10.7	10.7 Implicit Casts of Operations	Yes		Required	
MISRA23_10.8	10.8 The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes		Required	
MISRA23_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA23_11.2	11.2 Conversions shall not be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	performed between a pointer to an incomplete type and any other type				
MISRA23_11.3	11.3 A conversion shall not be performed between a pointer to object type and a pointer to a different object type	Yes		Required	
MISRA23_11.3.1	11.3.1 Variables of array type should not be declared	Yes		Advisory	
MISRA23_11.3.2	11.3.2 The declaration of an object should contain no more than two levels of pointer indirection	Yes		Advisory	
MISRA23_11.4	11.4 A conversion should not be performed between a pointer to object and an integer type	Yes		Advisory	
MISRA23_11.5	11.5 A conversion should not be performed from pointer to void into	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	pointer to object				
MISRA23_11.6.1	11.6.1 All variables should be initialized	Yes		Advisory	
MISRA23_11.6.2	11.6.2 The value of an object must not be read before it has been set	Yes		Mandatory	
MISRA23_11.6.3	11.6.3 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA23_11.8	11.8 A conversion shall not remove any const, volatile or _Atomic qualification from the type pointed to by a pointer	Yes		Required	
MISRA23_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA23_12.2.1	12.2.1 Bit-fields should not be declared	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_12.2.2	12.2.2 A bit-field shall have an appropriate type	Yes		Required	
MISRA23_12.2.3	12.2.3 A named bit-field with signed integer type shall not have a length of one bit	Yes		Required	
MISRA23_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA23_12.3.1	12.3.1 Unions	Yes		Required	
MISRA23_12.5	17.4 Size of Array Parameter	Yes		Mandatory	
MISRA23_12.6	12.6 Structure and union members of atomic objects shall not be directly accessed	Yes		Required	
MISRA23_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA23_13.1.1	13.1.1 Classes should not be inherited virtually	Yes		Advisory	
MISRA23_13.1.2	13.1.2 An accessible base class shall not be both virtual and non-virtual	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_13.3	in the same hierarchy 13.3 A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator	Yes		Advisory	
MISRA23_13.3.1	13.3.1 User-declared member functions shall use the virtual, override and final specifiers appropriately	Yes		Required	
MISRA23_13.3.2	13.3.2 Parameters in an overriding virtual function shall not specify different default arguments	Yes		Required	
MISRA23_13.3.3	13.3.3 The parameters in all declarations or overrides of a function shall either be unnamed or have identical names	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_13.3.4	13.3.4 A comparison of a potentially virtual pointer to member function shall only be with nullptr	Yes		Required	
MISRA23_13.4	13.4 The result of an assignment operator should not be used	Yes		Advisory	
MISRA23_13.5	13.5 The right hand operand of a logical && or operator shall not contain persistent side effects	Yes		Required	
MISRA23_13.13	13.13 Character Function Misuse	Yes		Mandatory	
MISRA23_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA23_14.1.1	14.1.1 Non-static data members should be either all private or all public	Yes		Advisory	
MISRA23_14.2	14.2 A for loop shall be well-formed	Yes		Required	
MISRA23_14.4	14.4 The	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type				
MISRA23_15.0.2	15.0.2 User-provided copy and move member functions of a class should have appropriate signatures	Yes		Advisory	
MISRA23_15.1.1	15.1.1 An object's dynamic type shall not be used from within its constructor or destructor	Yes		Required	
MISRA23_15.1.2	15.1.2 All constructors of a class should explicitly initialize all of its virtual base classes and immediate base classes	Yes		Advisory	
MISRA23_15.1.3	15.1.3 Conversion operators and constructors that are callable with a	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	single argument shall be explicit				
MISRA23_15.1.4	15.1.4 All direct, non-static data members of a class should be initialized before the class object is accessible	Yes		Required	
MISRA23_15.1.5	15.1.5 A class shall only define an initializer-list constructor when it is the only constructor	Yes		Required	
MISRA23_15.2	15.2 The goto statement shall jump to a label declared later in the same function	Yes		Required	
MISRA23_15.3	15.3 Goto Into or Between Blocks	Yes		Required	
MISRA23_15.8.1	15.8.1 User-provided copy assignment operators and move assignment operators shall handle self-assignment	Yes		Required	
MISRA23_16.5	16.5 A default label shall appear as either the first	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	or the last switch label of a switch statement				
MISRA23_16.5.1	16.5.1 The logical AND and logical OR operators shall not be overloaded	Yes		Required	
MISRA23_16.5.2	16.5.2 The address-of operator shall not be overloaded	Yes		Required	
MISRA23_16.6.1	16.6.1 Symmetrical operators should only be implemented as non-member functions	Yes		Advisory	
MISRA23_16.7	16.7 Switch Boolean	Yes		Required	
MISRA23_17.5	17.5 The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	Yes		Required	
MISRA23_17.8.1	17.8.1 Function templates shall not be explicitly specialized	Yes		Required	
MISRA23_17.9	17.9 Invalid	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_17.10	_Noreturn 17.10 A function declared with a _Noreturn function specifier shall have void return type	Yes		Required	
MISRA23_17.11	17.11 A function that never returns should be declared with a _Noreturn function specifier	Yes		Advisory	
MISRA23_17.12	17.12 A function identifier should only be used with either a preceding &, or with a parenthesized parameter list	Yes		Required	
MISRA23_17.13	17.13 A function type shall not be type qualified	Yes		Required	
MISRA23_18.1.1	18.1.1 An exception object shall not have pointer type	Yes		Required	
MISRA23_18.1.2	18.1.2 An empty throw shall only occur within the compound-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	statement of a catch handler				
MISRA23_18.3.1	18.3.1 There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes		Advisory	
MISRA23_18.3.2	18.3.2 An exception of class type shall be caught by const reference or reference	Yes		Required	
MISRA23_18.3.3	18.3.3 Handlers for a function-try-block of a constructor or destructor shall not refer to non-static members from their class or its bases	Yes		Required	
MISRA23_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	Yes		Advisory	
MISRA23_18.4.1	18.4.1 Exception-unfriendly functions shall be noexcept	Yes		Required	
MISRA23_18.5	18.5	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Declarations should contain no more than two levels of pointer nesting				
MISRA23_18.5.2	18.5.2 Program-terminating functions should not be used	Yes		Advisory	
MISRA23_18.7	18.7 Flexible Array Members	Yes		Required	
MISRA23_18.8	18.8 Variable-length arrays shall not be used	Yes		Required	
MISRA23_18.9	18.9 An object with temporary lifetime shall not undergo array-to-pointer conversion	Yes		Required	
MISRA23_18.10	18.10 Pointer to Variable-length Array	Yes		Mandatory	
MISRA23_19.0.1	19.0.1 A line whose first token is # shall be a valid preprocessing directive	Yes		Required	
MISRA23_19.0.2	19.0.2 Function-like macros shall not be defined	Yes		Required	
MISRA23_19.0.3	19.0.3 #include directives should only be preceded by preprocessor	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	directives or comments				
MISRA23_19.0.4	19.0.4 #undef should only be used for macros defined previously in the same file	Yes		Advisory	
MISRA23_19.1	19.1 An object shall not be assigned or copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA23_19.1.1	19.1.1 The defined preprocessor operator shall be used appropriately	Yes		Required	
MISRA23_19.1.2	19.1.2 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes		Required	
MISRA23_19.2	19.2 Unions	Yes		Advisory	
MISRA23_19.2.1	19.2.1 Precautions shall be taken in order to prevent the contents of a header file being included	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	more than once				
MISRA23_19.2.2	19.2.2 The #include directive shall be followed by either a <filename> or "filename" sequence	Yes		Required	
MISRA23_19.2.3	19.2.3 The ' or " or \ characters and the /* or // character sequences shall not occur in a header file name	Yes		Required	
MISRA23_19.3.1	19.3.1 The # and ## operators should not be used	Yes		Advisory	
MISRA23_19.3.2	19.3.2 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Yes		Required	
MISRA23_19.3.3	19.3.3 The argument to a mixed-use macro parameter shall not be subject to further expansion	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_19.3.4	19.3.4 Parentheses shall be used to ensure macro arguments are expanded appropriately	Yes		Required	
MISRA23_19.3.5	19.3.5 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Advisory	
MISRA23_19.6.1	19.6.1 The #pragma directive and the _Pragma operator should not be used	Yes		Advisory	
MISRA23_20.2	20.2 Invalid Header Name	Yes		Required	
MISRA23_20.4	20.4 Keyword Macros	Yes		Required	
MISRA23_20.5	20.5 Preprocessor #undef	Yes		Advisory	
MISRA23_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Required	
MISRA23_20.7	20.7 Expressions resulting from the expansion of macro parameters	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be enclosed in parentheses				
MISRA23_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
MISRA23_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation	Yes		Required	
MISRA23_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes		Required	
MISRA23_21.2	21.2 Reserved Identifiers or Macros	Yes		Required	
MISRA23_21.2.1	21.2.1 The library	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	functions atof, atoi, atol and atoll from library <cstdlib> shall not be used				
MISRA23_21.2.2	21.2.2 The string handling functions from <cstring>, <cstdlib>, <wchar> and <inttypes> shall not be used	Yes		Required	
MISRA23_21.2.3	21.2.3 The library function system from <cstdlib> shall not be used	Yes		Required	
MISRA23_21.2.4	21.2.4 The macro offsetof shall not be used	Yes		Required	
MISRA23_21.5	21.5 Standard Header signal.h	Yes		Required	
MISRA23_21.6	21.6 C Standard Library I/O Functions	Yes		Required	
MISRA23_21.6.1	21.6.1 Dynamic memory should not be used	Yes		Advisory	
MISRA23_21.6.2	21.6.2 Dynamic memory shall be managed automatically	Yes		Required	
MISRA23_21.6.3	21.6.3 Advanced memory	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	management shall not be used				
MISRA23_21.6.4	21.6.4 If a project defines either a sized or unsized version of a global operator delete, then both shall be defined	Yes		Required	
MISRA23_21.6.5	21.6.5 A pointer to an incomplete class type shall not be deleted	Yes		Required	
MISRA23_21.8	21.8 The Standard Library termination functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.10.1	21.10.1 The features of <cstdlib> shall not be used	Yes		Required	
MISRA23_21.10.2	21.10.2 The standard header file <setjmp> shall not be used	Yes		Required	
MISRA23_21.10.3	21.10.3 The facilities provided by the standard header file <signal> shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	not be used				
MISRA23_21.1 3	21.13 Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF	Yes		Mandatory	
MISRA23_21.1 4	21.14 The Standard Library function memcmp shall not be used to compare null terminated strings	Yes		Required	
MISRA23_21.1 5	21.15 Incompatible Pointers	Yes		Required	
MISRA23_21.1 6	21.16 Invalid Memory Compare	Yes			
MISRA23_21.1 7	21.17 Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters	Yes		Mandatory	
MISRA23_21.1 8	21.18 Out of Bounds with string.h	Yes			
MISRA23_21.1	21.19 The	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
9	pointers returned by the Standard Library functions localeconv, getenv, setlocale or, strerror shall only be used as if they have pointer to const-qualified type				
MISRA23_21.20	21.20 The pointer returned by the C++ Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function	Yes		Mandatory	
MISRA23_21.22	21.22 All operand arguments to any type-generic macros declared in <tgmath.h> shall have an appropriate	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_21.2 3	essential type 21.23 All operand arguments to any multi-argument type-generic macros declared in <tgmath.h> shall have the same standard type	Yes		Required	
MISRA23_21.2 4	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.2 5	21.25 Atomic Operations with Inconsistent Order	Yes		Required	
MISRA23_21.2 6	21.26 The Standard Library function mtx_timedlock () shall only be invoked on mutex objects of appropriate mutex type	Yes		Required	
MISRA23_22.1	22.1 All resources obtained dynamically by means of Standard Library	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	functions shall be explicitly released				
MISRA23_22.2	22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function	Yes		Mandatory	
MISRA23_22.3	22.3 The same file shall not be open for read and write access at the same time on different streams	Yes		Required	
MISRA23_22.3.1	22.3.1 The assert macro shall not be used with a constant-expression	Yes		Required	
MISRA23_22.4	22.4 There shall be no attempt to write to a stream which has been opened as read-only	Yes		Mandatory	
MISRA23_22.4.1	22.4.1 The literal value zero shall be the only value assigned to errno	Yes		Required	
MISRA23_22.5	22.5 Dereference of	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	FILE Pointer				
MISRA23_22.6	22.6 Use of Closed FILE Pointers (Partial)	Yes		Mandatory	
MISRA23_22.7	22.7 EOF Macro Misuse	Yes		Required	
MISRA23_22.8	22.8 The value of errno shall be set to zero prior to a call to an errno-setting-function	Yes		Required	
MISRA23_22.9	22.9 The value of errno shall be tested against zero after calling an errno-setting-function	Yes		Required	
MISRA23_22.10	22.10 The value of errno shall only be tested when the last function to be called was an errno-setting-function	Yes		Required	
MISRA23_22.11	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA23_22.12	22.12 Thread Data Misuse	Yes		Mandatory	
MISRA23_22.13	22.13 Thread	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
3	objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration				
MISRA23_22.1 5	22.15 Thread synchronization objects and thread-specific storage pointers shall not be destroyed until after all threads accessing them have terminated	Yes		Required	
MISRA23_22.1 6	22.16 All mutex objects locked by a thread shall be explicitly unlocked by the same thread	Yes		Required	
MISRA23_22.1 7	22.17 No thread shall unlock a mutex or call <code>cnd_wait()</code> or <code>cnd_timedwait()</code> for a mutex it has not locked before	Yes		Required	
MISRA23_22.1 8	22.18 Non-recursive mutexes shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	not be recursively locked				
MISRA23_22.1 9	22.19 Mutexes with One Condition Variable	Yes		Required	
MISRA23_23.1 1.1	23.11.1 The raw pointer constructors of std::shared_ptr and std::weak_ptr should not be used	Yes		Advisory	
MISRA23_24.5 .1	24.5.1 The character handling functions from <cctype> and <cwctype> shall not be used	Yes		Required	
MISRA23_24.5 .2	24.5.2 The C++ Standard Library functions memcpy, memmove and memcmp from <cstring> shall not be used	Yes		Required	
MISRA23_25.5 .1	25.5.1 The setlocale and std::locale::global functions shall not be called	Yes		Required	
MISRA23_25.5 .2	25.5.2 The pointers returned by the C++	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Standard Library functions localeconv, getenv, setlocale or strerror must only be used as if they have pointer to const-qualified type				
MISRA23_25.5.3	25.5.3 The pointer returned by the C++ Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function	Yes		Mandatory	
MISRA23_26.3.1	26.3.1 std::vector should not be specialized with bool	Yes		Advisory	
MISRA23_28.3.1	28.3.1 Predicates shall not have persistent side effects	Yes		Required	
MISRA23_28.6.1	28.6.1 The argument to	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	std::move shall be a non-const lvalue				
MISRA23_28.6.2	28.6.2 Forwarding references and std::forward shall be used together	Yes		Required	
MISRA23_28.6.3	28.6.3 An object shall not be used while in a potentially moved-from state	Yes		Required	
MISRA23_28.6.4	28.6.4 The result of std::remove, std::remove_if, std::unique and empty shall be used	Yes		Required	
MISRA23_30.0.1	30.0.1 The C Library input/output functions shall not be used	Yes		Required	
MISRA23_30.0.2	30.0.2 Reads and writes on the same file stream shall be separated by a positioning operation	Yes		Required	
MISRA23_DIR_4.2	Directive 4.2 All usage of assembly language should be documented	Yes		Advisory	
MISRA23_DIR_4.7	Directive 4.7 If a function	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	returns error information, then that error information shall be tested				
MISRA23_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA23_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they are interchangeable	Yes		Advisory	
MISRA23_DIR_4.12	Directive 4.12 Dynamic Memory Allocation	Yes		Required	
MISRA23_DIR_4.13	Directive 4.13 Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_DIR_5.3	Directive 5.3 There shall be no dynamic thread creation	Yes		Required	
MISRA25_1.4	1.4 Emergent language features shall not be used	Yes		Required	
MISRA25_1.5	1.5 Obsolescent language features shall not be used	Yes		Required	
MISRA25_6.2	6.2 Single-bit named bit-fields shall not be of a signed type	Yes		Required	
MISRA25_7.6	7.6 The small integer variants of the minimum-width integer constant macros shall not be used	Yes		Required	
MISRA25_8.5	8.5 An external object or function shall be declared once in one and only one file	Yes		Required	
MISRA25_8.14	8.14 The restrict type qualifier shall not be used	Yes		Required	
MISRA25_8.18	8.18 There shall be no tentative definitions in a header file	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_8.19	8.19 There should be no external declarations in a source file	Yes		Advisory	
MISRA25_9.1	9.1 The value of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
MISRA25_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA25_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA25_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA25_11.8	11.8 A conversion shall not remove any const, volatile or _Atomic qualification from the type	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	pointed to by a pointer				
MISRA25_11.11	11.11 Pointers shall not be implicitly compared to NULL	Yes		Required	
MISRA25_12.2	12.2 The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand	Yes		Required	
MISRA25_13.5	13.5 The right hand operand of a logical && or operator shall not contain persistent side effects	Yes		Required	
MISRA25_17.12	17.12 A function identifier should only be used with either a preceding &, or with a parenthesized parameter list	Yes		Advisory	
MISRA25_18.7	18.7 Flexible array members shall not be declared	Yes		Required	
MISRA25_18.10	18.10 Pointers to variably-	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	modified array types shall not be used				
MISRA25_19.1	19.1 An object shall not be assigned or copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA25_19.3	19.3 A union member shall not be read unless it has been previously set	Yes		Required	
MISRA25_20.7	20.7 Expressions resulting from the expansion of macro parameters shall be appropriately delimited	Yes		Required	
MISRA25_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
MISRA25_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_20.12	evaluation 20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes		Required	
MISRA25_21.17	21.17 Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters	Yes		Mandatory	
MISRA25_21.19	21.19 The pointers returned by the Standard Library functions localeconv, getenv, setlocale or strerror shall only be used as if they have pointer to const-qualified	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_21.2 0	type 21.20 The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function	Yes		Mandatory	
MISRA25_21.2 1	21.21 The Standard Library function system of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.2 4	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.2 5	21.25 All memory synchronization operations shall be executed in sequentially consistent order	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_22.5	22.5 A pointer to a FILE object shall not be dereferenced	Yes		Mandatory	
MISRA25_22.7	22.7 The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF	Yes		Required	
MISRA25_22.1 1	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA25_22.1 2	22.12 Thread objects, thread synchronization objects, and thread-specific storage pointers shall only be accessed by the appropriate Standard Library functions	Yes		Mandatory	
MISRA25_22.1 4	22.14 Thread synchronization objects shall be initialized	No		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	before being accessed				
MISRA25_22.19	22.19 A condition variable shall be associated with at most one mutex object	Yes		Required	
MISRA25_22.20	22.20 Thread-specific storage pointers shall be created before being accessed	No		Mandatory	
MISRA25_23.1	23.1 A generic selection should only be expanded from a macro	Yes		Advisory	
MISRA25_23.3	23.3 A generic selection should contain at least one non-default association	Yes		Advisory	
MISRA25_23.7	23.7 A generic selection that is expanded from a macro should evaluate its argument only once	Yes		Advisory	
MISRA25_23.8	23.8 A default association shall appear as either the first or the last association of a generic	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	selection				
MISRA25_DIR_1.2	Directive 1.2 The use of language extensions should be minimized (Partial)	Yes		Advisory	
MISRA25_DIR_4.7	Directive 4.7 If a function returns error information, then that error information shall be tested	Yes		Required	
MSC30-C	Do not use the rand() function for generating pseudorandom numbers	Yes			Medium
MSC37-C	Ensure that control never reaches the end of a non-void function	Yes			High
MSC41-C	Never hard code sensitive information	No			High
MSC50-CPP	Do not use the rand() function for generating pseudorandom numbers	Yes			Medium
MSC51-CPP	Ensure your random number generator is properly seeded	Yes			Medium
MSC52-CPP	Value-returning functions must	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	return a value from all exit paths				
MSC53-CPP	Do not return from a function declared [[noreturn]]	Yes			Medium
MSC54-CPP	A signal handler must be a plain old function	Yes			High
OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes			Low
OOP51-CPP	Do not slice derived objects	Yes			Low
OOP52-CPP	Do not delete a polymorphic object without a virtual destructor	Yes			Low
OOP53-CPP	Write constructor member initializers in the canonical order	Yes			Medium
OOP54-CPP	Gracefully handle self-copy assignment	Yes			Low
OOP55-CPP	Do not use pointer-to-member operators to access nonexistent members	No			High
OOP56-CPP	Honor	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
OOP57-CPP	replacement handler requirements Prefer special member functions and overloaded operators to C Standard Library functions	Yes			High
OOP58-CPP	Copy operations must not mutate the source object	Yes			Low
POS44-C	Do not use signals to terminate threads	Yes			Low
POS47-C	Do not use threads that can be canceled asynchronously	Yes			Medium
POS48-C	Do not unlock or destroy another POSIX thread's mutex	Yes			Medium
POS49-C	When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed	No			Medium
POS50-C	Declare	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	objects shared between POSIX threads with appropriate storage durations				
POS51-C	Avoid deadlock with POSIX threads by locking in predefined order	Yes			Low
POWER_OF_TEN_01	1 Simple Control Flow	Yes			
POWER_OF_TEN_02	2 Loops with Fixed Limits	Yes			
POWER_OF_TEN_03	3 No Dynamic Memory Allocation	Yes			
POWER_OF_TEN_04	4 Short Functions	Yes			
POWER_OF_TEN_05	5 Use Assertion Statements	Yes			
POWER_OF_TEN_06	6 Declarations at Lowest Scope	Yes			
POWER_OF_TEN_07_A	7A Check Parameters and Return Values - Ignored Return Values	Yes			
POWER_OF_TEN_07_B	7B Check Parameters and Return Values - Unchecked Parameters and Return Values	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
POWER_OF_TEN_08	8 Limit Preprocessor Usage	Yes			
POWER_OF_TEN_09_A	9A Restrict Pointer Usage - Multiple Dereferences	Yes			
POWER_OF_TEN_09_B	9B Restrict Pointer Usage - Other	Yes			
POWER_OF_TEN_10	10 All Compiler Warnings	Yes			
PRE30-C	Do not create a universal character name through concatenation	Yes			Low
RECOMMENDED_00	Commented Out Code	Yes			
RECOMMENDED_01	Definitions in Header Files	Yes			
RECOMMENDED_02	Files too long	Yes			
RECOMMENDED_03	Floating Equality Test	Yes			
RECOMMENDED_04	Functions Too Long	Yes			
RECOMMENDED_05	Functions shall not be declared implicitly	Yes			
RECOMMENDED_06	Goto Statements	Yes			
RECOMMENDED_07	Macros shall not be #define'd or #undef'd within a block	Yes			
RECOMMENDED_08	Magic Numbers	Yes			
RECOMMENDED	Nested	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
D_09	Comments				
RECOMMENDED	Overly	Yes			
D_10	Complex Functions				
RECOMMENDED	Trigraphs shall	Yes			
D_11	not be used				
RECOMMENDED	Unreachable	Yes			
D_12	Code				
RECOMMENDED	Unused	Yes			
D_13	Functions				
RECOMMENDED	Unused C and	Yes			
D_14	C++ Local Variables				
RECOMMENDED	Unused Static	Yes			
D_15	Globals				
RECOMMENDED	Variables	Yes			
D_16	should be commented				
RECOMMENDED	Upper limit	Yes			
D_17	shall not be modified within the bounds of the loop				
RECOMMENDED	Comments	Yes			
D_19	Indicating Future Fixes				
RECOMMENDED	Duplicate Code	Yes			
D_20					
SIG35-C	Do not return from a computational exception signal handler	No			Low
STI_FRIENDS	Unnecessary Friends	Yes		Recommended	High
STI_SPECIAL_MEMBER_FUNCTIONS	Special Member Functions	Yes		Recommended	High
STI_UNUSED	Unused Entities	Yes		Recommended	High
STR34-C	Cast	No			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	characters to unsigned char before converting to larger integer sizes				
STR50-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes			High
STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes			High
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes			High
STR53-CPP	Range check element access	Yes			High
WIN30-C	Properly pair allocation and deallocation functions	Yes			Low