	All Rules	Advisory Rules	Document Rules	Mandatory Rules	Recommended Rules	Required Rules
Understand % Coverage	94%	95%	8%	95%	100%	92%
Understand Coverage	2,679	146	1	36	3	792
Total Rules	2,846	153	12	38	3	862

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	Yes	Automated	Required	

	member function shall be used				
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 functions that override it	Yes	Automated	Required	
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	No	Automated	Required	

	Language Standard				
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 process shall be set in compliance with project policies	No	Non-automated	Required	
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	No	Non-automated	Required	

	safety-related software, the tool confidence level (TCL) shall be determined				
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 the C++ Language Standard basic source character set shall be used in the source code	Yes	Automated	Required	
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	Yes	Automated	Required	

	\ shall not occur as a last character of a C++ comment				
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 outside of the file	No	Non-automated	Required	
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 enumeration name shall not be hidden by a variable, function or enumerator declaration in	Yes	Automated	Required	

	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	
A3-1-1	It shall be possible to include any header file in multiple	Yes	Automated	Required	

	translation units without violating the One Definition Rule				
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	Yes	Partially Automated	Required	

	intended to be inlined (2) it is 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	Yes	Automated	Required	

	enum or enum class 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 <, <=, >, >=				
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	Yes	Automated	Required	

	shall be bool				
A5-0-3	No more than 2 levels of pointer indirection	Yes	Automated	Required	
A5-0-4	Pointer arithmetic shall not be used with pointers to non-final classes	Yes	Automated	Required	
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 defined inside another lambda expression	Yes	Automated	Advisory	
A5-1-9	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_ca	Yes	Automated	Required	

	st shall not be used				
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 beyond its range Part B	Yes	Automated	Required	
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	Yes	Automated	Required	

	division or remainder operators shall not be equal to zero				
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 function shall only be tested for equality with null-pointer-constant	Yes	Automated	Required	
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 not be used	Yes	Automated	Required	
A6-5-2	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-	Yes	Automated	Advisory	

	counter initialization and modification				
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 the type that is a typedef or a using name	Yes	Automated	Required	
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	Yes	Automated	Required	

	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				
A7-1-6	The typedef specifier shall not be used	Yes	Automated	Required	
A7-1-7	Each expression statement and identifier declaration shall be placed on a separate line	Yes	Automated	Required	
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,	Yes	Automated	Required	

	structure, or enumeration shall not be declared in the definition of its type				
A7-2-1	An expression with enum underlying type shall only have values correspondin g 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 shall be initialized	Yes	Automated	Required	
A7-2-5	Enumerations should be used to represent sets of	No	Non-automated	Advisory	

	related named constants				
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 shall be used if the return type depends on	Yes	Automated	Required	

	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	Yes	Automated	Required	

	passed by 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	
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"	Yes	Non-automated	Required	

	Relationship				
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	
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	Yes	Non-automated	Advisory	

	interface classes				
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 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	Yes	Automated	Required	
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	Yes	Automated	Required	

	others of these five special member functions shall be declared as well.				
A12-0-2	Bitwise operations and operations that assume data representation in memory shall not be performed on objects.	Yes	Automated	Required	
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	Yes	Automated	Required	

	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.				
A12-1-4	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes	Automated	Required	
A12-1-5	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	No	Automated	Required	

	constructors from the base class shall use inheriting constructors				
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 initializers.	Yes	Automated	Required	
A12-7-1	If the behavior of a user-defined special member function is identical to implicitly defined special member	Yes	Automated	Required	

	function, then it shall be defined =default or be left undefined.				
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 and base classes using copy semantics.	Yes	Automated	Required	

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 operators	Yes	Automated	Required	

	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	Yes	Automated	Required	

	defined explicit				
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	Yes	Non-automated	Advisory	

	suitable for this template				
A14-5-1	A template constructor shall not participate in overload resolution for a single argument of the enclosing class type	Yes	Automated	Required	
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	Yes	Automated	Required	

	by the template				
A14-7-2	Template specialization shall be declared in the same file as the primary template	Yes	Automated	Required	
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	No	Non-automated	Required	

	function shall be considered				
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	
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	No	Non-automated	Required	

	time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions processing				
A15-1-1	Only instances of types derived from std::exception should be thrown	Yes	Automated	Advisory	
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	Yes	Partially Automated	Required	

	delete them				
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 deallocate the object's resources and it shall throw an exception	Yes	Partially Automated	Required	
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	Yes	Partially Automated	Required	

	Main Function				
A15-3-4	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines	Yes	Non-automated	Required	
A15-3-5	A class type exception shall be caught by reference or const reference	Yes	Automated	Required	
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	Yes	Automated	Required	

	noexcept(<trueondition>), then it shall not exit with an exception				
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 overrider	Yes	Automated	Required	
A15-4-4	A declaration of non-throwing function shall contain noexcept specification	Yes	Automated	Required	
A15-4-5	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations and for all its	Yes	Automated	Required	

	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 std::terminate() function shall not be called implicitly	Yes	Automated	Required	
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	Yes	Automated	Required	

	include directives (slow)				
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 to standard namespaces	Yes	Automated	Required	
A18-0-1	The C library	Yes	Automated	Required	

	facilities shall only be accessed through C++ library headers				
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 <clocale> (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 shall have a	Yes	Automated	Required	

	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	
A18-5-6	An analysis	No	Non-	Required	

	shall be performed to analyze the failure modes of dynamic memory management		automated		
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	Yes	Automated	Required	

	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				
A18-9-3	The std::move shall not be used on objects declared const or const&	Yes	Automated	Required	
A18-9-4	An argument to std::forward 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 std::unique_ptr shall be used to represent exclusive ownership	Yes	Automated	Required	
A20-8-3	A std::shared_p	Yes	Automated	Required	

	tr shall be used to represent shared ownership				
A20-8-4	A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required	Yes	Automated	Required	
A20-8-5	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	Yes	Automated	Required	

	char				
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 Function Objects Copied Incorrectly	Yes	Automated	Required	
A25-4-1	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 std::rand()	Yes	Automated	Required	
A26-5-2	Random number engines shall	Yes	Automated	Required	

	not be default-initialized				
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 a file stream shall not be used without an intervening flush or positioning call	Yes	Automated	Required	
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_CYCLE)	Yes			
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 predefined order	No			Low
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	Yes			Low

	released on exceptional conditions				
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 thread	Yes			Low
CPP_A000	Assembler instructions shall only be introduced using the asm declaration.	Yes			
CPP_A001	Assembly language	Yes			

	shall be encapsulated and isolated.				
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 Documented	Yes			
CPP_A009	User-defined copy and move assignment operators	Yes			

	should use user-defined no-throw swap function.				
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 the ref-qualifier &.	Yes			

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,	Yes			

	Unsigned, or Signed Bit-fields				
CPP_B001	(Fuzzy parser) Bit-fields shall only be declared with an appropriate type	Yes			
CPP_B002	Enum Bit-fields	Yes			
CPP_B003	The underlying bit representations 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	Yes			

	reference				
CPP_C005	Members in function-try-blocks in constructors or destructors	Yes			
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	Yes			

	Constructor				
CPP_C013	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes			
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-	Yes			

	statement				
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-control-variable other than the loop-counter shall not be modified within condition or expression	Yes			
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	Yes			

	the duration of the loop				
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			
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	Yes			

	most closely-enclosing compound statement is the body of a switch statement				
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-empty switch-clause	Yes			
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	Yes			
CPP_C038	Before preprocessing, a null statement shall only	Yes			

	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				
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 is modified in statement shall have type bool	Yes			
CPP_C041	Do statements should not be used	Yes			
CPP_C042	For-init-statement and expression should not perform actions other	Yes			

	than loop-counter initialization and modification				
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 order	Yes			
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	Yes	Non-	Required	

	Control to a Try or Catch Block Using Goto or Switch Statement		automated		
CPP_C049	Class Constructor with Parameter Type std::initializer_list	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 cannot finish object initialization, then it shall deallocate the object's resources	Yes			

	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 statement shall not have	Yes			

	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 checkedvalues	Yes			
CPP_CF010	Legacy for statements should be	Yes			

	simple				
CPP_CF011	Goto Into or Between Blocks	Yes			
CPP_CM000	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_BUGPRONE_ASS	Assert Side Effect	Yes			High

ERT_SIDE_EFFECT					
CPP_CT_BUGPRONE_BRANCH_CLONE	Branch Clone	Yes			High
CPP_CT_BUGPRONE_COPY_CONSTRUCTOR_INIT	Copy Constructor Init	Yes			High
CPP_CT_BUGPRONE_INFINITE_LOOP	Infinte Loop	Yes			High
CPP_CT_BUGPRONE_MACRO_REPEATED_SIDE_EFFECTS	Macro Side Effects	Yes			High
CPP_CT_BUGPRONE_NOT_NULL_TERMINATED_RESULT	Missing Null Terminator	Yes			High
CPP_CT_BUGPRONE_REDUNDANT_BRANCH_CONDITION	Redundant Condition	Yes			High
CPP_CT_CPPCOREGUIDELINES_AVOID_NON_CONST_GLOBAL_VARIABLES	Non-const Globals	Yes			
CPP_CT_MISC_CONST_CORRECTNESS	Missing Const for Locals and Parameters	Yes			
CPP_CT_MODERNIZE_USE_DEFAULT_MEMBER_INIT	Default Member Init	Yes			

CPP_CT_MODERNIZE_USE_EQUALS_DEFAULT	Default Member Function	Yes			
CPP_CT_MODERNIZE_USE_EQUALS_DELETE	Delete Member Function	Yes			
CPP_CT_MODERNIZE_USE_NULLPTR	Null Pointer Keyword	Yes			
CPP_CT_READABILITY_DELETE_NULL_POINTER	Delete Null Pointer	Yes			High
CPP_CT_READABILITY_MAKE_MEMBER_FUNCTION_CONST	Non-const Member Functions	Yes			
CPP_CT_READABILITY_NON_CONST_PARAMETER	Non-const Parameters	Yes			
CPP_CT_READABILITY_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	Yes			

	destructor				
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 hierarchy	Yes			
CPP_D006	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-	Yes			

	Declaration Clash				
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 one and only one file	Yes			
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	Yes			

	the || operator shall not be overloaded				
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			
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	Yes			

	Set Data Ptr to 0				
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 keyword shall not be used	Yes			
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 storage class specifier	Yes			
CPP_D043	Static and thread-local objects shall	Yes			

	be constant-initialized				
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 call, (2) to declare	Yes			

	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 Classes	Yes			
CPP_D054	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	Yes			

	these five special 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	Yes			

	using NSDMI instead.				
CPP_D064	All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes			
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	Yes			

	user-defined special member function is identical to implicitly defined special member function, then it shall be defined =default or be left undefined.				
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	Yes			

	exception. A noexcept exception specification shall be added to these functions as appropriate				
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 variable shall not be set to a value where it is being declared	Yes			
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	Yes			

	tentative definitions in a header file				
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 be limited	Yes			
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	Yes			

	constants that are used as argument or return variables shall be placed in an include file				
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			
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	Yes			

	versions should be defined.				
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			
CPP_DD017	Avoid cycles during initialization of static objects	Yes			
CPP_DD018	Obey the one-definition	Yes			

	rule				
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 correspondin g uses of new and delete	Yes			
CPP_DD024	Postpone variable definitions as long as possible	Yes			
CPP_DD025	Avoid hiding inherited names	Yes			
CPP_DD026	Never redefine an	Yes			

	inherited non-virtual function				
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	Yes			

	shall not hide a variable declared in an outer scope				
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	Yes			

	indicated by the function				
CPP_E010	Inconsistent Exception-Specification	Yes			
CPP_E011	No "errno" allowed	Yes			
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	Yes			

	operator				
CPP_E016	Character Operators	Yes			
CPP_E017	Code Slicing Should Not Occur	Yes			
CPP_E018	Expressions with type enum or enum class 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 <, <=, >, >=	Yes			
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	Yes			

	Hexadecimal Sequences				
CPP_E022	Nonstandard Escape Sequences	Yes			
CPP_E023	Expression uses operand of side-effect more than once	Yes			
CPP_E024	Signed operands to modulus or division operator	Yes			
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	Yes			

	literals of unsigned type.				
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			
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 typeid	Yes			
CPP_E040	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	Yes			

	shall have type bool				
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 hand operand.	Yes			
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	Yes			

	not contain side effects				
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 contain persistent side effects	Yes			
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 std::exception should be	Yes			

	thrown				
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 exception shall be caught by reference or const reference	Yes			
CPP_E063	Catch-all (ellipsis and std::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions	Yes			

	that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines				
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 ordering predicate	Yes			
CPP_E069	Assignment in SubExpressions	Yes			
CPP_E070	Boolean operators	Yes			
CPP_E071	Expressions of essentially character type shall not	Yes			

	be used inappropriately in addition and subtraction operations				
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 shall not be assigned to an object with a narrower essential type	Yes			

	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 object type	Yes			
CPP_E081	A conversion should not be	Yes			

	performed between a pointer to object and an integer type				
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			
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_EH000	Program shall not be abruptly terminated	Yes			
CPP_EH001	The std::terminate() function shall not be called implicitly	Yes			
CPP_EH002	Library objects shall not generate error messages directly	Yes			
CPP_EH003	Destructors should not throw exceptions	Yes			
CPP_EH004	Exceptions should be caught only by reference	Yes			
CPP_EH005	A declaration	Yes			

	of non-throwing function shall contain noexcept specification				
CPP_EH006	If a function is declared to be noexcept, noexcept(true) or noexcept(<true condition>), 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			
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	Yes			

	shall not be 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	Yes			

	virtual function shall either use the same default arguments as 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 correspondin g 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	Yes			

	be used				
CPP_F025	All functions with void return type shall have external side effect(s)	Yes			
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	Yes			

	together				
CPP_F030	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined	Yes			
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	Yes			

	should be inlined.				
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			
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-	Yes			

	const raw pointers or references to private or protected data owned by the class				
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 functions with identical parameter types and noexcept	Yes			
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	Yes			

	participate in overload resolution for a single argument of the enclosing class type				
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			
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 overrider	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 overriding methods, should be kept constant	Yes			
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-	Yes			

	length argument lists should not be used				
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., a += b, a-=b etc.), the resulting behavior should remain consistent with the expected outcome of the operator.	Yes			
CPP_F063	Member function arguments should not share the same name	Yes			

	as class state variables				
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 correspondin g to a parameter declared to have an array type shall have an appropriate number of elements	Yes			
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			
CPP_H000	The #include directive shall be followed	Yes			

	by either a <filename> or "filename" sequence				
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	Yes			

	Rule				
CPP_H007	Unnecessary #Includes	Yes			
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	Yes			

	identical to names of library header files				
CPP_H014	All public header files shall be capable of being included by a C++ file as well as a C file	Yes			
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	Yes			

	referencing should be used, when appropriate, over direct inclusion when documenting dependencies within a header file.				
CPP_H020	The standard header file <tgmath.h> shall not be used	Yes			
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	Yes			

	functions with external linkage shall be unique				
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 space shall be distinct	Yes			
CPP_I008	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	Yes			

	shall be a unique identifier				
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 be hidden by a variable, function or enumerator declaration in the same scope	Yes			
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	Yes			

	non-member object with external or internal linkage should not be reused				
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 identifier	Yes			
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 Functions of <cstring>	Yes			
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	Yes			

	Function Names				
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 generator is properly seeded	Yes			
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	Yes			

	implicitly converted to const_iterator				
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 used	Yes			
CPP_L026	Alternate input and output operations on a file stream shall not be used without an	Yes			

	intervening flush or positioning call				
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 <clocale> (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			
CPP_L034	Use of the iostream library is preferred	Yes			

	over stdio.h				
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 Functions	Yes			
CPP_L045	Standard	Yes			

	Header signal.h				
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			

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	Yes			

	be used to deallocate arrays that have been allocated with the new[] operator				
CPP_M012	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	Yes			

	correspondin g delete[] operator should be provided.				
CPP_M016	Overloaded new operator should not hide the global new operator	Yes			
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	Yes			

	durations				
CPP_M022	All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes			
CPP_M023	A block of memory shall only be freed if it was allocated by means of a Standard Library function	Yes			
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:	Yes			

	Macros				
CPP_N006	Naming Convention: Namespaces	Yes			
CPP_N007	Naming Convention: Parameters	Yes			
CPP_N008	Naming Convention: Structs	Yes			
CPP_N009	Naming Convention: Typedefs	Yes			
CPP_N010	Naming Convention: Unions	Yes			
CPP_N011	Naming Convention: Variables	Yes			
CPP_N012	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes			
CPP_N013	Naming Convention: Header File Names	Yes			
CPP_N014	Naming Convention: Implementation File Names	Yes			
CPP_N015	Implementation files, that are defined	Yes			

	locally in the project, should have a file name extension of ".cpp"				
CPP_N016	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters	Yes			
CPP_N017	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 data being pointed to	Yes			
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	Yes			

	std::shared_ptr if ownership sharing is not required				
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			
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	Yes			

	be preceded by other preprocessor directives or comments				
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			
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			
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,	Yes			

	the asterisk shall be placed with the variable				
CPP_P034	Invalid Header Name	Yes			
CPP_P035	std::make_shared 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			
CPP_P039	String literals shall not be assigned to non-constant pointers	Yes			
CPP_P040	Only nullptr literal shall be	Yes			

	used as the null-pointer-constant				
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 array of objects shall	Yes			

	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	Yes			

	Integer Cast				
CPP_P052	A parameter shall be passed by reference if it can't be NULL	Yes			
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 be cast to a pointer to a derived class by means of dynamic_cast	Yes			
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	Yes			

	between a pointer to function and any other type				
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 they point to the same array	Yes			
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 only be used in one of the two standard forms	Yes			
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 Parentheses in Macros	Yes			
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	Yes			

	have an appropriate form				
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 preprocessing directives shall be #define'd before evaluation	Yes			
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	Yes			

	operand to these operators				
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) not be a base of another struct or class, (4) not inherit from another struct or class	Yes			
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_POINTERS	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 ";" with the exception of the null statement	Yes			
CPP_ST002	Equal signs should be aligned when they occur in a series of	Yes			

	assignment operators				
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 the function level or at the module level.	Yes			
CPP_ST008	Blank lines should be	Yes			

	used to separate distinct algorithmic parts				
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	Yes			

	characters				
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	Yes			

	level				
CPP_ST021	Placement of braces for constructs shall be consistent within a module	Yes			
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	Yes			

	function internal data				
CPP_ST028	Data type naming conventions	Yes			
CPP_ST029	Immutable data naming conventions	Yes			
CPP_ST030	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	Yes			

	as an unsigned char				
CPP_T002	The std::vector<bool> specialization shall not be used	Yes			
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	Yes			

	integral conversion shall not change the signedness of the underlying type				
CPP_T009	There shall be no implicit floating-integral conversions	Yes			
CPP_T010	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	Yes			

	conversion shall not change the signedness of the underlying type of a cvalue expression				
CPP_T014	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			
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	Yes			

	values				
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 derived class should not be performed on polymorphic types	Yes			
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	Yes			

	Pointer Decay				
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 correspondin g to the enumerators of the enumeration	Yes			
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 && shall always be moved from.	Yes			
CPP_T034	"forward" parameters declared as T && shall always be forwarded.	Yes			
CPP_T035	"in" parameters for "cheap to copy" types	Yes			

	shall be passed by value.				
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 arithmetic operator and a bitwise operator shall return a "prvalue"	Yes			
CPP_T042	A relational operator shall return a boolean value	Yes			
CPP_T043	If "operator[]"	Yes			

	is to be overloaded with a non-const version, const version shall also be implemented				
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 operators should not be used	Yes			
CPP_T050	Types shall be explicitly specified	Yes			
CPP_T051	C-style	Yes			

	strings shall not be used				
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 casting shall be used when performing calculations with a mix of	Yes			

	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 of Operations	Yes			
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			
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			

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	Yes			

	variables being given values that are not 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	Yes			

	names shall be used instead				
CPP_V024	Variables of type char shall be explicitly qualified as signed or unsigned when used to store numbers	Yes			
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	Yes			

	the constness attribute of a variable.				
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 provided by const_cast and dynamic_cast.	Yes			
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	Yes			

	of C-style arrays whenever possible.				
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 hiding should be implemented.	Yes			
CPP_V039	Within an object, most instance variables should be accessed directly. Methods should be used to set variables whose values	Yes			

	are determined by an algorithm.				
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_VALUE	Absolute Value Proper Usage	Yes			
CPP_WARN_ABSTRACT_FINAL_CLASS	Abstract Classes Should Not Be Final or Sealed	Yes			
CPP_WARN_ABSTRACT_VBASE_INIT	No Useless Init for Abstract Virtual Base	Yes			
CPP_WARN_ADDRESS_OF	Do Not Take the Address	Yes			

_PACKED_MEMBER	of Packed Members				
CPP_WARN_ADDRESS_OF_TEMPORARY	Do Not Take the Address of Temporary Objects	Yes			High
CPP_WARN_AIX_COMPAT	IBM AIX Compatibility with Byte Alignment	Yes			
CPP_WARN_ALIGN_MISMATCH	Match Byte Alignment of Arguments	Yes			
CPP_WARN_ALLOCA	Do Not Use Certain Allocation Functions	Yes			
CPP_WARN_ALLOCA_WITH_ALIGN_ALIGNOF	Correct Usage of Second Argument of Certain Allocation Functions	Yes			
CPP_WARN_ALWAYS_INLINE_COROUTINE	Always Inline Coroutine Functions	Yes			
CPP_WARN_AMBIGUOUS_DELETE	Ambiguous Delete	Yes			
CPP_WARN_AMBIGUOUS_ELLIPSIS	Ambiguous Ellipsis	Yes			
CPP_WARN_AMBIGUOUS_MACRO	Ambiguous Macro	Yes			
CPP_WARN_AMBIGUOUS_MEMBER_TEMPLATE	Ambiguous Member Template	Yes			
CPP_WARN_	Ambiguous	Yes			

AMBIGUOUS_REVERSED_OPERATOR	Reversed Operator				
CPP_WARN_ANALYZER_INCOMPATIBLE_PLUGIN	Analyzer Incompatible Plugin	Yes			
CPP_WARN_ANON_ENUM_ENUM_CONVERSION	Anon-Enum Enum Conversion	Yes			
CPP_WARN_ANONYMOUS_PACK_PARENS	Anonymous Pack Parentheses	Yes			
CPP_WARN_ARC_BRIDGE_CASTS_DISALLOWED_IN_NONARC	ARC (Automatic Reference Counting) Bridge Casts Disallowed in Non-ARC	Yes			
CPP_WARN_ARC_MAYBE_REPEATED_USE_OF_WEAK	ARC Maybe Repeated Use of Weak	Yes			
CPP_WARN_ARC_RETAIN_CYCLES	ARC Retain Cycles	Yes			
CPP_WARN_ARC_UNSAFE_RETAINED_ASSIGN	ARC Unsafe Retained Assign	Yes			
CPP_WARN_ARGUMENT_OUTSIDE_RANGE	Argument Outside Range	Yes			
CPP_WARN_ARGUMENT_UNDEFINED_BEHAVIOUR	Argument Undefined Behaviour	Yes			

CPP_WARN_ARRAY_BOUNDS	Array Bounds	Yes			High
CPP_WARN_ARRAY_BOUNDS_POINTER_ARITHMETIC	Array Bounds Pointer Arithmetic	Yes			
CPP_WARN_ARRAY_PARAMETER	Array Parameter	Yes			
CPP_WARN_ASM_OPERAND_WIDTHS	Assembly Operand Widths	Yes			
CPP_WARN_ASSIGN_ENUM	Assign Enum	Yes			
CPP_WARN_ASSUME	Discarded Side Effects to __assume Function	Yes			
CPP_WARN_ATOMIC_ACCESS	Atomic Access	Yes			
CPP_WARN_ATOMIC_ALIGNMENT	Atomic Alignment	Yes			
CPP_WARN_ATOMIC_IMPLICIT_SEQ_CST	Atomic Implicitly Sequentially-Consistent	Yes			
CPP_WARN_ATOMIC_MEMORY_ORDERING	Atomic Memory Ordering	Yes			
CPP_WARN_AUTO_DISABLE_VPTR_SANITIZER	Auto Disable Virtual Pointer Sanitizer	Yes			
CPP_WARN_AUTO_STORAGE_CLASS	Auto Storage Class	Yes			

CPP_WARN_AVAILABILITY	Availability Attribute	Yes			High
CPP_WARN_AVR_RTLIB_LINKING_QUIRKS	AVR RTLIB (Real-Time Library) Linking Quirks	Yes			
CPP_WARN_BACKEND_PLUGIN	Backend Plugin	Yes			
CPP_WARN_BACKSLASH_NEWLINE_ESCAPE	Backslash Newline Escape	Yes			High
CPP_WARN_BAD_FUNCTION_CAST	Do Not Cast from Function Call of One Type to Another	Yes			
CPP_WARN_BIND_TO_TEMPORARY_COPY	Bind to Temporary Copy	Yes			
CPP_WARN_BINDING_IN_CONDITION	Binding in Condition	Yes			
CPP_WARN_BIT_INT_EXTENSION	Bit Int Extension	Yes			
CPP_WARN_BITFIELD_CONSTANT_CONVERSION	Bitfield Constant Conversion	Yes			High
CPP_WARN_BITFIELD_ENUM_CONVERSION	Bitfield Enum Conversion	Yes			
CPP_WARN_BITFIELD_WIDTH	Do Not Exceed Bit-Field Width	Yes			High
CPP_WARN_BITWISE_CO	Bitwise Conditional	Yes			

NDITIONAL_PARENTHESES	Parentheses				
CPP_WARN_BITWISE_INSTEAD_OF_LOGICAL	Bitwise Instead of Logical	Yes			
CPP_WARN_BITWISE_OP_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_ALIGNMENT	Builtin Assume Aligned Alignment	Yes			
CPP_WARN_BUILTIN_MACRO_REDEFINED	Builtin Macro Redefined	Yes			High
CPP_WARN_BUILTIN_MEMCPY_CHK_SIZE	Builtin Memcpy Check Size	Yes			High
CPP_WARN_BUILTIN_REQUIRES_HEADER	Builtin Requires Header	Yes			
CPP_WARN_C2X_EXTENS	C2X Extensions	Yes			

IONS					
CPP_WARN_C11_EXTENSIONS	C11 Extensions	Yes			
CPP_WARN_C99_COMPAT	C99 Compatibility	Yes			
CPP_WARN_C99_DESIGNATOR	C99 Designator	Yes			
CPP_WARN_C99_EXTENSIONS	C99 Extensions	Yes			
CPP_WARN_CALL_TO_PURE_VIRTUAL_FROM_CTOR_DTOR	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_QUAL	Cast Qualifiers	Yes			
CPP_WARN_CAST_QUAL_UNRELATED	Cast Qualifiers Unrelated	Yes			High
CPP_WARN_CHAR_SUBSCRIPTS	Char Subscripts	Yes			
CPP_WARN_CLANG_CL_P	Clang-CL Precompiled	Yes			

CH	Headers				
CPP_WARN_CLASS_CONVERSION	Class Conversion	Yes			High
CPP_WARN_CLASS_VARARGS	Class Variadic Arguments	Yes			
CPP_WARN_CMSE_UNION_LEAK	CMSE (Cortex-M Support for Security Extension) Union Leak	Yes			
CPP_WARN_COMMA	Comma Operator Misuse	Yes			
CPP_WARN_COMMENT	Comment Misuse	Yes			
CPP_WARN_COMPARE_DISTINCT_POINTER_TYPES	Compare Distinct Pointer Types	Yes			High
CPP_WARN_COMPLEX_COMPONENT_INIT	Complex Component Init	Yes			
CPP_WARN_COMPOUND_TOKEN_SPLIT_BY_MACRO	Compound Token Split by Macro	Yes			High
CPP_WARN_COMPOUND_TOKEN_SPLIT_BY_SPACE	Compound Token Split by Space	Yes			
CPP_WARN_CONDITIONAL_TYPE_MISMATCH	Conditional Type Mismatch	Yes			High
CPP_WARN_CONDITIONAL_UNINITIALI	Conditional Uninitialized	Yes			

ZED					
CPP_WARN_CONSTANT_CONVERSION	Constant Conversion	Yes			High
CPP_WARN_CONSTANT_EVALUATED	Constant Evaluated	Yes			
CPP_WARN_CONSTANT_LOGICAL_OPERAND	Constant Logical Operand	Yes			High
CPP_WARN_CONSTEXPR_NOT_CONST	Constexpr Not Const	Yes			
CPP_WARN_CONSUMED	Consumable Attribute	Yes			
CPP_WARN_CONVERSION	Type Conversion	Yes			
CPP_WARN_COROUTINE	Coroutine Return Type	Yes			
CPP_WARN_COROUTINE_MISSING_UNHANDLED_EXCEPTION	Coroutine Missing Unhandled Exception	Yes			
CPP_WARN_COVERED_SWITCH_DEFAULT	Covered Switch Default	Yes			
CPP_WARN_CPP_COMPAT	C++ Compatibility	Yes			
CPP_WARN_CPP2B_EXTENSIONS	C++2B Extensions	Yes			
CPP_WARN_CPP11_COMPAT	C++11 Compatibility	Yes			

CPP_WARN_CPP11_COMPAT_DEPRECATED_WRITABLE_STRINGS	C++11 Compatibility Deprecated Writable Strings	Yes			High
CPP_WARN_CPP11_COMPAT_RESERVED_USER_DEFINED_LITERAL	C++11 Compatibility Reserved User Defined Literal	Yes			
CPP_WARN_CPP11_EXTENSIONS	C++11 Extensions	Yes			
CPP_WARN_CPP11_EXTRA_SEMI	C++11 Extra Semicolon	Yes			
CPP_WARN_CPP11_INLINE_NAMESPACE	C++11 Inline Namespace	Yes			
CPP_WARN_CPP11_LONG_LONG	C++11 Long Long	Yes			
CPP_WARN_CPP11_NARROWING	C++11 Narrowing	Yes			
CPP_WARN_CPP14_ATTRIBUTE_EXTENSIONS	C++14 Attribute Extensions	Yes			
CPP_WARN_CPP14_BINARY_LITERAL	C++14 Binary Literal	Yes			
CPP_WARN_CPP14_EXTENSIONS	C++14 Extensions	Yes			
CPP_WARN_CPP17_ATTRIBUTE_EXTENSIONS	C++17 Attribute Extensions	Yes			
CPP_WARN_	C++17	Yes			

CPP17_COMPAT_MANGLING	Compatibility Mangling				
CPP_WARN_CPP17_EXTENSIONS	C++17 Extensions	Yes			
CPP_WARN_CPP20_ATTRIBUTE_EXTENSIONS	C++20 Attribute Extensions	Yes			
CPP_WARN_CPP20_COMPAT	C++20 Compatibility	Yes			
CPP_WARN_CPP20_DESIGNATOR	C++20 Designator	Yes			
CPP_WARN_CPP20_EXTENSIONS	C++20 Extensions	Yes			
CPP_WARN_CPP98_COMPAT	C++98 Compatibility	Yes			
CPP_WARN_CPP98_COMPAT_BIND_TO_TEMPORARY_COPY	C++98 Compatibility Bind to Temporary Copy	Yes			
CPP_WARN_CPP98_COMPAT_EXTRA_SEMI	C++98 Compatibility Extra Semicolon	Yes			
CPP_WARN_CPP98_COMPAT_LOCAL_TYPE_TEMPLATE_ARGS	C++98 Compatibility Local Type Template Args	Yes			
CPP_WARN_CPP98_COMPAT_PEDANTIC	C++98 Compatibility Pedantic	Yes			
CPP_WARN_CPP98_COM	C++98 Compatibility	Yes			

PAT_UNNAMED_TYPE_TEMPLATE_ARGS	Unnamed Type Template Args				
CPP_WARN_CPP98_CPP11_COMPAT_BINARY_LITERAL	C++98 C++11 Compatibility Binary Literal	Yes			
CPP_WARN_CTAD_MAYBE_UNSUPPORTED	CTAD (Class Template Argument Deduction) Maybe Unsupported	Yes			
CPP_WARN_CXX_ATTRIBUTE_EXTENSION	C++ Attribute Extension	Yes			
CPP_WARN_DANGLING	Dangling Pointers	Yes			High
CPP_WARN_DANGLING_ELSE	Dangling Else	Yes			
CPP_WARN_DANGLING_FIELD	Dangling Field	Yes			High
CPP_WARN_DANGLING_GSL	Dangling Pointers Found by Guidelines Support Library	Yes			High
CPP_WARN_DANGLING_INITIALIZER_LIST	Dangling Initializer List	Yes			High
CPP_WARN_DARWIN_SDK_SETTINGS	Darwin SDK Settings	Yes			
CPP_WARN_DATE_TIME	Date and Time Macros	Yes			

CPP_WARN_DEALLOC_IN_CATEGORY	Dealloc in Category	Yes			
CPP_WARN_DEBUG_COMPRESSION_UNAVAILABLE	Debug Compression Unavailable	Yes			
CPP_WARN_DECLARATION_AFTER_STATEMENT	Declaration After Statement	Yes			
CPP_WARN_DEFAULTED_FUNCTION_DELETED	Defaulted Function Deleted	Yes			High
CPP_WARN_DELEGATING_CTOR_CYCLES	Delegating Constructor Cycles	Yes			High
CPP_WARN_DELETE_ABSTRACT_NON_VIRTUAL_DTOR	Delete Abstract Non-Virtual Destructor	Yes			High
CPP_WARN_DELETE_INCOMPLETE	Delete Incomplete	Yes			High
CPP_WARN_DELETE_NON_ABSTRACT_NON_VIRTUAL_DTOR	Delete Non-Abstract Non-Virtual Destructor	Yes			
CPP_WARN_DEPRECATED	Deprecated	Yes			
CPP_WARN_DEPRECATED_ALTIVEC_SRC_COMPAT	Deprecated AltiVec Instruction Set Source Compatibility	Yes			
CPP_WARN_DEPRECATED_ANON_ENU	Deprecated Anon-Enum, Enum	Yes			

M_ENUM_CONVERSION	Conversion				
CPP_WARN_DEPRECATED_ARRAY_COMPARE	Deprecated Array Compare	Yes			
CPP_WARN_DEPRECATED_ATTRIBUTES	Deprecated Attributes	Yes			
CPP_WARN_DEPRECATED_BUILTINS	Deprecated Builtins	Yes			High
CPP_WARN_DEPRECATED_COMMA_SUBSCRIPT	Deprecated Comma Subscript	Yes			
CPP_WARN_DEPRECATED_COPY	Deprecated Copy	Yes			
CPP_WARN_DEPRECATED_COPY_WITH_DTOR	Deprecated Copy with Destructor	Yes			
CPP_WARN_DEPRECATED_COPY_WITH_USER_PROVIDED_COPY	Deprecated Copy with User Provided Copy	Yes			
CPP_WARN_DEPRECATED_COPY_WITH_USER_PROVIDED_DTOR	Deprecated Copy with User Provided Destructor	Yes			
CPP_WARN_DEPRECATED_COROUTINE	Deprecated Coroutine	Yes			
CPP_WARN_DEPRECATED_DECLARATIONS	Deprecated Declarations	Yes			High
CPP_WARN_	Deprecated	Yes			

DEPRECATED_DYNAMIC_EXCEPTION_SPEC	Dynamic Exception Spec				
CPP_WARN_DEPRECATED_EXPERIMENTAL_COROUTINE	Deprecated Experimental Coroutine	Yes			
CPP_WARN_DEPRECATED_IMPLEMENTATIONS	Deprecated Implementations	Yes			
CPP_WARN_DEPRECATED_INCREMENT_BOOL	Deprecated Increment Bool	Yes			High
CPP_WARN_DEPRECATED_NON_PROTOTYPE	Deprecated Non-Prototype	Yes			High
CPP_WARN_DEPRECATED_REGISTER	Deprecated Register	Yes			High
CPP_WARN_DEPRECATED_STATIC_ANALYZER_FLAG	Deprecated Static Analyzer Flag	Yes			
CPP_WARN_DEPRECATED_THIS_CAPTURE	Deprecated This Capture	Yes			
CPP_WARN_DEPRECATED_TYPE	Deprecated Type	Yes			
CPP_WARN_DEPRECATED_VOLATILE	Deprecated Volatile	Yes			High
CPP_WARN_DISABLED_MACRO_EXPA	Disabled Macro Expansion	Yes			

NSION					
CPP_WARN_DIVISION_BY_ZERO	Division by Zero	Yes			High
CPP_WARN_DLL_ATTRIBUTE_ON_REDECLARATION	DLL Attribute on Re-Declaration	Yes			
CPP_WARN_DLLEXPORT_EXPLICIT_INSTANTIATION_DECL	DLLexport Explicit Instantiation Decl	Yes			
CPP_WARN_DLLIMPORT_STATIC_FIELD_DEF	DLLimport Static Field Def	Yes			
CPP_WARN_DOCUMENTATION	Documentation Warnings	Yes			
CPP_WARN_DOCUMENTATION_DEPRECATED_SYNC	Documentation Deprecated Sync	Yes			
CPP_WARN_DOCUMENTATION_HTML	Documentation Html	Yes			
CPP_WARN_DOCUMENTATION_PEDANTIC	Documentation Pedantic	Yes			
CPP_WARN_DOCUMENTATION_UNKNOWN_COMMAND	Documentation Unknown Command	Yes			
CPP_WARN_DOLLAR_IN_IDENTIFIER_EXTENSION	Dollar in Identifier Extension	Yes			

CPP_WARN_DOUBLE_PROMOTION	Double Promotion	Yes			
CPP_WARN_DTOR_NAME	Destructor Name	Yes			
CPP_WARN_DTOR_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_MEMACCESS	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			
CPP_WARN_EMPTY_BODY	Control Loop Shall Not Have Empty Body	Yes			
CPP_WARN_EMPTY_DECOMPOSITION	Decomposition Group Shall Not Be Empty	Yes			
CPP_WARN_EMPTY_INIT_STMT	No Empty Initialization Statements	Yes			
CPP_WARN_EMPTY_TRANSLATION_UNIT	Empty Translation Unit	Yes			
CPP_WARN_ENUM_COMPARE	Enum Compare	Yes			High
CPP_WARN_ENUM_COMPARE_CONDITIONAL	Enum Compare Conditional	Yes			
CPP_WARN_ENUM_COMPARE_SWITCH	Enum Compare Switch	Yes			High
CPP_WARN_ENUM_CONVERSION	Enum Conversion	Yes			
CPP_WARN_ENUM_ENUM_CONVERSION	Enum Enum Conversion	Yes			
CPP_WARN_ENUM_FLOAT_CONVERSION	Enum Float Conversion	Yes			
CPP_WARN_ENUM_TOO_LARGE	Enum Too Large	Yes			

CPP_WARN_EXCEPTIONS	Exceptions	Yes			High
CPP_WARN_EXCESS_INITIALIZERS	Excess Initializers	Yes			
CPP_WARN_EXIT_TIME_DESTRUCTORS	Exit Time Destructors	Yes			
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	Extra Warnings	Yes			
CPP_WARN_EXTRA_QUALIFICATION	Extra Qualification	Yes			High
CPP_WARN_EXTRA_SEMI	Extra Semicolon	Yes			
CPP_WARN_EXTRA_SEMI_STMT	Extra Semicolon in Empty Expression Statement	Yes			
CPP_WARN_EXTRA_TOKENS	Extra Tokens	Yes			
CPP_WARN_	Final	Yes			

FINAL_DTOR_NON_FINAL_CLASS	Destructor Non-Final Class				
CPP_WARN_FINAL_MACRO	Final Macros Should Not Be Redefined	Yes			
CPP_WARN_FIXED_ENUM_EXTENSION	Fixed Enum Extension	Yes			
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	Format String	Yes			High
CPP_WARN_FORMAT_EXTRA_ARGS	Format Extra Args	Yes			High
CPP_WARN_	Format	Yes			High

FORMAT_INSUFFICIENT_ARGS	Insufficient Args				
CPP_WARN_FORMAT_INVALID_SPECIFIER	Format Invalid Specifier	Yes			High
CPP_WARN_FORMAT_NON_ISO	Format Non-ISO	Yes			
CPP_WARN_FORMAT_NONLITERAL	Format Non-Literal	Yes			
CPP_WARN_FORMAT_PEDANTIC	Format Pedantic	Yes			
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_NONHEAP_OBJECT	Free Non-Heap Object	Yes			High
CPP_WARN_FUNCTION_	Function Multi-Version	Yes			

MULTIVERSION					
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_ISEL	GlobalISel (Global Instruction Selection) Framework	Yes			
CPP_WARN_GNU_ALIGNOF_EXPRESSION	GNU Alignof Expression	Yes			
CPP_WARN_GNU_ANONYMOUS_STRUCT	GNU Anonymous Struct	Yes			
CPP_WARN_GNU_ARRAY_MEMBER_PAREN_INIT	GNU Array Member Parentheses Init	Yes			
CPP_WARN_GNU_AUTO_TYPE	GNU Auto Type	Yes			
CPP_WARN_GNU_BINARY_LITERAL	GNU Binary Literal	Yes			
CPP_WARN_GNU_CASE_RANGE	GNU Case Range	Yes			
CPP_WARN_GNU_COMPLEX_INTEGER	GNU Complex Integer	Yes			

CPP_WARN_GNU_COMPOUND_LITERAL_INITIALIZER	GNU Compound Literal Initializer	Yes			
CPP_WARN_GNU_CONDITIONAL_OMITTED_OPERAND	GNU Conditional Omitted Operand	Yes			
CPP_WARN_GNU_DESIGNATOR	GNU Designator	Yes			
CPP_WARN_GNU_EMPTY_INITIALIZER	GNU Empty Initializer	Yes			
CPP_WARN_GNU_EMPTY_STRUCT	GNU Empty Struct	Yes			
CPP_WARN_GNU_FLEXIBLE_ARRAY_INITIALIZER	GNU Flexible Array Initializer	Yes			
CPP_WARN_GNU_FLEXIBLE_ARRAY_UNION_MEMBER	GNU Flexible Array Union Member	Yes			
CPP_WARN_GNU_FOLDING_CONSTANT	GNU Folding Constant	Yes			
CPP_WARN_GNU_IMAGINARY_CONSTANT	GNU Imaginary Constant	Yes			
CPP_WARN_GNU_INCLUDE_NEXT	GNU Include Next	Yes			
CPP_WARN_GNU_INLINE_CPP_WITHO	GNU Inline Cpp Without Extern	Yes			

UT_EXTERN					
CPP_WARN_GNU_LABEL_AS_VALUE	GNU Label as Value	Yes			
CPP_WARN_GNU_LINE_MARKER	GNU Line Marker	Yes			
CPP_WARN_GNU_NULL_POINTER_ARITHMETIC	GNU Null Pointer Arithmetic	Yes			
CPP_WARN_GNU_POINTER_ARITH	GNU Pointer Arithmetic	Yes			
CPP_WARN_GNU_REDECLARED_ENUM	GNU Re-Declared Enum	Yes			
CPP_WARN_GNU_STATEMENT_EXPRESSION	GNU Statement Expression	Yes			
CPP_WARN_GNU_STATEMENT_EXPRESSION_FROM_MACRO_EXPANSION	GNU Statement Expression from Macro Expansion	Yes			
CPP_WARN_GNU_STATIC_FLOAT_INIT	GNU Static Float Init	Yes			
CPP_WARN_GNU_STRING_LITERAL_OPERATOR_TEMPLATE	GNU String Literal Operator Template	Yes			
CPP_WARN_GNU_UNION_CAST	GNU Union Cast	Yes			
CPP_WARN_GNU_VARIABLE_SIZED_TY	GNU Variable Sized Type Not at End	Yes			

PE_NOT_AT_END					
CPP_WARN_GNU_ZERO_LINE_DIRECTIVE	GNU Zero Line Directive	Yes			
CPP_WARN_GNU_ZERO_VARIADIC_MACRO_ARGUMENTS	GNU Zero Variadic Macro Arguments	Yes			
CPP_WARN_HEADER_GUARD	Header Guard	Yes			High
CPP_WARN_HEADER_HYGIENE	Header Hygiene	Yes			
CPP_WARN_IDIOMATIC_PARENTHESES	Idiomatic Parentheses	Yes			
CPP_WARN_IGNORED_ATTRIBUTES	Ignored Attributes	Yes			High
CPP_WARN_IGNORED_AVAILABILITY_WITHOUT_SDK_SETTINGS	Ignored Availability Without Sdk Settings	Yes			
CPP_WARN_IGNORED_OPTIMIZATION_ARGUMENT	Ignored Optimization Argument	Yes			
CPP_WARN_IGNORED_PRAGMA_INTRINSIC	Ignored Pragma Intrinsic	Yes			
CPP_WARN_IGNORED_PRAGMAS	Ignored Pragmas	Yes			
CPP_WARN_IGNORED_RE	Ignored Reference	Yes			High

FERENCE_QUALIFIERS	Qualifiers				
CPP_WARN_IMPLICIT_CONST_INT_FLOAT_CONVERSION	Implicit Const Int Float Conversion	Yes			High
CPP_WARN_IMPLICIT_CONVERSION_FLOATING_POINT_TO_BOOL	Implicit Conversion Floating Point to Bool	Yes			High
CPP_WARN_IMPLICIT_EXCEPTION_SPEC_MISMATCH	Implicit Exception Spec Mismatch	Yes			High
CPP_WARN_IMPLICIT_FALLTHROUGH	Implicit Fallthrough	Yes			
CPP_WARN_IMPLICIT_FALLTHROUGH_PER_FUNCTION	Implicit Fallthrough Per Function	Yes			
CPP_WARN_IMPLICIT_FIXED_POINT_CONVERSION	Implicit Fixed Point Conversion	Yes			High
CPP_WARN_IMPLICIT_FLOAT_CONVERSION	Implicit Float Conversion	Yes			
CPP_WARN_IMPLICIT_FUNCTION_DECLARATION	Implicit Function Declaration	Yes			
CPP_WARN_IMPLICIT_INT	Implicit Int	Yes			
CPP_WARN_IMPLICIT_INT	Implicit Int Conversion	Yes			

_CONVERSIO N					
CPP_WARN_I MPLICIT_INT _FLOAT_CON VERSION	Implicit Int Float Conversion	Yes			
CPP_WARN_I MPLICIT_RET AIN_SELF	Implicit Retain Self	Yes			
CPP_WARN_I MPLICITLY_U NSIGNED_LIT ERAL	Implicitly Unsigned Literal	Yes			High
CPP_WARN_I MPORT_PRE PROCESSOR _DIRECTIVE_ PEDANTIC	Import Preprocessor Directive Pedantic	Yes			
CPP_WARN_I NACCESSIBL E_BASE	Inaccessible Base	Yes			High
CPP_WARN_I NCLUDE_NE XT_ABSOLUT E_PATH	Include Next Absolute Path	Yes			
CPP_WARN_I NCLUDE_NE XT_OUTSIDE _HEADER	Include Next Outside Header	Yes			
CPP_WARN_I NCOMPATIBL E_EXCEPTIO N_SPEC	Incompatible Exception Spec	Yes			High
CPP_WARN_I NCOMPATIBL E_FUNCTION _POINTER_T YPES	Incompatible Function Pointer Types	Yes			
CPP_WARN_I NCOMPATIBL E_LIBRARY_R EDECLARATI	Incompatible Library Redeclaration	Yes			High

ON					
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			
CPP_WARN_INITIALIZER_OVERRIDES	Initializer Overrides	Yes			
CPP_WARN_INJECTED_CLASS_NAME	Injected Class Name	Yes			High
CPP_WARN_INLINE_ASM	Inline Assembly	Yes			
CPP_WARN_INLINE_NAMESPACE_REOPENED_NONINLINE	Inline Namespace Reopened Non-Inline	Yes			High
CPP_WARN_INLINE_NEW_DELETE	Inline New Delete	Yes			
CPP_WARN_INSTANTIATION_AFTER_SPECIALIZATION	Instantiation After Specialization	Yes			High
CPP_WARN_INT_CONVERSION	Int Conversion	Yes			
CPP_WARN_INT_IN_BOOL_CONTEXT	Int in Bool Context	Yes			
CPP_WARN_INT_TO_POINTER_CAST	Int to Pointer Cast	Yes			High
CPP_WARN_INT_TO_VOID_POINTER_CAST	Int to Void Pointer Cast	Yes			High
CPP_WARN_INTEGER_OVERFLOW	Integer Overflow	Yes			High
CPP_WARN_I	Interrupt	Yes			

NTERRUPT_SERVICE_ROUTINE	Service Routine				
CPP_WARN_INVALID_COMMAND_LINE_ARGUMENT	Invalid Command Line Argument	Yes			
CPP_WARN_INVALID_CONSTEXPR	Invalid Constexpr	Yes			
CPP_WARN_INVALID_IBOUTLET	Invalid IBOutletCollection (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	Invalid Noreturn Attribute	Yes			High
CPP_WARN_INVALID_OFFSETOF	Invalid Offsetof	Yes			High
CPP_WARN_INVALID_OR_NONEXISTENT_DIRECTORY	Invalid or Nonexistent Directory	Yes			
CPP_WARN_INVALID_PAR	Invalid Partial Specialization	Yes			

TIAL_SPECIALIZATION					
CPP_WARN_INVALID_PP_TOKEN	Invalid Preprocessor Token	Yes			High
CPP_WARN_INVALID_SOURCE_ENCODING	Invalid Source Encoding	Yes			
CPP_WARN_INVALID_TOKEN_PASTE	Invalid Token Paste	Yes			
CPP_WARN_INVALID_UTF8	Invalid UTF-8	Yes			
CPP_WARN_JUMP_SEH_FINALLY	Jump SEH (Structured Exception Handling) Finally	Yes			
CPP_WARN_KEYWORD_COMPAT	Keyword Compatibility	Yes			
CPP_WARN_KEYWORD_MACRO	Keyword Macro	Yes			
CPP_WARN_KNR_PROMOTED_PARAMETER	K&R Promoted Parameter	Yes			
CPP_WARN_LANGUAGE_EXTENSION_TOKEN	Language Extension Token	Yes			
CPP_WARN_LARGE_BY_VALUE_COPY	Large by Value Copy	Yes			High
CPP_WARN_LINKER_WARNINGS	Linker Warnings	Yes			
CPP_WARN_	Literal	Yes			High

LITERAL_CONVERSION	Conversion				
CPP_WARN_LITERAL_RANGE	Literal Range	Yes			High
CPP_WARN_LOCAL_TYPE_TEMPLATE_ARGS	Local Type Template Args	Yes			
CPP_WARN_LOGICAL_NOT_PARENTHESES	Logical Not Parentheses	Yes			High
CPP_WARN_LOGICAL_OP_PARENTHESES	Logical Operator Parentheses	Yes			
CPP_WARN_LONG_LONG	Long Long	Yes			
CPP_WARN_MACRO_REDEFINED	Macro Redefined	Yes			High
CPP_WARN_MAIN	Main Function Conventions	Yes			
CPP_WARN_MAIN_RETURN_TYPE	Main Return Type	Yes			High
CPP_WARN_MALFORMED_WARNING_CHECK	Malformed Warning Check	Yes			
CPP_WARN_MANY_BRACES_AROUND_SCALAR_INIT	Many Braces Around Scalar Init	Yes			High
CPP_WARN_MAX_TOKENS	Max Tokens	Yes			
CPP_WARN_MAX_UNSIG	Max Unsigned	Yes			High

NED_ZERO	Zero				
CPP_WARN_MEMSET_TRANSPOSED_ARGS	Memset Transposed Args	Yes			High
CPP_WARN_MEMSIZE_COMPARISON	Memsize Comparison	Yes			High
CPP_WARN_MICROSOFT_ABSTRACT	Microsoft Abstract	Yes			
CPP_WARN_MICROSOFT_ANON_TAG	Microsoft Anonymous Tag	Yes			
CPP_WARN_MICROSOFT_CAST	Microsoft Cast	Yes			
CPP_WARN_MICROSOFT_CHARIZE	Microsoft Charizing Operator	Yes			
CPP_WARN_MICROSOFT_COMMENT_PASTE	Microsoft Comment Paste	Yes			
CPP_WARN_MICROSOFT_CONST_INIT	Microsoft Const Init	Yes			
CPP_WARN_MICROSOFT_CPP_MACRO	Microsoft C++ Macro	Yes			
CPP_WARN_MICROSOFT_DEFAULT_ARG_REDEFINITION	Microsoft Default Arg Redefinition	Yes			
CPP_WARN_MICROSOFT_DRECTVE_SECTION	Microsoft Drectve Section	Yes			
CPP_WARN_MICROSOFT_	Microsoft End of File	Yes			

END_OF_FILE					
CPP_WARN_MICROSOFT_ENUM_FORWARD_REFERENCE	Microsoft Enum Forward Reference	Yes			
CPP_WARN_MICROSOFT_ENUM_VALUE	Microsoft Enum Value	Yes			
CPP_WARN_MICROSOFT_EXCEPTION_SPEC	Microsoft Exception Spec	Yes			
CPP_WARN_MICROSOFT_EXISTS	Microsoft Exists	Yes			
CPP_WARN_MICROSOFT_EXPLICIT_CONSTRUCTOR_CALL	Microsoft Explicit Constructor Call	Yes			
CPP_WARN_MICROSOFT_EXTRA_QUALIFICATION	Microsoft Extra Qualification	Yes			
CPP_WARN_MICROSOFT_FIXED_ENUM	Microsoft Fixed Enum	Yes			
CPP_WARN_MICROSOFT_FLEXIBLE_ARRAY	Microsoft Flexible Array	Yes			
CPP_WARN_MICROSOFT_GOTO	Microsoft Goto	Yes			
CPP_WARN_MICROSOFT_INACCESSIBLE_BASE	Microsoft Inaccessible Base	Yes			
CPP_WARN_MICROSOFT_	Microsoft Include	Yes			

INCLUDE					
CPP_WARN_MICROSOFT_MUTABLE_REFERENCE	Microsoft Mutable Reference	Yes			
CPP_WARN_MICROSOFT_PURE_DEFINITION	Microsoft Pure Definition	Yes			
CPP_WARN_MICROSOFT_REDECLARE_STATIC	Microsoft Redeclare Static	Yes			
CPP_WARN_MICROSOFT_SEALED	Microsoft Sealed	Yes			
CPP_WARN_MICROSOFT_STATIC_ASSERT	Microsoft Static Assert	Yes			
CPP_WARN_MICROSOFT_TEMPLATE	Microsoft Template	Yes			
CPP_WARN_MICROSOFT_TEMPLATE_SHADOW	Microsoft Template Shadow	Yes			
CPP_WARN_MICROSOFT_UNION_MEMBER_REFERENCE	Microsoft Union Member Reference	Yes			
CPP_WARN_MICROSOFT_UNQUALIFIED_FRIEND	Microsoft Unqualified Friend	Yes			
CPP_WARN_MICROSOFT_USING_DECL	Microsoft Using Decl	Yes			
CPP_WARN_MICROSOFT_VOID_PSEUD	Microsoft Void Pseudo Destructor	Yes			

O_DTOR					
CPP_WARN_MISEXPECT	Misuse of __builtin_expect()	Yes			
CPP_WARN_MISLEADING_INDENTATION	Misleading Indentation	Yes			
CPP_WARN_MISMATCHED_NEW_DELETE	Mismatched New Delete	Yes			High
CPP_WARN_MISMATCHED_TAGS	Mismatched Tags	Yes			
CPP_WARN_MISSING_BRACES	Missing Braces	Yes			
CPP_WARN_MISSING_CONSTINIT	Missing Constinit	Yes			
CPP_WARN_MISSING_DECLARATIONS	Missing Declarations	Yes			High
CPP_WARN_MISSING_EXCEPTION_SPEC	Missing Exception Spec	Yes			High
CPP_WARN_MISSING_FIELD_INITIALIZERS	Missing Field Initializers	Yes			
CPP_WARN_MISSING_METHOD_RETURN_TYPE	Missing Method Return Type	Yes			
CPP_WARN_MISSING_NORETURN	Missing Noreturn Attribute	Yes			
CPP_WARN_MISSING_PR	Missing Prototype for	Yes			

OTOTYPE_FOR_CC	Calling Convention				
CPP_WARN_MISSING_PROTOTYPES	Missing Prototypes	Yes			
CPP_WARN_MISSING_SELECTOR_NAME	Missing Selector Name	Yes			
CPP_WARN_MISSING_SYSROOT	Missing Sysroot	Yes			
CPP_WARN_MISSING_VARIABLE_DECLARATIONS	Missing Variable Declarations	Yes			
CPP_WARN_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			
CPP_WARN_MODULE_IMPORT_IN_EXTERN_C	Module Import in Extern C	Yes			
CPP_WARN_MODULES_AMBIGUOUS_INTERNAL_LINKAGE	Modules Ambiguous Internal Linkage	Yes			
CPP_WARN_MODULES_I	Modules Import	Yes			

MPORT_NESTED_REDUNDANT	Nested Redundant				
CPP_WARN_MSVC_NOT_FOUND	MSVC Not Found	Yes			
CPP_WARN_MULTICHAR	Multiple Characters in Character Literal	Yes			High
CPP_WARN_MULTIPLE_MOVE_VBASE	Multiple Move Virtual Base	Yes			High
CPP_WARN_NESTED_ANON_TYPES	Nested Anon-Types	Yes			
CPP_WARN_NEW_RETURNS_NULL	New Returns Null	Yes			High
CPP_WARN_NEWLINE_EOF	Newline EOF (End of File)	Yes			
CPP_WARN_NODEREF	Noderef Attirbute	Yes			High
CPP_WARN_NON_C_TYPEDEF_FOR_LINKAGE	Non-C Typedef for Linkage	Yes			High
CPP_WARN_NON_LITERAL_NULL_CONVERSION	Non-Literal Null Conversion	Yes			High
CPP_WARN_NON_POD_VARARGS	Non-POD (Plain Old Data) Variadic Arguments	Yes			High
CPP_WARN_NON_POWER_OF_TWO_ALIGNMENT	Non-Power of Two Alignment	Yes			High
CPP_WARN_	Non-Virtual	Yes			

NON_VIRTUAL_DTOR	Destructor				
CPP_WARN_NONNULL	Null as an Argument	Yes			High
CPP_WARN_NONPORTABLE_INCLUDE_PATH	Non-Portable Include Path	Yes			High
CPP_WARN_NONPORTABLE_SYSTEM_INCLUDE_PATH	Non-Portable System Include Path	Yes			
CPP_WARN_NONPORTABLE_VECTOR_INITIALIZATION	Non-Portable Vector Initialization	Yes			
CPP_WARN_NONTRIVIAL_MEMACCESS	Nontrivial Memory Access	Yes			High
CPP_WARN_NULL_ARITHMETIC	Null Arithmetic	Yes			High
CPP_WARN_NULL_CHARACTER	Null Character	Yes			High
CPP_WARN_NULL_CONVERSION	Null Conversion	Yes			High
CPP_WARN_NULL_DEREFERENCE	Null Dereference	Yes			High
CPP_WARN_NULL_POINTER_ARITHMETIC	Null Pointer Arithmetic	Yes			
CPP_WARN_NULL_POINTER_SUBTRACTION	Null Pointer Subtraction	Yes			

CPP_WARN_OBJC_BOOL_CONSTANT_CONVERSION	Objective-C Bool Constant Conversion	Yes			
CPP_WARN_OBJC_CIRCULAR_CONTAINER	Objective-C Circular Container	Yes			
CPP_WARN_OBJC_MULTIPLE_METHOD_NAMES	Objective-C Multiple Method Names	Yes			
CPP_WARN_OBJC_READONLY_WITH_SETTER_PROPERTY	Objective-C Readonly with Setter Property	Yes			
CPP_WARN_OBJC_SIGNED_CHAR_BOOL_IMPLICIT_FLOAT_CONVERSION	Objective-C Signed Char Bool Implicit Float Conversion	Yes			
CPP_WARN_OBJC_SIGNED_CHAR_BOOL_IMPLICIT_INT_CONVERSION	Objective-C Signed Char Bool Implicit Int Conversion	Yes			
CPP_WARN_ODR	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_CLAUSES	OpenMP Clauses	Yes			

CPP_WARN_OPENMP_LOOP_FORM	OpenMP Loop Form	Yes			
CPP_WARN_OPENMP_MAPPING	OpenMP Mapping	Yes			
CPP_WARN_OPENMP_TARGET	OpenMP Target	Yes			
CPP_WARN_OPTION_IGNORED	Option Ignored	Yes			
CPP_WARN_ORDERED_COMPARE_FUNCTION_POINTERS	Ordered Compare Function Pointers	Yes			High
CPP_WARN_OUT_OF_LINE_DECLARATION	Out of Line Declaration	Yes			
CPP_WARN_OUT_OF_SCOPE_FUNCTION	Out of Scope Function	Yes			High
CPP_WARN_OVER_ALIGNED	Over Aligned	Yes			
CPP_WARN_OVERLENGTH_STRINGS	Long String Literals	Yes			
CPP_WARN_OVERLOADED_SHIFT_OP_PARENTHESES	Overloaded Shift Operator Parentheses	Yes			High
CPP_WARN_OVERLOADED_VIRTUAL	Overloaded Virtual	Yes			
CPP_WARN_OVERRIDE_MODULE	Override Module	Yes			

CPP_WARN_OVERRIDING_T_OPTION	Overriding Slash T Option	Yes			
CPP_WARN_PACKED	Packed Attribute	Yes			
CPP_WARN_PADDED	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_DATE_TIME	PCH (Precompiled Header) Date Time	Yes			
CPP_WARN_PEDANTIC	Pedantic	Yes			
CPP_WARN_PEDANTIC_CORE_FEATURES	Pedantic Core Features	Yes			
CPP_WARN_PESSIMIZING_MOVE	Pessimizing Move	Yes			
CPP_WARN_POINTER_ARITH	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_COM	Pointer Integer Compare	Yes			High

PARE					
CPP_WARN_POINTER_SIGN	Pointer Sign	Yes			High
CPP_WARN_POINTER_TO_ENUM_CAST	Pointer to Enum Cast	Yes			High
CPP_WARN_POINTER_TO_INT_CAST	Pointer to Int Cast	Yes			High
CPP_WARN_POINTER_TYPE_MISMATCH	Pointer Type Mismatch	Yes			High
CPP_WARN_POISON_SYSTEM_DIRECTORIES	Poison System Directories	Yes			
CPP_WARN_POTENTIALLY_EVALUATED_EXPRESSION	Potentially Evaluated Expression	Yes			High
CPP_WARN_PRAGMA_CLANG_ATTRIBUTE	Pragma Clang Attribute	Yes			
CPP_WARN_PRAGMA_MESSAGES	Preprocessor #Pragma Messages	Yes			
CPP_WARN_PRAGMA_ONCE_OUTSIDE_HEADER	Pragma once Outside Header	Yes			High
CPP_WARN_PRAGMA_PACK	Pragma Pack	Yes			
CPP_WARN_PRAGMA_PACK_SUSPICIOUS_INCLUD	Pragma Pack Suspicious Include	Yes			

E					
CPP_WARN_PRAGMA_SYSTEM_HEADER_OUTSIDE_HEADER	Pragma System Header Outside Header	Yes			
CPP_WARN_PRAGMAS	Pragmas	Yes			
CPP_WARN_PRE_C2X_COMPAT	Pre C2X Compatibility	Yes			
CPP_WARN_PRE_CPP2B_COMPAT	Pre C++2B Compatibility	Yes			
CPP_WARN_PRE_CPP14_COMPAT	Pre C++14 Compatibility	Yes			
CPP_WARN_PRE_CPP17_COMPAT	Pre C++17 Compatibility	Yes			
CPP_WARN_PRE_CPP17_COMPAT_PEDANTIC	Pre C++17 Compatibility Pedantic	Yes			
CPP_WARN_PRE_CPP20_COMPAT	Pre C++20 Compatibility	Yes			
CPP_WARN_PRE_CPP20_COMPAT_PEDANTIC	Pre C++20 Compatibility Pedantic	Yes			
CPP_WARN_PRE_OPENMP_51_COMPAT	Pre OpenMP 51 Compatibility	Yes			
CPP_WARN_PREDEFINED_IDENTIFIER_OUTSIDE_FUNCTION	Predefined Identifier Outside Function	Yes			
CPP_WARN_	Private Extern	Yes			

PRIVATE_EXTERN					
CPP_WARN_PRIVATE_HEADER	Private Header	Yes			
CPP_WARN_PROFILE_INSTR_MISSING	Profile Instrumented Code Missing	Yes			
CPP_WARN_PROFILE_INSTR_OUT_OF_DATE	Profile Instrumented Code Out of Date	Yes			
CPP_WARN_PROFILE_INSTR_UNPROFILED	Profile Instrumented Code Unprofiled	Yes			
CPP_WARN_PSABI	PSABI (Processor-Specific Application Binary Interface)	Yes			
CPP_WARN_QUALIFIED_VOID_RETURN_TYPE	Qualified Void Return Type	Yes			High
CPP_WARN_RANGE_LOOP_BIND_REFERENCE	Range Loop Bind Reference	Yes			
CPP_WARN_RANGE_LOOP_CONSTRUCT	Range Loop Construct	Yes			
CPP_WARN_REDECLARED_CLASS_MEMBER	Re-Declared Class Member	Yes			High
CPP_WARN_REDUNDANT_CONSTEVAL_IF	Redundant Consteval If	Yes			High

CPP_WARN_REDUNDANT_MOVE	Redundant Move	Yes			
CPP_WARN_REDUNDANT_PARENS	Redundant Parentheses	Yes			
CPP_WARN_REGISTER	Register Keyword	Yes			
CPP_WARN_REINTERPRET_BASE_CLASS	Reinterpret Base Class	Yes			High
CPP_WARN_REORDER_CTOR	Reorder Constructor	Yes			
CPP_WARN_REORDER_INIT_LIST	Reorder Initializer List	Yes			High
CPP_WARN_RESERVED_IDENTIFIER	Reserved Identifier	Yes			
CPP_WARN_RESERVED_MACRO_IDENTIFIER	Reserved Macro Identifier	Yes			
CPP_WARN_RESERVED_USER_DEFINED_LITERAL	Reserved User Defined Literal	Yes			
CPP_WARN_RESTRICT_EXPANSION	Restrict Expansion	Yes			
CPP_WARN_RETAINED_LANGUAGE_LINKAGE	Retained Language Linkage	Yes			
CPP_WARN_RETURN_STACK_ADDRESS	Return Stack Address	Yes			High
CPP_WARN_	Return Type	Yes			High

RETURN_TYPE					
CPP_WARN_RETURN_TYPE_C_LINKAGE	Return Type C Linkage	Yes			High
CPP_WARN_REWRITE_NOT_BOOL	Rewrite Not Bool	Yes			
CPP_WARN_RTTI	Run-Time Type Information	Yes			
CPP_WARN_SARIF_FORMAT_UNSTABLE	SARIF Format Unstable	Yes			
CPP_WARN_SECTION	Section Attributes	Yes			
CPP_WARN_SELF_ASSIGN	Self Assign	Yes			
CPP_WARN_SELF_ASSIGN_FIELD	Self Assign Field	Yes			High
CPP_WARN_SELF_ASSIGN_OVERLOADED	Self Assign Overloaded	Yes			
CPP_WARN_SELF_MOVE	Self Move	Yes			
CPP_WARN_SENTINEL	Sentinel Attribute	Yes			
CPP_WARN_SERIALIZED_DIAGNOSTICS	Serialized Diagnostics	Yes			
CPP_WARN_SHADOW	Shadowing Identifiers	Yes			
CPP_WARN_SHADOW_FIELD	Shadowing Field	Yes			

CPP_WARN_SHADOW_FIELD_IN_CONSTRUCTOR	Shadowing Field in Constructor	Yes			
CPP_WARN_SHADOW_FIELD_IN_CONSTRUCTOR_MODIFIED	Shadowing Field in Constructor Modified	Yes			
CPP_WARN_SHADOW_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_OP_PARENTHESES	Shift Operator Parentheses	Yes			High
CPP_WARN_SHIFT_OVERFLOW	Shift Overflow	Yes			High
CPP_WARN_SHIFT_SIGN_OVERFLOW	Shift Sign Overflow	Yes			
CPP_WARN_SHORTEN_64_TO_32	Shorten Integer Type Width	Yes			
CPP_WARN_SIGN_COMPARE	Sign Compare	Yes			
CPP_WARN_SIGN_CONVERSION	Sign Conversion	Yes			

CPP_WARN_SIGNED_ENUM_BITFIELD	Signed Enum Bitfield	Yes			
CPP_WARN_SIGNED_UNSIGNED_WCHAR	Signed Unsigned Wchar	Yes			
CPP_WARN_SINGLE_BIT_BITFIELD_CONSTANT_CONVERSION	Single Bit Bitfield Constant Conversion	Yes			High
CPP_WARN_SIZEOF_ARRAY_ARGUMENT	Sizeof Array Argument	Yes			High
CPP_WARN_SIZEOF_ARRAY_DECAY	Sizeof Array Decay	Yes			High
CPP_WARN_SIZEOF_ARRAY_DIV	Sizeof Array Division	Yes			High
CPP_WARN_SIZEOF_POINTER_DIV	Sizeof Pointer Division	Yes			High
CPP_WARN_SIZEOF_POINTER_MEMACCESS	Sizeof Pointer Memory Access	Yes			High
CPP_WARN_SLASH_U_FILENAME	Slash U Filename	Yes			
CPP_WARN_SLH_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_IN_INLINE	Static in Inline	Yes			
CPP_WARN_STATIC_INLINE_EXPLICIT_INSTANTIATION	Static Inline Explicit Instantiation	Yes			High
CPP_WARN_STATIC_LOCAL_IN_INLINE	Static Local in Inline	Yes			High
CPP_WARN_STATIC_SELF_INIT	Static Self Init	Yes			High
CPP_WARN_STDLIBCXX_NOT_FOUND	LibStdC++ Headers Not Found	Yes			
CPP_WARN_STRICT_POTENTIALLY_DIRECT_SELECTOR	Strict Potentially Direct Selector	Yes			
CPP_WARN_STRICT_PROTOTYPES	Strict Prototypes	Yes			

CPP_WARN_STRICT_SELECTOR_MATCH	Strict Selector Match	Yes			
CPP_WARN_STRING_COMPARE	String Compare	Yes			High
CPP_WARN_STRING_CONCATENATION	String Concatenation	Yes			
CPP_WARN_STRING_CONVERSION	String Conversion	Yes			
CPP_WARN_STRING_PLUS_CHAR	String Plus Char	Yes			High
CPP_WARN_STRING_PLUS_INT	String Plus Int	Yes			High
CPP_WARN_STRLCPY_STRLCAT_SIZE	Strlcpy Strlcat Size	Yes			High
CPP_WARN_STRNCAT_SIZE	Strncat Size	Yes			High
CPP_WARN_SUGGEST_DESTRUCTOR_OVERRIDE	Suggest Destructor Override	Yes			
CPP_WARN_SUGGEST_OVERRIDE	Suggest Override	Yes			
CPP_WARN_SUPER_CLASS_METHOD_MISMATCH	Super Class Method Mismatch	Yes			
CPP_WARN_SUSPICIOUS_BZERO	Suspicious Argument for Bzero Function	Yes			
CPP_WARN_	Switch	Yes			High

SWITCH	Statements				
CPP_WARN_SWITCH_BOOL	Switch Bool	Yes			High
CPP_WARN_SWITCH_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_POINTER_COMPARE	Tautological Pointer Compare	Yes			High
CPP_WARN_T	Tautological	Yes			

AUTOLOGICAL_TYPE_LIMIT_COMPARE	Type Limit Compare				
CPP_WARN_TAUTOLOGICAL_UNDEFINED_COMPARE	Tautological Undefined Compare	Yes			High
CPP_WARN_TAUTOLOGICAL_UNSIGNED_CHAR_ZERO_COMPARE	Tautological Unsigned Char Zero Compare	Yes			
CPP_WARN_TAUTOLOGICAL_UNSIGNED_ENUM_ZERO_COMPARE	Tautological Unsigned Enum Zero Compare	Yes			
CPP_WARN_TAUTOLOGICAL_UNSIGNED_ZERO_COMPARE	Tautological Unsigned Zero Compare	Yes			
CPP_WARN_TAUTOLOGICAL_VALUE_RANGE_COMPARE	Tautological Value Range Compare	Yes			
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_SAF	Thread Safety Attributes	Yes			

ETY_ATTRIBUTES					
CPP_WARN_THREAD_SAFETY_BETA	Thread Safety Beta	Yes			
CPP_WARN_THREAD_SAFETY_NEGATIVE	Thread Safety Negative	Yes			
CPP_WARN_THREAD_SAFETY_PRECISE	Thread Safety Precise	Yes			
CPP_WARN_THREAD_SAFETY_REFERENCE	Thread Safety Reference	Yes			
CPP_WARN_THREAD_SAFETY_VERBOSE	Thread Safety Verbose	Yes			
CPP_WARN_TRIGRAPHS	Trigraphs	Yes			High
CPP_WARN_TYPE_SAFETY	Type Safety	Yes			High
CPP_WARN_TYPEDEF_REDEFINITION	Typedef Redefinition	Yes			High
CPP_WARN_TYPENAME_MISSING	Typename Missing	Yes			High
CPP_WARN_UNABLE_TO_OPEN_STATS_FILE	Unable to Open Stats File	Yes			
CPP_WARN_UNALIGNED_ACCESS	Unaligned Access	Yes			
CPP_WARN_UNALIGNED_QUALIFIER_I	Unaligned Qualifier Implicit Cast	Yes			

MPLICIT_CAST					
CPP_WARN_UNDEF	Undefined Macros	Yes			
CPP_WARN_UNDEF_PREFIX	Undefined Macros of a Certain Prefix	Yes			
CPP_WARN_UNDEFINED_BOOL_CONVERSION	Undefined Bool Conversion	Yes			High
CPP_WARN_UNDEFINED_FUNC_TEMPLATE	Undefined Function Template	Yes			
CPP_WARN_UNDEFINED_INLINE	Undefined Inline	Yes			High
CPP_WARN_UNDEFINED_INTERNAL	Undefined Internal	Yes			High
CPP_WARN_UNDEFINED_INTERNAL_TYPE	Undefined Internal Type	Yes			
CPP_WARN_UNDEFINED_REINTERPRET_CAST	Undefined Reinterpret Cast	Yes			
CPP_WARN_UNDEFINED_VAR_TEMPLATE	Undefined Var Template	Yes			High
CPP_WARN_UNDERALIGNED_EXCEPTION_OBJECT	Underaligned Exception Object	Yes			
CPP_WARN_UNEVALUATED_EXPRESSION	Unevaluated Expression	Yes			High

CPP_WARN_UNGUARDED_AVAILABILITY	Unguarded Availability	Yes			
CPP_WARN_UNGUARDED_AVAILABILITY_NEW	Unguarded Availability New	Yes			High
CPP_WARN_UNICODE	Unicode Escape Sequences	Yes			High
CPP_WARN_UNICODE_HOMOGLYPH	Unicode Homoglyph	Yes			High
CPP_WARN_UNICODE_WHITESPACE	Unicode Whitespace	Yes			High
CPP_WARN_UNICODE_ZERO_WIDTH	Unicode Zero Width	Yes			High
CPP_WARN_UNINITIALIZED	Uninitialized	Yes			
CPP_WARN_UNINITIALIZED_CONST_REFERENCE	Uninitialized Const Reference	Yes			
CPP_WARN_UNKNOWN_ARGUMENT	Unknown Argument	Yes			
CPP_WARN_UNKNOWN_ASSUMPTION	Unknown Assumption	Yes			
CPP_WARN_UNKNOWN_ATTRIBUTES	Unknown Attributes	Yes			
CPP_WARN_UNKNOWN_DIRECTIVES	Unknown Directives	Yes			High
CPP_WARN_	Unknown	Yes			High

UNKNOWN_ESCAPE_SEQUENCE	Escape Sequence				
CPP_WARN_UNKNOWN_PRAGMAS	Unknown Pragmas	Yes			
CPP_WARN_UNKNOWN_SANITIZERS	Unknown Sanitizers	Yes			
CPP_WARN_UNKNOWN_WARNING_OPTION	Unknown Warning Option	Yes			
CPP_WARN_UNNAMED_TYPE_TEMPLATE_ARGS	Unnamed Type Template Args	Yes			
CPP_WARN_UNNEEDED_INTERNAL_DECLARATION	Unneeded Internal Declaration	Yes			
CPP_WARN_UNNEEDED_MEMBER_FUNCTION	Unneeded Member Function	Yes			
CPP_WARN_UNQUALIFIED_STD_CAST_CALL	Unqualified Standard Cast Call	Yes			High
CPP_WARN_UNREACHABLE_CODE	Unreachable Code	Yes			
CPP_WARN_UNREACHABLE_CODE_BREAK	Unreachable Code Break	Yes			
CPP_WARN_UNREACHABLE_CODE_FALLTHROUGH	Unreachable Code Fallthrough	Yes			
CPP_WARN_UNREACHAB	Unreachable Code Generic	Yes			

LE_CODE_GENERIC_ASSOC	Assoc				
CPP_WARN_UNREACHABLE_CODE_LOOP_INCREMENT	Unreachable Code Loop Increment	Yes			
CPP_WARN_UNREACHABLE_CODE_RETURN	Unreachable Code Return	Yes			
CPP_WARN_UNSAFE_BUFFER_USAGE	Unsafe Buffer Usage	Yes			
CPP_WARN_UNSEQUENCED	Unsequenced Modifications	Yes			High
CPP_WARN_UNSUPPORTED_ABI	Unsupported ABI (Application Binary Interface)	Yes			
CPP_WARN_UNSUPPORTED_ABS	Unsupported Absolute Value Argument	Yes			
CPP_WARN_UNSUPPORTED_AVAILABILITY_GUARD	Unsupported Availability Guard	Yes			High
CPP_WARN_UNSUPPORTED_CB	Unsupported Compact Branches	Yes			
CPP_WARN_UNSUPPORTED_DLL_BASE_CLASS_TEMPLATE	Unsupported DLL Base Class Template	Yes			
CPP_WARN_UNSUPPORTED_FLOATIN	Unsupported Floating Point Option	Yes			

G_POINT_OPT					
CPP_WARN_UNSUPPORTED_FRIEND	Unsupported Friend	Yes			High
CPP_WARN_UNSUPPORTED_GPOPT	Unsupported GPopt (Gaussian Process Optimization)	Yes			
CPP_WARN_UNSUPPORTED_NAN	Unsupported Nan Argument	Yes			
CPP_WARN_UNSUPPORTED_TARGET_OPT	Unsupported Target Option	Yes			
CPP_WARN_UNSUPPORTED_VISIBILITY	Unsupported Visibility	Yes			
CPP_WARN_UNUSABLE_PARTIAL_SPECIALIZATION	Unusable Partial Specialization	Yes			
CPP_WARN_UNUSED_BUT_SET_PARAMETER	Unused but Set Parameter	Yes			
CPP_WARN_UNUSED_BUT_SET_VARIABLE	Unused but Set Variable	Yes			
CPP_WARN_UNUSED_COMMAND_LINE_ARGUMENT	Unused Command Line Argument	Yes			
CPP_WARN_UNUSED_COMPARISON	Unused Comparison	Yes			High
CPP_WARN_	Unused	Yes			

UNUSED_CONST_VARIABLE	Const Variable				
CPP_WARN_UNUSED_EXCEPTION_PARAMETER	Unused Exception Parameter	Yes			
CPP_WARN_UNUSED_FUNCTION	Unused Function	Yes			
CPP_WARN_UNUSED_GETTER_RETURN_VALUE	Unused Getter Return Value	Yes			
CPP_WARN_UNUSED_LABEL	Unused Label	Yes			
CPP_WARN_UNUSED_LAMBDA_CAPTURE	Unused Lambda Capture	Yes			
CPP_WARN_UNUSED_LOCAL_TYPEDEF	Unused Local Typedef	Yes			
CPP_WARN_UNUSED_MACROS	Unused Macros	Yes			
CPP_WARN_UNUSED_MEMBER_FUNCTION	Unused Member Function	Yes			
CPP_WARN_UNUSED_PARAMETER	Unused Parameter	Yes			
CPP_WARN_UNUSED_PRIVATE_FIELD	Unused Private Field	Yes			
CPP_WARN_UNUSED_PROPERTY_IVAR	Unused Property IVar (Instance Variable)	Yes			

CPP_WARN_UNUSED_RESULT	Unused Result	Yes			High
CPP_WARN_UNUSED_TEMPLATE	Unused Template	Yes			
CPP_WARN_UNUSED_VALUE	Unused Value	Yes			High
CPP_WARN_UNUSED_VARIABLE	Unused Variable	Yes			
CPP_WARN_UNUSED_VOLATILE_LVALUE	Unused Volatile Lvalue	Yes			High
CPP_WARN_USED_BUT_MARKED_UNUSED	Used but Marked Unused	Yes			
CPP_WARN_USER_DEFINED_LITERALS	User Defined Literals	Yes			High
CPP_WARN_USER_DEFINED_WARNINGS	User Defined Warnings	Yes			
CPP_WARN_VARARGS	Variadic Arguments	Yes			High
CPP_WARN_VARIADIC_MACROS	Variadic Macros	Yes			
CPP_WARN_VEC_ELEM_SIZE	Vector Element Size	Yes			
CPP_WARN_VECTOR_CONVERSION	Vector Conversion	Yes			
CPP_WARN_VEXING_PARSE	Vexing Parse Occurrences	Yes			High

CPP_WARN_VISIBILITY	Visibility of Declarations	Yes			High
CPP_WARN_VLA	VLA (Variable Length Array)	Yes			
CPP_WARN_VLA_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_PTR_DEREFERENCE	Void Pointer Dereference	Yes			High
CPP_WARN_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_POW	Xor Used as Power	Yes			High
CPP_WARN_ZERO_AS_NULL_POINTER_CONSTANT	Zero as Null Pointer Constant	Yes			
CPP_WARN_ZERO_LENGTH_ARRAY	Zero Length Array	Yes			
CTR50-CPP	Guarantee	Yes			High

	that container indices and iterators are within the valid range				
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
CTR57-CPP	Provide a valid ordering predicate	Yes			Low
CTR58-CPP	Predicate function	Yes			Low

	objects should not be mutable				
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 Wraparound	Yes			
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 or define a reserved identifier	No			
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	Yes			High

	namespaces				
DCL59-CPP	Do not define an unnamed namespace in a header file	Yes			Medium
DCL60-CPP	Obey the one-definition rule	Yes			High
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			
EFFECTIVEC	9. Virtual Call	Yes			

PP_09	in Constructor/ Destructor				
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			
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 correspondin g uses of new and delete	Yes			
EFFECTIVECPP_17	17. Store newed objects in smart pointers in	Yes			

	standalone statements				
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 return a reference when you must return an object	No			
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			
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	No			

	inheritance of implementation				
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-implemented-in-terms-of” through composition	No			
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 polymorphis	No			

	m				
EFFECTIVECPP_42	42. Understand the two meanings of typename	No			
EFFECTIVECPP_43	43. Know how to access names in templatized 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 templates when type conversions are desired	No			
EFFECTIVECPP_47	47. Use traits classes for information about types	No			
EFFECTIVECPP_48	48. Be aware of template metaprogramming	No			
EFFECTIVECPP_49	49. Understand	No			

	the behavior of the new-handler				
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 yourself with Boost	No			
ERR32-C	Do not rely on indeterminate values of errno	No			Low
ERR33-C	Detect and handle standard	Yes			High

	library errors				
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 function-try-block handler	Yes			Low
ERR54-CPP	Catch handlers should order their parameter types from most derived to least derived	Yes			Medium
ERR55-CPP	Honor exception specifications	Yes			Low
ERR56-CPP	Guarantee exception safety	No			
ERR57-CPP	Do not leak resources when	Yes			Low

	handling exceptions				
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
EXP43-C	Avoid undefined behavior when using	No			Medium

	restrict-qualified pointers				
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 of the correct type to va_start	Yes			Medium
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	No			High

	between characters read from a file and EOF or WEOF				
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	No			Low

	floating-point type				
FLP37-C	Do not use object representations to compare floating-point values	Yes			Low
HIS_01	1. Comment Density (COMF)	Yes			
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	Yes			

	(VOCF)				
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 (SNEW)	Yes			
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	No			Low

	integer precisions				
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 code	Yes	Automated	Required	
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	No	Automated	Required	

	code				
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, then that error information shall be tested	No	Non-automated	Required	
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	No	Non-automated	Required	

	common, defined interface				
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 octal or hexadecimal integer literals of unsigned type.	Yes	Automated	Required	
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	Yes	Automated	Required	

	declarations of an object or function shall have compatible types				
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 specifier	Yes	Automated	Required	
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	Yes	Automated	Required	

	parameter shall be token-for-token identical in all declarations and re-declarations				
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	
M4-5-3	Character Operators	Yes	Automated	Required	
M4-10-1	NULL shall not be used as an integer	Yes	Automated	Required	

	value				
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 conversion shall not	Yes	Automated	Required	

	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	Yes	Automated	Required	

	unsigned 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	Yes	Automated	Required	

	operand shall both address elements of the same array				
M5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes	Automated	Required	
M5-0-18	>, >=, <, <= 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	Yes	Automated	Required	

	class shall only be cast to a pointer to a derived class by means of dynamic_cast				
M5-2-3	Casts from a base class to a derived class should not be performed on polymorphic types	Yes	Automated	Advisory	
M5-2-6	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes	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	Yes	Automated	Required	

	other operators in an expression				
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	
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	Yes	Partially Automated	Required	

	less than the width in bits of the underlying type of the left hand operand.				
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	
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-	Yes	Partially	Required	

	point expressions shall not be directly or indirectly tested for equality or inequality		Automated		
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 forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes	Automated	Required	
M6-4-1	An if (condition) construct shall be followed by a compound	Yes	Automated	Required	

	statement. The else keyword shall be followed by either a compound statement, or another if statement				
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	
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 constant for the duration of the loop	Yes	Automated	Required	
M6-5-5	A loop-control-	Yes	Automated	Required	

	variable other than the loop-counter shall not be modified within condition or expression				
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 reference parameter in a function shall be declared as pointer to const or reference to const if the correspondin g object is not modified	Yes	Automated	Required	
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)	Yes	Automated	Required	

	shall not be used in header files.				
M7-4-1	Assembly Language Code Usage not Documented	Yes	Non-automated	Required	
M7-4-2	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	Yes	Non-automated	Required	

	object has ceased to exist.				
M8-0-1	Single Declarations	Yes	Automated	Required	
M8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Yes	Automated	Required	
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	Yes	Automated	Required	

	Function Returning Non-Const Pointer or Reference				
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 absolute positioning of bits representing a bit-field is required, then the behavior and packing of bit-fields shall be documented	No	Non-automated	Required	
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	Yes	Automated	Required	

	shall not be both virtual and non-virtual in the same hierarchy				
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	
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	
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	Yes	Automated	Required	

	members from this class or its bases				
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 single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last	Yes	Automated	Required	
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	Yes	Automated	Required	

	namespace.				
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 directives, except as operands to the defined operator	Yes	Automated	Required	
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	Yes	Automated	Required	

	and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related				
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 and objects shall not be reused	Yes	Automated	Required	
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>	Yes	Automated	Required	

	Library Functions				
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 <cstdio> shall not be used	Yes	Automated	Required	
MEM30-C	Do not access freed memory	No			High
MEM36-C	Do not modify the alignment of objects by calling realloc()	No			Low
MEM50-CPP	Do not access freed memory	No			High
MEM51-CPP	Properly deallocate dynamically	Yes			High

	allocated resources				
MEM52-CPP	Detect and handle memory allocation errors	Yes			High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No			High
MEM55-CPP	Honor replacement dynamic storage management requirements	No			
MEM57-CPP	Avoid using default operator new for over-aligned types	Yes			Medium
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	Yes			

	Parameters Count				
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 Reuse in Multiple C Name Spaces	Yes		Advisory	
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	Yes		Required	

	Names				
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 operand of # or ##	Yes		Required	
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	0-1-1 A	Yes		Required	

-1	project shall not contain unreachable code				
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 subsequently used	No		Required	
MISRA08_0-1-7	0-1-7 The value returned by a function having a non-void return type that is not an	Yes		Required	

	overloaded operator shall always be used				
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_0-2-1	0-2-1 An object shall not be assigned to an overlapping	No		Required	

	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	No		Required	

	to ISO/IEC 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 comment.	Yes		Required	
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	Yes		Required	

	qualification, if any) shall be a unique identifier				
MISRA08_2-10-5	2-10-5 The identifier name of a non-member object or function with static storage duration should not be reused	Yes		Advisory	
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	Yes		Required	

	literals of unsigned type.				
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_3-1-1	3-1-1 It shall be possible to include any header file in multiple translation units without violating the One Definition Rule	Yes		Required	
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	Yes		Required	

	function that is used in multiple translation units shall be declared in one and only one file				
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 linkage shall be declared in a header file	Yes		Required	
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	Yes		Required	

	parameter shall be token-for-token identical in all declarations and re-declarations				
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-point values shall not be used	Yes		Required	
MISRA08_4-5-1	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 ==	Yes		Required	

	and !=, the unary & operator, and the conditional operator				
MISRA08_4-5-2	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 operators <, <=, >, >=	Yes		Required	
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-	5-0-1 The	Yes		Required	

0-1	value of an expression shall be the same under any order of evaluation that the standard permits				
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 integral conversion shall not change the signedness of the underlying type	Yes		Required	
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	
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	
MISRA08_5-0-13	5-0-13 The condition of an if-statement and the	No		Required	

	condition of an iteration-statement shall have type bool				
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	No		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 be applied to objects of pointer type, except where they point to the same array	Yes		Required	
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	No		Required	

	class shall only be cast to a pointer to a derived class by means of dynamic_cast				
MISRA08_5-2-3	5-2-3 Casts from a base class to a derived class should not be performed on polymorphic types	Yes		Advisory	
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,	Yes		Required	

	including a pointer to function type				
MISRA08_5-2-7	5-2-7 An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly	No		Required	
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-	5-2-12 Array	Yes		Required	

2-12	to Pointer Decay				
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 to an expression whose underlying type is unsigned	No		Required	
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	Yes		Required	

	underlying type of the left hand operand.				
MISRA08_5-14-1	5-14-1 The right hand operand of a logical && or || operator shall not contain side effects	No		Required	
MISRA08_5-17-1	5-17-1 The semantic equivalence between a binary operator and its assignment operator form shall be preserved	No		Required	
MISRA08_5-18-1	5-18-1 The comma operator shall not be used	No		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	No		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 the first character following the null statement is a white-space character	Yes		Required	
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	Yes		Required	

	followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement				
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	No		Required	
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	Yes		Required	

	shall terminate every non-empty switch-clause				
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	No		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 shall not have floating-point type	Yes		Required	
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	Yes		Required	

	operand to <=, <, > or >=				
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-variable other than the loop-counter which is modified in statement shall have type bool	Yes		Required	
MISRA08_6-6-1	6-6-1 Any label	Yes		Required	

	referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement				
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 point of exit at the end of the function	Yes		Required	

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	Yes		Required	

	function main				
MISRA08_7-3-3	7-3-3 There shall be no unnamed namespaces in header files.	Yes		Required	
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	Yes		Required	

	language shall be encapsulated and isolated.				
MISRA08_7-5-1	7-5-1 A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes		Required	
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	Yes		Advisory	

	should not call themselves, either directly or indirectly.				
MISRA08_8-0-1	8-0-1 Single Declarations	Yes		Required	
MISRA08_8-3-1	8-3-1 Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Yes		Required	
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	Yes		Required	

	the function or it shall be preceded by &				
MISRA08_8-5-1	8-5-1 All variables shall have a defined value before they are used	Yes		Required	
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	Yes		Required	

	shall be made static, otherwise if it can be made const then it shall be made const.				
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 the behavior and packing of bit-fields shall be documented	No		Document	
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	Yes		Required	

	and non-virtual in the same hierarchy				
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 shall be declared with the virtual keyword.	Yes		Required	
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	Yes		Advisory	

	and virtual base classes				
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 protected or private in an abstract class	No		Required	
MISRA08_14-5-1	14-5-1 A non-member 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	Yes		Required	

	with a single parameter that is a generic parameter				
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 base shall be referred to using a qualified-id or this->	No		Required	
MISRA08_14-6-2	14-6-2 The function or operator chosen by overload resolution shall resolve to a function declared previously in	No		Required	

	the translation unit				
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 the declaration of their primary template	No		Required	
MISRA08_14-8-1	14-8-1 Overloaded function templates shall not be explicitly specialized	Yes		Required	

MISRA08_14-8-2	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	15-0-1 Exceptions shall only be used for error handling	No		Document	
MISRA08_15-0-2	15-0-2 An exception object should not have pointer type	Yes		Advisory	
MISRA08_15-0-3	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	15-1-1 The assignment-expression of a throw statement shall not itself cause an exception to be thrown	Yes		Required	
MISRA08_15-	15-1-2 NULL	Yes		Required	

1-2	Throw				
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 paths that could lead to that point	No		Required	
MISRA08_15-3-6	15-3-6 Order of Catch Blocks with	Yes		Required	

	Derived Classes				
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 #define'd or #undef'd in the global	Yes		Required	

	namespace				
MISRA08_16-0-3	16-0-3 Preprocessor #undef	Yes		Required	
MISRA08_16-0-4	16-0-4 Function-like macros shall not be defined	Yes		Required	
MISRA08_16-0-5	16-0-5 Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA08_16-0-6	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	16-0-7 Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as	Yes		Required	

	operands to 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 backslash character should not occur in a header file name	Yes		Advisory	
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-	17-0-1	Yes		Required	

0-1	Reserved or Standard Library Identifiers as Macros				
MISRA08_17-0-2	17-0-2 The names of standard library macros and objects shall not be reused	Yes		Required	
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	Yes		Required	

	abort, exit, getenv and system from library <cstdlib> shall not be used				
MISRA08_18-0-4	18-0-4 The time handling functions of library <ctime> shall not be used	Yes		Required	
MISRA08_18-0-5	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 should be no unused parameters in functions	Yes		Advisory	
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	
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	Yes		Required	

	is "pointer to const-qualified char"				
MISRA12_8.2	8.2 Use Named Parameters and Prototype Form	Yes			
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 of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
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	Yes		Required	

	partially initialized				
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 used inappropriately in addition and subtraction operations	Yes		Required	
MISRA12_10.3	10.3 The value of an expression shall not be assigned to	Yes		Required	

	an object with a narrower essential type or of a different essential type category				
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 shall not be cast to a different essential type category or a wider	Yes		Required	

	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 conversion should not be performed from pointer to void into	Yes		Advisory	

	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 containing an increment (+	Yes		Advisory	

	+) 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	Yes		Required	

	an iteration-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			
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	Yes		Advisory	

	should not be applied to an expression of pointer type				
MISRA12_18.5	18.5 Declarations should contain no more than two levels of pointer nesting	Yes		Advisory	
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	Yes		Required	

	macro argument				
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 controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
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,	Yes		Required	

	shall only be used as an operand to these operators				
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 Functions	Yes		Required	
MISRA12_22.1	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	Yes		Mandatory	

	opened as read-only				
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 language should be documented	Yes		Advisory	
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	Yes		Advisory	

	be hidden				
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 should be called in an appropriate sequence (Partial)	Yes		Advisory	
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	Yes		Required	

	be used				
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 should not be used (Partial)	Yes		Advisory	
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	Yes		Advisory	

	unused type declarations				
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 (C++17)	No		Required	
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	No		Required	

	critical unspecified behaviour				
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-Splicing in // Comments	Yes		Required	
MISRA23_5.8	5.8 Identifiers that define objects or functions with external linkage shall	Yes		Required	

	be unique				
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 names shall be terminated	Yes		Required	
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 declared, its size should be explicitly	Yes		Advisory	

	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	

	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	Yes		Advisory	

	object with 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	Yes		Required	

	with automatic storage duration to an object with a greater lifetime				
MISRA23_6.8.4	6.8.4 Member functions returning references to their object should be ref-qualified appropriately	Yes		Advisory	
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 shall be appropriate	Yes		Required	
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	Yes		Required	

	an object unless the object's type is "pointer to const-qualified char				
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 Integer Constant Macros	Yes		Required	
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	Yes		Required	

	expression appropriately explicit				
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 and Prototype Form	Yes		Required	
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	Yes		Required	

	a pointer or by reference				
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 pointer type	Yes		Required	
MISRA23_8.2.7	8.2.7 Pointer to Integer Cast	Yes		Advisory	
MISRA23_8.2.8	8.2.8 An object 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 used	Yes		Advisory	
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	
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	Yes		Required	

	with an explicit alignment specification shall specify the same alignment				
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 operator should not be used	No		Advisory	
MISRA23_8.19.1	8.19.1 The comma operator shall not be used.	Yes		Advisory	
MISRA23_8.2	8.20.1 An	Yes		Advisory	

0.1	unsigned arithmetic operation with constant operands should not wrap				
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 selection-statement shall be acompound-statement	Yes		Required	
MISRA23_9.4	9.4 An	Yes		Required	

	element of an object shall not be initialized more than once				
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	
MISRA23_9.6	9.6 An	Yes		Required	

	initializer using chained designators shall not contain initializers without designators				
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 appropriately initialized before being accessed	Yes		Mandatory	
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	Yes		Required	

	inappropriately in addition and subtraction operations				
MISRA23_10.2.1	10.2.1 An enumeration shall be defined with an explicit underlying type	Yes		Required	
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	Yes		Advisory	

	in header files				
MISRA23_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	
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 performed between a pointer to an incomplete type and any other type	Yes		Required	
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 pointer to object	Yes		Advisory	
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	Yes		Required	

	qualification from the type pointed to by a pointer				
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	
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	Yes		Required	

	members of atomic objects shall not be directly accessed				
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 in the same hierarchy	Yes		Required	
MISRA23_13.3	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	Yes		Required	

	member functions shall use the virtual, override and final specifiers appropriately				
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	
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 controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes		Required	
MISRA23_15.0.2	15.0.2 User-provided	Yes		Advisory	

	copy and move member functions of a class should have appropriate signatures				
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 single argument shall be explicit	Yes		Required	
MISRA23_15.1.4	15.1.4 All direct, non-static data members of a class should	Yes		Required	

	be initialized before the class object is accessible				
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 or the last switch label of a switch statement	Yes		Required	

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.	17.9 Invalid	Yes		Mandatory	

9	_Noreturn				
MISRA23_17.10	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	Yes		Required	

	compound-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-	Yes		Required	

	unfriendly functions shall be noexcept				
MISRA23_18.5	18.5 Declarations should contain no more than two levels of pointer nesting	Yes		Advisory	
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 preprocessin	Yes		Required	

	g directive				
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 directives or comments	Yes		Advisory	
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	Yes		Required	

	as the #if, #ifdef or #ifndef directive to which they are related				
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 more than once	Yes		Required	
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	
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	Yes		Advisory	

	should not be used				
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 shall be enclosed in parentheses	Yes		Required	
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 preprocessin	Yes		Required	

	g directives shall be #define'd before evaluation				
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 functions atof, atoi, atol and atoll from library <cstdlib> shall not be used	Yes		Required	
MISRA23_21.2.2	21.2.2 The string handling functions from <cstring>, <cstdlib>, <cwchar>	Yes		Required	

	and <cinttypes> shall not be used				
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 management shall not be used	Yes		Required	
MISRA23_21.6.4	21.6.4 If a project defines either a sized or	Yes		Required	

	unsized version of a global operator delete, then both shall be defined				
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 <cstdarg> shall not be used	Yes		Required	
MISRA23_21.10.2	21.10.2 The standard header file <csetjmp> shall not be used	Yes		Required	
MISRA23_21.10.3	21.10.3 The facilities provided by the standard header file <csignal> shall not be used	Yes		Required	
MISRA23_21.13	21.13 Any value passed	Yes		Mandatory	

	to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF				
MISRA23_21.14	21.14 The Standard Library function memcmp shall not be used to compare null terminated strings	Yes		Required	
MISRA23_21.15	21.15 Incompatible Pointers	Yes		Required	
MISRA23_21.16	21.16 Invalid Memory Compare	Yes			
MISRA23_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	
MISRA23_21.18	21.18 Out of Bounds with string.h	Yes			

MISRA23_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 type	Yes		Mandatory	
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	Yes		Mandatory	

	declared in <tgmath.h> shall have an appropriate essential type				
MISRA23_21.23	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_21.24	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.25	21.25 Atomic Operations with Inconsistent Order	Yes		Required	
MISRA23_21.26	21.26 The Standard Library function mtx_timedlock() shall only be invoked on mutex objects of appropriate mutex type	Yes		Required	
MISRA23_22.	22.1 All	Yes		Required	

1	resources obtained dynamically by means of Standard Library 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 FILE Pointer	Yes		Mandatory	
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-	Yes		Required	

	function				
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 objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration	Yes		Required	
MISRA23_22.15	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_22.16	22.16 All mutex	Yes		Required	

	objects locked by a thread shall be explicitly unlocked by the same thread				
MISRA23_22.17	22.17 No thread shall unlock a mutex or call cnd_wait() or cnd_timedwait() for a mutex it has not locked before	Yes		Required	
MISRA23_22.18	22.18 Non-recursive mutexes shall not be recursively locked	Yes		Required	
MISRA23_22.19	22.19 Mutexes with One Condition Variable	Yes		Required	
MISRA23_23.11.1	23.11.1 The raw pointer constructors of std::shared_ptr and std::unique_ptr should not be used	Yes		Advisory	
MISRA23_24.5.1	24.5.1 The character handling functions from	Yes		Required	

	<cctype> and <cwctype> shall not be used				
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++ Standard Library functions localeconv, getenv, setlocale or strerror must only be used as if they have pointer to const-qualified type	Yes		Mandatory	
MISRA23_25.5.3	25.5.3 The pointer returned by the C++	Yes		Mandatory	

	Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function				
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 std::move shall be a non-const lvalue	Yes		Required	
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	Yes		Required	

	not be used while in a potentially moved-from state				
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 returns error information, then that error information shall be tested	Yes		Required	
MISRA23_DI	Directive 4.8	Yes		Advisory	

R_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				
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	
MISRA23_DIR_5.3	Directive 5.3 There shall be no dynamic	Yes		Required	

	thread creation				
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	
MISRA25_8.1	8.19 There	Yes		Advisory	

9	should be no external declarations in a source file				
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	Yes		Required	

	or _Atomic qualification from the type 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.	18.7 Flexible	Yes		Required	

7	array members shall not be declared				
MISRA25_18.10	18.10 Pointers to variably-modified array types shall not be used	Yes		Mandatory	
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	Yes		Required	

	used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation				
MISRA25_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	
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.	21.19 The	Yes		Mandatory	

19	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				
MISRA25_21.20	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_21.21	21.21 The Standard Library function system of <stdlib.h> shall not be used	Yes		Required	

MISRA25_21.24	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.25	21.25 All memory synchronization operations shall be executed in sequentially consistent order	Yes		Required	
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_22.11	22.11 A thread that was previously either joined or detached shall not be	Yes		Required	

	subsequently joined nor detached				
MISRA25_22.12	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_22.14	22.14 Thread synchronization objects shall be initialized before being accessed	No		Mandatory	
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.	23.1 A	Yes		Advisory	

1	generic selection should only be expanded from a macro				
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 selection	Yes		Required	
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	Yes		Required	

	information, then that error information shall be tested				
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 return a value from all exit paths	Yes			Medium
MSC53-CPP	Do not return from a function	Yes			Medium

	declared [[noreturn]]				
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 replacement	Yes			Low

	handler requirements				
OOP57-CPP	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 objects shared between POSIX threads with appropriate storage durations	Yes			Medium
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 -	Yes			

	Unchecked Parameters and Return Values				
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	Yes			

	#define'd or #undef'd within a block				
RECOMMENDED_08	Magic Numbers	Yes			
RECOMMENDED_09	Nested Comments	Yes			
RECOMMENDED_10	Overly Complex Functions	Yes			
RECOMMENDED_11	Trigraphs shall not be used	Yes			
RECOMMENDED_12	Unreachable Code	Yes			
RECOMMENDED_13	Unused Functions	Yes			
RECOMMENDED_14	Unused C and C++ Local Variables	Yes			
RECOMMENDED_15	Unused Static Globals	Yes			
RECOMMENDED_16	Variables should be commented	Yes			
RECOMMENDED_17	Upper limit shall not be modified within the bounds of the loop	Yes			
RECOMMENDED_19	Comments Indicating Future Fixes	Yes			
RECOMMENDED_20	Duplicate Code	Yes			
SIG35-C	Do not return from a computational exception	No			Low

	signal handler				
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 characters to unsigned char before converting to larger integer sizes	No			Medium
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	Yes			Low

	deallocation functions				