Understand™
by SciTools

	All Rules	Advisory Rules	Mandatory Rules	Recommended Rules	Required Rules
Understand % Coverage	96%	88%	60%	100%	93%
Understand Coverage	2,680	150	15	1	831
Total Rules	2,804	170	25	1	893

Checks

Check ID	Check Name	Supported	Automation	Category	Severity
A0-1-1	A project shall not contain instances of non-volatile variables being given values that are not subsequently used	Yes	Automated	Required	
A0-1-2	The value returned by a function shall be used	Yes	Automated	Required	
A0-1-3	Every function defined in an anonymous namespace, or static function with internal linkage, or private member function shall be used	Yes	Automated	Required	
A0-1-4	There shall	Yes	Automated	Required	

	be no unused named parameters in non-virtual functions				
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	
A1-1-2	A warning level of the compilation process shall be set in compliance with project policies	No	Non-automated	Required	
A1-4-3	All code should compile free	Yes	Automated	Advisory	

	of compiler warnings				
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 \ shall not occur as a last character of a C++ comment	Yes	Automated	Required	
A2-7-2	Sections of code shall not be "commented out"	Yes	Non-automated	Required	
A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods	Yes	Automated	Required	

	shall be preceded by documentation				
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	Yes	Automated	Advisory	

	linkage should not be reused				
A2-10-6	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope	Yes	Automated	Required	
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			
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 translation units without violating the One Definition Rule	Yes	Automated	Required	
A3-1-2	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx"	Yes	Automated	Required	
A3-1-3	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes	Automated	Advisory	
A3-1-4	When an array with external linkage is	Yes	Automated	Required	

	declared, its size shall be stated explicitly				
A3-1-5	A function definition shall only be placed in a class definition if (1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template	Yes	Partially Automated	Required	
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	Yes	Automated	Required	

	objects shall be constant-initialized				
A3-9-1	Fixed Width Integers	Yes	Automated	Required	
A4-5-1	Expressions with type enum or enum class shall not be used as operands to built-in and overloaded operators other than the subscript operator [ ], the assignment operator =, the equality operators == and ! =, the unary & operator, and the relational operators <, <=, >, >=	Yes	Automated	Required	
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	Yes	Automated	Required	

	shall be the same under any order of evaluation that the standard permits				
A5-0-2	Condition of if statement shall be bool	Yes	Automated	Required	
A5-0-3	No more than 2 levels of pointer indirection	Yes	Automated	Required	
A5-0-4	Pointer arithmetic shall not be used 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	Yes	Automated	Required	

	lambda expression				
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	Yes	Automated	Required	

	shall not be used				
A5-2-3	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes	Automated	Required	
A5-2-4	reinterpret_cast shall not be used	Yes	Automated	Required	
A5-2-5A	An array or container shall not be accessed beyond its range (Part A)	Yes	Automated	Required	
A5-2-5B	An array or container shall not be accessed 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	Yes	Automated	Required	

	Pointers to Incomplete Class Types				
A5-5-1	A pointer to member shall not access non-existent class members	Yes	Automated	Required	
A5-6-1A	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-6-1B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes	Automated	Required	
A5-10-1	A pointer to member virtual 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	Yes	Automated	Required	

	not have floating-point type				
A6-5-3	Do statements should not be used	Yes	Automated	Advisory	
A6-5-4	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes	Automated	Advisory	
A6-6-1	The goto statement shall not be used.	Yes	Automated	Required	
A7-1-1	Constexpr or const specifiers shall be used for immutable data declaration	Yes	Automated	Required	
A7-1-2	The constexpr specifier shall be used for values that can be determined at compile time	Yes	Automated	Required	
A7-1-3	CV-qualifiers shall be	Yes	Automated	Required	

	placed on the right hand side of the type that is a typedef or a using name				
A7-1-4	The register keyword shall not be used	Yes	Automated	Required	
A7-1-5	The auto specifier shall not be used apart from following cases: (1) to declare that a variable has the same type as return type of a function call, (2) to declare that a variable has the same type as initializer of non-fundamental type, (3) to declare parameters of a generic lambda expression, (4) to declare a function template using trailing return type syntax	Yes	Automated	Required	

A7-1-6	The typedef specifier shall not be used	Yes	Automated	Required	
A7-1-7	Each expression statement and identifier declaration 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, structure, or enumeration shall not be declared in the definition of its type	Yes	Automated	Required	
A7-2-1	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes	Automated	Required	
A7-2-2	Enumeration underlying base type shall be	Yes	Automated	Required	

	explicitly defined				
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	
A7-3-1	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	Yes	Automated	Required	

	the [[noreturn]] attribute shall not return				
A8-2-1	When declaring function templates, the trailing return type syntax shall be used if the return type depends on the type of parameters.	Yes	Automated	Required	
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	Yes	Automated	Required	

	&& shall always be moved from.			
A8-4-6	"forward" parameters declared as T && shall always be forwarded.	Yes	Automated	Required
A8-4-7	"in" parameters for "cheap to copy" types shall be passed by value.	Yes	Automated	Required
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_p	Yes	Automated	Required

	tr				
A8-4-13	Invalid Use of std::shared_ptr	Yes	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	
A10-0-1	Public Inheritance	Yes	Non-automated	Required	

	not Used in a "is-a" 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	
A10-3-3	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	Yes	Non-automated	Advisory	

	based on 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,	Yes	Automated	Required	

	then all 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-	Yes	Automated	Required	

	defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.				
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	No	Automated	Required	

	and require all the 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	Yes	Automated	Required	

	implicitly defined special member 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	Yes	Automated	Required	

	members and base classes using copy semantics.				
A12-8-5	A copy assignment and a move assignment operators shall handle self-assignment.	Yes	Automated	Required	
A12-8-6	Copy and move constructors and copy assignment and move assignment operators shall be declared protected or defined "=delete" in base class.	Yes	Automated	Required	
A12-8-7	Assignment operators should be declared with the ref-qualifier &.	Yes	Automated	Advisory	
A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more	Yes	Automated	Required	

	letters				
A13-1-3	User defined literals operators shall only perform conversion of passed parameters	Yes	Automated	Required	
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	Yes	Automated	Required	

	shall also be implemented				
A13-5-2	All user-defined conversion operators shall be defined explicit	Yes	Automated	Required	
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	Yes	Automated	Required	

	digits, (3) for binary, every 4 digits				
A14-1-1	A template should check if a specific template argument is suitable for this template	Yes	Non-automated	Advisory	
A14-5-1	A template constructor 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	Yes	Automated	Advisory	

	union type declarations				
A14-7-1	A type used as a template argument shall provide all members that are used by the template	Yes	Automated	Required	
A14-7-2	Template specialization shall be declared in the same file as the primary template	Yes	Automated	Required	
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	No	Partially Automated	Required	

	either the strong guarantee or the nothrow guarantee				
A15-0-3	Exception safety guarantee of a called function shall be considered	No	Non-automated	Required	
A15-0-4	Unchecked exceptions shall be used to represent errors from which the caller cannot reasonably be expected to recover.	No	Non-automated	Required	
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	No	Partially Automated	Required	

	shall guarantee a deterministic worst-case time execution time				
A15-0-8	A worst-case execution time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions processing	No	Non-automated	Required	
A15-1-1	Only instances of types derived from 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	Yes	Partially Automated	Required	

	throw, the function shall place all objects/ resources that the function constructed in valid states or it shall delete them				
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	
A15-3-3	Unhandled Exceptions on Main Function	Yes	Partially Automated	Required	
A15-3-4	Catch-all (ellipsis and std::exceptio	Yes	Non-automated	Required	

	n) 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				
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 noexcept(<truecondition>), then it shall	Yes	Automated	Required	

	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 overriders.	Yes	Automated	Required	

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 include	Yes	Automated	Required	

	directives (slow)				
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	
A17-6-1	Non-standard entities shall not be added to standard namespaces	Yes	Automated	Required	
A18-0-1	The C library facilities shall only be accessed through C++ library headers	Yes	Automated	Required	
A18-0-2	The error state of a conversion from string to a numeric value shall be checked	Yes	Automated	Required	
A18-0-3	Library <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	Yes	Automated	Required	

	used			
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 noexcept function call operator	Yes	Automated	Required
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	Yes	Automated	Required

	allocate the memory				
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	
A18-5-7	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	
A18-5-11	operator "new" and operator "delete" shall be defined together	Yes	Automated	Required	
A18-9-1	The std::bind shall not be used	Yes	Automated	Required	
A18-9-2	Forwarding values to other functions	Yes	Automated	Required	

	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_ptr shall be used to	Yes	Automated	Required	

	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 char	Yes	Automated	Required	
A23-0-1	An iterator	Yes	Automated	Required	

	shall not be implicitly converted to const_iterator				
A25-1-1	Predicate Function Objects Copied Incorrectly	Yes	Automated	Required	
A26-5-1	Pseudorandom numbers shall not be generated using std::rand()	Yes	Automated	Required	
A26-5-2	Random number engines shall not be default-initialized	Yes	Automated	Required	
A27-0-1	Inputs from independent components shall be validated	Yes	Non-automated	Required	
A27-0-3	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 Staments (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
ARR36-C	Do not subtract or compare two pointers that do not refer to the same array	Yes			Medium
ARR37-C	Do not add or subtract an integer to a pointer to a non-array object	Yes			Medium
ARR38-C	Guarantee that library functions do not form invalid pointers	No			High
ARR39-C	Do not add or subtract a scaled integer to a pointer	Yes			High
CON30-C	Clean up	Yes			Medium

	thread-specific storage				
CON31-C	Do not destroy a mutex while it is locked	Yes			Medium
CON32-C	Prevent data races when accessing bit-fields from multiple threads	No			Medium
CON33-C	Avoid race conditions when using library functions	Yes			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
CON36-C	Wrap functions that can spuriously wake up in a loop	Yes			Low
CON37-C	Do not call signal() in a multithreaded program	Yes			Low
CON38-C	Preserve	Yes			Low

	thread safety and liveness when using condition variables				
CON39-C	Do not join or detach a thread that was previously joined or detached	Yes			Low
CON40-C	Do not refer to an atomic variable twice in an expression	Yes			Medium
CON41-C	Wrap functions that can fail spuriously in a loop	Yes			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 released on exceptional conditions	Yes			Low
CON52-CPP	Prevent data races when accessing bit-fields from multiple threads	Yes			Medium

CON53-CPP	Avoid deadlock by locking in a predefined order	No			Low
CON54-CPP	Wrap functions that can spuriously wake up in a loop	Yes			Medium
CON55-CPP	Preserve thread safety and liveness when using condition variables	Yes			Low
CON56-CPP	Do not speculatively lock a non-recursive mutex that is already owned by the calling thread	Yes			Low
CPP_A000	Assembler instructions only use asm keyword	Yes			
CPP_A001	Assembly language shall be encapsulated and isolated.	Yes			
CPP_A004	Parameter of assignment operator is a constant reference	Yes			
CPP_A005	Move and copy assignment	Yes			

	operators shall either move or respectively copy base classes and data members of a class, without any side effects				
CPP_A006	The asm declaration shall not be used.	Yes			
CPP_A007	Assembler instructions shall only be introduced using the asm declaration.	Yes			
CPP_A008	Assembly Language Code Usage not Documented	Yes			
CPP_A009	User-defined copy and move assignment operators should use user-defined no-throw swap function.	Yes			
CPP_A010	Move constructor shall not initialize its class	Yes			

	members and base classes using copy semantics.				
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, Unsigned, or Signed Bit-fields	Yes			
CPP_B001	(Fuzzy parser) Bit-fields shall only be declared with an appropriate	Yes			

	type				
CPP_B002	Enum Bit-fields	Yes			
CPP_B003	The underlying bit representations of floating-point values shall not be used	Yes			
CPP_B004	(Fuzzy parser) Named bit-fields with signed integer type shall have a length of more than one bit.	Yes			
CPP_B005	(Fuzzy parser) Single-bit named bit fields shall not be of a signed type	Yes			
CPP_B006	Bit-field Length	Yes			
CPP_C000	Commented Out Code	Yes			
CPP_C001	Line-splicing shall not be used in // comments	Yes			
CPP_C002	No Nested Comments	Yes			
CPP_C003	Only use /* comments	Yes			
CPP_C004	Parameter of copy	Yes			

	constructor is a constant 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	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member	Yes			
CPP_C009	Explicit Constructors	Yes			
CPP_C010	Incomplete constructor initialization list	Yes			

CPP_C011	An object's dynamic type shall not be used from the body of its constructor or destructor	Yes			
CPP_C012	Virtual Function Call In Constructor	Yes			
CPP_C013	The statement forming the body of a switch, while, do ... while or for statement shall be a compound 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	Yes			

	compound statement, or another if statement				
CPP_C017	The body of an iteration-statement or a selection-statement shall be a compound-statement	Yes			
CPP_C018	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes			
CPP_C019	A loop-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	Yes			

	shall only be used as an operand to <=, <, > or >=				
CPP_C021	The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes			
CPP_C022	The loop-counter shall not be modified within condition or statement	Yes			
CPP_C023	The goto statement shall jump to a label declared later in the same function body	Yes			
CPP_C024	No Continue Statements	Yes			
CPP_C025	Goto Statements	Yes			
CPP_C026	There should be no more than one break or goto statement used to terminate any iteration	Yes			

	statement				
CPP_C027	Member data in non-POD class types shall be private	Yes			
CPP_C028	A null statement shall only occur on a line by itself	Yes			
CPP_C029	Single exit point at end	Yes			
CPP_C030	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes			
CPP_C031	Switch Has Default	Yes			
CPP_C032	Every switch statement shall have at least two switch-clauses	Yes			
CPP_C033	An unconditional throw or break statement shall terminate every non-empty switch-	Yes			

	clause				
CPP_C034	Unreachable Code	Yes			
CPP_C035	No Backslash at End of Comment	Yes			
CPP_C036	If statements shall not have assignments in the conditions	Yes			
CPP_C037	Documentati on	Yes			
CPP_C038	Before preprocessin g, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character	Yes			
CPP_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 than loop-counter initialization and modification	Yes			
CPP_C043	Checked exceptions that could be thrown from a function shall be specified together with the function declaration and they shall be identical in all function declarations	Yes			

	and for all its overriders.				
CPP_C044	Continue Statement Used in a not Well-formed For Loop	Yes			
CPP_C046	Switch Statement not Well-formed	Yes			
CPP_C047	All if and else if constructs shall be terminated with an else clause	Yes			
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	Yes			

	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				
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	Yes			

	and respectively copy base classes and data members of a class, without any side effects				
CPP_CF000	The condition of a switch statement shall not have bool type	Yes			
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	Yes			

	Fixed Limits				
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_CT_BUG PRONE_ASS ERT_SIDE_EF FECT	Assert Side Effect	Yes			High
CPP_CT_BUG PRONE_BRA NCH_CLONE	Branch Clone	Yes			High
CPP_CT_BUG PRONE_COP Y_CONSTRU	Copy Constructor Init	Yes			High

CTOR_INIT						
CPP_CT_BUG PRONE_INFI NITE_LOOP	Infinte Loop	Yes				High
CPP_CT_BUG PRONE_MAC RO_REPEATE D_SIDE_EFFE CTS	Macro Side Effects	Yes				High
CPP_CT_BUG PRONE_NOT _NULL_TER MINATED_RE SULT	Missing Null Terminator	Yes				High
CPP_CT_BUG PRONE_RED UNDANT_BR ANCH_COND ITION	Redundant Condition	Yes				High
CPP_CT_MO DERNIZE_US E_DEFAULT_ MEMBER_INI T	Default Member Init	Yes				
CPP_CT_MO DERNIZE_US E_EQUALS_ DEFAULT	Default Member Function	Yes				
CPP_CT_MO DERNIZE_US E_EQUALS_ DELETE	Delete Member Function	Yes				
CPP_CT_MO DERNIZE_US E_NULLPTR	Null Pointer Keyword	Yes				
CPP_CT_REA DABILITY_DE LETE_NULL_ POINTER	Delete Null Pointer	Yes				High
CPP_CT_REA DABILITY_RE DUNDANT_C	Redundant Cast	Yes				High

ASTING					
CPP_D000	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes			
CPP_D002	Single Declarations	Yes			
CPP_D003	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation	Yes			
CPP_D004	A u or U suffix shall be applied to all integer constants that are represented in an unsigned type	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	Yes			

	Virtual Bases				
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	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier	Yes			
CPP_D010	= construct in enumerator list shall only be used on either the first item alone, or all	Yes			

	items explicitly.				
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 the || operator shall not be overloaded	Yes			
CPP_D020	The lowercase character L shall not be	Yes			

	used in a literal suffix				
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_D024	The restrict type qualifier shall not be used	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	Yes			

	unique				
CPP_D029	Destructor Set Data Ptr to 0	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	Yes			

	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				
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_D055	All Checks/ Language Specific/C and C++/ Destructors/ Non-Virtual Destructors in Base Classes	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	Yes			

	used.				
CPP_D060	If a class declares a copy or move operation, or a destructor, either via "=default", "=delete", or via a user-provided declaration, then all others of these five special member functions shall be declared as well.	Yes			
CPP_D061	Constructors shall explicitly initialize all virtual base classes, all direct non-virtual base classes and all non-static data members.	Yes			
CPP_D062	Both NSDMI and a non-static member initializer in a constructor shall not be used in the same type.	Yes			

CPP_D063	If all user–defined constructors of a class initialize data members with constant values that are the same across all constructors, then data members shall be initialized using NSDMI instead.	Yes			
CPP_D064	All constructors that are callable with a single 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	Yes			

	class should be declared final.				
CPP_D067	All class data members that are initialized by the constructor shall be initialized using member initializers.	Yes			
CPP_D068	If the behavior of a user-defined special member function is identical to implicitly defined special member function, then it shall be defined =default or be left undefined.	Yes			
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	Yes			

	template				
CPP_D071	All user-provided class destructors, deallocation functions, move constructors, move assignment operators and swap functions shall not exit with an exception. A noexcept exception specification shall be added to these functions as appropriate	Yes			
CPP_D072	Non-standard entities shall not be added to standard namespaces	Yes			
CPP_D073	There shall be one variable declaration per line	Yes			
CPP_D074	An external variable shall not be set to a value where it is being	Yes			

	declared				
CPP_D075	Incorrect Order of Initialization	Yes			
CPP_D076	If a class requires a virtual destructor but has nothing to undo from a constructor, an empty implementation should be provided.	Yes			
CPP_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 constants that are used as argument or return variables shall be placed in an include file	Yes			
CPP_DD006	Initializer lists shall be written in the order in which they are declared	Yes			
CPP_DD007	The private keyword should be used in class definitions	Yes			
CPP_DD008	Nesting template class definitions should be avoided.	Yes			
CPP_DD009	Assignment operators should check for self-assignment	Yes			
CPP_DD010	The use of friend classes should be avoided	Yes			

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

	syntactically ambiguous declarations				
CPP_DD017	Avoid cycles during initialization of static objects	Yes			
CPP_DD018	Obey the one-definition rule	Yes			
CPP_DD019	Arrays shall not be partially initialized	Yes			
CPP_DD020	An element of an object shall not be initialized more than once	Yes			
CPP_DD021	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes			
CPP_DD022	Make sure that objects are initialized before they are used	Yes			
CPP_DD023	Use the same form in corresponding uses of new and	Yes			

	delete				
CPP_DD024	Postpone variable definitions as long as possible	Yes			
CPP_DD025	Avoid hiding inherited names	Yes			
CPP_DD026	Never redefine an inherited non-virtual function	Yes			
CPP_E000	A class type exception shall always be caught by reference	Yes			
CPP_E001	There should be at least one exception handler to catch all otherwise unhandled exceptions	Yes			
CPP_E003	Catch Const References	Yes			
CPP_E004	Destructors Not Throw Exceptions	Yes			
CPP_E005	An empty throw (throw;) shall only be used in the compound-statement of a catch handler	Yes			

CPP_E006	Order of Catch Blocks with Derived Classes	Yes			
CPP_E007	An exception object should not have pointer type	Yes			
CPP_E008	Exceptions shall be raised only after start-up and before termination of the program	Yes			
CPP_E009	Exceptions thrown shall be the type indicated by the function	Yes			
CPP_E011	No "errno" allowed	Yes			
CPP_E012	NULL shall not be thrown explicitly	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	Yes			

	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				
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	Yes			

	operators == and ! =, the unary & operator, and the relational operators <, <=, >, >=				
CPP_E019	Avoid Trigraphs	Yes			
CPP_E020	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes			
CPP_E021	Octal and Hexadecimal Sequences	Yes			
CPP_E022	Escape sequences are standardized	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_E027	Only those escape sequences that are defined in	Yes			

	ISO/IEC 14882:2014 shall be used			
CPP_E028	Hexadecimal constants should be upper case	Yes			
CPP_E029	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes			
CPP_E030	Concatenating String Literals of Different Encodings	Yes	Automated	Required	
CPP_E031	Traditional C-style casts shall not be used	Yes			
CPP_E032	Infeasible Paths	Yes			
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	Yes			

	be included in every lambda expression				
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	Yes			

	increment (++) and decrement (--) operators shall not be mixed with other operators in an expression					
CPP_E044	Each operand of the ! operator, the logical && or the logical || operators shall have type bool	Yes				
CPP_E045	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes				
CPP_E046	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes				
CPP_E047	The ternary conditional	Yes				

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

	should not contain persistent side effects				
CPP_E053	Empty Throw	Yes			
CPP_E054	NULL Throw	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 thrown	Yes			
CPP_E058	An exception object shall not be a pointer	Yes			
CPP_E059	All thrown exceptions should be unique	Yes			
CPP_E060	If a function exits with an exception, then before a throw, the function shall place all objects/ resources that the function	Yes			

	constructed in valid states or it shall delete them				
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 that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines	Yes			
CPP_E064	Unhandled	Yes			

	Exceptions on Main Function				
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_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	Yes			

	operands of an operator in which the usual arithmetic conversions are performed shall have the same 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	Yes			

	type				
CPP_E080	A cast 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 performed between a pointer to object and an integer type	Yes			
CPP_E082	Initializer lists shall not contain persistent side effects	Yes			
CPP_E083	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes			
CPP_E084	The macro NULL shall be the only permitted form of integer null pointer	Yes			

	constant				
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_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 of non-throwing function shall	Yes			

	contain noexcept specification				
CPP_EH006	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes			
CPP_EH007	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point	Yes			
CPP_EH008	Exceptions thrown across execution boundaries	Yes			
CPP_EH009	New Method Throwing an Exception	Yes			
CPP_EH010	Use Assertion Statements	Yes			
CPP_EH011	Catch exceptions by lvalue reference	Yes			
CPP_F000	All prototype parameters	Yes			

	must have an identifier.				
CPP_F001	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once	Yes			
CPP_F002	Const member functions shall not return non-const pointers or references to class-data	Yes			
CPP_F003	Unused Functions	Yes			
CPP_F004	Functions with no parameters need explicit void keyword	Yes			
CPP_F005	Declare functions at file scope	Yes			
CPP_F006	A Function identifier shall either be used to call the function or it	Yes			

	shall be preceded by &				
CPP_F007	Functions must not return objects by value.	Yes			
CPP_F008	Functions shall not be defined using the ellipsis notation	Yes			
CPP_F009	Use Named Parameters and Prototype Form	Yes			
CPP_F010	Functions shall not be declared implicitly	Yes			
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	Member functions shall not return non-const handles to class-data	Yes			
CPP_F014	If a member	Yes			

	function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const				
CPP_F015	Missing parameter name in function declarations	Yes			
CPP_F016	variable numbers of arguments shall not be used.	Yes			
CPP_F017	Overloaded function templates shall not be explicitly specialized	Yes			
CPP_F018	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.	Yes			
CPP_F019	A pointer or reference	Yes			

	parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified				
CPP_F020	use the same identifier in definition and declaration of functions.	Yes			
CPP_F021	The features of <stdarg.h> shall not be used	Yes			
CPP_F022	Objects should not be passed by reference	Yes			
CPP_F023	A function parameter should not be modified	Yes			
CPP_F024	The value returned by a function shall be used	Yes			
CPP_F025	All functions with void return type shall have external side effect(s)	Yes			
CPP_F026	Every function defined in an anonymous	Yes			

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

	and unsized versions shall be defined				
CPP_F031	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes			
CPP_F032	A function shall not return a reference or a pointer to a parameter that is passed by reference to const.	Yes			
CPP_F033	Always return a value in non-void functions	Yes			
CPP_F034	Trivial accessor and mutator functions should be inlined.	Yes			
CPP_F035	Non-virtual public or protected member functions shall not be redefined in	Yes			

	derived classes				
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-const raw pointers or references to private or protected data owned by the class	Yes			
CPP_F042	If two	Yes			

	opposite operators are defined, one shall be defined in terms of the other				
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 participate in overload resolution for a single argument of the enclosing class type	Yes			
CPP_F048	A non-	Yes			

	member generic operator shall only be declared in a namespace that does not contain class (struct) type, enum type or union type declarations				
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	Yes			

	not be used				
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-length argument lists should not be used	Yes			
CPP_F060	A method that does not	Yes			

	change the visible properties of a class shall be declared const				
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 as class state variables	Yes			
CPP_F064	Member functions	Yes			

	should always be declared const unless they modify state variables				
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_F069	A signal handler must be a plain old function	Yes			
CPP_F070	Consider alternatives to virtual functions	Yes			
CPP_H001	The backslash character should not occur in a header file name	Yes			
CPP_H002	The ', ", /* or // characters shall not occur in a	Yes			

	header file name				
CPP_H003	Definitions in Header Files	Yes			
CPP_H004	There shall be no unnamed namespaces in header files.	Yes			
CPP_H005	Objects or functions with external linkage shall be declared in a header file	Yes			
CPP_H006	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule	Yes			
CPP_H007	Unnecessary #Includes	Yes			
CPP_H008	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes			

CPP_H009	Header files, that are defined locally in the project, shall have a file name extension of one of: ".h", ".hpp" or ".hxx"	Yes			
CPP_H010	Header File Name	Yes			
CPP_H011	Absolute path names shall not be used for header files	Yes			
CPP_H012	All references to header files shall be listed one per line	Yes			
CPP_H013	Names of private header files should not be identical to names of library header files	Yes			
CPP_H014	All public header files shall be capable of being included by a C++ file as well as a C file	Yes			
CPP_H016	If prototypes, typedefs,	Yes			

	macros, structure definitions, or enums are needed in multiple modules, they shall be placed in header files				
CPP_H017	C++ version of the header file should be used	Yes			
CPP_H018	When including C Standard Library header files, C++ Standard Library header files without a '.h' appended should be used	Yes			
CPP_H019	Forward referencing should be used, when appropriate, over direct inclusion when documenting dependencies within a header file.	Yes			
CPP_H020	The standard header file <tgmath.h>	Yes			

	shall not be used				
CPP_H021	The standard header file <setjmp.h> shall not be used	Yes			
CPP_I000	A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes			
CPP_I001	Different identifiers shall be typographically unambiguous	Yes			
CPP_I002	External identifiers shall be distinct	Yes			
CPP_I003	Identifiers that define objects or functions with external linkage shall be unique	Yes			
CPP_I005	Identifier name reuse	Yes			
CPP_I006	Identifiers shall be distinct from macro names	Yes			
CPP_I007	Identifiers declared in the same scope and	Yes			

	name space shall be distinct				
CPP_I008	Identifiers that define objects or functions with internal linkage should be unique	Yes			
CPP_I009	Macro identifiers shall be distinct	Yes			
CPP_I010	The identifier name of a non-member object or function with static storage duration should not be reused	Yes			
CPP_I011	Identifier name significance	Yes			
CPP_I012	Static name reuse	Yes			
CPP_I013	A tag name shall be a unique identifier	Yes			
CPP_I014	A typedef name shall be a unique identifier.	Yes			
CPP_I015	No identifier in one name space should have the same spelling	Yes			

	as an identifier in another name space.				
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 non-member object with external or internal linkage should not be	Yes			

	reused				
CPP_I021	Universal character names shall be used only inside character or string literals	Yes			
CPP_I022	Similiar 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	Yes			

	facilities of <csignal> shall not be used				
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 Function Names	Yes			
CPP_L013	Avoid including	Yes			

	stdio.h				
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 implicitly converted to	Yes			

	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 intervening	Yes			

	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_L033	Reserved Builtin Macros	Yes			
CPP_L034	Use of the iostream library is preferred over stdio.h	Yes			
CPP_M000	Dynamic heap 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	Yes			

	maximum field width should be specified				
CPP_M008	Dynamically allocated memory shall be set to some value prior to its use as an rvalue or in a test	Yes			
CPP_M009	Memory that has been freed shall not be referenced	Yes			
CPP_M010	The new[] and delete[] operators shall be used for the allocation and deallocation of memory resources	Yes			
CPP_M011	The delete[] operator shall be used to deallocate arrays that have been allocated with the new[] operator	Yes			
CPP_M012	The delete[] operator shall be called in the	Yes			

	destructor for all member pointers in an object that are pointing to memory that was dynamically allocated by that object				
CPP_M013	Users shall provide a copy constructor, destructor and assignment operator for a class that uses dynamic memory allocation	Yes			
CPP_M014	The operator new should be called with the nothrow option.	Yes			
CPP_M015	When overloading the new[] operator, a corresponding delete[] operator should be provided.	Yes			
CPP_M016	Overloaded new operator should not hide the	Yes			

	global new operator				
CPP_M017	All local allocations made in a method, other than the destructor, should be deallocated prior to exiting the method.	Yes			
CPP_M018	Dynamic Memory Usage on Realtime Phase	Yes			
CPP_M019	No Dynamic Memory Allocation	Yes			
CPP_M020	Properly pair allocation and deallocation functions	Yes			
CPP_M021	Declare objects shared between POSIX threads with appropriate storage durations	Yes			
CPP_N000	Naming Convention: Classes	Yes			
CPP_N001	Naming Convention: Enumerator	Yes			

CPP_N002	Naming Convention: Enums	Yes			
CPP_N003	Naming Convention: Files	Yes			
CPP_N004	Naming Convention: Functions	Yes			
CPP_N005	Naming Convention: Macros	Yes			
CPP_N006	Naming Convention: Namespaces	Yes			
CPP_N007	Naming Convention: Parameters	Yes			
CPP_N008	Naming Convention: Structs	Yes			
CPP_N009	Naming Convention: 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	Yes			

	Convention: Header File Names				
CPP_N014	Naming Convention: Implementation File Names	Yes			
CPP_N015	Implementation files, that are defined locally in the project, should have a file name extension of ".cpp"	Yes			
CPP_N016	User defined suffixes of the user defined literal operators shall start with underscore followed by 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	Yes			

	preprocessin g directives				
CPP_P006	std::make_un ique shall be used to construct objects owned by std::unique_p tr	Yes			
CPP_P007	A std::unique_p tr shall be used over std::shared_ ptr if ownership sharing is not required	Yes			
CPP_P008	Do Not Use #define	Yes			
CPP_P009	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			
CPP_P011	Ifndef Wrappers or Pragma Once	Yes			
CPP_P012	File Include Matching Header	Yes			
CPP_P013	Function-like macros shall	Yes			

	not be defined				
CPP_P014_A	Restrict Pointer Usage - Multiple Dereferences	Yes			
CPP_P014_B	Restrict Pointer Usage - Other	Yes			
CPP_P015	Inactive Code	Yes			
CPP_P017	#include directives in a file shall only be preceded by other preprocessor directives or comments	Yes			
CPP_P018	A macro shall not be defined with the same name as a keyword	Yes			
CPP_P019	Macros in Blocks	Yes			
CPP_P020	C++ macros shall only be used for include guards, type qualifiers, or storage class specifiers	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 identifiers, macros and functions in the standard library shall not be defined, redefined or undefined	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_P026	avoid #undef	Yes			
CPP_P028	A smart pointer shall only be used as a parameter type if it expresses lifetime semantics	Yes			
CPP_P029	A project	Yes			

	should not contain unused macro declarations				
CPP_P030	Invalid Use of std::shared_ptr	Yes			
CPP_P031	Invalid Use of std::unique_ptr	Yes			
CPP_P032	Cyclic Structure of std::shared_ptr	Yes			
CPP_P033	For pointer declarations, the asterisk shall be placed with the variable	Yes			
CPP_P034	Const Member Function Returning Non-Const Pointer or Reference	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	Yes			

	ownership				
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 used as the null-pointer-constant	Yes			
CPP_P041	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes			
CPP_P042	Pointer arithmetic shall not be used with pointers to non-final	Yes			

	classes				
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 not be passed to a smart pointer of single object type	Yes			
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	Yes			

	tested for equality with null-pointer-constant				
CPP_P049	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes			
CPP_P050	Literal zero (0) shall not be used as the null-pointer-constant.	Yes			
CPP_P051	Pointer to Integer Cast	Yes			
CPP_P052	A parameter 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_P060	Prefer pass-by-reference-to-const to pass by value	Yes			
CPP_P061	Shared Pointer Capture	Yes			
CPP_PR000	#define and #undef shall not be used on a reserved identifier or	Yes			

	reserved macro name				
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	Yes			

	two standard forms				
CPP_PR021	The names of standard library macros and objects shall not be reused	Yes			
CPP_PR030	The #pragma directive shall not be used	Yes			
CPP_PR031	#error directive shall not be used	Yes			
CPP_PR032	The # and ## operators should not be used	Yes			
CPP_PR033	The macro offsetof shall not be used	Yes			
CPP_PR034	There shall be no unused include directives (slow)	Yes			
CPP_PR036	Invalid Preprocessor Directives	Yes			
CPP_PR037	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to	Yes			

	the defined operator				
CPP_PR038	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			
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_S000	no unions	Yes			
CPP_S001	Flexible array members shall not be declared	Yes			
CPP_S002	Incorrect Initializer Lists	Yes			
CPP_S003	A type defined as struct shall: (1) provide only public data members, (2)	Yes			

	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				
CPP_S004	Unions Shall not be Used	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	Yes			

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

CPP_ST008	Blank lines should be used to separate distinct algorithmic parts	Yes			
CPP_ST009	Parentheses should be used in lengthy logical and arithmetic expressions for clarity.	Yes			
CPP_ST010	Items should be logically grouped	Yes			
CPP_ST011	Inline functions should be used instead of macros	Yes			
CPP_ST012	Names that differ in case only or that look similar should not be used.	Yes			
CPP_ST013	Switch statements should be used instead of deeply nested else-ifs when testing a	Yes			

The first row above continues: shall be placed at the function level or at the module level.

	variable for multiple values				
CPP_ST014	No line of code should extend beyond 80 characters	Yes			
CPP_ST015	Incrementing and decrementing control variables in loops	Yes			
CPP_ST016	Calls to free should have an if test around them if it is uncertain that the pointer has been properly allocated.	Yes			
CPP_ST017	White space shall not be used in the following places	Yes			
CPP_ST018	Continuation lines shall be indented at least one level from the line being continued	Yes			
CPP_ST019	Statements under case labels shall be indented one level	Yes			

CPP_ST020	For the if-else, while, do, and for control structure, the statement(s) shall be indented one level	Yes			
CPP_ST021	Placement of braces for constructs 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	Yes			

	before and after it				
CPP_ST026	Class naming conventions	Yes			
CPP_ST027	Naming conventions for class data members vs. member function internal data	Yes			
CPP_ST028	Data type naming conventions	Yes			
CPP_ST029	Immutable data naming 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	Yes			

	of the basic numerical types				
CPP_T001	Arguments to character-handling functions shall be representable as an unsigned char	Yes			
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 integral conversion shall not change the signedness of the underlying type	Yes			
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	Yes			

	shall not increase the size of the underlying type of a cvalue expression				
CPP_T013	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes			
CPP_T014	If the bitwise operators ~ and << are 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	Yes			

	character values				
CPP_T016	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes			
CPP_T017	The first operand of a conditional-operator shall have type bool	Yes			
CPP_T018	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes			
CPP_T019	C-style Array	Yes			
CPP_T020	Casts from a base class to a 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 Pointer Decay	Yes			
CPP_T024	NULL shall not be used as an integer value	Yes			
CPP_T025	CV-qualifiers shall be placed on the right hand side of the type that is a typedef or a using name	Yes			
CPP_T026	The typedef specifier shall not be used	Yes			
CPP_T027	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration	Yes			
CPP_T028	Enumeration	Yes			

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

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

	conversion of passed parameters				
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[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes			
CPP_T044	Undocument ed Use of Floating-point Arithmetic	Yes			
CPP_T045	Undocument ed Use of Scaled-integer or Fixed-point Arithmetic	Yes			
CPP_T046	Assigning Object to an Overlapping Object	Yes			
CPP_T047	Data types used for	Yes			

	interfacing				
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 strings shall not be used	Yes			
CPP_T052	String-to-Number Conversion Handling	Yes			
CPP_T053	A type used as a template argument shall provide all members that are used by the template	Yes			
CPP_T054A	An array or container shall not be accessed beyond its range (Part A)	Yes			
CPP_T054B	An array or container shall not be accessed	Yes			

	beyond its range Part B				
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 signed and unsigned values.	Yes			
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	Yes			

	pointer to integer or integer to pointer				
CPP_T061	All Checks/ Language Specific/C and C++/ Types/Use Const whenever possible	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	A project should not contain unused tag declarations	Yes			
CPP_U006	A project shall not contain unused type declarations	Yes			
CPP_U007	Unused Labels	Yes			
CPP_U010	Unused Entities	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_V008	Unused Static Global	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	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy	Yes			
CPP_V014	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	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_V017	A project shall not	Yes			

	contain instances of non-volatile 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	Yes			

	from type initialization, otherwise symbolic 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	Yes			

	should be used exclusively for altering 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	Yes			

	containers (vector, list, map, etc.) should be used instead 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	Yes			

	directly. Methods should be used to set variables whose values are determined by an algorithm.					
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_PACKED_MEMBER	Do Not Take the Address of Packed Members	Yes				

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_REVERSED_	Ambiguous Reversed Operator	Yes			

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_BOU	Array Bounds	Yes			High

NDS					
CPP_WARN_ ARRAY_BOU NDS_POINTE R_ARITHMET IC	Array Bounds Pointer Arithmetic	Yes			
CPP_WARN_ ARRAY_PARA METER	Array Parameter	Yes			
CPP_WARN_ ASM_OPERA ND_WIDTHS	Assembly Operand Widths	Yes			
CPP_WARN_ ASSIGN_ENU M	Assign Enum	Yes			
CPP_WARN_ ASSUME	Discarded Side Effects to __assume Function	Yes			
CPP_WARN_ ATOMIC_AC CESS	Atomic Access	Yes			
CPP_WARN_ ATOMIC_ALI GNMENT	Atomic Alignment	Yes			
CPP_WARN_ ATOMIC_IMP LICIT_SEQ_C ST	Atomic Implicitly Sequentially- Consistent	Yes			
CPP_WARN_ ATOMIC_ME MORY_ORDE RING	Atomic Memory Ordering	Yes			
CPP_WARN_ AUTO_DISAB LE_VPTR_SA NITIZER	Auto Disable Virtual Pointer Sanitizer	Yes			
CPP_WARN_ AUTO_STOR AGE_CLASS	Auto Storage Class	Yes			
CPP_WARN_	Availability	Yes			High

AVAILABILITY	Attribute				
CPP_WARN_ AVR_RTLIB_L INKING_QUIR KS	AVR RTLIB (Real-Time Library) Linking Quirks	Yes			
CPP_WARN_ BACKEND_P LUGIN	Backend Plugin	Yes			
CPP_WARN_ BACKSLASH _NEWLINE_E SCAPE	Backslash Newline Escape	Yes			High
CPP_WARN_ BAD_FUNCTI ON_CAST	Do Not Cast from Function Call of One Type to Another	Yes			
CPP_WARN_ BIND_TO_TE MPORARY_C OPY	Bind to Temporary Copy	Yes			
CPP_WARN_ BINDING_IN_ CONDITION	Binding in Condition	Yes			
CPP_WARN_ BIT_INT_EXT ENSION	Bit Int Extension	Yes			
CPP_WARN_ BITFIELD_CO NSTANT_CO NVERSION	Bitfield Constant Conversion	Yes			High
CPP_WARN_ BITFIELD_EN UM_CONVER SION	Bitfield Enum Conversion	Yes			
CPP_WARN_ BITFIELD_WI DTH	Do Not Exceed Bit-Field Width	Yes			High
CPP_WARN_ BITWISE_CO NDITIONAL_	Bitwise Conditional Parentheses	Yes			

PARENTHESE S					
CPP_WARN_ BITWISE_INS TEAD_OF_LO GICAL	Bitwise Instead of Logical	Yes			
CPP_WARN_ BITWISE_OP _PARENTHES ES	Bitwise Operator Parentheses	Yes			
CPP_WARN_ BOOL_CONV ERSION	Bool Conversion	Yes			High
CPP_WARN_ BOOL_OPER ATION	Bool Operation	Yes			
CPP_WARN_ BRACED_SC ALAR_INIT	Braced Scalar Init	Yes			
CPP_WARN_ BRANCH_PR OTECTION	Branch Protection	Yes			
CPP_WARN_ BUILTIN_ASS UME_ALIGNE D_ALIGNME NT	Builtin Assume Aligned Alignment	Yes			
CPP_WARN_ BUILTIN_MA CRO_REDEFI NED	Builtin Macro Redefined	Yes			High
CPP_WARN_ BUILTIN_ME MCPY_CHK_ SIZE	Builtin Memcpy Check Size	Yes			High
CPP_WARN_ BUILTIN_REQ UIRES_HEAD ER	Builtin Requires Header	Yes			
CPP_WARN_ C2X_EXTEN SIONS	C2X Extensions	Yes			

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_PCH	Clang-CL Precompiled Headers	Yes			

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_UNINITIALIZED	Conditional Uninitialized	Yes			

CPP_WARN_ CONSTANT_ CONVERSION	Constant Conversion	Yes			High
CPP_WARN_ CONSTANT_ EVALUATED	Constant Evaluated	Yes			
CPP_WARN_ CONSTANT_ LOGICAL_OP ERAND	Constant Logical Operand	Yes			High
CPP_WARN_ CONSTEXPR _NOT_CONS T	Constexpr Not Const	Yes			
CPP_WARN_ CONSUMED	Consumable Attribute	Yes			
CPP_WARN_ CONVERSIO N	Type Conversion	Yes			
CPP_WARN_ COROUTINE	Coroutine Return Type	Yes			
CPP_WARN_ COROUTINE _MISSING_U NHANDLED_ EXCEPTION	Coroutine Missing Unhandled Exception	Yes			
CPP_WARN_ COVERED_S WITCH_DEFA ULT	Covered Switch Default	Yes			
CPP_WARN_ CPP_COMPA T	C++ Compatibility	Yes			
CPP_WARN_ CPP2B_EXTE NSIONS	C++2B Extensions	Yes			
CPP_WARN_ CPP11_COMP AT	C++11 Compatibility	Yes			
CPP_WARN_	C++11	Yes			High

CPP11_COMPAT_DEPRECATED_WRITABLE_STRINGS	Compatibility Deprecated Writable Strings				
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_CPP17_COM	C++17 Compatibility	Yes			

PAT_MANGLI NG	Mangling				
CPP_WARN_ CPP17_EXTE NSIONS	C++17 Extensions	Yes			
CPP_WARN_ CPP20_ATTR IBUTE_EXTE NSIONS	C++20 Attribute Extensions	Yes			
CPP_WARN_ CPP20_COM PAT	C++20 Compatibility	Yes			
CPP_WARN_ CPP20_DESI GNATOR	C++20 Designator	Yes			
CPP_WARN_ CPP20_EXTE NSIONS	C++20 Extensions	Yes			
CPP_WARN_ CPP98_COM PAT	C++98 Compatibility	Yes			
CPP_WARN_ CPP98_COM PAT_BIND_T O_TEMPORA RY_COPY	C++98 Compatibility Bind to Temporary Copy	Yes			
CPP_WARN_ CPP98_COM PAT_EXTRA_ SEMI	C++98 Compatibility Extra Semicolon	Yes			
CPP_WARN_ CPP98_COM PAT_LOCAL_ TYPE_TEMPL ATE_ARGS	C++98 Compatibility Local Type Template Args	Yes			
CPP_WARN_ CPP98_COM PAT_PEDANT IC	C++98 Compatibility Pedantic	Yes			
CPP_WARN_ CPP98_COM PAT_UNNAM	C++98 Compatibility Unnamed	Yes			

ED_TYPE_TE MPLATE_AR GS	Type Template Args				
CPP_WARN_ CPP98_CPP1 1_COMPAT_B INARY_LITER AL	C++98 C++11 Compatibility Binary Literal	Yes			
CPP_WARN_ CTAD_MAYB E_UNSUPPO RTED	CTAD (Class Template Argument Deduction) Maybe Unsupported	Yes			
CPP_WARN_ CXX_ATTRIB UTE_EXTENS ION	C++ Attribute Extension	Yes			
CPP_WARN_ DANGLING	Dangling Pointers	Yes			High
CPP_WARN_ DANGLING_E LSE	Dangling Else	Yes			
CPP_WARN_ DANGLING_F IELD	Dangling Field	Yes			High
CPP_WARN_ DANGLING_ GSL	Dangling Pointers Found by Guidelines Support Library	Yes			High
CPP_WARN_ DANGLING_I NITIALIZER_ LIST	Dangling Initializer List	Yes			High
CPP_WARN_ DARWIN_SD K_SETTINGS	Darwin SDK Settings	Yes			
CPP_WARN_ DATE_TIME	Date and Time Macros	Yes			
CPP_WARN_	Dealloc in	Yes			

DEALLOC_IN_CATEGORY	Category				
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_CO NVERSION	Conversion				
CPP_WARN_ DEPRECATED _ARRAY_CO MPARE	Deprecated Array Compare	Yes			
CPP_WARN_ DEPRECATED _ATTRIBUTE S	Deprecated Attributes	Yes			
CPP_WARN_ DEPRECATED _BUILTINS	Deprecated Builtins	Yes			High
CPP_WARN_ DEPRECATED _COMMA_SU BSCRIPT	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_PRO VIDED_COPY	Deprecated Copy with User Provided Copy	Yes			
CPP_WARN_ DEPRECATED _COPY_WITH _USER_PRO VIDED_DTOR	Deprecated Copy with User Provided Destructor	Yes			
CPP_WARN_ DEPRECATED _COROUTINE	Deprecated Coroutine	Yes			
CPP_WARN_ DEPRECATED _DECLARATI ONS	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_BOD Y	Control Loop Shall Not Have Empty Body	Yes			
CPP_WARN_ EMPTY_DEC OMPOSITION	Decompositi on Group Shall Not Be Empty	Yes			
CPP_WARN_ EMPTY_INIT_ STMT	No Empty Initialization Statements	Yes			
CPP_WARN_ EMPTY_TRA NSLATION_U NIT	Empty Translation Unit	Yes			
CPP_WARN_ ENUM_COMP ARE	Enum Compare	Yes			High
CPP_WARN_ ENUM_COMP ARE_CONDIT IONAL	Enum Compare Conditional	Yes			
CPP_WARN_ ENUM_COMP ARE_SWITCH	Enum Compare Switch	Yes			High
CPP_WARN_ ENUM_CONV ERSION	Enum Conversion	Yes			
CPP_WARN_ ENUM_ENU M_CONVERS ION	Enum Enum Conversion	Yes			
CPP_WARN_ ENUM_FLOA T_CONVERSI ON	Enum Float Conversion	Yes			
CPP_WARN_ ENUM_TOO_ LARGE	Enum Too Large	Yes			

CPP_WARN_ EXCEPTIONS	Exceptions	Yes			High
CPP_WARN_ EXCESS_INIT IALIZERS	Excess Initializers	Yes			
CPP_WARN_ EXIT_TIME_D ESTRUCTOR S	Exit Time Destructors	Yes			
CPP_WARN_ EXPANSION_ TO_DEFINED	Expansion to Defined	Yes			
CPP_WARN_ EXPORT_UN NAMED	Export Unnamed	Yes			
CPP_WARN_ EXPORT_USI NG_DIRECTI VE	Export Using Directive	Yes			
CPP_WARN_ EXTERN_C_C OMPAT	Extern C Compatibility	Yes			High
CPP_WARN_ EXTERN_INIT IALIZER	Extern Initializer	Yes			High
CPP_WARN_ EXTRA	Extra Warnings	Yes			
CPP_WARN_ EXTRA_QUA LIFICATION	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_TOKE NS	Extra Tokens	Yes			
CPP_WARN_	Final	Yes			

FINAL_DTOR _NON_FINAL _CLASS	Destructor Non-Final Class				
CPP_WARN_ FINAL_MACR O	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_AR RAY_EXTENS IONS	Flexible Array Extensions	Yes			
CPP_WARN_ FLOAT_CON VERSION	Float Conversion	Yes			
CPP_WARN_ FLOAT_EQUA L	Float Equal	Yes			
CPP_WARN_ FLOAT_OVER FLOW_CONV ERSION	Float Overflow Conversion	Yes			
CPP_WARN_ FLOAT_ZERO _CONVERSIO N	Float Zero Conversion	Yes			
CPP_WARN_ FOR_LOOP_ ANALYSIS	For Loop Analysis	Yes			
CPP_WARN_ FORMAT	Format String	Yes			High
CPP_WARN_ FORMAT_EX TRA_ARGS	Format Extra Args	Yes			High
CPP_WARN_	Format	Yes			High

FORMAT_INS UFFICIENT_A RGS	Insufficient Args				
CPP_WARN_ FORMAT_INV ALID_SPECIF IER	Format Invalid Specifier	Yes			High
CPP_WARN_ FORMAT_NO N_ISO	Format Non-ISO	Yes			
CPP_WARN_ FORMAT_NO NLITERAL	Format Non-Literal	Yes			
CPP_WARN_ FORMAT_PE DANTIC	Format Pedantic	Yes			
CPP_WARN_ FORMAT_SE CURITY	Format Security	Yes			High
CPP_WARN_ FORMAT_TY PE_CONFUSI ON	Format Type Confusion	Yes			
CPP_WARN_ FORMAT_ZE RO_LENGTH	Format Zero Length	Yes			High
CPP_WARN_ FORTIFY_SO URCE	Fortify Source	Yes			High
CPP_WARN_ FOUR_CHAR _CONSTANT S	Four Char Constants	Yes			
CPP_WARN_ FRAME_ADD RESS	Frame Address	Yes			
CPP_WARN_ FREE_NONH EAP_OBJECT	Free Non-Heap Object	Yes			High
CPP_WARN_ FUNCTION_	Function Multi-Version	Yes			

MULTIVERSI ON					
CPP_WARN_ FUSE_LD_PA TH	Fuse LD Path	Yes			
CPP_WARN_ GCC_COMPA T	GCC Compatibility	Yes			
CPP_WARN_ GLOBAL_CO NSTRUCTOR S	Global Constructors	Yes			
CPP_WARN_ GLOBAL_ISE L	GlobalISel (Global Instruction Selection) Framework	Yes			
CPP_WARN_ GNU_ALIGN OF_EXPRESS ION	GNU Alignof Expression	Yes			
CPP_WARN_ GNU_ANONY MOUS_STRU CT	GNU Anonymous Struct	Yes			
CPP_WARN_ GNU_ARRAY _MEMBER_P AREN_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_COMPL EX_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_I	Ignored	Yes			High

GNORED_RE FERENCE_Q UALIFIERS	Reference Qualifiers				
CPP_WARN_I MPLICIT_CO NST_INT_FL OAT_CONVE RSION	Implicit Const Int Float Conversion	Yes			High
CPP_WARN_I MPLICIT_CO NVERSION_F LOATING_PO INT_TO_BOO L	Implicit Conversion Floating Point to Bool	Yes			High
CPP_WARN_I MPLICIT_EXC EPTION_SPE C_MISMATC H	Implicit Exception Spec Mismatch	Yes			High
CPP_WARN_I MPLICIT_FAL LTHROUGH	Implicit Fallthrough	Yes			
CPP_WARN_I MPLICIT_FAL LTHROUGH_ PER_FUNCTI ON	Implicit Fallthrough Per Function	Yes			
CPP_WARN_I MPLICIT_FIX ED_POINT_C ONVERSION	Implicit Fixed Point Conversion	Yes			High
CPP_WARN_I MPLICIT_FLO AT_CONVER SION	Implicit Float Conversion	Yes			
CPP_WARN_I MPLICIT_FU NCTION_DE CLARATION	Implicit Function Declaration	Yes			
CPP_WARN_I MPLICIT_INT	Implicit Int	Yes			
CPP_WARN_I	Implicit Int	Yes			

MPLICIT_INT _CONVERSIO N	Conversion					
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_LI TERAL	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_ABSOLU TE_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_	Incompatible Library Redeclaratio	Yes				High

REDECLARAT ION	n				
CPP_WARN_I NCOMPATIBL E_MS_STRU CT	Incompatible Microsoft Struct	Yes			
CPP_WARN_I NCOMPATIBL E_POINTER_ TYPES	Incompatible Pointer Types	Yes			High
CPP_WARN_I NCOMPATIBL E_POINTER_ TYPES_DISC ARDS_QUALI FIERS	Incompatible Pointer Types Discards Qualifiers	Yes			High
CPP_WARN_I NCOMPATIBL E_SYSROOT	Incompatible Sysroot	Yes			
CPP_WARN_I NCOMPLETE _IMPLEMENT ATION	Incomplete Implementati on	Yes			
CPP_WARN_I NCOMPLETE _SETJMP_DE CLARATION	Incomplete Setjmp Declaration	Yes			
CPP_WARN_I NCONSISTE NT_DLLIMPO RT	Inconsistent DLLimport	Yes			
CPP_WARN_I NCONSISTE NT_MISSING _DESTRUCT OR_OVERRID E	Inconsistent Missing Destructor Override	Yes			
CPP_WARN_I NCONSISTE NT_MISSING _OVERRIDE	Inconsistent Missing Override	Yes			High
CPP_WARN_I	Increment	Yes			High

NCREMENT_ BOOL	Bool				
CPP_WARN_I NFINITE_REC URSION	Infinite Recursion	Yes			
CPP_WARN_I NITIALIZER_ OVERRIDES	Initializer Overrides	Yes			
CPP_WARN_I NJECTED_CL ASS_NAME	Injected Class Name	Yes			High
CPP_WARN_I NLINE_ASM	Inline Assembly	Yes			
CPP_WARN_I NLINE_NAME SPACE_REOP ENED_NONI NLINE	Inline Namespace Reopened Non-Inline	Yes			High
CPP_WARN_I NLINE_NEW_ DELETE	Inline New Delete	Yes			
CPP_WARN_I NSTANTIATI ON_AFTER_S PECIALIZATI ON	Instantiation After Specializatio n	Yes			High
CPP_WARN_I NT_CONVER SION	Int Conversion	Yes			
CPP_WARN_I NT_IN_BOOL _CONTEXT	Int in Bool Context	Yes			
CPP_WARN_I NT_TO_POIN TER_CAST	Int to Pointer Cast	Yes			High
CPP_WARN_I NT_TO_VOID _POINTER_C AST	Int to Void Pointer Cast	Yes			High
CPP_WARN_I NTEGER_OV	Integer Overflow	Yes			High

ERFLOW					
CPP_WARN_INTERRUPT_SERVICE_ROUTINE	Interrupt Service Routine	Yes			
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_DIRECTOR	Invalid or Nonexistent Directory	Yes			

Y					
CPP_WARN_INVALID_PARTIAL_SPECIALIZATION	Invalid Partial Specialization	Yes			
CPP_WARN_INVALID_PP_TOKEN	Invalid Preprocessor Token	Yes			High
CPP_WARN_INVALID_SOURCE_ENCODING	Invalid Source Encoding	Yes			
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	Yes			

LINKER_WAR NINGS	Warnings				
CPP_WARN_ LITERAL_CO NVERSION	Literal Conversion	Yes			High
CPP_WARN_ LITERAL_RA NGE	Literal Range	Yes			High
CPP_WARN_ LOCAL_TYPE _TEMPLATE_ ARGS	Local Type Template Args	Yes			
CPP_WARN_ LOGICAL_NO T_PARENTHE SES	Logical Not Parentheses	Yes			High
CPP_WARN_ LOGICAL_OP _PARENTHES ES	Logical Operator Parentheses	Yes			
CPP_WARN_ LONG_LONG	Long Long	Yes			
CPP_WARN_ MACRO_RED EFINED	Macro Redefined	Yes			High
CPP_WARN_ MAIN	Main Function Conventions	Yes			
CPP_WARN_ MAIN_RETUR N_TYPE	Main Return Type	Yes			High
CPP_WARN_ MALFORMED _WARNING_ CHECK	Malformed Warning Check	Yes			
CPP_WARN_ MANY_BRAC ES_AROUND _SCALAR_INI T	Many Braces Around Scalar Init	Yes			High
CPP_WARN_	Max Tokens	Yes			

MAX_TOKEN S					
CPP_WARN_ MAX_UNSIG NED_ZERO	Max Unsigned Zero	Yes			High
CPP_WARN_ MEMSET_TR ANSPOSED_ ARGS	Memset Transposed Args	Yes			High
CPP_WARN_ MEMSIZE_C OMPARISON	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_P ASTE	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_AR G_REDEFINIT ION	Microsoft Default Arg Redefinition	Yes			
CPP_WARN_ MICROSOFT_	Microsoft Drectve	Yes			

DRECTVE_SE CTION	Section				
CPP_WARN_ MICROSOFT_ END_OF_FILE	Microsoft End of File	Yes			
CPP_WARN_ MICROSOFT_ ENUM_FORW ARD_REFERE NCE	Microsoft Enum Forward Reference	Yes			
CPP_WARN_ MICROSOFT_ ENUM_VALU E	Microsoft Enum Value	Yes			
CPP_WARN_ MICROSOFT_ EXCEPTION_ SPEC	Microsoft Exception Spec	Yes			
CPP_WARN_ MICROSOFT_ EXISTS	Microsoft Exists	Yes			
CPP_WARN_ MICROSOFT_ EXPLICIT_CO NSTRUCTOR _CALL	Microsoft Explicit Constructor Call	Yes			
CPP_WARN_ MICROSOFT_ EXTRA_QUA LIFICATION	Microsoft Extra Qualification	Yes			
CPP_WARN_ MICROSOFT_ FIXED_ENUM	Microsoft Fixed Enum	Yes			
CPP_WARN_ MICROSOFT_ FLEXIBLE_AR RAY	Microsoft Flexible Array	Yes			
CPP_WARN_ MICROSOFT_ GOTO	Microsoft Goto	Yes			
CPP_WARN_ MICROSOFT_	Microsoft Inaccessible	Yes			

INACCESSIBLE_BASE	Base					
CPP_WARN_MICROSOFT_INCLUDE	Microsoft Include	Yes				
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_	Microsoft Using Decl	Yes				

USING_DECL					
CPP_WARN_ MICROSOFT_ VOID_PSEUD O_DTOR	Microsoft Void Pseudo Destructor	Yes			
CPP_WARN_ MISEXPECT	Misuse of __builtin_exp ect()	Yes			
CPP_WARN_ MISLEADING _INDENTATIO N	Misleading Indentation	Yes			
CPP_WARN_ MISMATCHE D_NEW_DEL ETE	Mismatched New Delete	Yes			High
CPP_WARN_ MISMATCHE D_TAGS	Mismatched Tags	Yes			
CPP_WARN_ MISSING_BR ACES	Missing Braces	Yes			
CPP_WARN_ MISSING_CO NSTINIT	Missing Constinit	Yes			
CPP_WARN_ MISSING_DE CLARATIONS	Missing Declarations	Yes			High
CPP_WARN_ MISSING_EX CEPTION_SP EC	Missing Exception Spec	Yes			High
CPP_WARN_ MISSING_FIE LD_INITIALIZ ERS	Missing Field Initializers	Yes			
CPP_WARN_ MISSING_ME THOD_RETU RN_TYPE	Missing Method Return Type	Yes			
CPP_WARN_	Missing	Yes			

MISSING_NO RETURN	Noreturn Attribute					
CPP_WARN_ MISSING_PR OTOTYPE_F OR_CC	Missing Prototype for Calling Convention	Yes				
CPP_WARN_ MISSING_PR OTOTYPES	Missing Prototypes	Yes				
CPP_WARN_ MISSING_SE LECTOR_NA ME	Missing Selector Name	Yes				
CPP_WARN_ MISSING_SY SROOT	Missing Sysroot	Yes				
CPP_WARN_ MISSING_VA RIABLE_DEC LARATIONS	Missing Variable Declarations	Yes				
CPP_WARN_ MISSPELLED _ASSUMPTI ON	Misspelled Assumption	Yes				
CPP_WARN_ MODULE_CO NFLICT	Module Conflict	Yes				
CPP_WARN_ MODULE_FIL E_CONFIG_M ISMATCH	Module File Config Mismatch	Yes				
CPP_WARN_ MODULE_FIL E_EXTENSIO N	Module File Extension	Yes				
CPP_WARN_ MODULE_IM PORT_IN_EX TERN_C	Module Import in Extern C	Yes				
CPP_WARN_ MODULES_A MBIGUOUS_I	Modules Ambiguous Internal	Yes				

NTERNAL_LI NKAGE	Linkage				
CPP_WARN_ MODULES_I MPORT_NES TED_REDUN DANT	Modules Import Nested Redundant	Yes			
CPP_WARN_ MSVC_NOT_ FOUND	MSVC Not Found	Yes			
CPP_WARN_ MULTICHAR	Multiple Characters in Character Literal	Yes			High
CPP_WARN_ MULTIPLE_M OVE_VBASE	Multiple Move Virtual Base	Yes			High
CPP_WARN_ NESTED_AN ON_TYPES	Nested Anon-Types	Yes			
CPP_WARN_ NEW_RETUR NS_NULL	New Returns Null	Yes			High
CPP_WARN_ NEWLINE_EO F	Newline EOF (End of File)	Yes			
CPP_WARN_ NODEREF	Noderef Attirbute	Yes			High
CPP_WARN_ NON_C_TYP EDEF_FOR_LI NKAGE	Non-C Typedef for Linkage	Yes			High
CPP_WARN_ NON_LITERA L_NULL_CO NVERSION	Non-Literal Null Conversion	Yes			High
CPP_WARN_ NON_POD_V ARARGS	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_DTOR	Non-Virtual Destructor	Yes			
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_ARITHME	Null Pointer Arithmetic	Yes			

TIC					
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	Yes			

OPENMP_51_EXTENSIONS	Extensions				
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_ OVERLOADE D_VIRTUAL	Overloaded Virtual	Yes			
CPP_WARN_ OVERRIDE_M ODULE	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_ PARENTHESE S	Parentheses	Yes			
CPP_WARN_ PARENTHESE S_EQUALITY	Parentheses Equality	Yes			High
CPP_WARN_ PASS_FAILE D	Pass Failed	Yes			
CPP_WARN_ PCH_DATE_T IME	PCH (Precompiled Header) Date Time	Yes			
CPP_WARN_ PEDANTIC	Pedantic	Yes			
CPP_WARN_ PEDANTIC_C ORE_FEATUR ES	Pedantic Core Features	Yes			
CPP_WARN_ PESSIMIZING _MOVE	Pessimizing Move	Yes			
CPP_WARN_ POINTER_AR ITH	Pointer Arithmetic	Yes			High
CPP_WARN_ POINTER_BO OL_CONVER	Pointer Bool Conversion	Yes			High

SION					
CPP_WARN_POINTER_COMPARE	Pointer Compare	Yes			High
CPP_WARN_POINTER_INTEGER_COMPARE	Pointer Integer Compare	Yes			High
CPP_WARN_POINTER_SIGN	Pointer Sign	Yes			High
CPP_WARN_POINTER_TO_ENUM_CAST	Pointer to Enum Cast	Yes			High
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_INCLUDE	Pragma Pack Suspicious Include	Yes			
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_FU NCTION	Predefined Identifier Outside Function	Yes			
CPP_WARN_ PRIVATE_EXT ERN	Private Extern	Yes			
CPP_WARN_ PRIVATE_HE ADER	Private Header	Yes			
CPP_WARN_ PROFILE_INS TR_MISSING	Profile Instrumented Code Missing	Yes			
CPP_WARN_ PROFILE_INS TR_OUT_OF_ DATE	Profile Instrumented Code Out of Date	Yes			
CPP_WARN_ PROFILE_INS TR_UNPROFI LED	Profile Instrumented Code Unprofiled	Yes			
CPP_WARN_ PSABI	PSABI (Processor-Specific Application Binary Interface)	Yes			
CPP_WARN_ QUALIFIED_V OID_RETURN _TYPE	Qualified Void Return Type	Yes			High
CPP_WARN_ RANGE_LOO P_BIND_REF ERENCE	Range Loop Bind Reference	Yes			
CPP_WARN_ RANGE_LOO P_CONSTRU CT	Range Loop Construct	Yes			
CPP_WARN_ REDECLARE	Re-Declared Class	Yes			High

D_CLASS_M EMBER	Member				
CPP_WARN_ REDUNDANT _CONSTEVA L_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_ REINTERPRE T_BASE_CLA SS	Reinterpret Base Class	Yes			High
CPP_WARN_ REORDER_C TOR	Reorder Constructor	Yes			
CPP_WARN_ REORDER_IN IT_LIST	Reorder Initializer List	Yes			High
CPP_WARN_ RESERVED_I DENTIFIER	Reserved Identifier	Yes			
CPP_WARN_ RESERVED_ MACRO_IDE NTIFIER	Reserved Macro Identifier	Yes			
CPP_WARN_ RESERVED_U SER_DEFINE D_LITERAL	Reserved User Defined Literal	Yes			
CPP_WARN_ RESTRICT_E XPANSION	Restrict Expansion	Yes			
CPP_WARN_ RETAINED_L ANGUAGE_LI	Retained Language Linkage	Yes			

NKAGE					
CPP_WARN_ RETURN_STA CK_ADDRES S	Return Stack Address	Yes			High
CPP_WARN_ RETURN_TY PE	Return Type	Yes			High
CPP_WARN_ RETURN_TY PE_C_LINKA GE	Return Type C Linkage	Yes			High
CPP_WARN_ REWRITE_NO T_BOOL	Rewrite Not Bool	Yes			
CPP_WARN_ RTTI	Run-Time Type Information	Yes			
CPP_WARN_ SARIF_FORM AT_UNSTABL E	SARIF Format Unstable	Yes			
CPP_WARN_ SECTION	Section Attributes	Yes			
CPP_WARN_ SELF_ASSIG N	Self Assign	Yes			
CPP_WARN_ SELF_ASSIG N_FIELD	Self Assign Field	Yes			High
CPP_WARN_ SELF_ASSIG N_OVERLOA DED	Self Assign Overloaded	Yes			
CPP_WARN_ SELF_MOVE	Self Move	Yes			
CPP_WARN_ SENTINEL	Sentinel Attribute	Yes			
CPP_WARN_ SERIALIZED_ DIAGNOSTIC	Serialized Diagnostics	Yes			

S					
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_6	Shorten Integer Type	Yes			

4_TO_32	Width				
CPP_WARN_ SIGN_COMP ARE	Sign Compare	Yes			
CPP_WARN_ SIGN_CONVE RSION	Sign Conversion	Yes			
CPP_WARN_ SIGNED_ENU M_BITFIELD	Signed Enum Bitfield	Yes			
CPP_WARN_ SIGNED_UNS IGNED_WCH AR	Signed Unsigned Wchar	Yes			
CPP_WARN_ SINGLE_BIT_ BITFIELD_CO NSTANT_CO NVERSION	Single Bit Bitfield Constant Conversion	Yes			High
CPP_WARN_ SIZEOF_ARR AY_ARGUME NT	Sizeof Array Argument	Yes			High
CPP_WARN_ SIZEOF_ARR AY_DECAY	Sizeof Array Decay	Yes			High
CPP_WARN_ SIZEOF_ARR AY_DIV	Sizeof Array Division	Yes			High
CPP_WARN_ SIZEOF_POIN TER_DIV	Sizeof Pointer Division	Yes			High
CPP_WARN_ SIZEOF_POIN TER_MEMAC CESS	Sizeof Pointer Memory Access	Yes			High
CPP_WARN_ SLASH_U_FI LENAME	Slash U Filename	Yes			
CPP_WARN_ SLH_ASM_G	SLH (Speculative	Yes			

OTO	Load Hardening) Assembly Goto				
CPP_WARN_ SOMETIMES _UNINITIALIZ ED	Sometimes Uninitialized	Yes			
CPP_WARN_ SOURCE_US ES_OPENMP	Source Uses OpenMP	Yes			
CPP_WARN_ SPIR_COMPA T	SPIR (Sampler Initializer) Compatibility	Yes			
CPP_WARN_ STACK_EXHA USTED	Stack Exhausted	Yes			
CPP_WARN_ STACK_PROT ECTOR	Stack Protector	Yes			
CPP_WARN_ STATIC_FLOA T_INIT	Static Float Init	Yes			
CPP_WARN_ STATIC_IN_I NLINE	Static in Inline	Yes			
CPP_WARN_ STATIC_INLI NE_EXPLICIT _INSTANTIAT ION	Static Inline Explicit Instantiation	Yes			High
CPP_WARN_ STATIC_LOC AL_IN_INLIN E	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	Yes			

SUPER_CLASS_METHOD_MISMATCH	Method Mismatch				
CPP_WARN_SUSPICIOUS_BZERO	Suspicious Argument for Bzero Function	Yes			
CPP_WARN_SWITCH	Switch Statements	Yes			High
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_COM	Tautological Constant Out of Range Compare	Yes			High

PARE					
CPP_WARN_ TAUTOLOGIC AL_OVERLAP _COMPARE	Tautological Overlap Compare	Yes			
CPP_WARN_ TAUTOLOGIC AL_POINTER _COMPARE	Tautological Pointer Compare	Yes			High
CPP_WARN_ TAUTOLOGIC AL_TYPE_LI MIT_COMPA RE	Tautological Type Limit Compare	Yes			
CPP_WARN_ TAUTOLOGIC AL_UNDEFIN ED_COMPAR E	Tautological Undefined Compare	Yes			High
CPP_WARN_ TAUTOLOGIC AL_UNSIGNE D_CHAR_ZE RO_COMPAR E	Tautological Unsigned Char Zero Compare	Yes			
CPP_WARN_ TAUTOLOGIC AL_UNSIGNE D_ENUM_ZE RO_COMPAR E	Tautological Unsigned Enum Zero Compare	Yes			
CPP_WARN_ TAUTOLOGIC AL_UNSIGNE D_ZERO_CO MPARE	Tautological Unsigned Zero Compare	Yes			
CPP_WARN_ TAUTOLOGIC AL_VALUE_R ANGE_COMP ARE	Tautological Value Range Compare	Yes			
CPP_WARN_	TCB (Trusted	Yes			

TCB_ENFOR CEMENT	Computing Base) Enforcement				
CPP_WARN_ TENTATIVE_ DEFINITION_I NCOMPLETE _TYPE	Tentative Definition Incomplete Type	Yes			High
CPP_WARN_ THREAD_SA FETY_ANALY SIS	Thread Safety Analysis	Yes			
CPP_WARN_ THREAD_SA FETY_ATTRIB UTES	Thread Safety Attributes	Yes			
CPP_WARN_ THREAD_SA FETY_BETA	Thread Safety Beta	Yes			
CPP_WARN_ THREAD_SA FETY_NEGAT IVE	Thread Safety Negative	Yes			
CPP_WARN_ THREAD_SA FETY_PRECI SE	Thread Safety Precise	Yes			
CPP_WARN_ THREAD_SA FETY_REFER ENCE	Thread Safety Reference	Yes			
CPP_WARN_ THREAD_SA FETY_VERBO SE	Thread Safety Verbose	Yes			
CPP_WARN_ TRIGRAPHS	Trigraphs	Yes			High
CPP_WARN_ TYPE_SAFET Y	Type Safety	Yes			High
CPP_WARN_ TYPEDEF_RE	Typedef Redefinition	Yes			High

DEFINITION					
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 MPLICIT_CAS T	Unaligned Qualifier Implicit Cast	Yes			
CPP_WARN_ UNDEF	Undefined Macros	Yes			
CPP_WARN_ UNDEF_PREF IX	Undefined Macros of a Certain Prefix	Yes			
CPP_WARN_ UNDEFINED_ BOOL_CONV ERSION	Undefined Bool Conversion	Yes			High
CPP_WARN_ UNDEFINED_ FUNC_TEMP LATE	Undefined Function Template	Yes			
CPP_WARN_ UNDEFINED_ INLINE	Undefined Inline	Yes			High
CPP_WARN_ UNDEFINED_ INTERNAL	Undefined Internal	Yes			High
CPP_WARN_ UNDEFINED_ INTERNAL_T YPE	Undefined Internal Type	Yes			
CPP_WARN_ UNDEFINED_	Undefined Reinterpret	Yes			

REINTERPRET_CAST	Cast				
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_ESCAPE_SEQUENCE	Unknown Escape Sequence	Yes			High
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	Unqualified Standard Cast Call	Yes			High

_CALL					
CPP_WARN_ UNREACHAB LE_CODE	Unreachable Code	Yes			
CPP_WARN_ UNREACHAB LE_CODE_BR EAK	Unreachable Code Break	Yes			
CPP_WARN_ UNREACHAB LE_CODE_FA LLTHROUGH	Unreachable Code Fallthrough	Yes			
CPP_WARN_ UNREACHAB LE_CODE_GE NERIC_ASSO C	Unreachable Code Generic Assoc	Yes			
CPP_WARN_ UNREACHAB LE_CODE_LO OP_INCREME NT	Unreachable Code Loop Increment	Yes			
CPP_WARN_ UNREACHAB LE_CODE_RE TURN	Unreachable Code Return	Yes			
CPP_WARN_ UNSEQUENC ED	Unsequence d Modifications	Yes			High
CPP_WARN_ UNSUPPORT ED_ABI	Unsupported ABI (Application Binary Interface)	Yes			
CPP_WARN_ UNSUPPORT ED_ABS	Unsupported Absolute Value Argument	Yes			
CPP_WARN_ UNSUPPORT ED_AVAILABI LITY_GUARD	Unsupported Availability Guard	Yes			High

CPP_WARN_ UNSUPPORT ED_CB	Unsupported Compact Branches	Yes			
CPP_WARN_ UNSUPPORT ED_DLL_BAS E_CLASS_TE MPLATE	Unsupported DLL Base Class Template	Yes			
CPP_WARN_ UNSUPPORT ED_FLOATIN G_POINT_OP T	Unsupported Floating Point Option	Yes			
CPP_WARN_ UNSUPPORT ED_FRIEND	Unsupported Friend	Yes			High
CPP_WARN_ UNSUPPORT ED_GPOPT	Unsupported GPopt (Gaussian Process Optimization)	Yes			
CPP_WARN_ UNSUPPORT ED_NAN	Unsupported Nan Argument	Yes			
CPP_WARN_ UNSUPPORT ED_TARGET_ OPT	Unsupported Target Option	Yes			
CPP_WARN_ UNSUPPORT ED_VISIBILIT Y	Unsupported Visibility	Yes			
CPP_WARN_ UNUSABLE_ PARTIAL_SP ECIALIZATIO N	Unusable Partial Specializatio n	Yes			
CPP_WARN_ UNUSED_BU T_SET_PARA METER	Unused but Set Parameter	Yes			
CPP_WARN_	Unused but	Yes			

UNUSED_BUT_SET_VARIABLE	Set Variable					
CPP_WARN_UNUSED_COMMAND_LINE_ARGUMENT	Unused Command Line Argument	Yes				
CPP_WARN_UNUSED_COMPARISON	Unused Comparison	Yes				High
CPP_WARN_UNUSED_CONST_VARIABLE	Unused Const Variable	Yes				
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_ME	Unused Member	Yes				

MBER_FUNC TION	Function				
CPP_WARN_ UNUSED_PA RAMETER	Unused Parameter	Yes			
CPP_WARN_ UNUSED_PRI VATE_FIELD	Unused Private Field	Yes			
CPP_WARN_ UNUSED_PR OPERTY_IVA R	Unused Property IVar (Instance Variable)	Yes			
CPP_WARN_ UNUSED_RE SULT	Unused Result	Yes			High
CPP_WARN_ UNUSED_TE MPLATE	Unused Template	Yes			
CPP_WARN_ UNUSED_VA LUE	Unused Value	Yes			High
CPP_WARN_ UNUSED_VA RIABLE	Unused Variable	Yes			
CPP_WARN_ UNUSED_VO LATILE_LVAL UE	Unused Volatile Lvalue	Yes			High
CPP_WARN_ USED_BUT_ MARKED_UN USED	Used but Marked Unused	Yes			
CPP_WARN_ USER_DEFIN ED_LITERAL S	User Defined Literals	Yes			High
CPP_WARN_ USER_DEFIN ED_WARNIN GS	User Defined Warnings	Yes			
CPP_WARN_	Variadic	Yes			High

VARARGS	Arguments				
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	Yes			High

WRITABLE_S TRINGS	Strings				
CPP_WARN_ XOR_USED_ AS_POW	Xor Used as Power	Yes			High
CPP_WARN_ ZERO_AS_N ULL_POINTE R_CONSTAN T	Zero as Null Pointer Constant	Yes			
CPP_WARN_ ZERO_LENG TH_ARRAY	Zero Length Array	Yes			
CTR50-CPP	Guarantee that container indices and iterators are within the valid range	Yes			High
CTR51-CPP	Use valid references, pointers, and iterators to reference elements of a container	Yes			High
CTR52-CPP	Guarantee that library functions do not overflow	Yes			High
CTR53-CPP	Use valid iterator ranges	Yes			High
CTR54-CPP	Do not subtract iterators that do not refer to the same container	Yes			Medium
CTR55-CPP	Do not use an additive	Yes			

	operator on an iterator if the result would overflow				
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 objects should not be mutable	Yes			Low
DCL30-C-A	Declare objects with appropriate storage durations - assigning addresses	Yes			High
DCL30-C-B	Declare objects with appropriate storage durations - returning addresses	Yes			High
DCL31-C	Declare identifiers before using them	Yes			Low
DCL36-C	Do not declare an identifier with conflicting linkage classification	Yes			Medium

	s				
DCL37-C	Do not declare or define a reserved identifier	Yes			Low
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
DCL41-C	Do not declare variables inside a switch statement before the first case label	Yes			Medium
DCL50-CPP	Do not define a C-style variadic function	Yes			High
DCL52-CPP	Never qualify a reference type with const or	Yes			Low

	volatile				
DCL54-CPP	Overload allocation and deallocation functions as a pair in the same scope	Yes			Low
DCL55-CPP	Avoid information leakage when passing a class object across a trust boundary	No			Low
DCL56-CPP	Avoid cycles during initialization of static objects	Yes			Low
DCL57-CPP	Do not let exceptions escape from destructors or deallocation functions	Yes			Low
DCL58-CPP	Do not modify the standard namespaces	Yes			High
DCL59-CPP	Do not define an unnamed namespace in a header file	Yes			Medium
DCL60-CPP	Obey the one-definition rule	Yes			High
EFFECTIVEC PP_02	2. Do Not Use #define	Yes			
EFFECTIVEC	3. Use Const	Yes			

PP_03	whenever possible				
EFFECTIVEC PP_04	4. Make sure that objects are initialized before they are used	Yes			
EFFECTIVEC PP_07	7. Non-Virtual Destructors in Base Classes	Yes			
EFFECTIVEC PP_16	16. Use the same form in corresponding uses of new and delete	Yes			
EFFECTIVEC PP_17	17. Store newed objects in smart pointers in standalone statements	Yes			
EFFECTIVEC PP_20	20. Prefer pass-by-reference-to-const to pass by value	Yes			
EFFECTIVEC PP_22	22. Datamembers should be declared private	Yes			
EFFECTIVEC PP_26	26. Postpone variable definitions as long as possible	Yes			
EFFECTIVEC PP_27	27. Minimize casting	Yes			
EFFECTIVEC	33. Avoid	Yes			

PP_33	hiding inherited names				
EFFECTIVEC PP_35	35. Consider alternatives to virtual functions	Yes			
EFFECTIVEC PP_36	36. Never redefine an inherited non-virtual function	Yes			
ENV30-C	Do not modify the object referenced by the return value of certain functions	Yes			Low
ENV31-C	Do not rely on an environment pointer following an operation that may invalidate it	Yes			Low
ENV32-C	All exit handlers must return normally	Yes			Medium
ENV33-C	Do not call system()	Yes			High
ENV34-C	Do not store pointers returned by certain functions	Yes			Low
ERR30-C	Take care when reading errno	Yes			Medium

ERR32-C	Do not rely on indeterminate values of errno	No			Low
ERR33-C	Detect and handle standard library errors	Yes			High
ERR34-C	Detect errors when converting a string to a number	Yes			Medium
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
ERR57-CPP	Do not leak	Yes			Low

	resources when handling exceptions				
ERR58-CPP	Handle all exceptions thrown before main() begins executing	Yes			Low
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
EXP30-C-A	Do not depend on the order of evaluation for side effects - calls	Yes			Medium
EXP30-C-B	Do not depend on the order of evaluation for side effects - other	Yes			Medium
EXP32-C	Do not access a volatile object through a nonvolatile	Yes			Low

	reference				
EXP33-C	Do not read uninitialized memory	Yes			High
EXP34-C	Do not dereference null pointers	Yes			High
EXP35-C	Do not modify objects with temporary lifetime	No			Low
EXP36-C	Do not cast pointers into more strictly aligned pointer types	Yes			Low
EXP37-C	Call functions with the correct number and type of arguments	Yes			Medium
EXP39-C	Do not access a variable through a pointer of an incompatible type	Yes			Medium
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 restrict-	No			Medium

	qualified pointers				
EXP44-C	Do not rely on side effects in operands to sizeof, _Alignof, or _Generic	Yes			Low
EXP45-C	Do not perform assignments in selection statements	Yes			Low
EXP46-C	Do not use a bitwise operator with a Boolean-like operand	Yes			Low
EXP47-C	Do not call va_arg with an argument of the incorrect type	Yes			Medium
EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes			Medium
EXP51-CPP	Do not delete an array through a pointer of the incorrect type	Yes			Low
EXP52-CPP	Do not rely on side effects in unevaluated operands	Yes			Low
EXP53-CPP	Do not read	Yes			High

	uninitialized memory				
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
EXP61-CPP	A lambda object must not outlive any of its reference captured objects	Yes			High
EXP62-CPP	Do not access the	Yes			High

	bits of an object representation that are not part of the object's value representation				
EXP63-CPP	Do not rely on the value of a moved-from object	Yes			Medium
FIO30-C	Exclude user input from format strings	Yes			High
FIO32-C	Do not perform operations on devices that are only appropriate for files	No			Medium
FIO34-C	Distinguish between characters read from a file and EOF or WEOF	No			High
FIO37-C	Do not assume that fgets() or fgetws() returns a nonempty string when successful	Yes			High
FIO38-C	Do not copy a FILE object	Yes			Low
FIO39-C	Do not alternately	Yes			Low

	input and output from a stream without an intervening flush or positioning call				
FIO40-C	Reset strings on fgets() or fgetws() failure	Yes			Low
FIO41-C	Do not call getc(), putc(), getwc(), or putwc() with a stream argument that has side effects	Yes			Low
FIO42-C	Close files when they are no longer needed	Yes			Medium
FIO44-C	Only use values for fsetpos() that are returned from fgetpos()	Yes			Medium
FIO45-C	Avoid TOCTOU race conditions while accessing files	Yes			High
FIO46-C	Do not access a closed file	Yes			Medium
FIO47-C	Use valid	Yes			High

	format strings				
FIO50-CPP	Do not alternately input and output from a file stream without an intervening positioning call	Yes			Low
FIO51-CPP	Close files when they are no longer needed	Yes			Medium
FLP30-C	Do not use floating-point variables as loop counters	Yes			Low
FLP32-C	Prevent or detect domain and range errors in math functions	No			Medium
FLP34-C	Ensure that floating-point conversions are within range of the new type	No			Low
FLP36-C	Preserve precision when converting integral values to floating-point type	No			Low
FLP37-C	Do not use object representatio	Yes			Low

	ns to compare floating-point values				
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 Staments(ST MT)	Yes			
HIS_09	9. Number of call levels(LEVEL)	Yes			
HIS_10	10. Number of return points (RETURN)	Yes			
HIS_11	11. Language scope(VOCF)	Yes			
HIS_12	12. Recursion (AP_CG_CYC LE)	Yes			
HIS_13	13.	Yes			

	Statements Changed (SCHG)				
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			
INT30-C	Ensure that unsigned integer operations do not wrap	Yes			High
INT31-C	Ensure that unsigned integer operations do not result in lost or misinterpreted data	Yes			High
INT32-C	Ensure that operations on signed integers do not result in overflow	No			High
INT33-C	Division by Zero	Yes			Low

INT34-C	Do not shift an expression by a negative number of bits or by greater than or equal to the number of bits that exist in the operand	No			Low
INT35-C	Use correct integer precisions	No			Low
INT36-C	Converting a pointer to integer or integer to pointer	Yes			Low
INT50-CPP	Do not cast to an out-of-range enumeration value	Yes			Medium
M0-1-1	A project shall not contain unreachable 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	Yes	Automated	Required	

	contain non-volatile POD variables having only one use.				
M0-1-8	All functions with void return type shall have external side effect(s)	Yes	Automated	Required	
M0-1-10	Every defined function shall be called at least once.	Yes	Automated	Advisory	
M0-2-1	Assigning Object to an Overlapping Object	Yes	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	
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 typographical	Yes	Automated	Required	

	ly unambiguous				
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 declarations of an object or function shall have compatible types	Yes	Automated	Required	
M3-2-2	The One Definition Rule	Yes	Automated	Required	
M3-2-3	A type, object or function that is used in	Yes	Automated	Required	

	multiple translation units shall be declared in one and only one file				
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 parameter shall be token-for-token identical in all declarations and re-declarations	Yes	Automated	Required	
M3-9-3	The underlying bit representatio	Yes	Automated	Required	

	ns of floating-point values shall not be used				
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 value	Yes	Automated	Required	
M4-10-2	Literal zero (0) shall not be used as the null-pointer-constant.	Yes	Automated	Required	
M5-0-2	Limited dependence should be placed on C+	Yes	Automated	Advisory	

	+ operator precedence rules in expressions			
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 reduce the size of the underlying type	Yes	Automated	Required
M5-0-7	There shall be no explicit floating-integral conversions of a cvalue expression	Yes	Automated	Required

M5-0-8	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes	Automated	Required	
M5-0-9	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes	Automated	Required	
M5-0-10	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand	Yes	Automated	Required	
M5-0-11	The plain char type	Yes	Automated	Required	

	shall only be used for the storage and use of character values				
M5-0-12	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes	Automated	Required	
M5-0-14	The first operand of a conditional-operator shall have type bool	Yes	Automated	Required	
M5-0-15	Array indexing over pointer arithmetic	Yes	Automated	Required	
M5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes	Automated	Required	
M5-0-17	Subtraction between pointers shall only be applied to	Yes	Automated	Required	

	pointers that address elements of the same array				
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	
M5-2-3	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	Yes	Automated	Required	

	pointer to a function to any other pointer type, including a pointer to function type				
M5-2-8	An object with integer type or pointer to void type shall not be converted to an object with pointer type.	Yes	Automated	Required	
M5-2-9	Pointer to Integer Cast	Yes	Automated	Required	
M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes	Automated	Required	
M5-2-11	The comma operator, && operator and the || operator shall not be overloaded	Yes	Automated	Required	
M5-2-12	Array to Pointer Decay	Yes	Automated	Required	
M5-3-1	Each	Yes	Automated	Required	

	operand of the ! operator, the logical && or the logical || operators shall have type bool			
M5-3-2	Unary Minus Operator Applied to an Expression with an Unsigned Type	Yes	Automated	Required
M5-3-3	The unary & operator shall not be overloaded	Yes	Automated	Required
M5-3-4	Evaluation of the operand to the sizeof operator shall not contain side effects	Yes	Automated	Required
M5-8-1	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes	Partially Automated	Required
M5-14-1	The right hand operand of a	Yes	Automated	Required

	logical &&, || operators shall not contain side effects				
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	
M6-2-1	Assignment operators shall not be used in sub-expressions	Yes	Automated	Required	
M6-2-2	Floating-point expressions shall not be directly or indirectly tested for equality or inequality	Yes	Partially Automated	Required	
M6-2-3	Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment,	Yes	Automated	Required	

	provided that the first character following the null statement is a white-space character				
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 statement. The else keyword shall be followed by either a compound statement, or another if statement	Yes	Automated	Required	
M6-4-2	All if and else if constructs shall be terminated with an else clause	Yes	Automated	Required	
M6-4-3	Switch Statement	Yes	Automated	Required	

	not Well-formed				
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	Yes	Automated	Required	

	condition, the loop-counter shall only be used as an operand to <=, <, > or >=				
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-variable other than the loop-counter shall not be modified within condition or expression	Yes	Automated	Required	
M6-5-6	A loop-control-variable other than the loop-counter which is modified in	Yes	Automated	Required	

	statement shall have type bool				
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 corresponding object is not modified	Yes	Automated	Required	

M7-3-2	The identifier main shall not be used for a function other than the global function main	Yes	Automated	Required	
M7-3-3	There shall be no unnamed namespaces in header files.	Yes	Automated	Required	
M7-3-4	Using-directives shall not be used.	Yes	Automated	Required	
M7-3-6	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.	Yes	Automated	Required	
M7-4-1	Assembly Language Code Usage 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	Yes	Automated	Required	

	language shall be encapsulated and isolated.				
M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes	Non-automated	Required	
M7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Yes	Non-automated	Required	
M8-0-1	Single Declarations	Yes	Automated	Required	
M8-3-1	Parameters in an overriding virtual function shall either use the same default arguments as the function they override,	Yes	Automated	Required	

	or else shall not specify any default arguments.				
M8-4-2	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.	Yes	Automated	Required	
M8-4-4	A function identifier shall either be used to call the function or it shall be preceded by &.	Yes	Automated	Required	
M8-5-2	Incorrect Initializer Lists	Yes	Automated	Required	
M9-3-1	Const Member Function Returning Non-Const Pointer or Reference	Yes	Automated	Required	
M9-3-3	If a member function can be made static then it shall be made static, otherwise if it can be made	Yes	Automated	Required	

	const then it shall be made const				
M9-6-4	Bit-field Length	Yes	Automated	Required	
M10-1-1	Class Derived From Virtual Bases	Yes	Automated	Advisory	
M10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes	Automated	Required	
M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes	Automated	Required	
M10-2-1	Similiar 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	
M15-1-1	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	Yes	Automated	Required	

	termination of the program			
M15-3-3	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases	Yes	Automated	Required
M15-3-4	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point	Yes	Automated	Required
M15-3-6	Order of Catch Blocks with Derived Classes	Yes	Automated	Required
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	
M16-2-3	Include	Yes	Automated	Required	

	guards shall be provided				
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> Library Functions	Yes	Automated	Required	
M18-0-4	Time Handling Functions of <ctime>	Yes	Automated	Required	
M18-0-5	Unbounded Functions of <cstring>	Yes	Automated	Required	
M18-2-1	The macro	Yes	Automated	Required	

	offsetof shall not be used				
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
MEM31-C	Free dynamically allocated memory when no longer needed	Yes			Medium
MEM33-C	Allocate and copy structures containing a flexible array member dynamically	Yes			Low
MEM34-C	Only free memory allocated dynamically	Yes			High
MEM35-C	Allocate sufficient memory for	Yes			High

	an object				
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 allocated resources	Yes			High
MEM52-CPP	Detect and handle memory allocation errors	Yes			High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No			High
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	Yes			

	Complexity				
METRIC_04	Program Unit Max Length	Yes			
METRIC_05	Program Unit Max Nesting Depth	Yes			
METRIC_06	Program Unit Parameters Count	Yes			
METRIC_07	Program Unit Path Count	Yes			
METRIC_08	Program Unit Statement Count	Yes			
METRIC_09	Coupling Between Object Classes	Yes			
METRIC_11	Depth of Inheritance Tree	Yes			
METRIC_12	Lack of Cohesion in Methods	Yes			
METRIC_13	Maintainability Index	Yes			
MISRA04_2.1	2.1 Assembly language shall be encapsulated and isolated.	Yes		Required	
MISRA04_2.2	2.2 only use /* comments	Yes		Required	
MISRA04_2.3	2.3 The character sequence /* shall not be used within a comment.	Yes		Required	
MISRA04_2.4	2.4 Sections of code	Yes		Advisory	

	should not be "commented out"				
MISRA04_4.1	4.1 Only those escape sequences that are defined in the ISO C standard shall be used	Yes		Required	
MISRA04_4.2	4.2 Trigraphs shall not be used	Yes		Required	
MISRA04_5.1	5.1 Identifiers shall not rely on the significance of more than 31 characters	Yes		Required	
MISRA04_5.2	5.2 Shadowed Identifiers	Yes		Required	
MISRA04_5.3	5.3 A typedef name shall be a unique identifier.	Yes		Required	
MISRA04_5.4	5.4 A tag name shall be a unique identifier	Yes		Required	
MISRA04_5.5	5.5 No object or function identifier with static storage duration should be reused	Yes		Advisory	
MISRA04_5.6	5.6 No identifier in one name space should	Yes		Advisory	

	have the same spelling as an identifier in another name space.				
MISRA04_5.7	5.7 No identifier name should be reused	Yes		Advisory	
MISRA04_6.1	6.1 The plain char type shall only be used for the storage and use of character values	Yes		Required	
MISRA04_6.2	6.2 Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes		Required	
MISRA04_6.3	6.3 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA04_6.4	6.4 Bit fields shall only be defined to be of type unsigned int	Yes		Required	

	or signed int.				
MISRA04_6.5	6.5 Bit fields of signed type shall be at least 2 bits long.(Fuzzy parser)	Yes		Required	
MISRA04_7.1	7.1 Octal constants (other than zero) and octal escape sequences shall not be used.	Yes		Required	
MISRA04_8.3	8.3 For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical	Yes		Required	
MISRA04_8.5	8.5 No definitions of objects or functions in a header file	Yes		Required	
MISRA04_8.6	8.6 Functions shall be declared at file scope	Yes		Required	
MISRA04_8.7	8.7 Objects shall be local if only accessed from one	Yes		Required	

	function				
MISRA04_8.8	8.8 An external object or function shall be declared in one and only one file	Yes		Required	
MISRA04_8.9	8.9 An identifier with external linkage shall have exactly one external definition	Yes		Required	
MISRA04_8.10	8.10 prefer internal linkage over external whenever possible	Yes		Required	
MISRA04_8.11	8.11 Use the static keyword for internal linkage	Yes		Required	
MISRA04_8.12	8.12 When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation	Yes		Required	
MISRA04_9.3	9.3 = construct in enumerator list shall only be used on	Yes		Required	

	either the first item alone, or all items explicitly.				
MISRA04_10.5	10.5 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	
MISRA04_10.6	10.6 A U suffix shall be applied to all constants of unsigned type	Yes		Required	
MISRA04_12.6	12.6 The operands of logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean	Yes		Advisory	

	should not be used as operands to operators other than (&&, ||, !, =, ==, != and ?:)				
MISRA04_12.8	12.8 The right-hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left-hand operand.	Yes		Required	
MISRA04_12.12	12.12 The underlying bit representations of floating-point values shall not be used	Yes		Required	
MISRA04_12.13	12.13 The increment (++) and decrement (--) operators should not be mixed with other operators in an expression	Yes		Advisory	
MISRA04_13.3	13.3	Yes		Required	

3	Floating-point expressions shall not be tested for equality or inequality				
MISRA04_13.6	13.6 Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop	Yes		Required	
MISRA04_14.1	14.1 There shall be no unreachable code	Yes		Required	
MISRA04_14.3	14.3 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	
MISRA04_14.4	14.4 The	Yes		Required	

4	goto statement shall not be used				
MISRA04_14.5	14.5 The continue statement shall not be used	Yes		Required	
MISRA04_14.6	14.6 For any iteration statement there shall be at most one break statement used for loop termination	Yes		Required	
MISRA04_14.7	14.7 A function shall have a single point of exit at the end of the function	Yes		Required	
MISRA04_14.8	14.8 The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes		Required	
MISRA04_14.9	14.9 An if (expression) construct shall be followed by a compound statement. The else	Yes		Required	

	keyword shall be followed by either a compound statement, or another if statement				
MISRA04_14.10	14.10 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA04_15.1	15.1 A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes		Required	
MISRA04_15.2	15.2 An unconditional break statement shall terminate every non-empty switch clause	Yes		Required	
MISRA04_15.3	15.3 The final clause of a switch statement shall be the default clause	Yes		Required	
MISRA04_15.	15.5 Every	Yes		Required	

5	switch statement shall have at least one case clause				
MISRA04_16.1	16.1 Functions shall not be defined with variable numbers of arguments.	Yes		Required	
MISRA04_16.2	16.2 Functions shall not call themselves, either directly or indirectly.	Yes		Required	
MISRA04_16.3	16.3 All prototype parameters must have an identifier.	Yes		Required	
MISRA04_16.4	16.4 use the same identifier in definition and declaration of functions.	Yes		Required	
MISRA04_16.5	16.5 Functions with no parameters need explicit void keyword	Yes		Required	
MISRA04_16.8	16.8 Always return a value in non-void functions	Yes		Required	
MISRA04_16.9	16.9 A function	Yes		Required	

	identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty				
MISRA04_17.3	17.3 >, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes		Required	
MISRA04_17.5	17.5 No more than 2 levels of pointer indirection	Yes		Advisory	
MISRA04_17.6	17.6 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	
MISRA04_18.4	18.4 Unions shall not be used	Yes		Required	
MISRA04_19.	19.1 #include	Yes		Advisory	

1	statements in a file should only be preceded by other preprocessor directives or comments				
MISRA04_19.2	19.2 Non-standard characters should not occur in header file names in #include directives	Yes		Advisory	
MISRA04_19.3	19.3 The #include directive shall be followed by either a <filename> or "filename" sequence	Yes		Required	
MISRA04_19.5	19.5 Macros shall not be #define'd or #undef'd within a block	Yes		Required	
MISRA04_19.6	19.6 #undef shall not be used	Yes		Required	
MISRA04_19.7	19.7 A function should be used in preference to a function-like macro	Yes		Advisory	

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_19.11	19.11 All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator	No		Required	
MISRA04_19.12	19.12 There shall be at most one	Yes		Required	

	occurrence of the # or ## operators in a single macro definition				
MISRA04_19.13	19.12 The # and ## operators should not be used	Yes		Advisory	
MISRA04_19.14	19.14 The defined preprocessor operator shall only be used in one of the two standard forms	Yes		Required	
MISRA04_19.15	19.15 Precautions shall be taken in order to prevent the contents of a header file being included twice	Yes		Required	
MISRA04_19.17	20.14 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they	Yes		Required	

	are related				
MISRA04_20.1	20.1 Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined	Yes		Required	
MISRA04_20.2	20.2 The names of standard library macros and objects shall not be reused	Yes		Required	
MISRA04_20.4	20.4 Dynamic heap memory allocation shall not be used	Yes		Required	
MISRA04_20.5	20.5 The error indicator "errno" shall not be used	Yes		Required	
MISRA04_20.6	20.6 The macro offsetof, in library <stddef.h>, shall not be used	Yes		Required	
MISRA04_20.7	20.7 The setjmp macro and the longjmp	Yes		Required	

	function shall not be used				
MISRA04_20.8	20.8 The signal handling facilities of <signal.h> shall not be used	Yes		Required	
MISRA04_20.9	20.9 The input output library <stdio.h> shall not be used in production code	Yes		Required	
MISRA04_20.10	20.10 The library functions atof, atoi and atol from library <stdlib.h> shall not be used	Yes		Required	
MISRA04_20.11	20.11 The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used	Yes		Required	
MISRA04_20.12	20.12 The time handling functions of library <time.h> shall not be	Yes		Required	

	used				
MISRA04_21.1	21.1 Minimisation of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/ techniques; (b) dynamic analysis tools/ techniques; (c) explicit coding of checks to handle run-time faults.	Yes		Required	
MISRA08_0-1-1	0-1-1 A project shall not contain unreachable code	Yes		Required	
MISRA08_0-1-2	0-1-2 Infeasible Paths	Yes		Required	
MISRA08_0-1-3	0-1-3 A project shall not contain unused variables	Yes		Required	
MISRA08_0-1-4	0-1-4 A project shall not contain non-volatile POD variables having only one use.	Yes		Required	

MISRA08_0-1-5	0-1-5 A project shall not contain unused type declarations	Yes		Required	
MISRA08_0-1-7	0-1-7 The value returned by a function having a non-void return type that is not an overloaded operator shall always be used	Yes		Required	
MISRA08_0-1-8	0-1-8 All functions with void return type shall have external side effect(s)	Yes		Required	
MISRA08_0-1-10	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	Yes		Required	

	function and all the functions that override it				
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"	Yes		Required	
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-	2-10-4 A	Yes		Required	

10-4	class, union or enum name (including 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_2-13-1	2-13-1 escape sequences are standardized	Yes		Required	
MISRA08_2-13-2	2-13-2 Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes		Required	
MISRA08_2-13-3	2-13-3 A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes		Required	
MISRA08_2-	2-13-4 Literal	Yes		Required	

13-4	suffixes shall be upper case				
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_3-1-3	3-1-3 When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization	Yes		Required	
MISRA08_3-2-1	3-2-1 All declarations of an object or function shall have compatible	Yes		Required	

	types				
MISRA08_3-2-2	3-2-2 The One Definition Rule	Yes		Required	
MISRA08_3-2-3	3-2-3 A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes		Required	
MISRA08_3-2-4	3-2-4 An identifier with external linkage shall have exactly one definition	Yes		Required	
MISRA08_3-3-1	3-3-1 Objects or functions with external 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	Yes		Required	

	at Lowest Scope				
MISRA08_3-9-1	3-9-1 The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations	Yes		Required	
MISRA08_3-9-2	3-9-2 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA08_3-9-3	3-9-3 The underlying bit representations of floating-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	Yes		Required	

	other than the assignment operator =, the logical operators &&, ||, !, the equality operators == 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	Yes		Required	

	used as an integer value				
MISRA08_4-10-2	4-10-2 Literal zero (0) shall not be used as the null-pointer-constant.	Yes		Required	
MISRA08_5-0-2	5-0-2 Limited dependence should be placed on C++ operator precedence rules in expressions	Yes		Advisory	
MISRA08_5-0-3	5-0-3 A cvalue expression shall not be implicitly converted to a different underlying type	Yes		Required	
MISRA08_5-0-4	5-0-4 An implicit 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_5-0-14	5-0-14 The first operand of a conditional-	Yes		Required	

	operator shall have type bool				
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	Yes		Required	

	shall only be applied to operands of unsigned underlying type				
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_5-2-5	5-2-5 A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes		Required	
MISRA08_5-2-6	5-2-6 A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type	Yes		Required	
MISRA08_5-2-8	5-2-8 An object with integer type or pointer to void type shall not be converted to an object with pointer	Yes		Required	

	type.				
MISRA08_5-2-9	5-2-9 Pointer to Integer Cast	Yes		Advisory	
MISRA08_5-2-10	5-2-10 The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes		Advisory	
MISRA08_5-2-11	5-2-11 The comma operator, && operator and the || operator shall not be overloaded	Yes		Required	
MISRA08_5-2-12	5-2-12 Array to Pointer Decay	Yes		Required	
MISRA08_5-3-1	5-3-1 Each operand of the ! operator, the logical && or the logical || operators shall have type bool	Yes		Required	
MISRA08_5-3-3	5-3-3 The unary & operator shall not be overloaded	Yes		Required	
MISRA08_5-	5-3-4	Yes		Required	

3-4	Evaluation of the operand to the sizeof operator shall not contain side effects				
MISRA08_5-8-1	5-8-1 The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes		Required	
MISRA08_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	Yes		Required	

	character following the null statement is a white-space character				
MISRA08_6-3-1	6-3-1 The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes		Required	
MISRA08_6-4-1	6-4-1 An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement	Yes		Required	
MISRA08_6-4-2	6-4-2 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA08_6-4-4	6-4-4 A switch-label shall only be	Yes		Required	

	used when the most closely-enclosing compound statement is the body of a switch statement				
MISRA08_6-4-5	6-4-5 An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes		Required	
MISRA08_6-4-6	6-4-6 The final clause of a switch statement shall be the default-clause	Yes		Required	
MISRA08_6-4-8	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 operand to <=, <, > or >=	Yes		Required	
MISRA08_6-5-3	6-5-3 The loop-counter shall not be modified within condition or statement	Yes		Required	
MISRA08_6-5-4	6-5-4 The loop-counter shall be modified by one of: --, ++, -= n, or += n; where n remains constant for the duration of the loop	Yes		Required	
MISRA08_6-5-5	6-5-5 A loop-control-variable other than the loop-counter shall not be modified within condition or expression	Yes		Required	
MISRA08_6-	6-5-6 A	Yes		Required	

5-6	loop-control-variable other than the loop-counter which is modified in statement shall have type bool				
MISRA08_6-6-1	6-6-1 Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement	Yes		Required	
MISRA08_6-6-2	6-6-2 The goto statement shall jump to a label declared later in the same function body	Yes		Required	
MISRA08_6-6-4	6-6-4 For any iteration statement there shall be no more than one break or goto statement used for loop	Yes		Required	

	termination				
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_7-3-2	7-3-2 The identifier	Yes		Required	

	main shall not be used for a function other than the global 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 Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier	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-	7-4-2	Yes		Required	

4-2	Assembler instructions shall only be introduced using the asm declaration.				
MISRA08_7-4-3	7-4-3 Assembly language shall be encapsulated and isolated.	Yes		Required	
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_7-5-4	7-5-4 Functions should not	Yes		Advisory	

	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	Yes		Required

	be used to call 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	Automated	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 functions shall not return non-const pointers or references to class-data	Yes		Required	
MISRA08_9-3-2	9-3-2 Member functions shall not return non-const handles to	Yes		Required	

	class-data				
MISRA08_9-3-3	9-3-3 If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const	Yes		Required	
MISRA08_9-5-1	9-5-1 Unions shall not be used	Yes		Required	
MISRA08_9-6-2	9-6-2 Bool, Unsigned, or Signed Bit-fields	Yes		Required	
MISRA08_9-6-3	9-6-3 Enum Bit-fields	Yes		Required	
MISRA08_9-6-4	9-6-4 (Fuzzy parser) Named bit-fields with signed integer type shall have a length of more than one bit	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	Yes		Required	

	used in a diamond hierarchy				
MISRA08_10 -1-3	10-1-3 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA08_10 -3-1	10-3-1 There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy	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 A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual	Yes		Required	

MISRA08_11-0-1	11-0-1 Member data in non-POD class types shall be private	Yes		Required	
MISRA08_12-1-1	12-1-1 An object's dynamic type shall not be used from the body of its constructor or destructor	Yes		Required	
MISRA08_12-1-2	12-1-2 Explicitly call all immediate and virtual base classes	Yes		Advisory	
MISRA08_12-1-3	12-1-3 All constructors that are callable with a single argument of fundamental type shall be declared explicit.	Yes		Required	
MISRA08_12-8-1	12-8-1 A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member	Yes		Required	

MISRA08_14-5-2	14-5-2 A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter	Yes		Required	
MISRA08_14-5-3	14-5-3 A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter	Yes		Required	
MISRA08_14-7-1	14-7-1 All class templates, function templates, class template member functions and class template static members shall be instantiated at least once	Yes		Required	

MISRA08_14 -8-1	14-8-1 Overloaded function templates shall not be explicitly specialized	Yes		Required	
MISRA08_15 -0-2	15-0-2 An exception object should not have pointer type	Yes		Advisory	
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 -1-2	15-1-2 NULL shall not be thrown explicitly	Yes		Required	
MISRA08_15 -1-3	15-1-3 An empty throw (throw;) shall only be used in the compound- statement of a catch handler	Yes		Required	
MISRA08_15 -3-1	15-3-1 Exceptions shall be raised only after start-up and before termination of the	Yes		Required	

	program				
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_15-3-5	15-3-5 A class type exception shall always be caught by reference	Yes		Required	
MISRA08_15-3-6	15-3-6 Order of Catch Blocks with Derived Classes	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_16-0-1	16-0-1 #include	Yes		Required	

	directives in a file shall only be preceded by other preprocessor directives or comments				
MISRA08_16-0-2	16-0-2 Macros shall only be #define'd or #undef'd in the global namespace	Yes		Required	
MISRA08_16-0-3	16-0-3 #undef shall not be used	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,	Yes		Required	

	unless it is used as the operand of # or ##				
MISRA08_16-0-7	16-0-7 Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes		Required	
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_16-2-1	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 C++ macros shall only be used for include guards, type	Yes		Required	

	qualifiers, or storage class specifiers				
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_16-3-1	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_17-0-1	17-0-1 Reserved identifiers, macros and functions in the standard library shall	Yes		Required	

	not be defined, redefined or undefined				
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_17-0-5	17-0-5 The setjmp macro and the longjmp function shall not be used	Yes		Required	
MISRA08_18-0-1	18-0-1 The C library shall not be used	Yes		Required	
MISRA08_18-0-2	18-0-2 The library functions atof, atoi and atol from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-3	18-0-3 The library functions abort, exit, getenv and system from library	Yes		Required	

	<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 heap memory allocation shall not be used.	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_1.1	1.1 The	Yes		Required	

	program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementati on's translation limits				
MISRA12_1.2	1.2 Language extensions should not be used	No		Advisory	
MISRA12_1.3	1.3 There shall be no occurrence of undefined or critical unspecified behaviour	No		Required	
MISRA12_2.1	2.1 A project shall not contain unreachable code	Yes		Required	
MISRA12_2.2	2.2 There shall be no dead code	No		Required	
MISRA12_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA12_2.4	2.4 A project should not contain unused tag declarations	Yes		Advisory	
MISRA12_2.5	2.5 A project	Yes		Advisory	

	should not contain unused macro declarations				
MISRA12_2.6	2.6 Unused Labels	Yes		Required	
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 shall not be used in // comments	Yes		Required	
MISRA12_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA12_4.2	4.2 Trigraphs should not be used	Yes		Advisory	
MISRA12_5.1	5.1 External identifiers shall be distinct	Yes		Required	
MISRA12_5.2	5.2 Identifiers declared in the same scope and name space shall be distinct	Yes		Required	

MISRA12_5.3	5.3 Shadowed Identifiers	Yes		Required	
MISRA12_5.4	5.4 Macro identifiers shall be distinct	Yes		Required	
MISRA12_5.5	5.5 Identifiers shall be distinct from macro names	Yes		Required	
MISRA12_5.6	5.6 A typedef name shall be a unique identifier	Yes		Required	
MISRA12_5.7	5.7 A tag name shall be a unique identifier	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	Yes		Required	

	type				
MISRA12_6.2	6.2 Single-bit named bit fields shall not be of a signed type	Yes		Required	
MISRA12_7.1	7.1 Octal constants shall not be used	Yes		Required	
MISRA12_7.2	7.2 A u or U suffix shall be applied to all integer constants that are represented in an unsigned type	Yes		Required	
MISRA12_7.3	7.3 The lowercase character L shall not be used in a literal suffix	Yes		Required	
MISRA12_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	No		Required	
MISRA12_8.1	8.1 Types shall be explicitly specified	Yes		Required	
MISRA12_8.2	8.2 Use	Yes			

	Named Parameters and Prototype Form				
MISRA12_8.3	8.3 All declarations of an object or function shall use the same names and type qualifiers	Yes		Required	
MISRA12_8.4	8.4 A compatible declaration shall be visible when an object or function with external linkage is defined	Yes		Required	
MISRA12_8.5	8.5 An external object or function shall be declared once in one and only one file	Yes		Required	
MISRA12_8.6	8.6 An identifier with external linkage shall have exactly one external definition	Yes		Required	
MISRA12_8.7	8.7 Functions and objects should not be defined with	Yes		Advisory	

	external linkage if they are referenced in only one translation unit				
MISRA12_8.8	8.8 Use the static keyword for internal linkage	Yes		Required	
MISRA12_8.9	8.9 Objects shall be local if only accessed from one function	Yes		Advisory	
MISRA12_8.1 0	8.10 Non-static Inline Functions	Yes		Required	
MISRA12_8.1 1	8.11 When an array with external linkage is declared, its size should be explicitly specified	Yes		Advisory	
MISRA12_8.1 2	8.12 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA12_8.1 3	8.13 A pointer	No		Advisory	

	should point to a const-qualified type whenever possible				
MISRA12_8.14	8.14 The restrict type qualifier shall not be used	Yes		Required	
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 partially initialized	Yes		Required	
MISRA12_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA12_9.5	9.5 Where designated initializers are used to initialize an array object the size of	Yes		Required	

	the array shall be specified explicitly				
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	No		Required	
MISRA12_10.3	10.3 The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category	No		Required	
MISRA12_10.4	10.4 Both operands of an operator in which the usual arithmetic	Yes		Required	

	conversions are performed shall have the same essential type category				
MISRA12_10.5	10.5 The value of an expression should not be cast to an inappropriate essential type	Yes		Advisory	
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 If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have	No		Required	

	wider essential type				
MISRA12_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	
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	Yes		Required	

	to a different object type				
MISRA12_11.4	11.4 A conversion should not be performed between a pointer to object and an integer type	Yes		Required	
MISRA12_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA12_11.6	11.6 A cast shall not be performed between pointer to void and an arithmetic type	Yes		Required	
MISRA12_11.7	11.7 A cast shall not be performed between pointer to object and a non-integer arithmetic type	Yes		Required	
MISRA12_11.8	11.8 A cast shall not remove any const or volatile qualification from the type	Yes		Required	

	pointed to by a pointer				
MISRA12_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA12_12.1	12.1 The precedence of operators within expressions should be made explicit	No		Advisory	
MISRA12_12.2	12.2 The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes		Required	
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	No		Advisory	

	unsigned integer wrap-around				
MISRA12_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA12_13.2	13.2 The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders	No		Required	
MISRA12_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	
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_13.6	13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects	Yes		Mandatory	
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	No		Required	
MISRA12_14.3	14.3 Controlling expressions shall not be invariant	No		Required	
MISRA12_14.4	14.4 The controlling expression of an if statement and the controlling expression of an iteration-statement	Yes		Required	

	shall have essentially Boolean type				
MISRA12_15.1	15.1 The goto statement should not be used	Yes		Advisory	
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 Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement	Yes		Required	
MISRA12_15.4	15.4 There should be no more than one break or goto statement used to terminate any iteration statement	Yes		Advisory	
MISRA12_15.5	15.5 A function should have	Yes		Advisory	

	a single point of exit at the end				
MISRA12_15.6	15.6 The body of an iteration-statement or a selection-statement shall be a compound-statement	Yes		Required	
MISRA12_15.7	15.7 All if ... else if constructs shall be terminated with an else statement	Yes		Required	
MISRA12_16.1	Switch Statement not Well-formed	Yes		Required	
MISRA12_16.2	16.2 A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes		Required	
MISRA12_16.3	16.3 An unconditional break statement shall terminate every switch-clause	Yes		Required	

MISRA12_16.4	16.4 Every switch statement shall have a default label	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_16.6	16.6 Every switch statement shall have at least two switch-clauses	Yes		Required	
MISRA12_16.7	16.7 A switch-expression shall not have essentially Boolean type	No		Required	
MISRA12_17.1	17.1 The features of <stdarg.h> shall not be used	Yes		Required	
MISRA12_17.2	17.2 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA12_17.3	17.3 A function shall	Yes		Mandatory	

	not be declared implicitly				
MISRA12_17.4	17.4 Always return a value in non-void functions	Yes		Required	
MISRA12_17.5	17.5 The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements	No		Advisory	
MISRA12_17.6	17.6 The declaration of an array parameter shall not contain the static keyword between the [ ]	Yes		Mandatory	
MISRA12_17.7	17.7 The value returned by a function having non-void return type shall be used	Yes		Required	
MISRA12_17.8	17.8 A function parameter should not be	Yes		Advisory	

	modified				
MISRA12_18.1	18.1 A pointer resulting from arithmetic on a pointer operand shall address	No		Required	
MISRA12_18.2	18.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA12_18.3	18.3 The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object	Yes		Required	
MISRA12_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	No		Advisory	
MISRA12_18.5	18.5 Declarations should contain no	No		Advisory	

	more than two levels of pointer nesting				
MISRA12_18.6	18.6 The address of an object with automatic storage shall not be copied to another object that persists after the first object has ceased to exist	Yes		Required	
MISRA12_18.7	18.7 Flexible array members shall not be declared	Yes		Required	
MISRA12_18.8	18.8 Variable-length array types shall not be used	No		Required	
MISRA12_19.1	19.1 An object shall not be assigned or copied to an overlapping object	No		Mandatory	
MISRA12_19.2	19.2 The Union keyword should not be used	Yes		Advisory	
MISRA12_20.1	20.1 #include directives	Yes		Advisory	

	should only be preceded by preprocessor directives or comments				
MISRA12_20.2	20.2 The ', " or backslash characters and the /* or // character sequences shall not occur in a header file name	Yes		Required	
MISRA12_20.3	20.3 The #include directive shall be followed by either a \<filename\> or "filename" sequence	Yes		Required	
MISRA12_20.4	20.4 A macro shall not be defined with the same name as a keyword	Yes		Required	
MISRA12_20.5	20.5 #undef should not be used	Yes		Advisory	
MISRA12_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro	Yes		Required	

	argument				
MISRA12_20.7	20.7 Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	No		Required	
MISRA12_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	No		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	No		Required	
MISRA12_20.10	20.10 The # and ## operators should not be used	Yes		Advisory	
MISRA12_20.11	20.11 A macro parameter immediately following a # operator shall	Yes		Required	

	not immediately be followed by a ## operator			
MISRA12_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	No		Required
MISRA12_20.13	20.13 Invalid Preprocessor Directives	Yes		Required
MISRA12_20.14	20.14 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes		Required
MISRA12_21.1	21.1 #define and #undef shall not be used on a reserved	Yes		Required

	identifier or reserved macro name				
MISRA12_21.2	21.2 Reserved Identifiers or Macros	Yes		Required	
MISRA12_21.3	21.3 The memory allocation and deallocation functions of <stdlib.h> shall not be used	Yes		Required	
MISRA12_21.4	21.4 The standard header file <setjmp.h> shall not be used	Yes		Required	
MISRA12_21.5	21.5 The standard header file signal.h shall not be used	Yes		Required	
MISRA12_21.6	21.6 The Standard Library input/output functions shall not be used	Yes		Required	
MISRA12_21.7	21.7 The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used	Yes		Required	
MISRA12_21.	21.8 The	Yes		Required	

8	library functions abort, exit, getenv and system of <stdlib.h> shall not be used				
MISRA12_21.9	21.9 The library functions bsearch and qsort of <stdlib.h> shall not be used	Yes		Required	
MISRA12_21.10	21.10 The Standard Library time and date functions shall not be used	Yes		Required	
MISRA12_21.11	21.11 The standard header file <tgmath.h> shall not be used	Yes		Required	
MISRA12_21.12	21.12 The exception handling features of <fenv.h> should not be used	Yes		Advisory	
MISRA12_22.1	22.1 All resources obtained dynamically by means of Standard	No		Required	

	Library functions shall be explicitly released				
MISRA12_22.2	22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function	No		Mandatory	
MISRA12_22.3	22.3 The same file shall not be open for read and write access at the same time on different streams	No		Required	
MISRA12_22.4	22.4 There shall be no attempt to write to a stream which has been opened as read-only	No		Mandatory	
MISRA12_22.5	22.5 A pointer to a FILE object shall not be dereferenced	No		Mandatory	
MISRA12_22.6	22.6 The value of a pointer to a FILE shall not be used after the	No		Mandatory	

	associated stream has been closed				
MISRA12_DIR_1.1	Directive 1.1 Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	No		Required	
MISRA12_DIR_2.1	Directive 2.1 All source files shall compile without any compilation errors	Yes		Required	
MISRA12_DIR_3.1	Directive 3.1 All code shall be traceable to documented requirements	No		Required	
MISRA12_DIR_4.1	Directive 4.1 Run-time failures shall be minimized	No		Required	
MISRA12_DIR_4.2	Directive 4.2 All usage of assembly language should be documented	No		Advisory	
MISRA12_DIR_4.3	Directive 4.3 Assembly language	Yes		Required	

	shall be encapsulated and isolated.				
MISRA12_DIR_4.4	Directive 4.4 Sections of code should not be "commented out"	Yes		Advisory	
MISRA12_DIR_4.5	Directive 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Yes		Advisory	
MISRA12_DIR_4.6	Directive 4.6 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA12_DIR_4.7	Directive 4.7 If a function generates error information, then that error information shall be tested	No		Required	
MISRA12_DIR_4.8	Directive 4.8 If a pointer to	Yes		Advisory	

	a structure or union is never dereferenced within a translation unit, then the implementation of the object should 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	No		Advisory	
MISRA12_DIR_4.10	Directive 4.10 Precautions shall be taken in order to prevent the contents of a header file being included more than once	Yes		Required	
MISRA12_DIR_4.11	Directive 4.11 The validity of values passed to library functions shall be checked	No		Required	

MISRA12_DIR_4.12	Directive 4.12 Dynamic memory allocation shall not be used	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	No		Advisory	
MISRA23_0.0.1	0.0.1 A function shall not contain unreachable statements	Yes		Required	
MISRA23_0.1.2	0.1.2 The value returned by a function shall be used	Yes		Required	
MISRA23_0.2.1	0.2.1 Variables with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.2	0.2.2 A named function parameter shall be used at least once	Yes		Required	
MISRA23_0.2.3	0.2.3 Types with limited	Yes		Advisory	

	visibility should be used at least once				
MISRA23_0.2.4	0.2.4 Functions with limited visibility should be used at least once	Yes		Advisory	
MISRA23_1.1	1.1 The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation's translation limits	Yes		Required	
MISRA23_1.2	1.2 Language extensions should not be used	No		Advisory	
MISRA23_1.3	1.3 There shall be no occurrence of undefined or critical unspecified behaviour	No		Required	
MISRA23_2.1	2.1 A project shall not contain unreachable code	Yes		Required	
MISRA23_2.2	2.2 A project shall not	No		Required	

	contain dead code				
MISRA23_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA23_2.4	2.4 A project should not contain unused tag declarations	Yes		Advisory	
MISRA23_2.5	2.5 A project should not contain unused macro declarations	Yes		Advisory	
MISRA23_2.6	2.6 A function should not contain unused label declarations	Yes		Required	
MISRA23_2.7	2.7 A function should not contain unused parameters	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 shall not be used in // comments	Yes		Required	

MISRA23_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA23_4.2	4.2 Trigraphs should not be used	Yes		Advisory	
MISRA23_5.0.1	5.0.1 Trigraph-like sequences should not be used	Yes		Required	
MISRA23_5.1	5.1 External identifiers shall be distinct	Yes		Required	
MISRA23_5.2	5.2 Identifiers declared in the same scope and name space shall be distinct	Yes		Required	
MISRA23_5.3	5.3 An identifier declared in an inner scope shall not hide an identifier declared in an outer scope	Yes		Required	
MISRA23_5.4	5.4 Macro identifiers shall be distinct	Yes		Required	
MISRA23_5.5	5.5 Identifiers shall be distinct from macro names	Yes		Required	

MISRA23_5.6	5.6 A typedef name shall be a unique identifier	Yes		Required	
MISRA23_5.7	5.7 A tag name shall be a unique identifier	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 shall not be used in // comments	Yes		Required	
MISRA23_5.8	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes		Required	
MISRA23_5.9	5.9 Identifiers that define objects or functions with internal linkage should be	Yes		Advisory	

	unique				
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	Yes		Required	

	character in a literal suffix				
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 specified	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	Yes		Required	

	appropriate type				
MISRA23_6.2	6.2 Single-bit named bit fields shall not be of a signed type	Yes		Required	
MISRA23_6.2.4	6.2.4 A header file shall not contain definitions of functions or objects that are non-inline and have external linkage	Yes		Required	
MISRA23_6.3	6.3 A bit field shall not be declared as a member of a union	Yes		Required	
MISRA23_6.4.2	6.4.2 Derived classes shall not conceal functions that are inherited from their bases	Yes		Required	
MISRA23_6.5.1	6.5.1 A function or object with external linkage should be introduced in a header file	Yes		Advisory	
MISRA23_6.5.2	6.5.2 Internal linkage should be	Yes		Advisory	

	specified appropriately				
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_6.8.2	6.8.2 A function must not return a reference or a pointer to a local variable with automatic storage duration	Yes		Mandatory	
MISRA23_6.9.1	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	Yes		Required	

	conversion from type bool			
MISRA23_7.0.3	7.0.3 The numerical value of a character shall not be used	Yes	Required	
MISRA23_7.1	7.1 Octal constants shall not be used	Yes	Required	
MISRA23_7.2	7.2 A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type	Yes	Required	
MISRA23_7.3	7.3 The lowercase character "l" shall not be used in a literal suffix	Yes	Required	
MISRA23_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	No	Required	
MISRA23_7.1	7.11.1 nullptr	Yes	Required	

1.1	shall be the only form of the null-pointer-constant				
MISRA23_7.11.2	7.11.2 Array to Pointer Decay	Yes		Required	
MISRA23_8.1	8.1 Types shall be explicitly specified	Yes		Required	
MISRA23_8.1.1	8.1.1 A non-transient lambda shall not implicitly capture this	Yes		Required	
MISRA23_8.1.2	8.1.2 Variables should be captured explicitly in a non-transient lambda	Yes		Advisory	
MISRA23_8.2	8.2 Use Named Parameters and Prototype Form	Yes		Required	
MISRA23_8.2.3	8.2.3 A cast shall not remove any const or volatile qualification from the type accessed via a pointer or by reference	Yes		Required	
MISRA23_8.2.5	8.2.5 reinterpret_c	Yes		Required	

	ast shall not be used				
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_8.3	8.3 All declarations of an object or function	Yes		Required	

	shall use the same names and type qualifiers				
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_8.4	8.4 A compatible declaration shall be visible when an object or function with external linkage is defined	Yes		Required	
MISRA23_8.5	8.5 An external object or function shall be declared once in one and only one file	Yes		Required	
MISRA23_8.6	8.6 An identifier with external linkage shall have exactly one external definition	Yes		Required	

MISRA23_8.7	8.7 Functions and objects should not be defined with external linkage if they are referenced in only one translation unit	Yes		Advisory	
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_8.8	8.8 The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage	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_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA23_8.11	8.11 When an array with external linkage is declared, its size should be explicitly specified	Yes		Advisory	
MISRA23_8.12	8.12 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA23_8.13	8.13 A pointer should point to a const-qualified type whenever possible	No		Advisory	
MISRA23_8.14	8.14 The restrict type qualifier shall not be used	Yes		Required	
MISRA23_8.14.1	8.14.1 The right-hand operand of a logical && or || operator should not contain persistent side effects	Yes		Advisory	

MISRA23_8.15	8.15 All declarations of an object with an explicit alignment specification shall specify the same alignment	Yes		Required	
MISRA23_8.16	8.16 The alignment specification of zero should not appear in an object declaration	Yes		Advisory	
MISRA23_8.17	8.17 At most one explicit alignment specifier should appear in an object declaration	Yes		Advisory	
MISRA23_8.19.1	8.19.1 The comma operator shall not be used.	Yes		Advisory	
MISRA23_9.1	9.1 The value of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
MISRA23_9.2	9.2 The initializer for an aggregate	Yes		Required	

	or union shall be enclosed in braces			
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 element of an object shall not be initialized more than once	Yes	Required	
MISRA23_9.4.1	9.4.1 All if ... else if constructs shall be terminated with an else statement	Yes	Required	
MISRA23_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes	Required	

MISRA23_9.5.2	9.5.2 A for-range-initializer shall contain at most one function call	Yes		Required	
MISRA23_9.6.1	9.6.1 The goto statement should not be used	Yes		Advisory	
MISRA23_9.6.2	9.6.2 A goto statement shall reference a label in a surrounding block	Yes		Required	
MISRA23_9.6.3	9.6.3 The goto statement shall jump to a label declared later in the function body	Yes		Required	
MISRA23_9.6.4	9.6.4 A function declared with the [[noreturn]] attribute shall not return	Yes		Required	
MISRA23_9.6.5	9.6.5 A function with non-void return type shall return a value on all paths	Yes		Required	
MISRA23_9.7	9.7 Atomic	Yes		Mandatory	

	objects shall be appropriately initialized before being accessed				
MISRA23_10.0.1	10.0.1 A declaration should not declare more than one variable or member variable	Yes		Advisory	
MISRA23_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA23_10.1.1	10.1.1 The target type of a pointer or lvalue reference parameter should be const-qualified appropriately	Yes		Advisory	
MISRA23_10.1.2	10.1.2 The volatile qualifier shall be used appropriately	Yes		Required	
MISRA23_10.2	10.2 Expressions of essentially character type shall not be used	No		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_10.3	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	No		Required	
MISRA23_10.3.1	10.3.1 There should be no unnamed namespaces in header files	Yes		Advisory	
MISRA23_10.4	10.4 Both operands of an operator in which the usual	Yes		Required	

	arithmetic conversions are performed shall have the same essential type category				
MISRA23_10.4.1	10.4.1 The asm declaration shall not be used	Yes		Required	
MISRA23_10.5	10.5 The value of an expression should not be cast to an inappropriate essential type	Yes		Advisory	
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 If a composite expression is used as one operand of an operator in which the usual arithmetic conversions	No		Required	

	are performed then the other operand shall not have wider essential type				
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.	11.3 A cast	Yes		Required	

3	shall not be performed between a pointer to object type and a pointer to a different object type				
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		Required	
MISRA23_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA23_11.6	11.6 A cast shall not be performed between	Yes		Required	

	pointer to void and an arithmetic type				
MISRA23_11.6.1	11.6.1 All variables should be initialized	Yes		Advisory	
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_11.7	11.7 A cast shall not be performed between pointer to object and a non-integer arithmetic type	Yes		Required	
MISRA23_11.8	11.8 A conversion shall not remove any const, volatile or _Atomic qualification from the type pointed to by a pointer	Yes		Required	
MISRA23_11.9	11.9 The macro NULL shall be the	Yes		Required	

	only permitted form of integer null pointer constant				
MISRA23_12.1	12.1 The precedence of operators within expressions should be made explicit	No		Advisory	
MISRA23_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	
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	Yes		Required	

	of one bit				
MISRA23_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA23_12.3.1	12.3.1 The union keyword shall not be used	Yes		Required	
MISRA23_12.4	12.4 Evaluation of constant expressions should not lead to unsigned integer wrap-around	No		Advisory	
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_13.2	13.2 The value of an expression and its persistent side effects	No		Required	

	shall be the same under all permitted evaluation orders			
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 member functions shall use the virtual, override and final specifiers appropriately	Yes	Required	
MISRA23_13.3.2	13.3.2 Parameters in an overriding virtual function shall not specify different default arguments	Yes	Required	
MISRA23_13.	13.3.3 The	Yes	Required	

3.3	parameters in all declarations or overrides of a function shall either be unnamed or have identical names				
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_13.6	13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects	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	Yes		Advisory	

	members should be either all private or all public				
MISRA23_14.2	14.2 A for loop shall be well-formed	No		Required	
MISRA23_14.3	14.3 Controlling expressions shall not be invariant	No		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 copy and move member functions of a class should have appropriate signatures	Yes		Advisory	
MISRA23_15.1	15.1 The goto statement should not be used	Yes		Advisory	
MISRA23_15.1.1	15.1.1 An object's	Yes		Required	

	dynamic type shall not be used from within its constructor or destructor				
MISRA23_15.1.3	15.1.3 Conversion operators and constructors that are callable with a single argument shall be explicit	Yes		Required	
MISRA23_15.1.5	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 Any label referenced by a goto statement shall be declared in the same	Yes		Required	

	block, or in any block enclosing the goto statement			
MISRA23_15.4	15.4 There should be no more than one break or goto statement used to terminate any iteration statement	Yes	Advisory	
MISRA23_15.5	15.5 A function should have a single point of exit at the end	Yes	Advisory	
MISRA23_15.6	15.6 The body of an iteration-statement or a selection-statement shall be a compound-statement	Yes	Required	
MISRA23_15.7	15.7 All if ... else if constructs shall be terminated with an else statement	Yes	Required	
MISRA23_16.1	Switch Statement not Well-formed	Yes	Required	
MISRA23_16.	16.2 A switch	Yes	Required	

2	label shall only be used when the most closely-enclosing compound statement is the body of a switch statement				
MISRA23_16.3	16.3 An unconditional break statement shall terminate every switch-clause	Yes		Required	
MISRA23_16.4	16.4 Every switch statement shall have a default label	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	Yes		Required	

	overloaded				
MISRA23_16.6	16.6 Every switch statement shall have at least two switch-clauses	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 A switch-expression shall not have essentially Boolean type	No		Required	
MISRA23_17.1	17.1 The standard header file <stdarg.h> shall not be used	Yes		Required	
MISRA23_17.2	17.2 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA23_17.3	17.3 A function shall not be declared implicitly	Yes		Mandatory	

MISRA23_17.4	17.4 All exit paths from a function with non-void return type shall have an explicit return statement with an expression	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	No		Advisory	
MISRA23_17.6	17.6 The declaration of an array parameter shall not contain the static keyword between the [ ]	Yes		Mandatory	
MISRA23_17.7	The value returned by a function having non-void return type shall be used	Yes		Required	
MISRA23_17.	17.8 A	Yes		Advisory	

8	function parameter should not be modified				
MISRA23_17.8.1	17.8.1 Function templates shall not be explicitly specialized	Yes		Required	
MISRA23_17.10	17.10 A function declared with a _Noreturn function specifier shall have void return type	Yes		Required	
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	18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of	No		Required	

	the same array as that pointer operand				
MISRA23_18.1.1	18.1.1 An exception object shall not have pointer type	Yes		Required	
MISRA23_18.1.2	18.1.2 An empty throw shall only occur within the compound-statement of a catch handler	Yes		Required	
MISRA23_18.2	18.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA23_18.3	18.3 The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object	Yes		Required	
MISRA23_18.	18.3.1 There	Yes		Advisory	

3.1	should be at least one exception handler to catch all otherwise unhandled exceptions				
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	No		Advisory	
MISRA23_18.5	18.5 Declarations should contain no more than two levels of pointer	No		Advisory	

	nesting				
MISRA23_18.5.2	18.5.2 Program-terminating functions should not be used	Yes		Advisory	
MISRA23_18.6	18.6 The address of an object with automatic or thread-local storage shall not be copied to another object that persists after the first object has ceased to exist	Yes		Required	
MISRA23_18.7	18.7 Flexible array members shall not be declared	Yes		Required	
MISRA23_18.8	18.8 Variable-length array types shall not be used	No		Required	
MISRA23_19.0.1	19.0.1 A line whose first token is # shall be a valid preprocessing directive	Yes		Required	
MISRA23_19.0.2	19.0.2 Function-like macros shall	Yes		Required	

	not be defined				
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	No		Mandatory	
MISRA23_19.1.1	19.1.1 The defined preprocessor operator shall be used appropriately	Yes		Required	
MISRA23_19.1.3	19.1.3 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be defined prior to evaluation	No		Required	
MISRA23_19.2	19.2 The union keyword should not be used	Yes		Advisory	
MISRA23_19.2.1	19.2.1 Precautions shall be	Yes		Required	

	taken in order to prevent the contents of a header file being included more than once				
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_19.3.4	19.3.4 Parentheses shall be used to ensure macro arguments are expanded	Yes		Required	

	appropriately				
MISRA23_19.3.5	19.3.5 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Advisory	
MISRA23_19.6.1	19.6.1 The #pragma directive and the _Pragma operator should not be used	Yes		Advisory	
MISRA23_20.1	20.1 #include directives should only be preceded by preprocessor directives or comments	Yes		Advisory	
MISRA23_20.2	20.2 The ', " or backslash characters and the /* or // character sequences shall not occur in a header file name	Yes		Required	
MISRA23_20.3	20.3 The #include directive shall be followed by either a <filename>	Yes		Required	

	or "filename" sequence				
MISRA23_20.4	20.4 A macro shall not be defined with the same name as a keyword	Yes		Required	
MISRA23_20.5	20.5 #undef should not be used	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	No		Required	
MISRA23_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	No		Required	
MISRA23_20.9	20.9 All identifiers used in the controlling	No		Required	

	expression of #if or #elif preprocessing directives shall be #define'd before evaluation				
MISRA23_20.10	20.10 The # and ## operators should not be used	Yes		Advisory	
MISRA23_20.11	20.11 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Yes		Required	
MISRA23_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	No		Required	
MISRA23_20.13	20.13 A line whose first	Yes		Required	

	token is # shall be a valid preprocessing directive				
MISRA23_20.14	20.14 All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related	Yes		Required	
MISRA23_21.1	21.1 #define and #undef shall not be used on a reserved identifier or reserved macro name	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	Yes		Required	

	functions from <cstring>, <cstdlib>, <cwchar> 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_21.3	21.3 The memory allocation and deallocation functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.4	21.4 The standard header file <setjmp.h> shall not be used	Yes		Required	
MISRA23_21.5	21.5 The standard header file <signal.h> shall not be used	Yes		Required	
MISRA23_21.6	21.6 The	Yes		Required	

6	Standard Library input/output functions shall not be used				
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_21.6.4	21.6.4 If a project defines either a sized or unsized version of a global operator delete, then both shall be defined	Yes		Required	
MISRA23_21.6.5	21.6.5 A pointer to an incomplete class type shall not be deleted	Yes		Required	
MISRA23_21.7	21.7 The Standard Library functions atof, atoi, atol and atoll of <stdlib.h> shall not be used	Yes		Required	

MISRA23_21.8	21.8 The Standard Library termination functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.9	21.9 The library functions bsearch and qsort of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.10	21.10 The Standard Library time and date functions 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_21.11	21.11 The standard header file <tgmath.h> shall not be used	Yes		Advisory	
MISRA23_21.12	21.12 The standard	Yes		Required	

	header file <fenv.h> shall not be used				
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. 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	Yes		Mandatory	

	asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function				
MISRA23_21.20.3	21.20.3 The facilities provided by the standard header file <csignal> shall not be used	Yes		Required	
MISRA23_21.21	21.21 The Standard Library function system of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.24	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.26	21.26 The Standard Library function	Yes		Required	

	mtx_timedlock() shall only be invoked on mutex objects of appropriate mutex type				
MISRA23_22.1	22.1 All resources obtained dynamically by means of Standard Library functions shall be explicitly released	No		Required	
MISRA23_22.2	22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function	No		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	No		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	No		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 A pointer to a FILE object shall not be dereferenced	No		Mandatory	
MISRA23_22.6	22.6 The value of a pointer to a FILE shall not be used after the associated stream has been closed	No		Mandatory	
MISRA23_22.11	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA23_22.13	22.13 Thread objects, thread	Yes		Required	

	synchronizati on objects and thread- specific storage pointers shall have appropriate storage duration				
MISRA23_22. 17	22.17 No thread shall unlock a mutex or call cnd_wait() or cnd_timedwa it() for a mutex it has not locked before	Yes		Required	
MISRA23_24. 5.1	24.5.1 The character handling functions from <cctype> and <cwctype> shall not be used	Yes		Required	
MISRA23_24. 5.2	24.5.2 The C++ Standard Library functions memcpy, memmove and memcmp from <cstring> shall not be used	Yes		Required	

MISRA23_25.5.1	25.5.1 The setlocale and std::locale::global functions shall not be called	Yes		Required	
MISRA23_25.5.2	25.5.2 The pointers returned by the C++ 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++ Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the	Yes		Mandatory	

	same function				
MISRA23_26.3.1	26.3.1 std::vector should not be specialized with bool	Yes		Advisory	
MISRA23_28.6.1	28.6.1 The argument to std::move shall be a non-const lvalue	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_1.1	Directive 1.1 Any implementation-defined behaviour on which the output of the program depends shall be documented and understood	No		Required	
MISRA23_DIR_2.1	Directive 2.1 All source	Yes		Required	

	files shall compile without any compilation errors				
MISRA23_DIR_3.1	Directive 3.1 All code shall be traceable to documented requirements	No		Required	
MISRA23_DIR_4.1	Directive 4.1 Run-time failures shall be minimized	No		Required	
MISRA23_DIR_4.2	Directive 4.2 All usage of assembly language should be documented	No		Advisory	
MISRA23_DIR_4.3	Directive 4.3 Assembly language shall be encapsulated and isolated.	Yes		Required	
MISRA23_DIR_4.4	Directive 4.4 Sections of code should not be "commented out"	Yes		Advisory	
MISRA23_DIR_4.5	Directive 4.5 Identifiers in the same name space with overlapping visibility should be typographical	Yes		Advisory	

	ly unambiguous				
MISRA23_DIR_4.6	Directive 4.6 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA23_DIR_4.7	Directive 4.7 If a function returns error information, then that error information shall be tested	No		Required	
MISRA23_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA23_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they	No		Advisory	

	are interchangea ble				
MISRA23_DIR_4.10	Directive 4.10 Precautions shall be taken in order to prevent the contents of a header file being included more than once	Yes		Required	
MISRA23_DIR_4.11	Directive 4.11 The validity of values passed to library functions shall be checked	No		Required	
MISRA23_DIR_4.12	Directive 4.12 Dynamic memory allocation shall not be used	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	No		Advisory	
MISRA23_DIR_5.1	Directive 5.1 There shall	No		Required	

	be no data races between threads				
MISRA23_DIR_5.2	Directive 5.2 There shall be no deadlocks between threads	No		Required	
MISRA23_DIR_5.3	Directive 5.3 There shall be no dynamic thread creation	Yes		Required	
MSC30-C	Do not use the rand() function for generating pseudorandom numbers	Yes			Medium
MSC32-C	Properly seed pseudorandom number generators	Yes			Medium
MSC33-C	Do not pass invalid data to the asctime() function	Yes			High
MSC37-C	Ensure that control never reaches the end of a non-void function	Yes			High
MSC38-C	Do not treat a predefined identifier as an object if it might only be	Yes			Low

	implemented as a macro				
MSC39-C	Do not call va_arg() on a va_list that has an indeterminate value	Yes			Low
MSC40-C	Do not violate constraints	Yes			Low
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 declared [[noreturn]]	Yes			Medium
MSC54-CPP	A signal handler must be a plain old function	Yes			High

OOP50-CPP	Do not invoke virtual functions from constructors or destructors	Yes			Low
OOP51-CPP	Do not slice derived objects	Yes			Low
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 handler requirements	Yes			
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
POS30-C	Use the readlink() function properly	Yes			High

POS34-C	Do not call putenv() with a pointer to an automatic variable as the argument	Yes			High
POS35-C	Avoid race conditions while checking for the existence of a symbolic link	Yes			High
POS36-C	Observe correct revocation order while relinquishing privileges	Yes			High
POS37-C	Ensure that privilege relinquishme nt is successful	Yes			High
POS38-C	Beware of race conditions when using fork and file descriptors	Yes			Medium
POS39-C	Use the correct byte ordering when transferring data between systems	Yes			Medium
POS44-C	Do not use signals to terminate threads	Yes			Low
POS47-C	Do not use	Yes			Medium

	threads that can be canceled asynchronously				
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
POS52-C	Do not perform operations	Yes			Low

	that can block while holding a POSIX lock				
POS53-C	Do not use more than one mutex for concurrent waiting operations on a condition variable	Yes			Medium
POS54-C	Detect and handle POSIX library errors	Yes			High
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	Yes			

	and Return Values - 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
PRE31-C	Avoid side effects in arguments to unsafe macros	Yes			Low
PRE32-C	Do not use preprocessor directives in invocations of function-like macros	Yes			Low
RECOMMENDED_00	Commented Out Code	Yes			
RECOMMEN	Definitions in	Yes			

DED_01	Header Files				
RECOMMENDED_02	Files too long	Yes			
RECOMMENDED_03	Floating Equality Test	Yes			
RECOMMENDED_04	Functions Too Long	Yes			
RECOMMENDED_05	Functions shall not be declared implicitly	Yes			
RECOMMENDED_06	Goto Statements	Yes			
RECOMMENDED_07	Macros shall not be #define'd or #undef'd within a block	Yes			
RECOMMENDED_08	Magic Numbers	Yes			
RECOMMENDED_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	Variables	Yes			

DED_16	should be commented				
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			
SIG30-C	Call only asynchronous-safe functions within signal handlers	Yes			High
SIG31-C	Do not access shared objects in signal handlers	Yes			High
SIG34-C	Do not call signal() from within interruptible signal handlers	Yes			Low
SIG35-C	Do not return from a computational exception signal handler	No			Low
STI_UNUSED	Unused Entities	Yes		Recommended	High
STR30-C	Do not attempt to modify string	Yes			Low

	literals				
STR31-C	Guarantee that storage for strings has sufficient space for character data and the null terminator	Yes			High
STR32-C	Null-terminated strings passed to library functions	Yes			High
STR34-C	Cast characters to unsigned char before converting to larger integer sizes	No			Medium
STR37-C	Arguments to character-handling functions must be representable as an unsigned char	Yes			Low
STR38-C	Do not confuse narrow and wide character strings and functions	Yes			High
STR50-CPP	Guarantee that storage for strings	Yes			High

	has sufficient space for character data and the null terminator				
STR51-CPP	Do not attempt to create a std::string from a null pointer	Yes			High
STR52-CPP	Use valid references, pointers, and iterators to reference elements of a basic_string	Yes			High
STR53-CPP	Range check element access	Yes			High
WIN30-C	Properly pair allocation and deallocation functions	Yes			Low