

C

Checks

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

Check ID	Check Name	Supported	Automation	Category	Severity
	functions that override it				
A0-1-6	There should be no unused type declarations	Yes	Automated	Advisory	
A0-4-1	Floating-point implementation shall comply with IEEE 754 standard	No	Non-automated	Required	
A0-4-2	Type long double shall not be used	Yes	Automated	Required	
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 of compiler warnings	Yes	Automated	Advisory	
A2-3-1	Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes	Automated	Required	
A2-5-1	2-3-1 Trigraphs shall not be used	Yes	Automated	Required	
A2-5-2	Digraphs shall not be used	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A2-7-1	The character \ shall not occur as a last character of a C++ comment	Yes	Automated	Required	
A2-7-2	Sections of code shall not be "commented out"	Yes	Non-automated	Required	
A2-7-3	All declarations of "user-defined" types, static and non-static data members, functions and methods shall be preceded by documentation	Yes	Automated	Required	
A2-8-1	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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	with static storage duration or static function shall not be reused within a namespace				
A2-10-5	An identifier name of a function with static storage duration or a non-member object with external or internal linkage should not be reused	Yes	Automated	Advisory	
A2-10-6	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in 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	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	in the project, should have a file name extension of ".cpp"				
A3-1-4	When an array with external linkage is declared, its size shall be stated explicitly	Yes	Automated	Required	
A3-1-5	A function definition shall only be placed in a class definition if (1) the function is intended to be inlined (2) it is a member function template (3) it is a member function of a class template	Yes	Partially Automated	Required	
A3-3-1	Objects or functions with external linkage (including members of named namespaces) shall be declared in a header file	Yes	Automated	Required	
A3-3-2	Static and thread-local objects shall be constant-initialized	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 shall be the same under any order of evaluation that the standard permits	Yes	Automated	Required	
A5-0-2	Condition of if	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	statement shall be bool				
A5-0-3	No more than 2 levels of pointer indirection	Yes	Automated	Required	
A5-0-4	Pointer arithmetic shall not be used with pointers to non-final classes	Yes	Automated	Required	
A5-1-1	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead	Yes	Automated	Required	
A5-1-2	Variables shall not be implicitly captured in a lambda expression	Yes	Automated	Required	
A5-1-3	Parameter list (possibly empty) shall be included in every lambda expression	Yes	Automated	Required	
A5-1-4	A lambda expression object shall not outlive any of its reference-captured objects	Yes	Automated	Required	
A5-1-6	Specify Lambda Return	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
A5-1-7	Type A lambda shall not be an operand to decltype or typeid	Yes	Automated	Required	
A5-1-8	Lambda expressions should not be defined inside another lambda expression	Yes	Automated	Advisory	
A5-1-9	Identical unnamed lambda expressions shall be replaced with a named function or a named lambda expression	Yes	Automated	Advisory	
A5-2-1	dynamic_cast should not be used	Yes	Automated	Advisory	
A5-2-2	Traditional C-style casts shall not be used	Yes	Automated	Required	
A5-2-3	A cast shall not remove any const or volatile qualification from the type of a pointer or reference	Yes	Automated	Required	
A5-2-4	reinterpret_cast shall not be used	Yes	Automated	Required	
A5-2-5A	An array or	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	container shall not be accessed beyond its range (Part A)				
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	
A5-3-2	Before dereferencing a pointer, compare it with NULL	Yes	Partially Automated	Required	
A5-3-3	Deleting Pointers to Incomplete Class Types	Yes	Automated	Required	
A5-6-1A	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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A5-16-1	for equality with null-pointer-constant 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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	contain a single loop-counter which shall 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 placed on the right hand side of the type that is a typedef or a	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A7-1-4	using name 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	Yes	Automated	Required	

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

Check ID	Check Name	Supported	Automation	Category	Severity
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	
A7-5-2	Functions shall not call themselves, either directly or indirectly.	Yes	Automated	Required	
A7-6-1	Functions declared with the [[noreturn]] attribute shall not return	Yes	Automated	Required	
A8-4-2	Always return a value in non-void functions	Yes	Automated	Required	
A8-4-4	Multiple output values from a function should be returned as a struct or tuple.	Yes	Automated	Advisory	
A8-4-8	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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	can't be NULL				
A8-4-12	Invalid Use of std::unique_ptr	Yes	Automated	Required	
A8-4-13	Invalid Use of std::shared_ptr	Yes	Automated	Required	
A8-5-0	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 not Used in a "is-a" Relationship	Yes	Non-automated	Required	
A10-0-2	Membership or No	No	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	non-public inheritance shall be used to implement "has-a" relationship				
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 based on interface classes	Yes	Non-automated	Advisory	
A11-0-1	A non-POD	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	type should be defined as class				
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	
A12-1-6	Derived classes that do not need further explicit initialization and require all the constructors from the base class shall use inheriting constructors	No	Automated	Required	
A12-4-1	Destructor of a base class shall be public virtual, public override or protected non-virtual	Yes	Automated	Required	
A12-8-1	Move and copy constructors shall move and respectively copy base	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	classes and data members of a class, without any side effects				
A13-1-2	User defined suffixes of the user defined literal operators shall start with underscore followed by one or more letters	Yes	Automated	Required	
A13-1-3	User defined literals operators 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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A13-5-1	its argument shall not be overloaded If "operator[]" is to be overloaded with a non-const version, const version shall also be implemented	Yes	Automated	Required	
A13-5-2	All user-defined conversion operators shall be defined explicit	Yes	Automated	Required	
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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 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,	Yes	Automated	Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
A14-7-1	enum type or union type declarations 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 either the strong guarantee or the nothrow	No	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A15-0-3	guarantee 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 shall guarantee a deterministic worst-case time execution time	No	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A15-0-8	A worst-case execution time (WCET) analysis shall be performed to determine maximum execution time constraints of the software, covering in particular the exceptions processing	No	Non-automated	Required	
A15-1-1	Only instances of types derived from <code>std::exception</code> should be thrown	Yes	Automated	Advisory	
A15-1-3	All thrown exceptions should be unique	Yes	Automated	Advisory	
A15-1-4	If a function exits with an exception, then before a throw, the function shall place all objects/resources that the function constructed in valid states or it shall delete them	Yes	Partially Automated	Required	
A15-2-1	Constructors that are not <code>noexcept</code> shall not be invoked before	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A15-2-2	program startup 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::exception) handlers shall be used only in (a) main, (b) task main functions, (c) in functions that are supposed to isolate independent components and (d) when calling third-party code that uses exceptions not according to AUTOSAR C++14 guidelines	Yes	Non-automated	Required	
A15-3-5	A class type exception shall be caught by	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A15-4-1	reference or const reference Dynamic exception-specification shall not be used	Yes	Automated	Required	
A15-4-3	The noexcept specification of a function shall either be identical across all translation units, or identical or more restrictive between a virtual member function and an override	Yes	Automated	Required	
A15-4-5	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 overrides.	Yes	Automated	Required	
A15-5-1	All user-provided class destructors, deallocation functions,	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
A15-5-2	Program shall not be abruptly terminated	Yes	Automated	Required	
A15-5-3	The <code>std::terminate()</code> function shall not be called implicitly	Yes	Automated	Required	
A16-0-1	Incorrect Use of Pre-processor	Yes	Automated	Required	
A16-2-1	Header File Name	Yes	Automated	Required	
A16-2-2	There shall be no unused include directives (slow)	Yes	Automated	Required	
A16-6-1	<code>#error</code> directive shall not be used	Yes	Automated	Required	
A16-7-1	The <code>#pragma</code> directive shall not be used	Yes	Automated	Required	
A17-0-1	Reserved Builtin Macros	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 <locale> (locale.h)	Yes	Automated	Required	
A18-1-1	C-style Array	Yes	Automated	Required	
A18-1-2	The std::vector<bool> specialization shall not be used	Yes	Automated	Required	
A18-1-3	The std::auto_ptr type shall not be used	Yes	Automated	Required	
A18-1-4	A pointer pointing to an element of an array of objects shall not be passed to a smart pointer of single object type	Yes	Automated	Required	
A18-1-6	All std::hash	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	specializations for user-defined types shall have a noexcept function call operator				
A18-5-1	Functions malloc, calloc, realloc and free shall not be used	Yes	Automated	Required	
A18-5-2	Non-placement new or delete expressions shall not be used	Yes	Partially Automated	Required	
A18-5-3	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes	Automated	Required	
A18-5-4	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined	Yes	Automated	Required	
A18-5-7	Dynamic Memory Usage	Yes	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	on Realtime Phase				
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 shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding reference	Yes	Automated	Required	
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	

Check ID	Check Name	Supported	Automation	Category	Severity
A20-8-1	An already-owned pointer value shall not be stored in an unrelated smart pointer	Yes	Automated	Required	
A20-8-2	A <code>std::unique_ptr</code> shall be used to represent exclusive ownership	Yes	Automated	Required	
A20-8-3	A <code>std::shared_ptr</code> shall be used to represent shared ownership	Yes	Automated	Required	
A20-8-4	A <code>std::unique_ptr</code> shall be used over <code>std::shared_ptr</code> if ownership sharing is not required	Yes	Automated	Required	
A20-8-5	<code>std::make_unique</code> shall be used to construct objects owned by <code>std::unique_ptr</code>	Yes	Automated	Required	
A20-8-6	<code>std::make_shared</code> shall be used to construct objects owned by <code>std::shared_ptr</code>	Yes	Automated	Required	
A20-8-7	Cyclic Structure of	Yes	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A21-8-1	std::shared_ptr Arguments to character-handling functions shall be representable as an unsigned char	Yes	Automated	Required	
A23-0-1	An iterator shall not be implicitly converted to const_iterator	Yes	Automated	Required	
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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
A27-0-4	flush or positioning call C-style strings shall not be used	Yes	Automated	Required	
AC_00	No Control Code Characters	Yes			
AC_01	No Direct or Indirect Recursion	Yes			
AC_HIS_02	Number of Paths (PATH)	Yes			
AC_HIS_04	Cyclomatic Complexity (v(G))	Yes			
AC_HIS_05	Calling Functions (CALLING)	Yes			
AC_HIS_06	Called Functions (CALLS)	Yes			
AC_HIS_07	Function Parameters (PARAM)	Yes			
AC_HIS_08	Number of Statements (STMT)	Yes			
AC_HIS_09	Number of call levels (LEVEL)	Yes			
AC_HIS_10	Number of return points (RETURN)	Yes			
AC_HIS_11	Language scope (VOCF)	Yes			
AC_HIS_12	Recursion (AP_CG_CYCLE)	Yes			
AC_HIS_13	Statements Changed (SCHG)	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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 thread-specific storage	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
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 thread safety and liveness when using condition variables	Yes			Low
CON39-C	Do not join or detach a	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	thread that was previously joined or detached				
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
CON53-CPP	Avoid deadlock by locking in a predefined order	No			Low
CPP_A000	Assembler instructions shall only be introduced using the asm declaration.	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			

Check ID	Check Name	Supported	Automation	Category	Severity
	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_A008	Assembly Language Code Usage not Documented	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	qualified-id or this->				
CPP_A017	A template should check if a specific template argument is suitable for this template	Yes			
CPP_A0000	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 type	Yes			
CPP_B002	Enum Bit-fields	Yes			
CPP_B003	The underlying bit representation of floating-point values shall not be used	Yes			
CPP_B004	Named signed bit-field length	Yes			
CPP_B005	Signed single-bit named bit-fields	Yes			
CPP_B006	Bit-field Length	Yes			
CPP_C000	Commented Out Code	Yes			
CPP_C001	Line-Splicing	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	in // Comments				
CPP_C002	No Nested Comments	Yes			
CPP_C003	C99 / Comments	Yes			
CPP_C011	An object's dynamic type shall not be used from the body of its constructor or destructor	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 compound statement, or another if	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_C017	statement 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 shall only be used as an operand to <=, <, > or >=	Yes			
CPP_C021	The loop-counter shall be modified by one of: --, ++,	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	-= n, or += n; where n remains constant for the duration of the loop				
CPP_C022	The loop-counter shall not be modified within condition or statement	Yes			
CPP_C023	The goto statement shall jump to a label declared later in the same function body	Yes			
CPP_C024	No Continue Statements	Yes			
CPP_C025	Goto Statements	Yes			
CPP_C026	There should be no more than one break or goto statement used to terminate any iteration statement	Yes			
CPP_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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	compound statement is the body of a switch statement				
CPP_C031	Switch Has Default	Yes			
CPP_C032	Every switch statement shall have at least two switch-clauses	Yes			
CPP_C033	An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes			
CPP_C034	Unreachable Code	Yes			
CPP_C035	No Backslash at End of Comment	Yes			
CPP_C036	If statements shall not have assignments in the conditions	Yes			
CPP_C037	Documentation Before preprocessing,	Yes			
CPP_C038	a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	following the null statement is a white-space character				
CPP_C039	A switch statement shall have at least two case-clauses, distinct from the default label	Yes			
CPP_C040	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes			
CPP_C041	Do statements should not be used	Yes			
CPP_C042	For-init-statement and expression should not perform actions other than loop-counter initialization and modification	Yes			
CPP_C043	Checked exceptions that could be thrown from a function shall be specified together with	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	the function declaration and they shall be identical in all function declarations and for all its overriders.				
CPP_C044	Continue Statement Used in a not Well-formed For Loop	Yes			
CPP_C046	All switch statements shall be well-formed	Yes			
CPP_C047	All if and else if constructs shall be terminated with an else clause	Yes			
CPP_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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	program startup				
CPP_C052	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			
CPP_C053	Explicit Calls to Constructors of Temporary Objects	Yes			
CPP_C056	Move and copy constructors shall move and respectively copy base classes and data members of a class, without any side effects	Yes			
CPP_CF000	The condition of a switch statement shall not have 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	statements should have a default case				
CPP_CF003	Switch label unstructured	Yes			
CPP_CF004	The std::terminate() function shall not be called implicitly	Yes			
CPP_CF005	Program shall not be abruptly terminated	Yes			
CPP_CF006	Simple Control Flow	Yes			
CPP_CF007	Loops with Fixed Limits	Yes			
CPP_CF008	A for loop shall be well-formed	Yes			
CPP_CF009	The validity of values received from external sources shall be checked	Yes			
CPP_CF011	Goto Into or Between Blocks	Yes			
CPP_CM000	Comments shall precede code being commented and shall align with code they represent	Yes			
CPP_CM001	Each function shall end with a comment	Yes			
CPP_CM002	Timing delays shall be preceded by	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	comments explaining the delay				
CPP_CM004	All usage of assembly language should be documented	Yes			
CPP_CT_BUGP RONE_ASSERT _SIDE_EFFECT	Assert Side Effect	Yes			High
CPP_CT_BUGP RONE_BRANC H_CLONE	Branch Clone	Yes			High
CPP_CT_BUGP RONE_INFINIT E_LOOP	Infinte Loop	Yes			High
CPP_CT_BUGP RONE_MACRO _REPEATED_SI DE_EFFECTS	Macro Side Effects	Yes			High
CPP_CT_BUGP RONE_NOT_N ULL_TERMINA TED_RESULT	Missing Null Terminator	Yes			High
CPP_CT_BUGP RONE_REDUN DANT_BRANC H_CONDITION	Redundant Condition	Yes			High
CPP_CT_CPPC OREGUIDELIN ES_AVOID_NO N_CONST_GL OBAL_VARIAB LES	Non-const Globals	Yes			
CPP_CT_READ ABILITY_DELE TE_NULL_POI NTER	Delete Null Pointer	Yes			High
CPP_CT_READ ABILITY_NON_ CONST_PARA	Non-const Parameters	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
METER					
CPP_CT_READ ABILITY_REDUN DANT_CASTING	Redundant Cast	Yes			High
CPP_D000	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes			
CPP_D002	Single Declarations	Yes			
CPP_D003	Array Size Missing	Yes			
CPP_D004	Unsigned Integer Literal Without Suffix	Yes			
CPP_D005	A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes			
CPP_D006	Class Derived From Virtual Bases	Yes			
CPP_D007	A compatible declaration shall be visible when an object or function with external linkage is defined	Yes			
CPP_D008	A copy assignment operator shall be declared when there is a	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D010	template assignment operator with a parameter that is a generic parameter = construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes			
CPP_D011	Use the static keyword for internal linkage	Yes			
CPP_D012	An external object or function shall be declared in one and only one file	Yes			
CPP_D013	An identifier with external linkage shall have exactly one definition	Yes			
CPP_D015	Externals shall have the same type in the declaration and definition	Yes			
CPP_D017	Non-static Inline Functions	Yes			
CPP_D018	Literal suffixes shall be upper case	Yes			
CPP_D019	The comma operator, && operator and	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	the operator shall not be overloaded				
CPP_D020	Lowercase L Suffix	Yes			
CPP_D021	Narrow and wide string literals shall not be concatenated	Yes			
CPP_D022	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit	Yes			
CPP_D023	Single-Function Global Objects	Yes			
CPP_D024	Restrict Qualifier	Yes			
CPP_D026	The register keyword shall not be used	Yes			
CPP_D027	The unary & operator shall not be overloaded	Yes			
CPP_D028	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes			
CPP_D033	A function shall not be declared	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D035	implicitly 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	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	specifier in a declaration.				
CPP_D052	Use the same identifier in definition and declaration of functions.	Yes			
CPP_D053	Multiple Base Classes	Yes			
CPP_D054	Virtual function declaration shall contain exactly one of the three specifiers: (1) virtual, (2) override, (3) final	Yes			
CPP_D056	User-defined assignment operator shall not be virtual	Yes			
CPP_D057	Hierarchies should be based on interface classes	Yes			
CPP_D058	A non-POD type should be defined as class	Yes			
CPP_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			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D071	template 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 declared	Yes			
CPP_D075	Incorrect Order of Initialization	Yes			
CPP_D077	There shall be no tentative definitions in a	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_D078	header file There should be no external declarations in a source file	Yes			
CPP_DD000	The defines, typedefs, structures, externals, globals, statics, external prototypes, and local prototypes shall be grouped by category.	Yes			
CPP_DD001	Use of global functions and variables shall be limited	Yes			
CPP_DD002	Globals should not be used in macros	Yes			
CPP_DD003	There shall be a function prototype for all functions	Yes			
CPP_DD004	Prototypes for static functions shall include the static storage class	Yes			
CPP_DD005	Any defined constants that are used as argument or return variables shall be placed in an	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_DD013	include file 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_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_DD024	Postpone	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	variable definitions as long as possible				
CPP_DD027	A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes			
CPP_DD028	A pointer should point to a const-qualified type whenever possible	Yes			
CPP_DD029	Variable-length arrays shall not be used	Yes			
CPP_DD030	Storage Class Specifiers Not At Beginning	Yes			
CPP_DD031	Static Keyword in Array Brackets	Yes			
CPP_DD033	A variable declared in an inner scope shall not hide a variable declared in an outer scope	Yes			
CPP_DD034	There shall be no occurrence of undefined or critical unspecified behaviour	Yes			
CPP_E006	Order of Catch	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E008	Blocks with Derived Classes Exceptions shall be raised only after start-up and before termination of the program	Yes			
CPP_E011	No "errno" allowed	Yes			
CPP_E015	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator	Yes			
CPP_E016	Character Operators	Yes			
CPP_E018	Expressions with type enum or enum class shall not be used as operands to built-in and overloaded	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=				
CPP_E019	Avoid Trigraphs	Yes			
CPP_E020	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes			
CPP_E021	Octal and Hexadecimal Sequences	Yes			
CPP_E022	Nonstandard Escape Sequences	Yes			
CPP_E023	Expression uses operand of side-effect more than once	Yes			
CPP_E024	Signed operands to modulus or division operator	Yes			
CPP_E025	Floating	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E027	Equality Test Only those escape sequences that are defined in ISO/IEC 14882:2014 shall be used	Yes			
CPP_E028	Hexadecimal constants should be upper case	Yes			
CPP_E029	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes			
CPP_E030	Concatenating String Literals of Different Encodings	Yes	Automated	Required	
CPP_E031	Traditional C-style casts shall not be used	Yes			
CPP_E032	Infeasible Paths	Yes			
CPP_E034	Limited dependence should be placed on C++ operator precedence rules in expressions	Yes			
CPP_E035	Parameter list (possibly empty) shall be included in	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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 increment (++) and decrement (--) operators shall not be mixed	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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 operator shall not be used as a sub-expression	Yes			
CPP_E048	Each expression statement and identifier declaration shall be placed on a separate line	Yes			
CPP_E049	The comma	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	operator shall not be used.				
CPP_E050A	Evaluation of the operand to the typeid operator shall not contain side effects	Yes			
CPP_E050B	The right hand operand of the integer division or remainder operators shall not be equal to zero	Yes			
CPP_E051	Unary Minus Operator Applied to an Expression with an Unsigned Type	Yes			
CPP_E052	The right-hand operand of a logical && or operator should not contain persistent side effects	Yes			
CPP_E053	Side Effects in Generic Selections	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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	of types derived from <code>std::exception</code> should be thrown				
CPP_E059	All thrown exceptions should be unique	Yes			
CPP_E060	If a function exits with an exception, then before a throw, the function shall place all objects/resources that the function constructed in valid states or it shall delete them	Yes			
CPP_E061	Dynamic exception-specification shall not be used	Yes			
CPP_E062	A class type exception shall be caught by reference or const reference	Yes			
CPP_E063	Catch-all (ellipsis and <code>std::exception</code>) handlers shall be used only in (a) main, (b) task main functions, (c)	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
CPP_E064	Unhandled Exceptions on Main Function	Yes			
CPP_E065	Condition of if statement shall be bool	Yes			
CPP_E066	Const Should be placed on the left-hand side of the comparison	Yes			
CPP_E067	Floats shall not be tested for direct equality	Yes			
CPP_E069	Assignment in SubExpressions	Yes			
CPP_E070	Boolean operators	Yes			
CPP_E071	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E072	Int to Float Conversion	Yes			
CPP_E073	An implicit integral conversion shall not change the signedness of the underlying type	Yes			
CPP_E074	Operands shall not be of an inappropriate essential type	Yes			
CPP_E075	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes			
CPP_E076	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category	Yes			
CPP_E077	The value of a composite expression shall not be assigned to an object with wider essential	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E078	type The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes			
CPP_E079	Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes			
CPP_E080	A conversion shall not be performed between a pointer to object type and a pointer to a different object type	Yes			
CPP_E081	A conversion should not be 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type				
CPP_E084	The macro NULL shall be the only permitted form of integer null pointer constant	Yes			
CPP_E085	The result of an assignment operator should not be used	Yes			
CPP_E086	A loop counter shall not have essentially floating type	Yes			
CPP_E087	Minimize casting	Yes			
CPP_E088	Parentheses should be used to make the meaning of an expression appropriately explicit	Yes			
CPP_E089	An unsigned arithmetic operation with constant operands should not wrap	Yes			
CPP_E090	Controlling	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_E091	expressions shall not be invariant The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders	Yes			
CPP_E092	The operands of a logical && or shall be primary-expressions	Yes			
CPP_E093	The controlling expression of a for statement shall not contain any objects of floating type	Yes			
CPP_EH000	Program shall not be abruptly terminated	Yes			
CPP_EH001	The std::terminate() function shall not be called implicitly	Yes			
CPP_EH006	If a function is declared to be noexcept, noexcept(true) or noexcept(<truecondition>), then it shall not exit with an exception	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_EH007	Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point	Yes			
CPP_EH008	Exceptions thrown across execution boundaries	Yes			
CPP_EH009	New Method Throwing an Exception	Yes			
CPP_EH010	Use Assertion Statements	Yes			
CPP_EH011	Catch exceptions by lvalue reference	Yes			
CPP_F000	All prototype parameters must have an identifier.	Yes			
CPP_F003	Unused Functions	Yes			
CPP_F004	Missing Parameters	Yes			
CPP_F005	Declare functions at file scope	Yes			
CPP_F006	A Function identifier shall either be used to call the function or it shall be preceded by &	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_F009	Use Named Parameters and Prototype Form	Yes			
CPP_F010	Functions shall not be declared implicitly	Yes			
CPP_F012	The identifier main shall not be used for a function other than the global function main	Yes			
CPP_F015	Missing parameter name in function declarations	Yes			
CPP_F016	variable numbers of arguments shall not be used.	Yes			
CPP_F019	A pointer or reference parameter in a function shall be declared as pointer to const or reference to const if the corresponding object is not modified	Yes			
CPP_F020	Inconsistent Parameter Names	Yes			
CPP_F021	The features of <stdarg.h> shall not be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	used				
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 namespace, or static function with internal linkage, or private member function shall be used	Yes			
CPP_F027	There shall be no unused named parameters in non-virtual functions	Yes			
CPP_F028	There shall be no unused named parameters in the set of parameters for a virtual	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	function and all the functions that override it				
CPP_F029	operator "new" and operator "delete" shall be defined together	Yes			
CPP_F030	If a project has a sized or unsized version of operator "delete" globally defined, then both sized and unsized versions shall be defined	Yes			
CPP_F031	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes			
CPP_F033	Always return a value in non-void functions	Yes			
CPP_F035	Non-virtual public or protected member functions shall not be redefined in derived	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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 opposite operators are defined, one shall be defined in terms of the other	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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-member generic operator shall only be declared in a namespace that does not contain class	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	(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 override	Yes			
CPP_F051	A function should be inlined only if it has one or two lines of code	Yes			
CPP_F052	The function gets() should not be used	Yes			
CPP_F053	Every function shall have an explicitly declared return type.	Yes			
CPP_F054	Boolean functions shall explicitly return true or false	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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_F061	The type of the return and all method arguments (even type void) shall be specified when defining a method	Yes			
CPP_F066	Tail-Call Optimization	Yes			
CPP_F067	Functions declared with the [[noreturn]] attribute shall not return	Yes			
CPP_F068	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	elements				
CPP_F069	A signal handler must be a plain old function	Yes			
CPP_F072	A function should be used in preference to a function-like macro where they are interchangeable	Yes			
CPP_F073	If a function returns error information, then that error information shall be tested	Yes			
CPP_F074	Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes			
CPP_H001	The backslash character should not occur in a header file name	Yes			
CPP_H002	The ', ", /* or // characters shall not occur in a header file name	Yes			
CPP_H003	Definitions in	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_H005	Header Files 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_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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	be identical to names of library header files				
CPP_H014	All public header files shall be capable of being included by a C++ file as well as a C file	Yes			
CPP_H016	If prototypes, typedefs, macros, structure definitions, or enums are needed in multiple modules, they shall be placed in header files	Yes			
CPP_H018	When including C Standard Library header files, C++ Standard Library header files without a '.h' appended should be used	Yes			
CPP_H020	The standard header file <tgmath.h> shall not be used	Yes			
CPP_H021	The standard header file <setjmp.h> shall not be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_I000	used A class, union or enum name (including qualification, if any) shall be a unique identifier	Yes			
CPP_I001	Different identifiers shall be typographically unambiguous	Yes			
CPP_I002	External identifiers shall be distinct	Yes			
CPP_I003	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes			
CPP_I005	Identifier Reuse	Yes			
CPP_I006	Same Identifier and Macro Name	Yes			
CPP_I007	Identifiers declared in the same scope and name space shall be distinct	Yes			
CPP_I008	Identifiers that define objects or functions with internal linkage should be unique	Yes			
CPP_I009	Indistinct Macro	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Identifiers				
CPP_I011	Long Identifier Name	Yes			
	Significance				
CPP_I012	Static name reuse	Yes			
CPP_I013	A tag name shall be a unique identifier	Yes			
CPP_I015	Identifier Reuse in Multiple C Name Spaces	Yes			
CPP_I016	Reserved Identifiers or Macros	Yes			
CPP_I017	Shadowed Identifiers	Yes			
CPP_I018	A class or enumeration name shall not be hidden by a variable, function or enumerator declaration in the same scope	Yes			
CPP_I019	The identifier name of a non-member object with static storage duration or static function shall not be reused within a namespace	Yes			
CPP_I020	An identifier name of a function with	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	static storage duration or a non-member object with external or internal linkage should not be reused				
CPP_I021	Universal character names shall be used only inside character or string literals	Yes			
CPP_I022	Similar Entity Names within Multiple Inheritance	Yes			
CPP_I023	Uppercase 'O' shall not be used as an identifier	Yes			
CPP_I024	Lowercase 'l' shall not be used as an identifier	Yes			
CPP_L001	The C library shall not be used	Yes			
CPP_L002	The signal handling facilities of <csignal> shall not be used	Yes			
CPP_L003	The stream input/output library <cstdio> shall not be used	Yes			
CPP_L004	<cstdlib> Library	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Functions				
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	Including <stdio.h>	Yes			
CPP_L014	Library stdlib.h - avoid: abort, exit, getenv and system	Yes			
CPP_L016	The library <time.h> shall not be used	Yes			
CPP_L017	Inputs from independent components shall be validated	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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 const_iterator	Yes			
CPP_L022	An argument to std::forward shall not be subsequently used	Yes			
CPP_L023	The std::move shall not be used on objects declared const or const&	Yes			
CPP_L024	Forwarding values to other functions shall be done via: (1) std::move if the value is an rvalue reference, (2) std::forward if the value is forwarding	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	reference				
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 flush or positioning call	Yes			
CPP_L027	All std::hash specializations for user-defined types shall have a noexcept function call operator	Yes			
CPP_L028	The std::auto_ptr type shall not be used	Yes			
CPP_L029	Library <locale> (locale.h)	Yes			
CPP_L030	Avoid deadlock with POSIX threads by locking in predefined order	Yes			
CPP_L032	Bounds-checking Interfaces	Yes			
CPP_L033	Reserved Builtin Macros	Yes			
CPP_L034	Use of the iostream library is	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	preferred over stdio.h				
CPP_L035	Atomic Operations with Inconsistent Order	Yes			
CPP_L036	Character Function Misuse	Yes			
CPP_L037	Incompatible Pointers	Yes			
CPP_L038	There shall be no attempt to write to a stream which has been opened as read-only	Yes			
CPP_L039	Use of Closed FILE Pointers (Partial)	Yes			
CPP_L040	Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF	Yes			
CPP_L041	Out of Bounds with string.h	Yes			
CPP_L042	EOF Macro Misuse	Yes			
CPP_L043	Thread Data Misuse	Yes			
CPP_L044	C Standard Library I/O Functions	Yes			
CPP_L045	Standard Header	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_L048	signal.h The validity of values passed to library functions shall be checked	Yes			
CPP_M000	Dynamic Memory Allocation	Yes			
CPP_M001	The form of the delete expression shall match the form of the new expression used to allocate the memory	Yes			
CPP_M002	Non-placement new or delete expressions shall not be used	Yes			
CPP_M005	Uninitialized Memory Read	Yes			
CPP_M006	Functions malloc, calloc, realloc and free shall not be used	Yes			
CPP_M007	When reading strings a maximum field width should be specified	Yes			
CPP_M008	Dynamically allocated memory shall be set to some value prior to	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	its use as an rvalue or in a test				
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_M018	Dynamic Memory Usage on Realtime Phase	Yes			
CPP_M019	No Dynamic Memory Allocation	Yes			
CPP_M020	Properly pair allocation and deallocation functions	Yes			
CPP_M021	Declare objects shared between POSIX threads with appropriate storage	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_M022	durations All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes			
CPP_M023	A block of memory shall only be freed if it was allocated by means of a Standard Library function	Yes			
CPP_N001	Naming Convention: Enumerator	Yes			
CPP_N002	Naming Convention: Enums	Yes			
CPP_N003	Naming Convention: Files	Yes			
CPP_N005	Naming Convention: Macros	Yes			
CPP_N007	Naming Convention: Parameters	Yes			
CPP_N008	Naming Convention: Structs	Yes			
CPP_N009	Naming Convention: Typedefs	Yes			
CPP_N010	Naming Convention:	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_N012	Unions Only those characters specified in the C++ Language Standard basic source character set shall be used in the source code	Yes			
CPP_N013	Naming Convention: Header File Names	Yes			
CPP_N014	Naming Convention: Implementation File Names	Yes			
CPP_N015	Implementation files, that are defined 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	used as follows: (1) for decimal, every 3 digits, (2) for hexadecimal, every 2 digits, (3) for binary, every 4 digits				
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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	macro shall not contain tokens that look like preprocessing directives				
CPP_P006	std::make_unique shall be used to construct objects owned by std::unique_ptr	Yes			
CPP_P007	A std::unique_ptr shall be used over std::shared_ptr if ownership sharing is not required	Yes			
CPP_P008	Do Not Use #define	Yes			
CPP_P011	Ifndef Wrappers or Pragma Once	Yes			
CPP_P012	File Include Matching Header	Yes			
CPP_P013	Function-like macros shall not be defined	Yes			
CPP_P014_A	Restrict Pointer Usage - Multiple Dereferences	Yes			
CPP_P014_B	Restrict Pointer Usage - Other	Yes			
CPP_P015	Inactive Code	Yes			
CPP_P016	Size of Array Parameter	Yes			
CPP_P017	#include	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	directives in a file shall only be preceded by other preprocessor directives or comments				
CPP_P018	Keyword Macros	Yes			
CPP_P019	Macros in Blocks	Yes			
CPP_P020	Invalid Macro Usage	Yes			
CPP_P021	Before dereferencing a pointer, compare it with NULL	Yes			
CPP_P022	The pre-processor shall only be used for file inclusion and include guards	Yes			
CPP_P023	Reserved or Standard Library Identifiers as Macros	Yes			
CPP_P024	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Yes			
CPP_P026	Preprocessor	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	#undef				
CPP_P027	Dereference of FILE Pointer	Yes			
CPP_P029	Unused Macros	Yes			
CPP_P030	Invalid Use of std::shared_ptr	Yes			
CPP_P031	Invalid Use of std::unique_ptr	Yes			
CPP_P032	Cyclic Structure of std::shared_ptr	Yes			
CPP_P033	For pointer declarations, the asterisk shall be placed with the variable	Yes			
CPP_P034	Invalid Header Name	Yes			
CPP_P035	std::make_shared shall be used to construct objects owned by std::shared_ptr	Yes			
CPP_P036	A std::shared_ptr shall be used to represent shared ownership	Yes			
CPP_P037	A std::unique_ptr shall be used to represent exclusive ownership	Yes			
CPP_P038	An already-owned pointer value shall not	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
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 tested for equality with null-pointer-constant	Yes			
CPP_P049	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes			
CPP_P050	Literal zero (0) shall not be used as the null-pointer-constant.	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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_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_P061	Shared Pointer Capture	Yes			
CPP_P062	The +, -, += and -= operators should not be applied to an expression of pointer type	Yes			
CPP_P063	Declarations	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	should contain no more than two levels of pointer nesting				
CPP_P067	Pointers shall not be implicitly compared to NULL	Yes			
CPP_PR000	#define and #undef shall not be used on a reserved identifier or reserved macro name	Yes			
CPP_PR001	Include guards shall be provided	Yes			
CPP_PR002	Constants defined by #define shall be explicitly declared with uppercase suffixes	Yes			
CPP_PR003	Macros shall not be used to change language syntax	Yes			
CPP_PR004	Limit Preprocessor Usage	Yes			
CPP_PR005	#include directives should only be preceded by preprocessor directives or comments	Yes			
CPP_PR006	There shall be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	at most one occurrence of the # or ## operators in a single macro definition				
CPP_PR007	The defined preprocessor operator shall only be used in one of the two standard forms	Yes			
CPP_PR008	Generic Outside of Macro	Yes			
CPP_PR021	The names of standard library macros and objects shall not be reused	Yes			
CPP_PR030	The #pragma directive shall not be used	Yes			
CPP_PR031	#error directive shall not be used	Yes			
CPP_PR032	The # and ## operators should not be used	Yes			
CPP_PR033	The macro offsetof shall not be used	Yes			
CPP_PR034	There shall be no unused include directives (slow)	Yes			
CPP_PR036	Invalid Preprocessor Directives	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_PR037	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes			
CPP_PR038	Missing Parentheses in Macros	Yes			
CPP_PR039	Function-like Macro Containing Preprocessing Directives	Yes			
CPP_PR040	#include Directives Not Grouped Together	Yes			
CPP_PR041	Incorrect Use of Pre-processor	Yes			
CPP_PR042	The argument of an integer constant macro shall have an appropriate form	Yes			
CPP_PR043	An initializer using chained designators shall not contain initializers without designators	Yes			
CPP_PR044	Expressions	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	resulting from the expansion of macro parameters shall be enclosed in parentheses				
CPP_PR045	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation	Yes			
CPP_PR046	A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes			
CPP_PR047	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes			
CPP_S000	Unions	Yes			
CPP_S001	Flexible Array Members	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_S002	Incorrect Initializer Lists	Yes			
CPP_S003	A type defined as struct shall: (1) provide only public data members, (2) not provide any special member functions or methods, (3) not be a base of another struct or class, (4) not inherit from another struct or class	Yes			
CPP_S004	Unions Shall not be Used	Yes			
CPP_S005	Invalid Memory Compare	Yes			
CPP_S006	Unions	Yes			
CPP_S007	A union member shall not be read unless it has been previously set	Yes			
CPP_SA_DANGLING_POINTER	Dangling Pointer	Yes			High
CPP_SA_DEAD_STORES	Dead Stores	Yes			
CPP_SA_DIV_ZERO	Division by Zero	Yes			High
CPP_SA_LEAKS	Memory Leak	Yes			High
CPP_SA_NULL_PTR	Null Pointer Dereference	Yes			High
CPP_SA_STACK_ADDRESS_ESCAPE	Stack Address Escape	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
SCAPE					
CPP_SA_UNDE FINED_CALL	Undefined Call	Yes			High
CPP_SA_UNINI TIALIZED	Uninitialized Value	Yes			High
CPP_ST001	Not more than one space should precede a ";" with the exception of the null statement	Yes			
CPP_ST002	Equal signs should be aligned when they occur in a series of 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	should extend beyond column 80				
CPP_ST007	Declarations shall not be made within an individual block but shall be placed at the function level or at the module level.	Yes			
CPP_ST008	Blank lines should be 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	deeply nested else-ifs when testing a 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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	for control structure, the statement(s) shall be indented one level				
CPP_ST021	Placement of braces for constructs shall be consistent within a module	Yes			
CPP_ST022	Boolean expressions involving non-boolean values should always use an explicit test for equality or non-equality.	Yes			
CPP_ST023	At least one blank line shall be placed before comments	Yes			
CPP_ST024	Functions shall have at least one blank line between them	Yes			
CPP_ST025	Each area of declarations shall have at least one blank line before and after it	Yes			
CPP_ST029	Immutable data naming conventions	Yes			
CPP_ST031	Separate lines should be used	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	for each member declaration				
CPP_ST032	Indentation shall be at least three spaces, and consistent across modules	Yes			
CPP_ST033	Short Functions	Yes			
CPP_T000	Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes			
CPP_T001	Arguments to character-handling functions shall be representable as an unsigned char	Yes			
CPP_T002	The <code>std::vector<bool></code> specialization shall not be used	Yes			
CPP_T003	There should be no unused type declarations	Yes			
CPP_T004	Type long double shall not be used	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	reduce the size of the underlying type				
CPP_T011	There shall be no explicit floating-integral conversions of a cvalue expression	Yes			
CPP_T012	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression	Yes			
CPP_T013	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression	Yes			
CPP_T014	If the bitwise operators ~ and << are applied to an operand with an underlying type of unsigned char or unsigned short, the result shall be	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	immediately cast to the underlying type of the operand				
CPP_T015	The plain char type shall only be used for the storage and use of character values	Yes			
CPP_T016	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes			
CPP_T017	The first operand of a conditional-operator shall have type bool	Yes			
CPP_T018	Bitwise operators shall only be applied to operands of unsigned underlying type	Yes			
CPP_T019	C-style Array	Yes			
CPP_T020	Casts from a base class to a derived class should not be performed on polymorphic types	Yes			
CPP_T021	A cast shall not remove any const or	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	volatile qualification from the type of a pointer or reference				
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 underlying base type shall be explicitly	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_T029	defined In an enumeration, either (1) none, (2) the first or (3) all enumerators shall be initialized	Yes			
CPP_T032	Multiple output values from a function should be returned as a struct or tuple.	Yes			
CPP_T036	Output parameters shall not be used.	Yes			
CPP_T037	"in-out" parameters declared as T & shall be modified.	Yes			
CPP_T038	Fixed Width Integers	Yes			
CPP_T039	Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes			
CPP_T040	User defined literals operators shall only perform conversion of passed parameters	Yes			
CPP_T041	A binary arithmetic	Yes			

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

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_T064	Pointer to Variable-length Array	Yes			
CPP_T065	Unconditional Generic	Yes			
CPP_T066	Generic Default Not First or Last	Yes			
CPP_T067	Generic Expression Not Always Evaluated Once	Yes			
CPP_T068	The operands of bitwise operators and shift operators shall be appropriate	Yes			
CPP_T069	Integral promotion and the usual arithmetic conversions shall not change the signedness or the type category of an operand	Yes			
CPP_T070	Assignment between numeric types shall be appropriate	Yes			
CPP_U000	Digraphs shall not be used	Yes			
CPP_U001	Discarded return values.	Yes			
CPP_U002	Inline Functions have more than X	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	LOC				
CPP_U004	Unused Static Globals	Yes			
CPP_U005	Unused Tag Declarations	Yes			
CPP_U006	Unused Type Declarations	Yes			
CPP_U007	Unused Labels	Yes			
CPP_U010	Unused Entities	Yes			
CPP_U011	A project shall not contain dead code (Partial)	Yes			
CPP_U012	Language extensions should not be used (Partial)	Yes			
CPP_V000	Magic Numbers	Yes			
CPP_V001	One Variable per Line	Yes			
CPP_V002	Reference Symbols Spacing, (& *)	Yes			
CPP_V003	Declare each variable in a separate declaration statement	Yes			
CPP_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			

Check ID	Check Name	Supported	Automation	Category	Severity
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_V017	A project shall not contain instances of non-volatile variables being given values that are not subsequently used	Yes			
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	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	implicitly captured in a lambda expression				
CPP_V023	Literal values shall not be used apart from type initialization, otherwise symbolic names shall be used instead	Yes			
CPP_V024	Variables of type char shall be explicitly qualified as signed or unsigned when used to store numbers	Yes			
CPP_V025	Every variable shall be declared with a specific type	Yes			
CPP_V026	Local variables shall be initialized when declared	Yes			
CPP_V027	Globals in header files shall be ifdef'd	Yes			
CPP_V028	Constants should be declared as const values as opposed to #define directives.	Yes			
CPP_V029	The const_cast operator should be used	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	exclusively for altering the constness attribute of a variable.				
CPP_V032	Use of the reinterpret_cast operator should be avoided	Yes			
CPP_V034	STL containers (vector, list, map, etc.) should be used instead of C-style arrays whenever possible.	Yes			
CPP_V035	Objects that do not outlive a function shall have automatic storage duration	Yes			
CPP_V039	Within an object, most instance variables should be accessed directly. Methods should be used to set variables whose values are determined by an algorithm.	Yes			
CPP_V040	Mutexes with One Condition Variable	Yes			
CPP_WARN_A	Absolute Value	Yes			

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
RGUMENT_OUTSIDE_RANGE	Outside Range				
CPP_WARN_ARGUMENT_UNDEFINED_BEHAVIOUR	Argument Undefined Behaviour	Yes			
CPP_WARN_ARRAY_BOUNDS	Array Bounds	Yes			High
CPP_WARN_ARRAY_BOUNDS_POINTER_ARITHMETIC	Array Bounds Pointer Arithmetic	Yes			
CPP_WARN_ARRAY_PARAMETER	Array Parameter	Yes			
CPP_WARN_ASM_OPERAND_WIDTHS	Assembly Operand Widths	Yes			
CPP_WARN_ASSIGN_ENUM	Assign Enum	Yes			
CPP_WARN_ASSUME	Discarded Side Effects to <code>__assume</code> Function	Yes			
CPP_WARN_ATOMIC_ACCESS	Atomic Access	Yes			
CPP_WARN_ATOMIC_ALIGNMENT	Atomic Alignment	Yes			
CPP_WARN_ATOMIC_IMPLICITLY_SEQUENTIALLY_CONSISTENT	Atomic Implicitly Sequentially-Consistent	Yes			
CPP_WARN_ATOMIC_MEMORY_ORDERING	Atomic Memory Ordering	Yes			
CPP_WARN_AUTO_DISABLE_VIRTUAL_POINTER_SANITIZER	Auto Disable Virtual Pointer Sanitizer	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
TIONAL_PARENTHESSES	Parentheses				
CPP_WARN_BITWISE_INSTEAD_OF_LOGICAL	Bitwise Instead of Logical	Yes			
CPP_WARN_BITWISE_OPERATOR_PARENTHESSES	Bitwise Operator Parentheses	Yes			
CPP_WARN_BOOL_CONVERSION	Bool Conversion	Yes			High
CPP_WARN_BOOL_OPERATION	Bool Operation	Yes			
CPP_WARN_BRACED_SCALAR_INIT	Braced Scalar Init	Yes			
CPP_WARN_BRANCH_PROTECTION	Branch Protection	Yes			
CPP_WARN_BUILTIN_ASSUME_ALIGNED_ALIGNMENT	Builtin Assume Aligned Alignment	Yes			
CPP_WARN_BUILTIN_MACRO_REDEFINED	Builtin Macro Redefined	Yes			High
CPP_WARN_BUILTIN_MEMCPY_CHECK_SIZE	Builtin Malloc Check Size	Yes			High
CPP_WARN_BUILTIN_REQUIRES_HEADER	Builtin Requires Header	Yes			
CPP_WARN_C2X_EXTENSIONS	C2X Extensions	Yes			
CPP_WARN_C11_EXTENSIONS	C11 Extensions	Yes			
CPP_WARN_C99	C99	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
99_COMPAT	Compatibility				
C99_WARN_C	C99	Yes			
99_DESIGNATOR	Designator				
C99_WARN_C	C99	Yes			
99_EXTENSIONS	Extensions				
C99_WARN_C	Call to Pure Virtual from Constructor or Destructor	Yes			High
C99_WARN_C	Called once Parameter	Yes			
C99_WARN_C	Cast Align	Yes			
C99_WARN_C	Cast Calling Convention	Yes			
C99_WARN_C	Cast Function Type	Yes			
C99_WARN_C	Cast Qualifiers	Yes			
C99_WARN_C	Cast Qualifiers Unrelated	Yes			High
C99_WARN_C	Char Subscripts	Yes			
C99_WARN_C	Clang-CL Precompiled Headers	Yes			
C99_WARN_C	Class Conversion	Yes			High
C99_WARN_C	Class Variadic Arguments	Yes			
C99_WARN_C	CMSE (Cortex-	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
MSE_UNION_L EAK	M Support for Security Extension) Union Leak				
CPP_WARN_C OMMA	Comma Operator Misuse	Yes			
CPP_WARN_C OMMENT	Comment Misuse	Yes			
CPP_WARN_C OMPARE_DIST INCT_POINTER _TYPES	Compare Distinct Pointer Types	Yes			High
CPP_WARN_C OMPLEX_COM PONENT_INIT	Complex Component Init	Yes			
CPP_WARN_C OMPOUND_TO KEN_SPLIT_BY _MACRO	Compound Token Split by Macro	Yes			High
CPP_WARN_C OMPOUND_TO KEN_SPLIT_BY _SPACE	Compound Token Split by Space	Yes			
CPP_WARN_C ONDITIONAL_ TYPE_MISMAT CH	Conditional Type Mismatch	Yes			High
CPP_WARN_C ONDITIONAL_ UNINITIALIZED	Conditional Uninitialized	Yes			
CPP_WARN_C ONSTANT_CO NVERSION	Constant Conversion	Yes			High
CPP_WARN_C ONSTANT_EVA LUATED	Constant Evaluated	Yes			
CPP_WARN_C ONSTANT_LO GICAL_OPERA ND	Constant Logical Operand	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_C ONSTEXPR_N OT_CONST	Constexpr Not Const	Yes			
CPP_WARN_C ONSUMED	Consumable Attribute	Yes			
CPP_WARN_C ONVERSION	Type Conversion	Yes			
CPP_WARN_C OROUTINE	Coroutine Return Type	Yes			
CPP_WARN_C OROUTINE_MI SSING_UNHA NDLED_EXCEP TION	Coroutine Missing Unhandled Exception	Yes			
CPP_WARN_C OVERED_SWIT CH_DEFAULT	Covered Switch Default	Yes			
CPP_WARN_C PP_COMPAT	C++ Compatibility	Yes			
CPP_WARN_C PP2B_EXTENS IONS	C++2B Extensions	Yes			
CPP_WARN_C PP11_COMPAT	C++11 Compatibility	Yes			
CPP_WARN_C PP11_COMPAT _DEPRECATED _WRITABLE_S TRINGS	C++11 Compatibility Deprecated Writable Strings	Yes			High
CPP_WARN_C PP11_COMPAT _RESERVED_U SER_DEFINED _LITERAL	C++11 Compatibility Reserved User Defined Literal	Yes			
CPP_WARN_C PP11_EXTENS IONS	C++11 Extensions	Yes			
CPP_WARN_C PP11_EXTRA_ SEMI	C++11 Extra Semicolon	Yes			
CPP_WARN_C	C++11 Inline	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
PP11_INLINE_NAMESPACE	Namespace				
CPP_WARN_C PP11_LONG_LONG	C++11 Long	Yes			
CPP_WARN_C PP11_NARROWING	C++11 Narrowing	Yes			
CPP_WARN_C PP14_ATTRIBUTE_EXTENSIONS	C++14 Attribute Extensions	Yes			
CPP_WARN_C PP14_BINARY_LITERAL	C++14 Binary Literal	Yes			
CPP_WARN_C PP14_EXTENSIONS	C++14 Extensions	Yes			
CPP_WARN_C PP17_ATTRIBUTE_EXTENSIONS	C++17 Attribute Extensions	Yes			
CPP_WARN_C PP17_COMPAT_MANGLING	C++17 Compatibility Mangling	Yes			
CPP_WARN_C PP17_EXTENSIONS	C++17 Extensions	Yes			
CPP_WARN_C PP20_ATTRIBUTE_EXTENSIONS	C++20 Attribute Extensions	Yes			
CPP_WARN_C PP20_COMPAT	C++20 Compatibility	Yes			
CPP_WARN_C PP20_DESIGNATOR	C++20 Designator	Yes			
CPP_WARN_C PP20_EXTENSIONS	C++20 Extensions	Yes			
CPP_WARN_C C++98	C++98	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
PP98_COMPAT_T	Compatibility				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT_BIND_TO_TEMPORARY_COPY	Compatibility Bind to Temporary Copy				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT_EXTRA_SEMI	Compatibility Extra Semicolon				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT_LOCAL_TYPE_TEMPLATE_ARGS	Compatibility Local Type Template Args				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT_PEDANTIC	Compatibility Pedantic				
CPP_WARN_C	C++98	Yes			
PP98_COMPAT_UNNAMED_TYPE_TEMPLATE_ARGS	Compatibility Unnamed Type Template Args				
CPP_WARN_C	C++98 C++11	Yes			
PP98_CPP11_COMPAT_BINARY_LITERAL	Compatibility Binary Literal				
CPP_WARN_CTAD_MAYBE_UNSUPPOTED	CTAD (Class Template Argument Deduction) Maybe Unsupported	Yes			
CPP_WARN_CXX_ATTRIBUTE_EXTENSION	C++ Attribute Extension	Yes			
CPP_WARN_DANGLING_POINTERS	Dangling Pointers	Yes			High
CPP_WARN_DANGLING_ELSE	Dangling Else	Yes			

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

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_DEPRECATED_TYPE	Deprecated Type	Yes			
CPP_WARN_DEPRECATED_VOLATILE	Deprecated Volatile	Yes			High
CPP_WARN_DISABLED_MACRO_EXPANSION	Disabled Macro Expansion	Yes			
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_DEFINITION	DLLimport Static Field Def	Yes			
CPP_WARN_DOCUMENTATION_WARNINGS	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			

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

Check ID	Check Name	Supported	Automation	Category	Severity
MBEDDED_DIR DIRECTIVE	Directive				
CPP_WARN_E EMPTY_BODY	Control Loop Shall Not Have Empty Body	Yes			
CPP_WARN_E EMPTY_DECOM POSITION	Decomposition Group Shall Not Be Empty	Yes			
CPP_WARN_E EMPTY_INIT_ST MT	No Empty Initialization Statements	Yes			
CPP_WARN_E EMPTY_TRANSL ATION_UNIT	Empty Translation Unit	Yes			
CPP_WARN_E ENUM_COMPARE NUM_COMPAR E	Enum Compare	Yes			High
CPP_WARN_E ENUM_COMPARE NUM_COMPAR E_CONDITION AL	Enum Compare Conditional	Yes			
CPP_WARN_E ENUM_COMPARE NUM_COMPAR E_SWITCH	Enum Compare Switch	Yes			High
CPP_WARN_E ENUM_CONVER SION	Enum Conversion	Yes			
CPP_WARN_E ENUM_ENUM_C ONVERSION	Enum Enum Conversion	Yes			
CPP_WARN_E ENUM_FLOAT_C ONVERSION	Enum Float Conversion	Yes			
CPP_WARN_E ENUM_TOO_LA RGE	Enum Too Large	Yes			
CPP_WARN_EX EXCEPTIONS	Exceptions	Yes			High
CPP_WARN_EX EXCESS_INITI ALIZERS	Excess Initializers	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_E XIT_TIME_DES TRUCTORS	Exit Time Destructors	Yes			
CPP_WARN_E XPANSION_TO _DEFINED	Expansion to Defined	Yes			
CPP_WARN_E XPORT_UNNA MED	Export Unnamed	Yes			
CPP_WARN_E XPORT_USING _DIRECTIVE	Export Using Directive	Yes			
CPP_WARN_E XTERN_C_CO MPAT	Extern C Compatibility	Yes			High
CPP_WARN_E XTERN_INITIA LIZER	Extern Initializer	Yes			High
CPP_WARN_E XTRA	Extra Warnings	Yes			
CPP_WARN_E XTRA_QUALIFI CATION	Extra Qualification	Yes			High
CPP_WARN_E XTRA_SEMI	Extra Semicolon	Yes			
CPP_WARN_E XTRA_SEMI_S TMT	Extra Semicolon in Empty Expression Statement	Yes			
CPP_WARN_E XTRA_TOKENS	Extra Tokens	Yes			
CPP_WARN_FI NAL_DTOR_N ON_FINAL_CL ASS	Final Destructor Non-Final Class	Yes			
CPP_WARN_FI NAL_MACRO	Final Macros Should Not Be Redefined	Yes			
CPP_WARN_FI XED_ENUM_E	Fixed Enum Extension	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
XTENSION					
CPP_WARN_FI	Fixed Point	Yes			High
XED_POINT_O	Overflow				
VERFLOW					
CPP_WARN_FL	Flag Enum	Yes			High
AG_ENUM					
CPP_WARN_FL	Flexible Array	Yes			
EXIBLE_ARRAY	Extensions				
_EXTENSIONS					
CPP_WARN_FL	Float	Yes			
OAT_CONVERT	Conversion				
SION					
CPP_WARN_FL	Float Equal	Yes			
OAT_EQUAL					
CPP_WARN_FL	Float Overflow	Yes			
OAT_OVERFLOW	Conversion				
W_CONVERTI					
ON					
CPP_WARN_FL	Float Zero	Yes			
OAT_ZERO_CO	Conversion				
NVERSION					
CPP_WARN_F	For Loop	Yes			
OR_LOOP_AN	Analysis				
ALYSIS					
CPP_WARN_F	Format String	Yes			High
ORMAT					
CPP_WARN_F	Format Extra	Yes			High
ORMAT_EXTR	Args				
A_ARGS					
CPP_WARN_F	Format	Yes			High
ORMAT_INSUF	Insufficient				
FICIENT_ARGS	Args				
CPP_WARN_F	Format Invalid	Yes			High
ORMAT_INVALID	Specifier				
D_SPECIFIER					
CPP_WARN_F	Format Non-	Yes			
ORMAT_NON_I	ISO				
SO					
CPP_WARN_F	Format Non-	Yes			
ORMAT_NONLI	Literal				
TERAL					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_F ORMAT_PEDA NTIC	Format Pedantic	Yes			
CPP_WARN_F ORMAT_SECU RITY	Format Security	Yes			High
CPP_WARN_F ORMAT_TYPE_ CONFUSION	Format Type Confusion	Yes			
CPP_WARN_F ORMAT_ZERO _LENGTH	Format Zero Length	Yes			High
CPP_WARN_F ORTIFY_SOUR CE	Fortify Source	Yes			High
CPP_WARN_F OUR_CHAR_C ONSTANTS	Four Char Constants	Yes			
CPP_WARN_F AME_ADDRES S	Frame Address	Yes			
CPP_WARN_F EE_NONHEAP_ OBJECT	Free Non-Heap Object	Yes			High
CPP_WARN_F UNCTION_MU LTIVERSION	Function Multi- Version	Yes			
CPP_WARN_F USE_LD_PATH	Fuse LD Path	Yes			
CPP_WARN_G CC_COMPAT	GCC Compatibility	Yes			
CPP_WARN_G LOBAL_CONS TRUCTORS	Global Constructors	Yes			
CPP_WARN_G LOBAL_ISEL	GlobalSel (Global Instruction Selection) Framework	Yes			
CPP_WARN_G NU_ALIGNOF_ Expression	GNU Alignof Expression	Yes			

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

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
NORED_OPTIMIZATION_ARGUMENT	Optimization Argument				
CPP_WARN_IGNORED_PRAGMA_INTRINSIC	Ignored Pragma Intrinsic	Yes			
CPP_WARN_IGNORED_PRAGMAS	Ignored Pragma	Yes			
CPP_WARN_IGNORED_REFERENCE_QUALIFIERS	Ignored Reference Qualifiers	Yes			High
CPP_WARN_IMPLICIT_CONST_INT_FLOAT_CONVERSION	Implicit Const Int Float Conversion	Yes			High
CPP_WARN_IMPLICIT_CONVERSION_FLOATING_POINT_BOOL	Implicit Conversion Floating Point to Bool	Yes			High
CPP_WARN_IMPLICIT_EXCEPTION_SPEC_MISMATCH	Implicit Exception Spec Mismatch	Yes			High
CPP_WARN_IMPLICIT_FALLTHROUGH	Implicit Fallthrough	Yes			
CPP_WARN_IMPLICIT_FALLTHROUGH_PER_FUNCTION	Implicit Fallthrough Per Function	Yes			
CPP_WARN_IMPLICIT_FIXED_POINT_CONVERSION	Implicit Fixed Point Conversion	Yes			High
CPP_WARN_IMPLICIT_FLOAT_CONVERSION	Implicit Float Conversion	Yes			
CPP_WARN_IMPLICIT_FUNCTION	Implicit Function	Yes			

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_INFINITE_RECURSION	Infinite Recursion	Yes			
CPP_WARN_INITIALIZER_OVERRIDES	Initializer Overrides	Yes			
CPP_WARN_INJECTED_CLASS_NAME_S_NAME	Injected Class Name	Yes			High
CPP_WARN_INLINE_ASM	Inline Assembly	Yes			
CPP_WARN_INLINE_NAMESPACE	Inline Namespace	Yes			High
ACE_REOPENED_D_NONINLINE	Reopened Non-Inline				
CPP_WARN_INLINE_NEW_DELETE	Inline New Delete	Yes			
CPP_WARN_INSTANTIATION_AFTER_SPECIALIZATION	Instantiation After Specialization	Yes			High
CPP_WARN_INT_CONVERSION	Int Conversion	Yes			
CPP_WARN_INT_IN_BOOL_CONTEXT	Int in Bool Context	Yes			
CPP_WARN_INT_TO_POINTER_CAST	Int to Pointer Cast	Yes			High
CPP_WARN_INT_TO_VOID_POINTER_CAST	Int to Void Pointer Cast	Yes			High
CPP_WARN_INTEGER_OVERFLOW	Integer Overflow	Yes			High
CPP_WARN_INTERRUPT_SERVICE_ROUTINE	Interrupt Service Routine	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_IN	Invalid	Yes			
VALID_COMM	Command Line				
AND_LINE_AR	Argument				
GUMENT					
CPP_WARN_IN	Invalid	Yes			
VALID_CONST	Constexpr				
EXPR					
CPP_WARN_IN	Invalid	Yes			
VALID_IBOUTL	IBOutletCollect				
ET	ion (Interface Builder Outlet Collection)				
CPP_WARN_IN	Invalid	Yes			
VALID_INITIALI	Initializer from				
ZER_FROM_SY	System Header				
STEM_HEADE					
R					
CPP_WARN_IN	Invalid iOS	Yes			
VALID_IOS_DE	Deployment				
PLOYMENT_TA	Target				
RGET					
CPP_WARN_IN	Invalid No	Yes			High
VALID_NO_BUI	Builtin Names				
LTIN_NAMES					
CPP_WARN_IN	Invalid	Yes			High
VALID_NORET	Noreturn				
URN	Attribute				
CPP_WARN_IN	Invalid	Yes			High
VALID_OFFSET	Offsetof				
OF					
CPP_WARN_IN	Invalid or	Yes			
VALID_OR_NO	Nonexistent				
NEXISTENT_DI	Directory				
RECTORY					
CPP_WARN_IN	Invalid Partial	Yes			
VALID_PARTIA	Specialization				
L_SPECIALIZA					
TION					
CPP_WARN_IN	Invalid	Yes			High
VALID_PP_TOK	Preprocessor				
EN	Token				
CPP_WARN_IN	Invalid Source	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
VALID_SOURCE_ENCODING	Encoding				
CPP_WARN_INVALID_TOKEN	Invalid Token	Yes			
VALID_TOKEN_PASTE	Paste				
CPP_WARN_INVALID_UTF8	Invalid UTF-8	Yes			
CPP_WARN_JUMP_SEH_UNWIND_FINALLY	Jump SEH (Structured Exception Handling) Finally	Yes			
CPP_WARN_KEYWORD_COMPATIBILITY	Keyword Compatibility	Yes			
CPP_WARN_KEYWORD_MACRO	Keyword Macro	Yes			
CPP_WARN_KR_PROMOTED_PARAMETER	K&R Promoted Parameter	Yes			
CPP_WARN_LANGUAGE_EXTENSION_TOKEN	Language Extension Token	Yes			
CPP_WARN_LARGE_BY_VALUE_COPY	Large by Value Copy	Yes			High
CPP_WARN_LINKER_WARNINGS	Linker Warnings	Yes			
CPP_WARN_LITERAL_CONVERSION	Literal Conversion	Yes			High
CPP_WARN_LITERAL_RANGE	Literal Range	Yes			High
CPP_WARN_LOCAL_TYPE_TEMPLATE_ARGS	Local Type Template Args	Yes			
CPP_WARN_LOGICAL_NOT	Logical Not	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
LOGICAL_NOT_PARENTHESSES	Parentheses				
CPP_WARN_LOGICAL_OP_PARENTHESSES	Logical Operator Parentheses	Yes			
CPP_WARN_LONG_LONG	Long Long	Yes			
CPP_WARN_MACRO_REDEFINED	Macro Redefined	Yes			High
CPP_WARN_MAIN_FUNCTION_CONVENTIONS	Main Function Conventions	Yes			
CPP_WARN_MAIN_RETURN_TYPE	Main Return Type	Yes			High
CPP_WARN_MALFORMED_WARNING_CHECK_K	Malformed Warning Check	Yes			
CPP_WARN_MANY_BRACES_AROUND_SCALAR_INIT	Many Braces Around Scalar Init	Yes			High
CPP_WARN_MAX_TOKENS	Max Tokens	Yes			
CPP_WARN_MAX_UNSIGNED_ZERO	Max Unsigned Zero	Yes			High
CPP_WARN_MEMSET_TRANSPOSED_ARGS	Memset Transposed Args	Yes			High
CPP_WARN_MEMSIZE_COMPARISON	Memsize Comparison	Yes			High
CPP_WARN_MICROSOFT_ABSTRACT	Microsoft Abstract	Yes			
CPP_WARN_MICROSOFT_ANONYMOUS_TAG	Microsoft Anonymous Tag	Yes			
CPP_WARN_MICROSOFT_CAST	Microsoft Cast	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CROSOFT_CAST					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_CHARIZE	Charizing Operator				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_COMMENT_PASTE	Comment Paste				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_CONST_INIT	Const Init				
CPP_WARN_MI	Microsoft C++	Yes			
CROSOFT_CPP_MACRO	Macro				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_DEFAULT_ARG	Default Arg				
CROSOFT_REDEFINITION	Redefinition				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_DIRECTIVE_SECTION	Drectve Section				
CPP_WARN_MI	Microsoft End	Yes			
CROSOFT_END_OF_FILE	of File				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_ENUM_FORWARD	Enum Forward				
CROSOFT_REFERENCE	Reference				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_ENUM_VALUE	Enum Value				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_EXCEPTION_SPEC	Exception Spec				
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_EXISTS	Exists				
CPP_WARN_MI	Microsoft	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CROSOFT_EXP	Explicit				
LICIT_CONSTR	Constructor				
UCTOR_CALL	Call				
CPP_WARN_MI	Microsoft Extra	Yes			
CROSOFT_EXT	Qualification				
RA_QUALIFICA					
TION					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_FIX	Fixed Enum				
ED_ENUM					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_FLE	Flexible Array				
XIBLE_ARRAY					
CPP_WARN_MI	Microsoft Goto	Yes			
CROSOFT_GO					
TO					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_INA	Inaccessible				
CCESSIBLE_B	Base				
ASE					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_INC	Include				
LUDE					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_MU	Mutable				
TABLE_REFER	Reference				
ENCE					
CPP_WARN_MI	Microsoft Pure	Yes			
CROSOFT_PU	Definition				
RE_DEFINITIO					
N					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_RE	Redeclare				
DECLARE_STA	Static				
TIC					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_SEA	Sealed				
LED					
CPP_WARN_MI	Microsoft	Yes			
CROSOFT_STA	Static Assert				
TIC_ASSERT					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_MI_CROSOFT_TEMPLATE	Microsoft Template	Yes			
CPP_WARN_MI_CROSOFT_TEMPLATE_SHA_DOW	Microsoft Template Shadow	Yes			
CPP_WARN_MI_CROSOFT_UNION_MEMBER_REFERENCE	Microsoft Union Member Reference	Yes			
CPP_WARN_MI_CROSOFT_UNQUALIFIED_FRIEND_IEND	Microsoft Unqualified Friend	Yes			
CPP_WARN_MI_CROSOFT_USING_DECLING_DECL	Microsoft Using Decl	Yes			
CPP_WARN_MI_CROSOFT_VOID_PSEUDO_D_PSEUDO_DT_OR	Microsoft Void Pseudo Destructor	Yes			
CPP_WARN_MI_SEXPECT	Misuse of __builtin_expect()	Yes			
CPP_WARN_MI_SLEADING_INDENTATION	Misleading Indentation	Yes			
CPP_WARN_MI_SMATCHED_NEW_DELETE	Mismatched New Delete	Yes			High
CPP_WARN_MI_SMATCHED_TAGS	Mismatched Tags	Yes			
CPP_WARN_MI_SSING_BRACES	Missing Braces	Yes			
CPP_WARN_MI_SSING_CONST_INIT	Missing Constinit	Yes			
CPP_WARN_MI	Missing	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
SSING_DECLARATIONS	Declarations				
CPP_WARN_MI	Missing	Yes			High
SSING_EXCEPTION_SPEC	Exception Spec				
CPP_WARN_MI	Missing Field	Yes			
SSING_FIELD_INITIALIZERS	Initializers				
CPP_WARN_MI	Missing	Yes			
SSING_METHOD_RETURN_TYPE	Method Return Type				
CPP_WARN_MI	Missing	Yes			
SSING_NORETURN_ATTRIBUTE	Noreturn Attribute				
CPP_WARN_MI	Missing	Yes			
SSING_PROTO_TYPE_FOR_CALLING_CONVENTION	Prototype for Calling Convention				
CPP_WARN_MI	Missing	Yes			
SSING_PROTO_TYPES	Prototypes				
CPP_WARN_MI	Missing	Yes			
SSING_SELECTOR_NAME	Selector Name				
CPP_WARN_MI	Missing	Yes			
SSING_SYSROOT	Sysroot				
CPP_WARN_MI	Missing	Yes			
SSING_VARIABLE_DECLARATIONS	Variable Declarations				
CPP_WARN_MI	Misspelled	Yes			
SSPELLED_ASSUMPTION	Assumption				
CPP_WARN_M	Module	Yes			
MODULE_CONFLICT	Conflict				
CPP_WARN_M	Module File	Yes			
MODULE_FILE_CONFIG_MISMATCH	Config Mismatch				

Check ID	Check Name	Supported	Automation	Category	Severity
ATCH					
CPP_WARN_M	Module File	Yes			
MODULE_FILE_EXTENSION	Extension				
CPP_WARN_M	Module Import	Yes			
MODULE_IMPORT_IN_EXTERN_C	in Extern C				
T_IN_EXTERN_C					
CPP_WARN_M	Modules	Yes			
MODULES_AMBIGUOUS_INTERNAL_LINKAGE	Ambiguous Internal Linkage				
CPP_WARN_M	Modules	Yes			
MODULES_IMPORT_NESTED_REDUNDANT	Import Nested Redundant				
CPP_WARN_M	MSVC Not Found	Yes			
SVC_NOT_FOUND					
CPP_WARN_M	Multiple Characters in Character Literal	Yes			High
ULTICHAR					
CPP_WARN_M	Multiple Move Virtual Base	Yes			High
MULTIPLE_MOVE_VIRTUAL_BASE					
CPP_WARN_N	Nested Anon-Types	Yes			
NESTED_ANON_TYPES					
CPP_WARN_N	New Returns Null	Yes			High
NEW_RETURNS_NULL					
CPP_WARN_N	Newline EOF (End of File)	Yes			
NEWLINE_EOF					
CPP_WARN_N	Noderef Attribute	Yes			High
NO_DEREF					
CPP_WARN_N	Non-C Typedef for Linkage	Yes			High
NON_C_TYPEDEF_FOR_LINKAGE					
F_FOR_LINKAGE					
CPP_WARN_N	Non-Literal	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
ON_LITERAL_NULL_CONVERSION	Null Conversion				
CPP_WARN_NON_POD_ON_POD_VARS_ARGS	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_DESTRUCTOR	Non-Virtual Destructor	Yes			
CPP_WARN_NULL_AS_ARGUMENT	Null as an Argument	Yes			High
CPP_WARN_NON_PORTABLE_INCLUDE_PATH_H	Non-Portable Include Path	Yes			High
CPP_WARN_NON_PORTABLE_SYSTEM_INCLUDE_PATH	Non-Portable System Include Path	Yes			
CPP_WARN_NON_PORTABLE_VECTOR_INITIALIZATION	Non-Portable Vector Initialization	Yes			
CPP_WARN_NONTRIVIAL_MEMORY_ACCESS	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	Null	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
ULL_DEREFERENCE	Dereference				
CPP_WARN_NULL_POINTER_ARITHMETIC	Null Pointer Arithmetic	Yes			
CPP_WARN_NULL_POINTER_SUBTRACTION	Null Pointer Subtraction	Yes			
CPP_WARN_OBJC_BOOL_CONSTANT_CONVERSION	Objective-C Bool Constant Conversion	Yes			
CPP_WARN_OBJC_CIRCULAR_CONTAINER	Objective-C Circular Container	Yes			
CPP_WARN_OBJC_MULTIPLE_METHOD_NAMES	Objective-C Multiple Method Names	Yes			
CPP_WARN_OBJC_READONLY_WITH_SETTER_PROPERTY	Objective-C Readonly with Setter Property	Yes			
CPP_WARN_OBJC_SIGNED_CHAR_BOOL_IMPLICIT_FLOAT_CONVERSION	Objective-C Signed Char Bool Implicit Float Conversion	Yes			
CPP_WARN_OBJC_SIGNED_CHAR_BOOL_IMPLICIT_INT_CONVERSION	Objective-C Signed Char Bool Implicit Int Conversion	Yes			
CPP_WARN_ONE_DEFINITION_RULE	One Definition Rule	Yes			High
CPP_WARN_OLD_STYLE_CAST	Old Style Cast	Yes			
CPP_WARN_OPENMP_51_EXTENSIONS	OpenMP 51 Extensions	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_OVERRIDING_T_OPTION	Overriding Slash T Option	Yes			
CPP_WARN_PACKED_ATTRIBUTE	Packed Attribute	Yes			
CPP_WARN_PADDING_ADDED	Implicit Padding	Yes			
CPP_WARN_PARENTHESES_ARENTHESES	Parentheses	Yes			
CPP_WARN_PARENTHESES_ARENTHESES_EQUALITY	Parentheses Equality	Yes			High
CPP_WARN_PASS_FAILED_ASS_FAILED	Pass Failed	Yes			
CPP_WARN_PCH_DATE_TIME	PCH (Precompiled Header) Date Time	Yes			
CPP_WARN_PEDANTIC	Pedantic	Yes			
CPP_WARN_PEDANTIC_CORE_FEATURES	Pedantic Core Features	Yes			
CPP_WARN_PESSIMIZING_MOVE_OVE	Pessimizing Move	Yes			
CPP_WARN_POINTER_ARITHMETIC_H	Pointer Arithmetic	Yes			High
CPP_WARN_POINTER_BOOL_CONVERSION	Pointer Bool Conversion	Yes			High
CPP_WARN_POINTER_COMPARE	Pointer Compare	Yes			High
CPP_WARN_POINTER_INTEGER_COMPARE	Pointer Integer Compare	Yes			High
CPP_WARN_POINTER_SIGN	Pointer Sign	Yes			High

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

Check ID	Check Name	Supported	Automation	Category	Severity
RE_C2X_COM PAT	Compatibility				
CPP_WARN_P RE_CPP2B_CO MPAT	Pre C++2B Compatibility	Yes			
CPP_WARN_P RE_CPP14_CO MPAT	Pre C++14 Compatibility	Yes			
CPP_WARN_P RE_CPP17_CO MPAT	Pre C++17 Compatibility	Yes			
CPP_WARN_P RE_CPP17_CO MPAT_PEDANT IC	Pre C++17 Compatibility Pedantic	Yes			
CPP_WARN_P RE_CPP20_CO MPAT	Pre C++20 Compatibility	Yes			
CPP_WARN_P RE_CPP20_CO MPAT_PEDANT IC	Pre C++20 Compatibility Pedantic	Yes			
CPP_WARN_P RE_OPENMP_5 1_COMPAT	Pre OpenMP 51 Compatibility	Yes			
CPP_WARN_P REDEFINED_ID ENTIFIER_OUT SIDE_FUNCTI ON	Predefined Identifier Outside Function	Yes			
CPP_WARN_P PRIVATE_EXTER N	Private Extern	Yes			
CPP_WARN_P PRIVATE_HEAD ER	Private Header	Yes			
CPP_WARN_P ROFILE_INSTR _MISSING	Profile Instrumented Code Missing	Yes			
CPP_WARN_P ROFILE_INSTR	Profile Instrumented	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
_OUT_OF_DATAE	Code Out of Date				
CPP_WARN_PROFILE_INSTR_UNPROFILED	Profile Instrumented Code Unprofiled	Yes			
CPP_WARN_PSABI_SABI	PSABI (Processor-Specific Application Binary Interface)	Yes			
CPP_WARN_QUALIFIED_VOID_RETURN_TYPEPE	Qualified Void Return Type	Yes			High
CPP_WARN_RANGE_LOOP_BIND_REFERENCEIND_REFERENCCE	Range Loop Bind Reference	Yes			
CPP_WARN_RANGE_LOOP_CONSTRUCTONSTRUCT	Range Loop Construct	Yes			
CPP_WARN_REDECLARED_CLASS_MEMBERCLASS_MEMBERR	Re-Declared Class Member	Yes			High
CPP_WARN_REDUNDANT_CONSTEVAL_IFONSTEVAL_IF	Redundant Consteval If	Yes			High
CPP_WARN_REDUNDANT_MOVEOVE	Redundant Move	Yes			
CPP_WARN_REDUNDANT_PARENTHESESARENS	Redundant Parentheses	Yes			
CPP_WARN_REGISTER_KEYWORD	Register Keyword	Yes			
CPP_WARN_REINTERPRET_BASE_CLASSASE_CLASS	Reinterpret Base Class	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_R EORDER_CTOR	Reorder Constructor	Yes			
CPP_WARN_R EORDER_INIT_LIST	Reorder Initializer List	Yes			High
CPP_WARN_R ESERVED_IDENTIFIER	Reserved Identifier	Yes			
CPP_WARN_R ESERVED_MACRO_IDENTIFIER	Reserved Macro Identifier	Yes			
CPP_WARN_R ESERVED_USER_DEFINED_LITERAL	Reserved User Defined Literal	Yes			
CPP_WARN_R ESTRICT_EXPANSION	Restrict Expansion	Yes			
CPP_WARN_R ETAINED_LINKAGE	Retained Language Linkage	Yes			
CPP_WARN_R ETURN_STACK_ADDRESS	Return Stack Address	Yes			High
CPP_WARN_R ETURN_TYPE	Return Type	Yes			High
CPP_WARN_R ETURN_TYPE_C_LINKAGE	Return Type C Linkage	Yes			High
CPP_WARN_R EWRITE_NOT_BOOL	Rewrite Not Bool	Yes			
CPP_WARN_R TTI	Run-Time Type Information	Yes			
CPP_WARN_S ARIF_FORMAT_UNSTABLE	SARIF Format Unstable	Yes			
CPP_WARN_SE	Section	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
E_VALUE					
CPP_WARN_S	Shift Operator	Yes			High
HIFT_OP_PARE	Parentheses				
NTHESSES					
CPP_WARN_S	Shift Overflow	Yes			High
HIFT_OVERFL					
OW					
CPP_WARN_S	Shift Sign	Yes			
HIFT_SIGN_OV	Overflow				
ERFLOW					
CPP_WARN_S	Shorten	Yes			
HORTEN_64_T	Integer Type				
O_32	Width				
CPP_WARN_SI	Sign Compare	Yes			
GN_COMPARE					
CPP_WARN_SI	Sign	Yes			
GN_CONVERSI	Conversion				
ON					
CPP_WARN_SI	Signed Enum	Yes			
GNED_ENUM_	Bitfield				
BITFIELD					
CPP_WARN_SI	Signed	Yes			
GNED_UNSIG	Unsigned				
NED_WCHAR	Wchar				
CPP_WARN_SI	Single Bit	Yes			High
NGLE_BIT_BIT	Bitfield				
FIELD_CONST	Constant				
ANT_CONVER	Conversion				
SION					
CPP_WARN_SI	Sizeof Array	Yes			High
ZEOF_ARRAY_	Argument				
ARGUMENT					
CPP_WARN_SI	Sizeof Array	Yes			High
ZEOF_ARRAY_	Decay				
DECAY					
CPP_WARN_SI	Sizeof Array	Yes			High
ZEOF_ARRAY_	Division				
DIV					
CPP_WARN_SI	Sizeof Pointer	Yes			High
ZEOF_POINTE	Division				
R_DIV					

Check ID	Check Name	Supported	Automation	Category	Severity
CPP_WARN_SI ZEOF_POINTE R_MEMACCES S	Sizeof Pointer Memory Access	Yes			High
CPP_WARN_SL ASH_U_FILEN AME	Slash U Filename	Yes			
CPP_WARN_SL H_ASM_GOTO	SLH (Speculative Load Hardening) Assembly Goto	Yes			
CPP_WARN_S OMETIMES_U NINITIALIZED	Sometimes Uninitialized	Yes			
CPP_WARN_S OURCE_USES_ OPENMP	Source Uses OpenMP	Yes			
CPP_WARN_S PIR_COMPAT	SPIR (Sampler Initializer) Compatibility	Yes			
CPP_WARN_ST ACK_EXHAUS TED	Stack Exhausted	Yes			
CPP_WARN_ST ACK_PROTECT OR	Stack Protector	Yes			
CPP_WARN_ST ATIC_FLOAT_I NIT	Static Float Init	Yes			
CPP_WARN_ST ATIC_IN_INLIN E	Static in Inline	Yes			
CPP_WARN_ST ATIC_INLINE_E XPPLICIT_INSTA NTIATION	Static Inline Explicit Instantiation	Yes			High
CPP_WARN_ST ATIC_LOCAL_I N_INLINE	Static Local in Inline	Yes			High
CPP_WARN_ST	Static Self Init	Yes			High

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

Check ID	Check Name	Supported	Automation	Category	Severity
RRIDE					
CPP_WARN_S	Super Class	Yes			
UPER_CLASS_	Method				
METHOD_MIS	Mismatch				
MATCH					
CPP_WARN_S	Suspicious	Yes			
USPICIOUS_B	Argument for				
ZERO	Bzero Function				
CPP_WARN_S	Switch	Yes			High
WITCH	Statements				
CPP_WARN_S	Switch Bool	Yes			High
WITCH_BOOL					
CPP_WARN_S	Switch Enum	Yes			
WITCH_ENUM					
CPP_WARN_S	Sync Fetch	Yes			
YNC_FETCH_A	And Nand				
ND_NAND_SE	Semantics				
MANTICS_CHA	Changed				
NGED					
CPP_WARN_TA	Target Clones	Yes			
RGET_CLONES	Mixed				
_MIXED_SPECI	Specifiers				
FIERS					
CPP_WARN_TA	Tautological	Yes			
UTOLOGICAL_	Bitwise				
BITWISE_COM	Compare				
PARE					
CPP_WARN_TA	Tautological	Yes			
UTOLOGICAL_	Compare				
COMPARE					
CPP_WARN_TA	Tautological	Yes			High
UTOLOGICAL_	Constant				
CONSTANT_C	Compare				
OMPARE					
CPP_WARN_TA	Tautological	Yes			High
UTOLOGICAL_	Constant Out				
CONSTANT_O	of Range				
UT_OF_RANGE	Compare				
_COMPARE					
CPP_WARN_TA	Tautological	Yes			
UTOLOGICAL_	Overlap				

Check ID	Check Name	Supported	Automation	Category	Severity
OVERLAP_CO MPARE	Compare				
CPP_WARN_TA UTOLOGICAL_ POINTER_COM PARE	Tautological Pointer Compare	Yes			High
CPP_WARN_TA UTOLOGICAL_ TYPE_LIMIT_C OMPARE	Tautological Type Limit Compare	Yes			
CPP_WARN_TA UTOLOGICAL_ UNDEFINED_C OMPARE	Tautological Undefined Compare	Yes			High
CPP_WARN_TA UTOLOGICAL_ UNSIGNED_CH AR_ZERO_CO MPARE	Tautological Unsigned Char Zero Compare	Yes			
CPP_WARN_TA UTOLOGICAL_ UNSIGNED_EN UM_ZERO_CO MPARE	Tautological Unsigned Enum Zero Compare	Yes			
CPP_WARN_TA UTOLOGICAL_ UNSIGNED_ZE RO_COMPARE	Tautological Unsigned Zero Compare	Yes			
CPP_WARN_TA UTOLOGICAL_ VALUE_RANGE _COMPARE	Tautological Value Range Compare	Yes			
CPP_WARN_T CB_ENFORCE MENT	TCB (Trusted Computing Base) Enforcement	Yes			
CPP_WARN_TE NTATIVE_DEFI NITION_INCO MPLETE_TYPE	Tentative Definition Incomplete Type	Yes			High
CPP_WARN_T HREAD_SAFET	Thread Safety Analysis	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
Y_ANALYSIS					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Attributes				
Y_ATTRIBUTES					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Beta				
Y_BETA					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Negative				
Y_NEGATIVE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Precise				
Y_PRECISE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Reference				
Y_REFERENCE					
CPP_WARN_T	Thread Safety	Yes			
HREAD_SAFET	Verbose				
Y_VERBOSE					
CPP_WARN_TR	Trigraphs	Yes			High
IGRAPHS					
CPP_WARN_TY	Type Safety	Yes			High
PE_SAFETY					
CPP_WARN_TY	Typedef	Yes			High
PEDEF_REDEFI	Redefinition				
NITION					
CPP_WARN_TY	Typename	Yes			High
PENAME_MISS	Missing				
ING					
CPP_WARN_U	Unable to	Yes			
NABLE_TO_OP	Open Stats File				
EN_STATS_FIL					
E					
CPP_WARN_U	Unaligned	Yes			
NALIGNED_AC	Access				
CESS					
CPP_WARN_U	Unaligned	Yes			
NALIGNED_QU	Qualifier				
ALIFIER_IMPLI	Implicit Cast				
CIT_CAST					
CPP_WARN_U	Undefined	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NDEF	Macros				
CPP_WARN_U	Undefined	Yes			
NDEF_PREFIX	Macros of a Certain Prefix				
CPP_WARN_U	Undefined	Yes			High
NDEFINED_BO	Bool				
OL_CONVERSION	Conversion				
CPP_WARN_U	Undefined	Yes			
NDEFINED_FU	Function				
NC_TEMPLATE	Template				
CPP_WARN_U	Undefined	Yes			High
NDEFINED_INL	Inline				
CPP_WARN_U	Undefined	Yes			High
NDEFINED_INT	Internal				
ERNAL					
CPP_WARN_U	Undefined	Yes			
NDEFINED_INT	Internal Type				
ERNAL_TYPE					
CPP_WARN_U	Undefined	Yes			
NDEFINED_REI	Reinterpret				
NTERPRET_CA	Cast				
ST					
CPP_WARN_U	Undefined Var	Yes			High
NDEFINED_VA	Template				
R_TEMPLATE					
CPP_WARN_U	Underaligned	Yes			
NDERALIGNED	Exception				
EXCEPTION	Object				
OBJECT					
CPP_WARN_U	Unevaluated	Yes			High
NEVALUATED_	Expression				
EXPRESSION					
CPP_WARN_U	Unguarded	Yes			
NGUARDED_A	Availability				
VAILABILITY					
CPP_WARN_U	Unguarded	Yes			High
NGUARDED_A	Availability				
VAILABILITY_N	New				
EW					

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

Check ID	Check Name	Supported	Automation	Category	Severity
NKNOWN_WARNING_OPTION	Warning Option				
CPP_WARN_UNNAMED_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_STANDARD_CAST_CALL	Unqualified Standard Cast Call	Yes			High
CPP_WARN_UNREACHABLE_CODE	Unreachable Code	Yes			
CPP_WARN_UNREACHABLE_CODE_BREAK	Unreachable Code Break	Yes			
CPP_WARN_UNREACHABLE_CODE_FALLTHROUGH	Unreachable Code Fallthrough	Yes			
CPP_WARN_UNREACHABLE_CODE_GENERIC_ASSOC	Unreachable Code Generic Assoc	Yes			
CPP_WARN_UNREACHABLE_CODE_LOOP_INCREMENT	Unreachable Code Loop Increment	Yes			
CPP_WARN_UNREACHABLE_CODE_RETURN	Unreachable Code Return	Yes			
CPP_WARN_UNSAFE_BUFFER	Unsafe Buffer	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
NSAFE_BUFFER_USAGE	Usage				
CPP_WARN_UNSEQUENCED	Unsequenced Modifications	Yes			High
CPP_WARN_UNNSUPPORTED_ABI	Unsupported ABI (Application Binary Interface)	Yes			
CPP_WARN_UNNSUPPORTED_ABS	Unsupported Absolute Value Argument	Yes			
CPP_WARN_UNNSUPPORTED_AVAILABILITY_GUARD	Unsupported Availability Guard	Yes			High
CPP_WARN_UNNSUPPORTED_CB	Unsupported Compact Branches	Yes			
CPP_WARN_UNNSUPPORTED_DLL_BASE_CLASS_TEMPLATE	Unsupported DLL Base Class Template	Yes			
CPP_WARN_UNNSUPPORTED_FLOATING_POINT_OPTION	Unsupported Floating Point Option	Yes			
CPP_WARN_UNNSUPPORTED_FRIEND	Unsupported Friend	Yes			High
CPP_WARN_UNNSUPPORTED_GPOPT	Unsupported GPOpt (Gaussian Process Optimization)	Yes			
CPP_WARN_UNNSUPPORTED_NAN	Unsupported Nan Argument	Yes			
CPP_WARN_UNNSUPPORTED_TARGET_OPTION	Unsupported Target Option	Yes			

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

Check ID	Check Name	Supported	Automation	Category	Severity
NUSED_LOCA L_TYPEDEF	Typedef				
CPP_WARN_U NUSED_MACR OS	Unused Macros OS	Yes			
CPP_WARN_U NUSED_MEMB ER_FUNCTION	Unused Member Function	Yes			
CPP_WARN_U NUSED_PARA METER	Unused Parameter Meter	Yes			
CPP_WARN_U NUSED_PRIVA TE_FIELD	Unused Private Field	Yes			
CPP_WARN_U NUSED_PROP ERTY_IVAR	Unused Property IVar (Instance Variable)	Yes			
CPP_WARN_U NUSED_RESUL T	Unused Result	Yes			High
CPP_WARN_U NUSED_TEMP LATE	Unused Template Late	Yes			
CPP_WARN_U NUSED_VALUE	Unused Value	Yes			High
CPP_WARN_U NUSED_VARIA BLE	Unused Variable BLE	Yes			
CPP_WARN_U NUSED_VOLAT ILE_LVALUE	Unused Volatile Lvalue ILE_LVALUE	Yes			High
CPP_WARN_U SED_BUT_MA RKED_UNUSE D	Used but Marked Unused D	Yes			
CPP_WARN_U SER_DEFINED _LITERALS	User Defined Literals _LITERALS	Yes			High
CPP_WARN_U SER_DEFINED_ Warnings	User Defined Warnings	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
WARNINGS					
CPP_WARN_VA_ARGS	Variadic Arguments	Yes			High
CPP_WARN_VA_MACRO	Variadic Macros	Yes			
OS					
CPP_WARN_VE_C_ELEM_SIZE	Vector Element Size	Yes			
CPP_WARN_VE_CTOR_CONVERSION	Vector Conversion	Yes			
CPP_WARN_VE_VEXING_PARSE	Vexing Parse Occurrences	Yes			High
CPP_WARN_VI_SIBILITY	Visibility of Declarations	Yes			High
CPP_WARN_VL_A	VLA (Variable Length Array)	Yes			
CPP_WARN_VL_A_EXTENSION	VLA (Variable Length Array) Extension	Yes			
CPP_WARN_VOID_POINTER_TO_ENUM_CAST	Void Pointer to Enum Cast	Yes			High
CPP_WARN_VOID_POINTER_TO_INT_CAST	Void Pointer to Int Cast	Yes			High
CPP_WARN_VOID_POINTER_DEREFERENCE	Void Pointer Dereference	Yes			High
CPP_WARN_WARNINGS	Preprocessor #Warnings	Yes			
CPP_WARN_WASM_EXCEPTI_ON_SPEC	Wasm Exception Spec	Yes			
CPP_WARN_WEAK_VTABLES	Weak VTables (Virtual Tables)	Yes			
CPP_WARN_WRITABLE_STRINGS	Writable Strings	Yes			High
NGS					
CPP_WARN_XOR_USED_AS	Xor Used as	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
OR_USED_AS_POWER	Power				
CPP_WARN_ZERO_AS_NULL_POINTER_CONSTANT	Zero as Null Pointer Constant	Yes			
CPP_WARN_ZERO_LENGTH_ARRAY	Zero Length Array	Yes			
CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	Yes			
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('Command Injection')	Yes			
CWE-119A	Improper Restriction of Operations within the Bounds of a Memory Buffer(Part A: Read)	Yes			
CWE-119B	Improper Restriction of Operations within the Bounds of a Memory Buffer(Part B: Write)	Yes			
CWE-125	Out-of-bounds Read	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
CWE-190	Integer Overflow or Wraparound	Yes			
CWE-476	NULL Pointer Dereference	Yes			
CWE-787	Out-of-bounds Write	Yes			
CWE-798	Use of Hard-coded Credentials (Partial)	Yes			
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 classifications	Yes			Medium
DCL37-C	Do not declare or define a reserved identifier	Yes			Low
DCL38-C	Use the correct syntax when declaring a flexible array	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
DCL39-C	member 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
DCL52-CPP	Never qualify a reference type with const or volatile	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
DCL60-CPP	Obey the one-definition rule	Yes			High
EFFECTIVECPP_02	2. Do Not Use #define	Yes			
EFFECTIVECPP_03	3. Use Const whenever possible	Yes			
EFFECTIVECPP	4. Make sure	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
_04	that objects are initialized before they are used				
EFFECTIVECPP_26	26. Postpone variable definitions as long as possible	Yes			
EFFECTIVECPP_27	27. Minimize casting	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

Check ID	Check Name	Supported	Automation	Category	Severity
ERR33-C	Detect and handle standard library errors	Yes			High
ERR34-C	Detect errors when converting a string to a number	Yes			Medium
ERR52-CPP	Do not use setjmp() or longjmp()	Yes			Low
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 reference	Yes			Low
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

Check ID	Check Name	Supported	Automation	Category	Severity
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-qualified pointers	No			Medium
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	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	the incorrect type				
EXP50-CPP	Do not depend on the order of evaluation for side effects	Yes			Medium
EXP53-CPP	Do not read uninitialized memory	Yes			High
EXP54-CPP	Do not access an object outside of its lifetime	Yes			High
EXP55-CPP	Do not access a cv-qualified object through a cv-unqualified type	Yes			Medium
EXP56-CPP	Do not call a function with a mismatched language linkage	No			Low
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
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	No			High

Check ID	Check Name	Supported	Automation	Category	Severity
	characters read from a file and EOF or WEOF				
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 input and output from a stream without an intervening flush or positioning call	Yes			Low
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	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
	race conditions while accessing files				
FIO46-C	Do not access a closed file	Yes			Medium
FIO47-C	Use valid format strings	Yes			High
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	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	object representations 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 Statements (STMT)	Yes			
HIS_09	9. Number of call levels (LEVEL)	Yes			
HIS_10	10. Number of return points (RETURN)	Yes			
HIS_11	11. Language scope (VOCF)	Yes			
HIS_12	12. Recursion (AP_CG_CYCL E)	Yes			
HIS_13	13. Statements	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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	No			Low

Check ID	Check Name	Supported	Automation	Category	Severity
	than or equal to the number of bits that exist in the operand				
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	
M0-1-8	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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M0-4-1	Overlapping Object (Partial) 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 typographically unambiguous	Yes	Automated	Required	
M2-13-2	Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes	Automated	Required	
M2-13-3	A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes	Automated	Required	
M2-13-4	Literal suffixes shall be upper case	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M3-1-2	Functions shall not be declared at block scope	Yes	Automated	Required	
M3-2-1	All declarations of an object or function shall have compatible types	Yes	Automated	Required	
M3-2-2	The One Definition Rule	Yes	Automated	Required	
M3-2-3	A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes	Automated	Required	
M3-2-4	An identifier with external linkage shall have exactly one definition	Yes	Automated	Required	
M3-3-2	If a function has internal linkage then all redeclarations shall include the static storage class 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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	function parameter shall be token-for-token identical in all declarations and re-declarations				
M3-9-3	The underlying bit representation of floating-point values shall not be used	Yes	Automated	Required	
M4-5-1	Expressions with type bool shall not be used as operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator	Yes	Automated	Required	
M4-5-3	Character Operators	Yes	Automated	Required	
M4-10-1	NULL shall not be used as an integer value	Yes	Automated	Required	
M4-10-2	Literal zero (0) shall not be	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	used as the null-pointer-constant.				
M5-0-2	Limited dependence should be placed on C++ operator precedence rules in expressions	Yes	Automated	Advisory	
M5-0-3	A cvalue expression shall not be implicitly converted to a different underlying type	Yes	Automated	Required	
M5-0-4	An implicit integral conversion shall not change the signedness of the underlying type	Yes	Automated	Required	
M5-0-5	There shall be no implicit floating-integral conversions	Yes	Automated	Required	
M5-0-6	An implicit integral or floating-point conversion shall not reduce the size of the underlying type	Yes	Automated	Required	
M5-0-7	There shall be	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	no explicit floating-integral conversions of a cvalue expression				
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	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-0-11	The plain char type shall only be used for the storage and use of character values	Yes	Automated	Required	
M5-0-12	Signed char and unsigned char type shall only be used for the storage and use of numeric values	Yes	Automated	Required	
M5-0-14	The first operand of a conditional-operator shall have type bool	Yes	Automated	Required	
M5-0-15	Array indexing over pointer arithmetic	Yes	Automated	Required	
M5-0-16	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array	Yes	Automated	Required	
M5-0-17	Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes	Automated	Required	
M5-0-18	>, >=, <, <=	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be applied to objects of pointer type, except where they point to the same array				
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 pointer to a function to any other pointer type, including a pointer to function type	Yes	Automated	Required	
M5-2-8	An object with integer type or pointer to void type shall not be converted to an object	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	with pointer type.				
M5-2-9	Pointer to Integer Cast	Yes	Automated	Required	
M5-2-10	The increment (++) and decrement (--) operators shall not be mixed with other operators in an expression	Yes	Automated	Required	
M5-2-11	The comma operator, && operator and the operator shall not be overloaded	Yes	Automated	Required	
M5-2-12	Array to Pointer Decay	Yes	Automated	Required	
M5-3-1	Each operand of the ! operator, the logical && or the logical operators shall have type bool	Yes	Automated	Required	
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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M5-8-1	side effects The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left hand operand.	Yes	Partially Automated	Required	
M5-14-1	The right hand operand of a logical &&, operators shall not contain side effects	Yes	Automated	Required	
M5-17-1	The semantic equivalence between a binary operator and its assignment operator form shall be preserved	Yes	Non-automated	Required	
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	Yes	Partially Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M6-2-3	inequality Before preprocessing, a null statement shall only occur on a line by itself; it may be followed by a comment, provided that the first character following the null statement is a white-space character	Yes	Automated	Required	
M6-3-1	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes	Automated	Required	
M6-4-1	An if (condition) construct shall be followed by a compound 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	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be terminated with an else clause				
M6-4-3	A switch statement shall be a well-formed switch statement	Yes	Automated	Required	
M6-4-4	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement	Yes	Automated	Required	
M6-4-5	An unconditional throw or break statement shall terminate every non-empty switch-clause	Yes	Automated	Required	
M6-4-6	The final clause of a switch statement shall be the default-clause	Yes	Automated	Required	
M6-4-7	The condition of a switch statement shall not have bool type	Yes	Automated	Required	
M6-5-2	If loop-counter is not modified by -- or ++, then, within	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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 statement shall have type bool	Yes	Automated	Required	
M6-6-1	Any label referenced by a goto	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	statement shall be declared in the same block, or in a block enclosing the goto statement				
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	
M7-3-4	Using-directives shall not be used.	Yes	Automated	Required	
M7-4-1	Assembly Language Code Usage	Yes	Non-automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M7-4-2	not Documented Assembler instructions shall only be introduced using the asm declaration.	Yes	Automated	Required	
M7-4-3	Assembly language shall be encapsulated and isolated.	Yes	Automated	Required	
M7-5-1	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.	Yes	Non-automated	Required	
M7-5-2	The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.	Yes	Non-automated	Required	
M8-0-1	Single Declarations	Yes	Automated	Required	
M8-4-2	The identifiers used for the parameters in a re-declaration of	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	a function shall be identical to those in the declaration.				
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	
M9-6-4	Bit-field Length	Yes	Automated	Required	
M10-1-1	Class Derived From Virtual Bases	Yes	Automated	Advisory	
M10-1-2	A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes	Automated	Required	
M10-1-3	An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes	Automated	Required	
M10-2-1	Similar Entity Names within Multiple Inheritance	Yes	Automated	Advisory	
M10-3-3	A virtual function shall only be overridden by a pure virtual function if it is	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
M11-0-1	itself declared as pure virtual 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	
M15-3-1	Exceptions shall be raised only after start-up and	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	before termination of the program				
M16-0-1	#include Directives Not Grouped Together	Yes	Automated	Required	
M16-0-2	Macros shall only be #define'd or #undef'd in the global namespace.	Yes	Automated	Required	
M16-0-5	Function-like Macro Containing Preprocessing Directives	Yes	Automated	Required	
M16-0-6	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##	Yes	Automated	Required	
M16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 guards shall be provided	Yes	Automated	Required	
M16-3-1	There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes	Automated	Required	
M16-3-2	The # and ## operators should not be used	Yes	Automated	Advisory	
M17-0-2	The names of standard library macros and objects shall not be reused	Yes	Automated	Required	
M17-0-3	Standard Library Function Names	Yes	Automated	Required	
M17-0-5	The setjmp macro and the longjmp function shall not be used	Yes	Automated	Required	
M18-0-3	<cstdlib> Library Functions	Yes	Automated	Required	
M18-0-4	Time Handling	Yes	Automated	Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	Functions of <code><ctime></code>				
M18-0-5	Unbounded Functions of <code><cstring></code>	Yes	Automated	Required	
M18-2-1	The macro <code>offsetof</code> shall not be used	Yes	Automated	Required	
M18-7-1	The signal handling facilities of <code><csignal></code> shall not be used	Yes	Automated	Required	
M19-3-1	The error indicator <code>errno</code> shall not be used	Yes	Automated	Required	
M27-0-1	The stream input/output library <code><cstdio></code> 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	Yes			High

Check ID	Check Name	Supported	Automation	Category	Severity
	sufficient memory for 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
MEM52-CPP	Detect and handle memory allocation errors	Yes			High
MEM53-CPP	Explicitly construct and destruct objects when manually managing object lifetime	No			High
METRIC_00	Program Unit Call Count	Yes			
METRIC_01	Program Unit Callby Count	Yes			
METRIC_02	Program Unit Comment to Code Ratio	Yes			
METRIC_03	Program Unit Cyclomatic Complexity	Yes			
METRIC_04	Program Unit Max Length	Yes			
METRIC_05	Program Unit Max Nesting Depth	Yes			
METRIC_06	Program Unit Parameters Count	Yes			
METRIC_07	Program Unit	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Path Count				
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_1.1	1.1 All code shall conform to ISO/IEC 9899:1990 "Programming languages — C", amended and corrected by ISO/IEC 9899/COR1:1995, ISO/IEC 9899/AMD1:1995, and ISO/IEC 9899/COR2:1996	No		Required	
MISRA04_1.2	1.2 No reliance shall be placed on undefined or unspecified behaviour	No		Required	
MISRA04_1.3	1.3 Multiple compilers and/or languages shall only be used if there is a common	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	defined interface standard for object code to which the languages/compilers/assemblers conform				
MISRA04_1.4	1.4 The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers	No		Required	
MISRA04_1.5	1.5 Floating-point implementations should comply with a defined floating-point standard	No		Advisory	
MISRA04_2.1	2.1 Assembly language shall be encapsulated and isolated.	Yes		Required	
MISRA04_2.2	2.2 C99 / Comments	Yes		Required	
MISRA04_2.3	2.3 The character sequence /* shall not be used within a comment.	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA04_2.4	2.4 Sections of code should not be "commented out"	Yes		Advisory	
MISRA04_3.1	3.1 All usage of non-defined behaviour shall be documented	No		Required	
MISRA04_3.2	3.2 The character set and the corresponding encoding shall be documented	No		Required	
MISRA04_3.3	3.3 The implementation of integer division in the chosen compiler should be determined, documented and taken into account	No		Advisory	
MISRA04_3.4	3.4 All uses of the #pragma directive shall be documented and explained	No		Required	
MISRA04_3.5	3.5 The implementation defined behaviour and packing of bitfields shall be	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	documented if being relied upon				
MISRA04_3.6	3.6 All libraries used in production code shall be written to comply with the provisions of this document, and shall have been subject to appropriate validation	No		Required	
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 Long Identifier Name Significance	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_5.6	5.6 Identifier Reuse in Multiple C	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Name Spaces				
MISRA04_5.7	5.7 Identifier Reuse	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 or signed int.	Yes		Required	
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	zero) and octal escape sequences shall not be used.				
MISRA04_8.1	8.1 Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.	No		Required	
MISRA04_8.2	8.2 Whenever an object or function is declared or defined, its type shall be explicitly stated	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_8.4	8.4 If objects or functions are declared more than once their types shall be compatible.	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 function	Yes		Required	
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 Array Size Missing	Yes		Required	
MISRA04_9.1	9.1 All automatic variables shall have been	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	assigned a value before being used				
MISRA04_9.2	9.2 Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures	Yes		Required	
MISRA04_9.3	9.3 = construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes		Required	
MISRA04_10.1	10.1 The value of an expression of integer type shall not be implicitly converted to a different underlying type if: it is not a conversion to a wider integer type of the same signedness, or (a) the expression is complex, or (b) the expression is not constant and is a function	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	argument, or (c) the expression is not constant and is a return expression (d)				
MISRA04_10.2	10.2 The value of an expression of floating type shall not be implicitly converted to a different type if: it is not a conversion to a wider floating type, or (a) the expression is complex, or (b) the expression is a function argument, or (c) the expression is a return expression (d)	No		Required	
MISRA04_10.3	10.3 The value of a complex expression of integer type shall only be cast to a type of the same signedness that is no wider than the underlying type of the expression.	No		Required	
MISRA04_10.4	10.4 The value of a complex expression of	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	floating type shall only be cast to a floating type that is narrower or of the same size.				
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_11.1	11.1 Conversions shall not be performed between a pointer to a function and any type other than an integral type.	No		Required	
MISRA04_11.2	11.2 Conversions shall not be performed	No		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.				
MISRA04_11.3	11.3 A cast should not be performed between a pointer type and an integral type.	No		Advisory	
MISRA04_11.4	11.4 A cast should not be performed between a pointer to object type and a different pointer to object type	Yes		Advisory	
MISRA04_11.5	11.5 A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer	Yes		Required	
MISRA04_12.1	12.1 Limited dependence should be placed on C's operator precedence rules in	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA04_12.2	expressions 12.2 The value of an expression shall be the same under any order of evaluation that the standard permits	Yes		Required	
MISRA04_12.3	12.3 The sizeof operator shall not be used on expressions that contain side effects	Yes		Required	
MISRA04_12.4	12.4 The right-hand operand of a logical && or operator shall not contain side effects	Yes		Required	
MISRA04_12.5	12.5 The operands of a logical && or shall be primary-expressions	Yes		Required	
MISRA04_12.6	12.6 The operands of logical operators (&&, and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	operators other than (&&, , !, =, ==, != and ?:)				
MISRA04_12.7	12.7 Bitwise operators shall not be applied to operands whose underlying type is signed	Yes		Required	
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_12.9	12.9 The unary minus operator shall not be applied to an expression whose underlying type is unsigned	Yes		Required	
MISRA04_12.10	12.10 The comma operator shall not be used	Yes		Required	
MISRA04_12.11	12.11 Evaluation of constant unsigned integer expressions	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should not lead to wrap-around				
MISRA04_12.1 2	12.12 The underlying bit representations of floating-point values shall not be used	Yes		Required	
MISRA04_12.1 3	12.13 The increment (++) and decrement (--) operators should not be mixed with other operators in an expression	Yes		Advisory	
MISRA04_13.1	13.1 Assignment operators shall not be used in expressions that yield a Boolean value.	No		Required	
MISRA04_13.2	13.2 Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.	No		Advisory	
MISRA04_13.3	13.3 Floating-point expressions shall not be tested for equality or inequality	Yes		Required	
MISRA04_13.4	13.4 The	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	controlling expression of a for statement shall not contain any objects of floating type				
MISRA04_13.5	13.5 The three expressions of a for statement shall be concerned only with loop control.	No		Required	
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_13.7	13.7 Boolean operations whose results are invariant shall not be permitted	No		Required	
MISRA04_14.1	14.1 There shall be no unreachable code	Yes		Required	
MISRA04_14.2	14.2 All non-null statements shall either (a) have at least one side-effect however executed, or (b) cause	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	control flow to change (Partial)				
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 goto statement shall not be used	Yes		Required	
MISRA04_14.5	14.5 No Continue Statements	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	forming the body of a switch, while, do ... while or for statement shall be a compound statement				
MISRA04_14.9	14.9 An if (expression) 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	
MISRA04_14.10	14.10 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA04_15	15 The MISRA C switch syntax shall be used	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA04_15.2	statement 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_15.4	15.4 A switch expression shall not represent a value that is effectively Boolean	Yes		Required	
MISRA04_15.5	15.5 Every switch statement shall have at least one case clause	Yes		Required	
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	must have an identifier.				
MISRA04_16.4	16.4 Inconsistent Parameter Names	Yes		Required	
MISRA04_16.5	16.5 Missing Parameters	Yes		Required	
MISRA04_16.6	16.6 The number of arguments passed to a function shall match the number of parameters	Yes		Required	
MISRA04_16.7	16.7 A pointer parameter in a function prototype should be declared as pointer to const if the pointer is not used to modify the addressed object	Yes		Advisory	
MISRA04_16.8	16.8 All exit paths from a function with non-void return type shall have an explicit return statement with an expression	Yes		Required	
MISRA04_16.9	16.9 A function identifier shall only be used with either a preceding &, or	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	with a parenthesised parameter list, which may be empty				
MISRA04_16.10	16.10 Functions shall not be defined with variable numbers of arguments	No		Required	
MISRA04_17.1	17.1 Pointer arithmetic shall only be applied to pointers that address an array or array element	No		Required	
MISRA04_17.2	17.2 Pointer subtraction shall only be applied to pointers that address elements of the same array	No		Required	
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_17.4	17.4 Array indexing shall be the only allowed form of pointer arithmetic	Yes		Required	
MISRA04_17.5	17.5 No more than 2 levels of	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	pointer indirection				
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_18.1	18.1 All structure or union types shall be complete at the end of a translation unit.	No		Required	
MISRA04_18.2	18.2 An object shall not be assigned to an overlapping object	No		Required	
MISRA04_18.3	18.3 An area of memory shall not be reused for unrelated purposes	No		Required	
MISRA04_18.4	18.4 Unions	Yes		Required	
MISRA04_19.1	19.1 #include statements in a file should only be preceded by other preprocessor directives or comments	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
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_19.4	19.4 C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class specifier, or a do-while-zero construct.	No		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 Preprocessor #undef	Yes		Required	
MISRA04_19.7	19.7 A function should be used in preference	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	to a function-like macro				
MISRA04_19.8	19.8 A function-like macro shall not be invoked without all of its arguments.	No		Required	
MISRA04_19.9	19.9 Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA04_19.10	19.10 In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##	Yes		Required	
MISRA04_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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA04_19.1 2	19.12 There shall be at most one occurrence of the # or ## operators in a single macro definition	Yes		Required	
MISRA04_19.1 3	19.12 The # and ## operators should not be used	Yes		Advisory	
MISRA04_19.1 4	19.14 The defined preprocessor operator shall only be used in one of the two standard forms	Yes		Required	
MISRA04_19.1 5	19.15 Precautions shall be taken in order to prevent the contents of a header file being included twice	Yes		Required	
MISRA04_19.1 6	19.16 Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor	Yes		Required	
MISRA04_19.1 7	20.14 All #else, #elif and #endif	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	preprocessor directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related				
MISRA04_20.1	20.1 Reserved or Standard Library Identifiers as Macros	Yes		Required	
MISRA04_20.2	20.2 The names of standard library macros and objects shall not be reused	Yes		Required	
MISRA04_20.3	20.3 The validity of values passed to library functions shall be checked	No		Required	
MISRA04_20.4	20.4 Dynamic Memory Allocation	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	and the longjmp 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 Including <stdio.h>	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 used	Yes		Required	
MISRA04_21.1	21.1 Minimisation of run-time failures shall be ensured by the use of at least one of: (a) static	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	analysis tools/ techniques; (b) dynamic analysis tools/ techniques; (c) explicit coding of checks to handle run- time faults.				
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_0-1-5	0-1-5 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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	called				
MISRA08_2-3-1	2-3-1 Trigraphs shall not be used	Yes		Required	
MISRA08_2-5-1	2-5-1 Digraphs shall not be used	Yes		Advisory	
MISRA08_2-7-1	2-7-1 The character sequence /* shall not be used within a C-style comment.	Yes		Required	
MISRA08_2-7-2	2-7-2 Sections of code shall not be "commented out" using C-style comments	Yes		Required	
MISRA08_2-7-3	2-7-3 Sections of code should not be "commented out" using C++ comments	Yes		Advisory	
MISRA08_2-10-1	2-10-1 Different identifiers shall be typographically unambiguous	Yes		Required	
MISRA08_2-10-2	2-10-2 Shadowed Identifiers	Yes		Required	
MISRA08_2-10-4	2-10-4 A class, union or enum name (including qualification, if any) shall be a	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	unique identifier				
MISRA08_2-13-1	2-13-1 Nonstandard Escape Sequences	Yes		Required	
MISRA08_2-13-2	2-13-2 Octal constants (other than zero) and octal escape sequences (other than "\0") shall not be used.	Yes		Required	
MISRA08_2-13-3	2-13-3 A "U" suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.	Yes		Required	
MISRA08_2-13-4	2-13-4 Literal suffixes shall be upper case	Yes		Required	
MISRA08_2-13-5	2-13-5 Narrow and wide string literals shall not be concatenated	Yes		Required	
MISRA08_3-1-1	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
2	Functions shall not be declared at block scope				
MISRA08_3-1-3	3-1-3 Array Size Missing	Yes		Required	
MISRA08_3-2-1	3-2-1 All declarations of an object or function shall have compatible types	Yes		Required	
MISRA08_3-2-2	3-2-2 The One Definition Rule	Yes		Required	
MISRA08_3-2-3	3-2-3 A type, object or function that is used in multiple translation units shall be declared in one and only one file	Yes		Required	
MISRA08_3-2-4	3-2-4 An identifier with external linkage shall have exactly one definition	Yes		Required	
MISRA08_3-3-1	3-3-1 Objects or functions with external 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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall include the static storage class specifier				
MISRA08_3-4-1	3-4-1 Declarations at Lowest Scope	Yes		Required	
MISRA08_3-9-1	3-9-1 The types used for an object, a function return type, or a function parameter shall be token-for-token identical in all declarations and re-declarations	Yes		Required	
MISRA08_3-9-2	3-9-2 Typedefs that indicate size and signedness should be used in place of the basic numerical types	Yes		Advisory	
MISRA08_3-9-3	3-9-3 The underlying bit representations of floating-point values shall not be used	Yes		Required	
MISRA08_4-5-1	4-5-1 Expressions with type bool shall not be used as	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	operands to built-in operators other than the assignment operator =, the logical operators &&, , !, the equality operators == and !=, the unary & operator, and the conditional operator				
MISRA08_4-5-2	Expressions with type enum shall not be used as operands to built-in operators other than the subscript operator [], the assignment operator =, the equality operators == and !=, the unary & operator, and the relational operators <, <=, >, >=	Yes		Required	
MISRA08_4-5-3	Character Operators	Yes		Required	
MISRA08_4-10-1	NULL shall not be used as an	Yes		Required	

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

Check ID	Check Name	Supported	Automation	Category	Severity
5	shall be no implicit floating-integral conversions				
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_5-0-10	cvalue expression 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-operator shall have type bool	Yes		Required	
MISRA08_5-0-15	5-0-15 Array	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
15	indexing shall be the only form of pointer arithmetic				
MISRA08_5-0-17	5-0-17 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA08_5-0-18	5-0-18 >, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array	Yes		Required	
MISRA08_5-0-19	5-0-19 No more than 2 levels of pointer indirection	Yes		Required	
MISRA08_5-0-20	5-0-20 Non-constant operands to a binary bitwise operator shall have the same underlying type	Yes		Required	
MISRA08_5-0-21	5-0-21 Bitwise operators shall only be applied to operands of unsigned underlying type	Yes		Required	
MISRA08_5-2-3	5-2-3 Casts	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
3	from a base class to a derived class should not be performed on polymorphic types				
MISRA08_5-2-5	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 type.	Yes		Required	
MISRA08_5-2-9	5-2-9 Pointer to Integer Cast	Yes		Advisory	
MISRA08_5-2-10	5-2-10 The increment (++) and decrement (--) operators shall not be mixed with	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	other operators in an expression				
MISRA08_5-2-11	5-2-11 The comma operator, && operator and the operator shall not be overloaded	Yes		Required	
MISRA08_5-2-12	5-2-12 Array to Pointer Decay	Yes		Required	
MISRA08_5-3-1	5-3-1 Each operand of the ! operator, the logical && or the logical operators shall have type bool	Yes		Required	
MISRA08_5-3-2	5-3-2 The unary minus operator shall not be applied to an expression whose underlying type is unsigned	Yes		Required	
MISRA08_5-3-3	5-3-3 The unary & operator shall not be overloaded	Yes		Required	
MISRA08_5-3-4	5-3-4 Evaluation of the operand to the sizeof operator shall not contain side effects	Yes		Required	
MISRA08_5-8-1	5-8-1 The right	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	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.				
MISRA08_5-14-1	5-14-1 The right hand operand of a logical && or operator shall not contain side effects	Yes		Required	
MISRA08_5-18-1	5-18-1 The comma operator shall not be used	Yes		Required	
MISRA08_6-2-1	6-2-1 Assignment operators shall not be used in sub-expressions	Yes		Required	
MISRA08_6-2-2	6-2-2 Floating-point expressions shall not be directly or indirectly tested for equality or inequality	Yes		Required	
MISRA08_6-2-3	6-2-3 Before preprocessing, a null statement shall only occur on a line by itself; it	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	may be followed by a comment, provided that the first character following the null statement is a white-space character				
MISRA08_6-3-1	6-3-1 The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement	Yes		Required	
MISRA08_6-4-1	6-4-1 An if (condition) construct shall be followed by a compound statement. The else keyword shall be followed by either a compound statement, or another if statement	Yes		Required	
MISRA08_6-4-2	6-4-2 All if ... else if constructs shall be terminated with an else clause	Yes		Required	
MISRA08_6-4-3	6-4-3 A switch	Yes		Required	

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

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

Check ID	Check Name	Supported	Automation	Category	Severity
	expression				
MISRA08_6-5-6	6-5-6 A loop-control-variable other than the loop-counter which is modified in statement shall have type bool	Yes		Required	
MISRA08_6-6-1	6-6-1 Any label 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 termination	Yes		Required	
MISRA08_6-6-5	6-6-5 A function shall have a single point of exit at the end of the function	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 main shall not be used for a function other than the global function main	Yes		Required	
MISRA08_7-3-4	7-3-4 Using-directives shall not be used.	Yes		Required	
MISRA08_7-4-2	7-4-2 Assembler instructions shall only be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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 call themselves, either directly or indirectly.	Yes		Advisory	
MISRA08_8-0-1	8-0-1 Single Declarations	Yes		Required	
MISRA08_8-4-2	8-4-2 Use the	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
2	same identifier in definition and declaration of functions.				
MISRA08_8-4-3	8-4-3 Always return a value in non-void functions	Yes		Required	
MISRA08_8-4-4	8-4-4 A function identifier shall either be used to call the function or it shall be preceded by &	Yes		Required	
MISRA08_8-5-1	8-5-1 All variables shall have a defined value before they are used	Yes		Required	
MISRA08_8-5-2	8-5-2 Incorrect Initializer Lists	Yes		Required	
MISRA08_8-5-3	8-5-3 The = construct in enumerator list shall only be used on either the first item alone, or all items explicitly.	Yes		Required	
MISRA08_9-5-1	9-5-1 Unions	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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_9-6-4	9-6-4 Named signed bit-field length	Yes		Required	
MISRA08_10-1-2	10-1-2 A base class shall only be declared virtual if it is used in a diamond hierarchy	Yes		Required	
MISRA08_10-1-3	10-1-3 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA08_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_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_15-1-1	15-1-1 The assignment-expression of a throw statement shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	not itself cause an exception to be thrown				
MISRA08_15-3-1	15-3-1 Exceptions shall be raised only after start-up and before termination of the program	Yes		Required	
MISRA08_15-3-6	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_16-0-1	16-0-1 #include directives in a file shall only be preceded by other preprocessor directives or comments	Yes		Required	
MISRA08_16-0-2	16-0-2 Macros shall only be #define'd or #undef'd in the global namespace	Yes		Required	
MISRA08_16-0-3	16-0-3 Preprocessor #undef	Yes		Required	
MISRA08_16-0-4	16-0-4 Function-like macros shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA08_16-0-5	not be defined Arguments to a function-like macro shall not contain tokens that look like preprocessing directives	Yes		Required	
MISRA08_16-0-6	In the definition of a function-like macro, each instance of a parameter shall be enclosed in parentheses, unless it is used as the operand of # or ##	Yes		Required	
MISRA08_16-0-7	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator	Yes		Required	
MISRA08_16-0-8	Invalid Preprocessor Directives	Yes		Required	
MISRA08_16-1-1	The defined preprocessor operator shall only be used in	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	one of the two standard forms				
MISRA08_16-2-1	16-2-1 The pre-processor shall only be used for file inclusion and include guards	Yes		Required	
MISRA08_16-2-2	16-2-2 Invalid Macro Usage	Yes		Required	
MISRA08_16-2-3	16-2-3 Include guards shall be provided	Yes		Required	
MISRA08_16-2-4	16-2-4 The ' , " , /* or // characters shall not occur in a header file name	Yes		Required	
MISRA08_16-2-5	16-2-5 The backslash character should not occur in a header file name	Yes		Advisory	
MISRA08_16-3-1	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 or Standard	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	Library Identifiers as Macros				
MISRA08_17-0-2	17-0-2 The names of standard library macros and objects shall not be reused	Yes		Required	
MISRA08_17-0-3	17-0-3 Standard Library Function Names	Yes		Required	
MISRA08_17-0-5	17-0-5 The setjmp macro and the longjmp function shall not be used	Yes		Required	
MISRA08_18-0-1	18-0-1 The C library shall not be used	Yes		Required	
MISRA08_18-0-2	18-0-2 The library functions atof, atoi and atol from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-3	18-0-3 The library functions abort, exit, getenv and system from library <cstdlib> shall not be used	Yes		Required	
MISRA08_18-0-4	18-0-4 The time handling	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	functions of library <ctime> shall not be used				
MISRA08_18-0-5	18-0-5 Unbounded Functions of <cstring>	Yes		Required	
MISRA08_18-2-1	18-2-1 The macro offsetof shall not be used.	Yes		Required	
MISRA08_18-4-1	18-4-1 Dynamic Memory Allocation	Yes		Required	
MISRA08_18-7-1	18-7-1 The signal handling facilities of <csignal> shall not be used	Yes		Required	
MISRA08_19-3-1	19-3-1 The error indicator "errno" shall not be used.	Yes		Required	
MISRA08_27-0-1	27-0-1 The stream input/output library <stdio> shall not be used	Yes		Required	
MISRA12_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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_1.2	1.2 Language extensions should not be used (Partial)	Yes		Advisory	
MISRA12_1.3	1.3 There shall be no occurrence of undefined or critical unspecified behaviour (Partial)	Yes		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 (Partial)	Yes		Required	
MISRA12_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA12_2.4	2.4 Unused Tag Declarations	Yes		Advisory	
MISRA12_2.5	2.5 Unused Macros	Yes		Advisory	
MISRA12_2.6	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	a comment				
MISRA12_3.2	3.2 Line-Splicing 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 Indistinct Macro Identifiers	Yes		Required	
MISRA12_5.5	5.5 Same Identifier and Macro Name	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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	
MISRA12_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA12_6.2	6.2 Signed single-bit named bit-fields	Yes		Required	
MISRA12_7.1	7.1 Octal constants shall not be used	Yes		Required	
MISRA12_7.2	7.2 Unsigned Integer Literal Without Suffix	Yes		Required	
MISRA12_7.3	7.3 Lowercase L Suffix	Yes		Required	
MISRA12_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes		Required	
MISRA12_8.1	8.1 Types shall be explicitly specified	Yes		Required	
MISRA12_8.2	8.2 Use Named Parameters and Prototype Form	Yes		Required	
MISRA12_8.3	8.3 All declarations of	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	an object or function shall use the same names and type qualifiers				
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 external linkage if they are referenced in only one translation unit	Yes		Advisory	
MISRA12_8.8	8.8 Use the static keyword for internal linkage	Yes		Required	
MISRA12_8.9	8.9 Objects	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be local if only accessed from one function				
MISRA12_8.10	8.10 Non-static Inline Functions	Yes		Required	
MISRA12_8.11	8.11 When an array with external linkage is declared, its size should be explicitly specified	Yes		Advisory	
MISRA12_8.12	8.12 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA12_8.13	8.13 A pointer should point to a const-qualified type whenever possible	Yes		Advisory	
MISRA12_8.14	8.14 Restrict Qualifier	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	an aggregate or union shall be enclosed in braces				
MISRA12_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA12_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA12_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA12_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA12_10.2	10.2 Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations	Yes		Required	
MISRA12_10.3	10.3 The value of an expression shall not be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	assigned to an object with a narrower essential type or of a different essential type category				
MISRA12_10.4	10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes		Required	
MISRA12_10.5	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 Implicit Casts of Operations	Yes		Required	
MISRA12_10.8	10.8 The value of a composite expression shall not be cast to a	Yes		Required	

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

Check ID	Check Name	Supported	Automation	Category	Severity
	performed from pointer to void into pointer to object				
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 pointed to by a pointer	Yes		Required	
MISRA12_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA12_12.1	12.1 The precedence of operators within expressions should be	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_12.2	made explicit 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 unsigned integer wrap-around	Yes		Advisory	
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	Yes		Required	
MISRA12_13.3	13.3 A full expression containing an	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator				
MISRA12_13.4	13.4 The result of an assignment operator should not be used	Yes		Advisory	
MISRA12_13.5	13.5 The right hand operand of a logical && or operator shall not contain persistent side effects	Yes		Required	
MISRA12_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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_14.3	14.3 Controlling expressions shall not be invariant	Yes		Required	
MISRA12_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	
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 Goto Into or Between Blocks	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 a single point of	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_15.6	exit at the end 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	16.1 All switch statements shall be 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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
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 Switch Boolean	Yes		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 not be declared implicitly	Yes		Mandatory	
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	type shall have an appropriate number of elements				
MISRA12_17.6	17.6 Static Keyword in Array Brackets	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 modified	Yes		Advisory	
MISRA12_18.1	18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand (Partial)	Yes		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 <=	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be applied to objects of pointer type except where they point into the same object				
MISRA12_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	Yes		Advisory	
MISRA12_18.5	18.5 Declarations should contain no more than two levels of pointer nesting	Yes		Advisory	
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	Yes		Required	
MISRA12_18.8	18.8 Variable-length arrays shall not be used	Yes		Required	
MISRA12_19.1	19.1 An object shall not be assigned or	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	copied to an overlapping object (Partial)				
MISRA12_19.2	19.2 Unions	Yes		Advisory	
MISRA12_20.1	20.1 #include directives should only be preceded by preprocessor directives or comments	Yes		Advisory	
MISRA12_20.2	20.2 Invalid Header 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 Keyword Macros	Yes		Required	
MISRA12_20.5	20.5 Preprocessor #undef	Yes		Advisory	
MISRA12_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Required	
MISRA12_20.7	20.7 Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
MISRA12_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation	Yes		Required	
MISRA12_20.10	20.10 The # and ## operators should not be used	Yes		Advisory	
MISRA12_20.11	20.11 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Yes		Required	
MISRA12_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement,	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall only be used as an operand to these operators				
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 identifier or reserved macro name	Yes		Required	
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	<setjmp.h> shall not be used				
MISRA12_21.5	21.5 Standard Header signal.h	Yes		Required	
MISRA12_21.6	21.6 C Standard Library I/O Functions	Yes		Required	
MISRA12_21.7	21.7 The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used	Yes		Required	
MISRA12_21.8	21.8 The library functions abort, exit, getenv and system of <stdlib.h> shall not be used	Yes		Required	
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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_21.12	used 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 Library functions shall be explicitly released	Yes		Required	
MISRA12_22.2	22.2 A block of memory shall only be freed if it was allocated by means of a Standard Library function	Yes		Required	
MISRA12_22.3	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	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	read-only				
MISRA12_22.5	22.5 Dereference of FILE Pointer	Yes		Mandatory	
MISRA12_22.6	22.6 The value of a pointer to a FILE shall not be used after the associated stream has been closed (Partial)	Yes		Mandatory	
MISRA12_DIR_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	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	assembly language should be documented				
MISRA12_DIR_4.3	Directive 4.3 Assembly language shall be encapsulated and isolated.	Yes		Required	
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 returns error information, then that error information shall be tested	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_DIR_4.8	Directive 4.8 If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden	Yes		Advisory	
MISRA12_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they are interchangeable	Yes		Advisory	
MISRA12_DIR_4.10	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 (Partial)	Yes		Required	
MISRA12_DIR_4.12	Directive 4.12 Dynamic Memory Allocation	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA12_DIR_4.13	Directive 4.13 Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes		Advisory	
MISRA23_0.0.1	0.0.1 A function shall not contain unreachable statements	Yes		Required	
MISRA23_0.1.1	0.1.1 A value should not be unnecessarily written to a local object	No		Advisory	
MISRA23_0.1.2	0.1.2 The value returned by a function shall be used	Yes		Required	
MISRA23_0.2.1	0.2.1 Variables with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.2	0.2.2 A named function parameter shall be used at least once	Yes		Required	
MISRA23_0.2.3	0.2.3 Types with limited visibility should be used at least once	Yes		Advisory	
MISRA23_0.2.4	0.2.4	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
4	Functions with limited visibility should be used at least once				
MISRA23_0.3.1	0.3.1 Floating-point arithmetic should be used appropriately	No		Advisory	
MISRA23_0.3.2	0.3.2 A function call shall not violate the function's preconditions	No		Required	
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 (Partial)	Yes		Advisory	
MISRA23_1.3	1.3 There shall be no occurrence of undefined or critical unspecified behaviour	Yes		Required	
MISRA23_1.4	1.4 Bounds-checking Interfaces	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_1.5	1.5 Storage Class Specifiers Not At Beginning	Yes		Required	
MISRA23_2.1	2.1 A project shall not contain unreachable code	Yes		Required	
MISRA23_2.2	2.2 A project shall not contain dead code (Partial)	Yes		Required	
MISRA23_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA23_2.4	2.4 Unused Tag Declarations	Yes		Advisory	
MISRA23_2.5	2.5 Unused Macros	Yes		Advisory	
MISRA23_2.6	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_2.8	2.8 Unused Objects	Yes		Advisory	
MISRA23_3.1	3.1 The character sequences /* and // shall not be used within a comment	Yes		Required	
MISRA23_3.2	3.2 Line-Splicing in //	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	Comments				
MISRA23_4.1	4.1 Octal and Hexadecimal Sequences	Yes		Required	
MISRA23_4.2	4.2 Trigraphs should not be used	Yes		Advisory	
MISRA23_4.6.1	4.6.1 Operations on a memory location shall be sequenced appropriately	No		Required	
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 Indistinct Macro Identifiers	Yes		Required	
MISRA23_5.5	5.5 Same Identifier and Macro Name	Yes		Required	
MISRA23_5.6	5.6 A typedef name shall be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	a unique identifier				
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 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 unique	Yes		Advisory	
MISRA23_5.10.1	5.10.1 User-defined identifiers shall have an appropriate form	Yes		Required	
MISRA23_5.13.1	5.13.1 Within	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	character literals and non raw-string literals, \ shall only be used to form a defined escape sequence or universal character name				
MISRA23_5.13.2	5.13.2 Octal escape sequences, hexadecimal escape sequences and universal character names shall be terminated	Yes		Required	
MISRA23_5.13.3	5.13.3 Octal constants shall not be used	Yes		Required	
MISRA23_5.13.4	5.13.4 Unsigned integer literals shall be appropriately suffixed	Yes		Required	
MISRA23_5.13.5	5.13.5 The lowercase form of L shall not be used as the first character in a literal suffix	Yes		Required	
MISRA23_5.13.6	5.13.6 An integer-literal of type long long shall not use a single L	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	or l in any suffix				
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 appropriate type	Yes		Required	
MISRA23_6.2	6.2 Signed single-bit named bit-fields	Yes		Required	
MISRA23_6.2.1	6.2.1 The one-definition rule shall not be	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	violated				
MISRA23_6.2.2	6.2.2 All declarations of a variable or function shall have the same type	Yes		Required	
MISRA23_6.2.3	6.2.3 The source code used to implement an entity shall appear only once	Yes		Required	
MISRA23_6.2.4	6.2.4 A header file shall not 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_6.4.1	6.4.1 A variable declared in an inner scope shall not hide a variable declared in an outer scope	Yes		Required	
MISRA23_6.5.1	6.5.1 A function or object with external linkage should	Yes		Advisory	

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

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_6.9.1	6.9.1 The same type aliases shall be used in all declarations of the same entity	Yes		Required	
MISRA23_6.9.2	6.9.2 The names of the standard signed integer types and standard unsigned integer types should not be used	Yes		Advisory	
MISRA23_7.0.1	7.0.1 There shall be no conversion from type bool	Yes		Required	
MISRA23_7.0.2	7.0.2 There shall be no conversion to type bool	Yes		Required	
MISRA23_7.0.3	7.0.3 The numerical value of a character shall not be used	Yes		Required	
MISRA23_7.1	7.1 Octal constants shall not be used	Yes		Required	
MISRA23_7.2	7.2 Unsigned Integer Literal Without Suffix	Yes		Required	
MISRA23_7.3	7.3 Lowercase L Suffix	Yes		Required	
MISRA23_7.4	7.4 A string literal shall not be assigned to an object	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	unless the object's type is "pointer to const-qualified char				
MISRA23_7.5	7.5 The argument of an integer constant macro shall have an appropriate form	Yes		Mandatory	
MISRA23_7.6	7.6 Small Integer Constant Macros	Yes		Required	
MISRA23_7.11.1	7.11.1 nullptr shall be the only form of the null-pointer-constant	Yes		Required	
MISRA23_7.11.2	7.11.2 Array to Pointer Decay	Yes		Required	
MISRA23_7.11.3	7.11.3 A conversion from function type to pointer-to-function type shall only occur in appropriate contexts	Yes		Required	
MISRA23_8.0.1	8.0.1 Parentheses should be used to make the meaning of an expression appropriately	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	explicit				
MISRA23_8.1	8.1 Types shall be explicitly specified	Yes		Required	
MISRA23_8.1.2	8.1.2 Variables should be captured explicitly in a non-transient lambda	Yes		Advisory	
MISRA23_8.2	8.2 Use Named Parameters and Prototype Form	Yes		Required	
MISRA23_8.2.1	8.2.1 A virtual base class shall only be cast to a derived class by means of dynamic_cast	Yes		Required	
MISRA23_8.2.2	8.2.2 C-style casts and functional notation casts shall not be used	Yes		Required	
MISRA23_8.2.3	8.2.3 A cast shall not remove any const or volatile qualification from the type accessed via a pointer or by reference	Yes		Required	
MISRA23_8.2.5	8.2.5 reinterpret_cast shall not be used	Yes		Required	
MISRA23_8.2.6	8.2.6 An object	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
6	with integral, enumerated, or pointer to void type shall not be cast to a pointer type				
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_8.2.10	8.2.10 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA23_8.2.11	8.2.11 An argument passed via ellipsis shall have an appropriate type	Yes		Required	
MISRA23_8.3	8.3 All declarations of an object or function shall use the same names and type qualifiers	Yes		Required	
MISRA23_8.3.1	8.3.1 The built-in unary - operator should not be applied to an expression of	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_8.3.2	unsigned type 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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	only be applied to pointers that address elements of the same array				
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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_8.13	8.13 A pointer should point to a const-qualified type whenever possible	Yes		Advisory	
MISRA23_8.14	8.14 Restrict Qualifier	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_8.18.1	8.18.1 An object or	Yes		Mandatory	

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

Check ID	Check Name	Supported	Automation	Category	Severity
	of an iteration-statement or a selection-statement shall be a compound-statement				
MISRA23_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA23_9.4.1	9.4.1 All if ... else if constructs shall be terminated with an else statement	Yes		Required	
MISRA23_9.4.2	9.4.2 The structure of a switch statement shall be appropriate	Yes		Required	
MISRA23_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly	Yes		Required	
MISRA23_9.6	9.6 An initializer using chained designators shall not contain initializers without	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	designators				
MISRA23_9.6.1	9.6.1 The goto statement should not be used	Yes		Advisory	
MISRA23_9.6.2	9.6.2 A goto statement shall reference a label in a surrounding block	Yes		Required	
MISRA23_9.6.3	9.6.3 The goto statement shall jump to a label declared later in the function body	Yes		Required	
MISRA23_9.6.4	9.6.4 A function declared with the [[noreturn]] attribute shall not return	Yes		Required	
MISRA23_9.6.5	9.6.5 A function with non-void return type shall return a value on all paths	Yes		Required	
MISRA23_9.7	9.7 Atomic objects shall be appropriately initialized before being accessed	Yes		Mandatory	
MISRA23_10.0.1	10.0.1 A declaration should not declare more	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	than one variable or member variable				
MISRA23_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA23_10.1.1	10.1.1 The target type of a pointer or lvalue reference parameter should be const-qualified appropriately	Yes		Advisory	
MISRA23_10.1.2	10.1.2 The volatile qualifier shall be used appropriately	Yes		Required	
MISRA23_10.2	10.2 Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations	Yes		Required	
MISRA23_10.2.1	10.2.1 An enumeration shall be defined with an explicit underlying type	Yes		Required	
MISRA23_10.2.2	10.2.2 Unscoped	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	enumerations should not be declared				
MISRA23_10.3	10.3 The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category	Yes		Required	
MISRA23_10.4	10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes		Required	
MISRA23_10.4 .1	10.4.1 The asm declaration shall not be used	Yes		Required	
MISRA23_10.5	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	object with wider essential type				
MISRA23_10.7	10.7 Implicit Casts of Operations	Yes		Required	
MISRA23_10.8	10.8 The value of a composite expression shall not be cast to a different essential type category or a wider essential type	Yes		Required	
MISRA23_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA23_11.2	11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes		Required	
MISRA23_11.3	11.3 A conversion shall not be performed between a pointer to object type and a pointer to a different	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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		Advisory	
MISRA23_11.5	11.5 A conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA23_11.6	11.6 A cast shall not be performed between pointer to void and an arithmetic type	Yes		Required	
MISRA23_11.6.1	11.6.1 All variables should be initialized	Yes		Advisory	
MISRA23_11.6.3	11.6.3 Within an enumerator	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	list, the value of an implicitly-specified enumeration constant shall be unique				
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 only permitted form of integer null pointer constant	Yes		Required	
MISRA23_11.10	11.10 The _Atomic qualifier shall not be applied to the incomplete type void	Yes		Required	
MISRA23_12.1	12.1 The precedence of operators	No		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	within expressions should be made explicit				
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 of one bit	Yes		Required	
MISRA23_12.3	12.3 The comma operator shall not be used.	Yes		Advisory	
MISRA23_12.3.1	12.3.1 Unions	Yes		Required	
MISRA23_12.4	12.4 Evaluation of constant expressions should not lead to unsigned	No		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	integer wrap-around				
MISRA23_12.5	17.4 Size of Array Parameter	Yes		Mandatory	
MISRA23_12.6	12.6 Structure and union members of atomic objects shall not be directly accessed	Yes		Required	
MISRA23_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA23_13.1.1	13.1.1 Classes should not be inherited virtually	Yes		Advisory	
MISRA23_13.1.2	13.1.2 An accessible base class shall not be both virtual and non-virtual in the same hierarchy	Yes		Required	
MISRA23_13.2	13.2 The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders	Yes		Required	
MISRA23_13.3	13.3 A full expression containing an	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator				
MISRA23_13.3.1	13.3.1 User-declared member functions shall use the virtual, override and final specifiers appropriately	Yes		Required	
MISRA23_13.3.3	13.3.3 The parameters in all declarations or overrides of a function shall either be unnamed or have identical names	Yes		Required	
MISRA23_13.4	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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_13.6	13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects	Yes		Mandatory	
MISRA23_13.13	13.13 Character Function Misuse	Yes		Mandatory	
MISRA23_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA23_14.2	14.2 A for loop shall be well-formed	Yes		Required	
MISRA23_14.3	14.3 Controlling expressions shall not be invariant	Yes		Required	
MISRA23_14.4	14.4 The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type	Yes		Required	
MISRA23_15.1	15.1 The goto statement should not be used	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_15.1.1	15.1.1 An object's dynamic type shall not be used from within its constructor or destructor	Yes		Required	
MISRA23_15.1.2	15.1.2 All constructors of a class should explicitly initialize all of its virtual base classes and immediate base classes	Yes		Advisory	
MISRA23_15.1.4	15.1.4 All direct, non-static data members of a class should be initialized before the class object is accessible	Yes		Required	
MISRA23_15.2	15.2 The goto statement shall jump to a label declared later in the same function	Yes		Required	
MISRA23_15.3	15.3 Goto Into or Between Blocks	Yes		Required	
MISRA23_15.4	15.4 There should be no more than one break or goto statement used to terminate any	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	iteration statement				
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	16.1 All switch statements shall be well-formed	Yes		Required	
MISRA23_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	
MISRA23_16.3	16.3 An unconditional break statement shall terminate every switch-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_16.4	clause 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_16.6	16.6 Every switch statement shall have at least two switch-clauses	Yes		Required	
MISRA23_16.7	16.7 Switch Boolean	Yes		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	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	implicitly				
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	Yes		Required	
MISRA23_17.6	17.6 Static Keyword in Array Brackets	Yes		Mandatory	
MISRA23_17.7	17.7 The value returned by a function having non-void return type shall be used	Yes		Required	
MISRA23_17.8	17.8 A function parameter should not be modified	Yes		Advisory	
MISRA23_17.9	17.9 Invalid _Noreturn	Yes		Mandatory	
MISRA23_17.10	17.10 A function declared with a _Noreturn function	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	specifier shall have void return type				
MISRA23_17.11	17.11 A function that never returns should be declared with a _Noreturn function specifier	Yes		Advisory	
MISRA23_17.12	17.12 A function identifier should only be used with either a preceding &, or with a parenthesized parameter list	Yes		Required	
MISRA23_17.13	17.13 A function type shall not be type qualified	Yes		Required	
MISRA23_18.1	18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand	Yes		Required	
MISRA23_18.2	18.2 Subtraction between pointers shall only be applied to pointers that address	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_18.3	elements of the same array 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_18.3.2	18.3.2 An exception of class type shall be caught by const reference or reference	Yes		Required	
MISRA23_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	Yes		Advisory	
MISRA23_18.4.1	18.4.1 Exception-unfriendly functions shall be noexcept	Yes		Required	
MISRA23_18.5	18.5 Declarations should contain no more than two levels of pointer nesting	Yes		Advisory	
MISRA23_18.5.1	18.5.1 A noexcept function	No		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	should not attempt to propagate an exception to the calling function				
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	Yes		Required	
MISRA23_18.8	18.8 Variable-length arrays shall not be used	Yes		Required	
MISRA23_18.9	18.9 An object with temporary lifetime shall not undergo array-to-pointer conversion	Yes		Required	
MISRA23_18.10	18.10 Pointer to Variable-length Array	Yes		Mandatory	
MISRA23_19.0.1	19.0.1 A line	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
1	whose first token is # shall be a valid preprocessing directive				
MISRA23_19.0.2	19.0.2 Function-like macros shall not be defined	Yes		Required	
MISRA23_19.0.4	19.0.4 #undef should only be used for macros defined previously in the same file	Yes		Advisory	
MISRA23_19.1	19.1 An object shall not be assigned or copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA23_19.1.1	19.1.1 The defined preprocessor operator shall be used appropriately	Yes		Required	
MISRA23_19.1.3	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 Unions	Yes		Advisory	
MISRA23_19.2.1	19.2.1 Precautions	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be 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 appropriately	Yes		Required	
MISRA23_19.3.5	19.3.5 Tokens that look like a preprocessing	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	directive shall not occur within a macro argument				
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 Invalid Header Name	Yes		Required	
MISRA23_20.3	20.3 The #include directive shall be followed by either a <filename> or "filename" sequence	Yes		Required	
MISRA23_20.4	20.4 Keyword Macros	Yes		Required	
MISRA23_20.5	20.5 Preprocessor #undef	Yes		Advisory	
MISRA23_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Required	
MISRA23_20.7	20.7 Expressions	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	resulting from the expansion of macro parameters shall be enclosed in parentheses				
MISRA23_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
MISRA23_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation	Yes		Required	
MISRA23_20.10	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	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				
MISRA23_20.13	20.13 A line whose first token is # shall be a valid preprocessing directive	Yes		Required	
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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_21.2.1	21.2.1 The library functions atof, atoi, atol and atoll from <cstdlib> shall not be used	Yes		Required	
MISRA23_21.2.2	21.2.2 The string handling functions from <cstring>, <cstdlib>, <wchar> and <inttypes> shall not be used	Yes		Required	
MISRA23_21.2.3	21.2.3 The library function system from <cstdlib> shall not be used	Yes		Required	
MISRA23_21.2.4	21.2.4 The macro offsetof shall not be used	Yes		Required	
MISRA23_21.3	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 Standard Header	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	signal.h				
MISRA23_21.6	21.6 C Standard Library I/O Functions	Yes		Required	
MISRA23_21.6.1	21.6.1 Dynamic memory should not be used	Yes		Advisory	
MISRA23_21.7	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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_21.1 0.2	21.10.2 The standard header file <csignal> shall not be used	Yes		Required	
MISRA23_21.1 0.3	21.10.3 The facilities provided by the standard header file <csignal> shall not be used	Yes		Required	
MISRA23_21.11	21.11 The standard header file <tgmath.h> shall not be used	Yes		Advisory	
MISRA23_21.1 2	21.12 The standard header file <fenv.h> shall not be used	Yes		Required	
MISRA23_21.1 3	21.13 Any value passed to a function in <ctype.h> shall be representable as an unsigned char or be the value EOF	Yes		Mandatory	
MISRA23_21.1 4	21.14 The Standard Library function memcmp shall not be used to compare null terminated strings	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_21.1 5	21.15 Incompatible Pointers	Yes		Required	
MISRA23_21.1 6	21.16 Invalid Memory Compare	Yes			
MISRA23_21.1 7	21.17 Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters	Yes		Mandatory	
MISRA23_21.1 8	21.18 Out of Bounds with string.h	Yes			
MISRA23_21.1 9	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_21.2 0	21.20 The pointer returned by the C++ Standard	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function				
MISRA23_21.2 1	21.21 The Standard Library function system of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.2 2	21.22 All operand arguments to any type-generic macros declared in <tgmath.h> shall have an appropriate essential type	Yes		Mandatory	
MISRA23_21.2 3	21.23 All operand arguments to any multi-argument type-generic macros declared in <tgmath.h> shall have the same standard	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_21.2 4	type 21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA23_21.2 5	21.25 Atomic Operations with Inconsistent Order	Yes		Required	
MISRA23_21.2 6	21.26 The Standard Library function mtx_timedlock () shall only be invoked on mutex objects of appropriate mutex type	Yes		Required	
MISRA23_22.1	22.1 All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes		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	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_22.3	22.3 The same file shall not be open for read and write access at the same time on different streams	Yes		Required	
MISRA23_22.3.1	22.3.1 The assert macro shall not be used with a constant-expression	Yes		Required	
MISRA23_22.4	22.4 There shall be no attempt to write to a stream which has been opened as read-only	Yes		Mandatory	
MISRA23_22.4.1	22.4.1 The literal value zero shall be the only value assigned to errno	Yes		Required	
MISRA23_22.5	22.5 Dereference of FILE Pointer	Yes		Mandatory	
MISRA23_22.6	22.6 Use of Closed FILE Pointers (Partial)	Yes		Mandatory	
MISRA23_22.7	22.7 EOF Macro Misuse	Yes		Required	
MISRA23_22.8	22.8 The value of errno shall be set to zero prior to a call to an errno-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	setting-function				
MISRA23_22.9	22.9 The value of errno shall be tested against zero after calling an errno-setting-function	Yes		Required	
MISRA23_22.10	22.10 The value of errno shall only be tested when the last function to be called was an errno-setting-function	Yes		Required	
MISRA23_22.11	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA23_22.12	22.12 Thread Data Misuse	Yes		Mandatory	
MISRA23_22.13	22.13 Thread objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration	Yes		Required	
MISRA23_22.15	22.15 Thread synchronization	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	n objects and thread-specific storage pointers shall not be destroyed until after all threads accessing them have terminated				
MISRA23_22.1 6	22.16 All mutex objects locked by a thread shall be explicitly unlocked by the same thread	Yes		Required	
MISRA23_22.1 7	22.17 No thread shall unlock a mutex or call <code>cnd_wait()</code> or <code>cnd_timedwait()</code> for a mutex it has not locked before	Yes		Required	
MISRA23_22.1 8	22.18 Non-recursive mutexes shall not be recursively locked	Yes		Required	
MISRA23_22.1 9	22.19 Mutexes with One Condition Variable	Yes		Required	
MISRA23_23.1	23.1 Generic Outside of Macro	Yes		Advisory	
MISRA23_23.2	23.2 Side	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	Effects in Generic Selections				
MISRA23_23.3	23.3 Unconditional Generic	Yes		Advisory	
MISRA23_23.4	23.4 A generic association shall list an appropriate type	No		Required	
MISRA23_23.5	23.4 A generic selection should not depend on implicit pointer type conversion	No		Advisory	
MISRA23_23.6	23.4 The controlling expression of a generic selection shall have an essential type that matches its standard type	No		Advisory	
MISRA23_23.7	23.7 Generic Expression Not Always Evaluated Once	Yes		Advisory	
MISRA23_23.8	23.8 Generic Default Not First or Last	Yes		Required	
MISRA23_24.5 .1	24.5.1 The character handling functions from <cctype> and <cwctype>	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be used				
MISRA23_24.5.2	24.5.2 The C++ Standard Library functions memcpy, memmove and memcmp from <cstring> shall not be used	Yes		Required	
MISRA23_25.5.1	25.5.1 The setlocale and std::locale::global functions shall not be called	Yes		Required	
MISRA23_25.5.2	25.5.2 The pointers returned by the C++ Standard Library functions localeconv, getenv, setlocale or strerror must only be used as if they have pointer to const-qualified type	Yes		Mandatory	
MISRA23_25.5.3	25.5.3 The pointer returned by the C++ Standard Library functions asctime, ctime, gmtime, localtime,	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	localeconv, getenv, setlocale or strerror must not be used following a subsequent call to the same function				
MISRA23_26.3.1	26.3.1 std::vector should not be specialized with bool	Yes		Advisory	
MISRA23_28.6.1	28.6.1 The argument to std::move shall be a non-const lvalue	Yes		Required	
MISRA23_28.6.2	28.6.2 Forwarding references and std::forward shall be used together	Yes		Required	
MISRA23_28.6.3	28.6.3 An object shall not be used while in a potentially moved-from state	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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_DIR_1.1	operation 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 files shall compile without any compilation errors	Yes		Required	
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	Yes		Advisory	
MISRA23_DIR_4.3	Directive 4.3 Assembly language shall be encapsulated and isolated.	Yes		Required	
MISRA23_DIR_4.4	Directive 4.4	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
4.4	Sections of code should not be "commented out"				
MISRA23_DIR_4.5	Directive 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Yes		Advisory	
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	Yes		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	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA23_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they are interchangeable	Yes		Advisory	
MISRA23_DIR_4.10	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	Yes		Required	
MISRA23_DIR_4.12	Directive 4.12 Dynamic Memory Allocation	Yes		Required	
MISRA23_DIR_4.13	Directive 4.13 Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes		Advisory	
MISRA23_DIR_4.14	Directive 4.14	Yes		Directive	

Check ID	Check Name	Supported	Automation	Category	Severity
4.14	The validity of values received from external sources shall be checked				
MISRA23_DIR_5.1	Directive 5.1 There shall be no data races between threads	No		Required	
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	
MISRA25_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	No		Required	
MISRA25_1.3	1.3 There shall be no occurrence of undefined or critical unspecified behaviour	Yes		Required	
MISRA25_1.4	1.4 Emergent language features shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_1.5	not be used 1.5 Obsolescent language features shall not be used	Yes		Required	
MISRA25_2.1	2.1 A project shall not contain unreachable code	Yes		Required	
MISRA25_2.2	2.2 A project shall not contain dead code (Partial)	Yes		Required	
MISRA25_2.3	2.3 A project should not contain unused type declarations	Yes		Advisory	
MISRA25_2.4	2.4 A project should not contain unused tag declarations	Yes		Advisory	
MISRA25_2.5	2.5 A project should not contain unused macro declarations	Yes		Advisory	
MISRA25_2.6	2.6 A function should not contain unused label declarations	Yes		Advisory	
MISRA25_2.7	2.7 A function should not contain unused parameters	Yes		Advisory	
MISRA25_2.8	2.8 A project should not contain unused	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	object definitions				
MISRA25_3.1	3.1 The character sequences /* and // shall not be used within a comment	Yes		Required	
MISRA25_3.2	3.2 Line-splicing shall not be used in // comments	Yes		Required	
MISRA25_4.1	4.1 Octal and hexadecimal escape sequences shall be terminated	Yes		Required	
MISRA25_4.2	4.2 Trigraphs should not be used	Yes		Advisory	
MISRA25_5.1	5.1 External identifiers shall be distinct	Yes		Required	
MISRA25_5.2	5.2 Identifiers declared in the same scope and name space shall be distinct	Yes		Required	
MISRA25_5.3	5.3 An identifier declared in an inner scope shall not hide an identifier declared in an outer scope	Yes		Required	
MISRA25_5.4	5.4 Macro identifiers shall be distinct	Yes		Required	
MISRA25_5.5	5.5 Identifiers	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall be distinct from macro names				
MISRA25_5.6	5.6 A typedef name shall be a unique identifier	Yes		Required	
MISRA25_5.7	5.7 A tag name shall be a unique identifier	Yes		Required	
MISRA25_5.8	5.8 Identifiers that define objects or functions with external linkage shall be unique	Yes		Required	
MISRA25_5.9	5.9 Identifiers that define objects or functions with internal linkage should be unique	Yes		Advisory	
MISRA25_5.10	5.10 A reserved identifier or reserved macro name shall not be declared	Yes		Required	
MISRA25_6.1	6.1 Bit-fields shall only be declared with an appropriate type	Yes		Required	
MISRA25_6.2	6.2 Single-bit named bit-fields shall not be of a signed type	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_6.3	6.3 A bit-field shall not be declared as a member of a union	Yes		Required	
MISRA25_7.1	7.1 Octal constants shall not be used	Yes		Required	
MISRA25_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	
MISRA25_7.3	7.3 The lowercase character 'l' shall not be used in a literal suffix	Yes		Required	
MISRA25_7.4	7.4 A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char"	Yes		Required	
MISRA25_7.5	7.5 The argument of an integer constant macro shall have an appropriate form	Yes		Mandatory	
MISRA25_7.6	7.6 The small integer	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	variants of the minimum-width integer constant macros shall not be used				
MISRA25_8.1	8.1 Types shall be explicitly specified	Yes		Required	
MISRA25_8.2	8.2 Function types shall be in prototype form with named parameters	Yes		Required	
MISRA25_8.3	8.3 All declarations of an object or function shall use the same names and type qualifiers	Yes		Required	
MISRA25_8.4	8.4 A compatible declaration shall be visible when an object or function with external linkage is defined	Yes		Required	
MISRA25_8.5	8.5 An external object or function shall be declared once in one and only one file	Yes		Required	
MISRA25_8.6	8.6 An identifier with external linkage shall	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_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	
MISRA25_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	
MISRA25_8.9	8.9 An object should be defined at block scope if its identifier only appears in a single function	Yes		Advisory	
MISRA25_8.10	8.10 An inline function shall be declared with the static storage class	Yes		Required	
MISRA25_8.11	8.11 When an array with external linkage is declared, its size should be explicitly specified	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_8.12	8.12 Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique	Yes		Required	
MISRA25_8.13	8.13 A pointer should point to a const-qualified type whenever possible	Yes		Advisory	
MISRA25_8.14	8.14 The restrict type qualifier shall not be used	Yes		Required	
MISRA25_8.15	8.15 All declarations of an object with an explicit alignment specification shall specify the same alignment	Yes		Required	
MISRA25_8.16	8.16 The alignment specification of zero should not appear in an object declaration	Yes		Advisory	
MISRA25_8.17	8.17 At most one explicit alignment specifier should appear in an object declaration	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_8.18	8.18 There shall be no tentative definitions in a header file	Yes		Required	
MISRA25_8.19	8.19 There should be no external declarations in a source file	Yes		Advisory	
MISRA25_9.1	9.1 The value of an object with automatic storage duration shall not be read before it has been set	Yes		Mandatory	
MISRA25_9.2	9.2 The initializer for an aggregate or union shall be enclosed in braces	Yes		Required	
MISRA25_9.3	9.3 Arrays shall not be partially initialized	Yes		Required	
MISRA25_9.4	9.4 An element of an object shall not be initialized more than once	Yes		Required	
MISRA25_9.5	9.5 Where designated initializers are used to initialize an array object the size of the array shall be specified	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_9.6	explicitly 9.6 An initializer using chained designators shall not contain initializers without designators	Yes		Required	
MISRA25_9.7	9.7 Atomic objects shall be appropriately initialized before being accessed	Yes		Mandatory	
MISRA25_10.1	10.1 Operands shall not be of an inappropriate essential type	Yes		Required	
MISRA25_10.2	10.2 Expressions of essentially character type shall not be used in addition and subtraction operations	Yes		Required	
MISRA25_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	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	essential type category				
MISRA25_10.4	10.4 Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category	Yes		Required	
MISRA25_10.5	10.5 The value of an expression should not be cast to an inappropriate essential type	Yes		Advisory	
MISRA25_10.6	10.6 The value of a composite expression shall not be assigned to an object with wider essential type	Yes		Required	
MISRA25_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 wider essential type	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_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	
MISRA25_11.1	11.1 Conversions shall not be performed between a pointer to a function and any other type	Yes		Required	
MISRA25_11.2	11.2 Conversions shall not be performed between a pointer to an incomplete type and any other type	Yes		Required	
MISRA25_11.3	11.3 A conversion shall not be performed between a pointer to object type and a pointer to a different object type	Yes		Required	
MISRA25_11.4	11.4 A conversion shall not be performed between a pointer to	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_11.5	11.5 A object and an arithmetic type conversion should not be performed from pointer to void into pointer to object	Yes		Advisory	
MISRA25_11.6	11.6 A cast shall not be performed between pointer to void and an arithmetic type	Yes		Required	
MISRA25_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	
MISRA25_11.9	11.9 The macro NULL shall be the only permitted form of integer null pointer constant	Yes		Required	
MISRA25_11.10	11.10 The _Atomic qualifier shall not be applied to the incomplete type void	Yes		Required	
MISRA25_11.11	11.11 Pointers	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall not be implicitly compared to NULL				
MISRA25_12.1	12.1 The precedence of operators within expressions should be made explicit	No		Advisory	
MISRA25_12.2	12.2 The right hand operand of a shift operator shall lie in the range zero to one less than the width in bits of the essential type of the left hand operand	Yes		Required	
MISRA25_12.3	12.3 The comma operator should not be used	Yes		Advisory	
MISRA25_12.4	12.4 Evaluation of constant expressions should not lead to unsigned integer wrap-around	No		Advisory	
MISRA25_12.5	12.5 The sizeof operator shall not have an operand which is a function parameter declared as	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_12.6	"array of type" 12.6 Structure and union members of atomic objects shall not be directly accessed	Yes		Required	
MISRA25_13.1	13.1 Initializer lists shall not contain persistent side effects	Yes		Required	
MISRA25_13.2	13.2 The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders and shall be independent from thread interleaving	Yes		Required	
MISRA25_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	
MISRA25_13.4	13.4 The result	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	of an assignment operator should not be used				
MISRA25_13.5	13.5 The right hand operand of a logical && or operator shall not contain persistent side effects	Yes		Required	
MISRA25_13.6	13.6 The operand of the sizeof operator shall not contain any expression which has potential side effects	Yes		Required	
MISRA25_14.1	14.1 A loop counter shall not have essentially floating type	Yes		Required	
MISRA25_14.2	14.2 A for loop shall be well-formed	Yes		Required	
MISRA25_14.3	14.3 Controlling expressions shall not be invariant	Yes		Required	
MISRA25_14.4	14.4 The controlling expression of an if statement and the controlling expression of	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	an iteration-statement shall have essentially Boolean type				
MISRA25_15.1	15.1 The goto statement should not be used	Yes		Advisory	
MISRA25_15.2	15.2 The goto statement shall jump to a label declared later in the same function	Yes		Required	
MISRA25_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	
MISRA25_15.4	15.4 There should be no more than one break or goto statement used to terminate any iteration statement	Yes		Advisory	
MISRA25_15.5	15.5 A function should have a single point of exit at the end	Yes		Advisory	
MISRA25_15.6	15.6 The body of an iteration-statement or a selection-	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	statement shall be a compound-statement				
MISRA25_15.7	15.7 All if ... else if constructs shall be terminated with an else statement	Yes		Required	
MISRA25_16.1	16.1 All switch statements shall be well-formed	Yes		Required	
MISRA25_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	
MISRA25_16.3	16.3 An unconditional break statement shall terminate every switch-clause	Yes		Required	
MISRA25_16.4	16.4 Every switch statement shall have a default label	Yes		Required	
MISRA25_16.5	16.5 A default label shall appear as either the first	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	or the last switch label of a switch statement				
MISRA25_16.6	16.6 Every switch statement shall have at least two switch-clauses	Yes		Required	
MISRA25_16.7	16.7 A switch-expression shall not have essentially Boolean type	Yes		Required	
MISRA25_17.1	17.1 The features of <stdarg.h> shall not be used	Yes		Required	
MISRA25_17.2	17.2 Functions shall not call themselves, either directly or indirectly	Yes		Required	
MISRA25_17.3	17.3 A function shall not be declared implicitly	Yes		Mandatory	
MISRA25_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		Mandatory	
MISRA25_17.5	17.5 The function argument corresponding	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	to a parameter declared to have an array type shall have an appropriate number of elements				
MISRA25_17.7	17.7 The value returned by a function having non-void return type shall be used	Yes		Required	
MISRA25_17.8	17.8 A function parameter should not be modified	Yes		Advisory	
MISRA25_17.9	17.9 A function declared with a <code>_Noreturn</code> function specifier shall not return to its caller	Yes		Mandatory	
MISRA25_17.10	17.10 A function declared with a <code>_Noreturn</code> function specifier shall have void return type	Yes		Required	
MISRA25_17.11	17.11 A function that never returns should be declared with a <code>_Noreturn</code> function specifier	Yes		Advisory	
MISRA25_17.12	17.12 A	Yes		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
	function identifier should only be used with either a preceding &, or with a parenthesized parameter list				
MISRA25_17.13	17.13 A function type shall not be type qualified	Yes		Required	
MISRA25_18.1	18.1 A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand	Yes		Required	
MISRA25_18.2	18.2 Subtraction between pointers shall only be applied to pointers that address elements of the same array	Yes		Required	
MISRA25_18.3	18.3 The relational operators >, >=, < and <= shall not be applied to expressions of pointer type except where they point into the same	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	object				
MISRA25_18.4	18.4 The +, -, += and -= operators should not be applied to an expression of pointer type	Yes		Advisory	
MISRA25_18.5	18.5 Declarations should contain no more than two levels of pointer nesting	Yes		Advisory	
MISRA25_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	
MISRA25_18.7	18.7 Flexible array members shall not be declared	Yes		Required	
MISRA25_18.8	18.8 Variable-length arrays shall not be used	Yes		Required	
MISRA25_18.9	18.9 An object with temporary lifetime shall not undergo array-to-pointer conversion	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_18.1 0	18.10 Pointers to variably-modified array types shall not be used	Yes		Mandatory	
MISRA25_19.1	19.1 An object shall not be assigned or copied to an overlapping object (Partial)	Yes		Mandatory	
MISRA25_19.2	19.2 The union keyword should not be used	Yes		Advisory	
MISRA25_19.3	19.3 A union member shall not be read unless it has been previously set	Yes		Required	
MISRA25_20.1	20.1 #include directives shall only be preceded by preprocessor directives or comments	Yes		Advisory	
MISRA25_20.2	characters and the /* or // character sequences shall not occur in a header file name	Yes		Required	
MISRA25_20.3	20.3 The #include directive shall be followed by either a <filename> or "filename"	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	sequence				
MISRA25_20.4	20.4 A macro shall not be defined with the same name as a keyword	Yes		Required	
MISRA25_20.5	20.5 #undef should not be used	Yes		Advisory	
MISRA25_20.6	20.6 Tokens that look like a preprocessing directive shall not occur within a macro argument	Yes		Required	
MISRA25_20.7	20.7 Expressions resulting from the expansion of macro parameters shall be appropriately delimited	Yes		Required	
MISRA25_20.8	20.8 The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1	Yes		Required	
MISRA25_20.9	20.9 All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	before evaluation				
MISRA25_20.10	20.10 The # and ## preprocessor operators should not be used	Yes		Advisory	
MISRA25_20.11	20.11 A macro parameter immediately following a # operator shall not immediately be followed by a ## operator	Yes		Required	
MISRA25_20.12	20.12 A macro parameter used as an operand to the # or ## operators, which is itself subject to further macro replacement, shall only be used as an operand to these operators	Yes		Required	
MISRA25_20.13	20.13 A line whose first token is # shall be a valid preprocessing directive	Yes		Required	
MISRA25_20.14	20.14 All #else, #elif and #endif preprocessor	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	directives shall reside in the same file as the #if, #ifdef or #ifndef directive to which they are related				
MISRA25_20.15	20.15 #define and #undef shall not be used on a reserved identifier or reserved macro name	Yes		Required	
MISRA25_21.3	21.3 The memory allocation and deallocation functions of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.4	21.4 The standard header file <setjmp.h> shall not be used	Yes		Required	
MISRA25_21.5	21.5 The standard header file <signal.h> shall not be used	Yes		Required	
MISRA25_21.6	21.6 The Standard Library input/output functions shall not be used	Yes		Required	
MISRA25_21.7	21.7 The atof,	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	atoi, atol and atoll functions of <stdlib.h> shall not be used				
MISRA25_21.8	21.8 The Standard Library termination functions of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.9	21.9 The Standard Library functions bsearch and qsort of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.10	21.10 The Standard Library time and date functions shall not be used	Yes		Required	
MISRA25_21.11	21.11 The standard header file <tgmath.h> shall not be used	Yes		Required	
MISRA25_21.12	21.12 The standard header file <fenv.h> shall not be used	Yes		Required	
MISRA25_21.13	21.13 Any value passed to a function in <ctype.h> shall be	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	representable as an unsigned char or be the value EOF				
MISRA25_21.1 4	21.14 The Standard Library function memcmp shall not be used to compare null terminated strings	Yes		Required	
MISRA25_21.1 5	21.15 The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types	Yes		Required	
MISRA25_21.1 6	21.16 The pointer arguments to the Standard Library function memcmp shall point to either a pointer type, an essentially signed type, an essentially unsigned type, an essentially Boolean type	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	or an essentially enum type				
MISRA25_21.1 7	21.17 Use of the string handling functions from <string.h> shall not result in accesses beyond the bounds of the objects referenced by their pointer parameters	Yes		Mandatory	
MISRA25_21.1 8	21.18 The size_t argument passed to any function in <string.h> shall have an appropriate value	Yes		Mandatory	
MISRA25_21.1 9	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	
MISRA25_21.2 0	21.20 The pointer returned by	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function				
MISRA25_21.2 1	21.21 The Standard Library function system of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.2 2	21.22 All operand arguments to any type-generic macros declared in <tgmath.h> shall have an appropriate essential type	Yes		Mandatory	
MISRA25_21.2 3	21.23 All operand arguments to any multi-argument type-generic macros declared in <tgmath.h> shall have the	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	same standard type				
MISRA25_21.24	21.24 The random number generator functions of <stdlib.h> shall not be used	Yes		Required	
MISRA25_21.25	21.25 All memory synchronization operations shall be executed in sequentially consistent order	Yes		Required	
MISRA25_21.26	21.26 The Standard Library function mtx_timedlock () shall only be invoked on mutex objects of appropriate mutex type	Yes		Required	
MISRA25_22.1	22.1 All resources obtained dynamically by means of Standard Library functions shall be explicitly released	Yes		Required	
MISRA25_22.2	22.2 A block of memory shall only be freed if it was	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	allocated by means of a Standard Library function				
MISRA25_22.3	22.3 The same file shall not be open for read and write access at the same time on different streams	Yes		Required	
MISRA25_22.4	22.4 There shall be no attempt to write to a stream which has been opened as read-only	Yes		Mandatory	
MISRA25_22.5	22.5 A pointer to a FILE object shall not be dereferenced	Yes		Mandatory	
MISRA25_22.6	22.6 The value of a pointer to a FILE shall not be used after the associated stream has been closed (Partial)	Yes		Mandatory	
MISRA25_22.7	22.7 The macro EOF shall only be compared with the unmodified return value from any Standard	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	Library function capable of returning EOF				
MISRA25_22.8	22.8 The value of errno shall be set to zero prior to a call to an errno-setting-function	Yes		Required	
MISRA25_22.9	22.9 The value of errno shall be tested against zero after calling an errno-setting-function	Yes		Required	
MISRA25_22.10	22.10 The value of errno shall only be tested when the last function to be called was an errno-setting-function	Yes		Required	
MISRA25_22.11	22.11 A thread that was previously either joined or detached shall not be subsequently joined nor detached	Yes		Required	
MISRA25_22.12	22.12 Thread objects, thread synchronization objects, and thread-specific storage	Yes		Mandatory	

Check ID	Check Name	Supported	Automation	Category	Severity
	pointers shall only be accessed by the appropriate Standard Library functions				
MISRA25_22.1 3	22.13 Thread objects, thread synchronization objects and thread-specific storage pointers shall have appropriate storage duration	Yes		Required	
MISRA25_22.1 4	22.14 Thread synchronization objects shall be initialized before being accessed	No		Mandatory	
MISRA25_22.1 5	22.15 Thread synchronization objects and thread-specific storage pointers shall not be destroyed until after all threads accessing them have terminated	Yes		Required	
MISRA25_22.1 6	22.16 All mutex objects locked by a thread shall be explicitly unlocked by	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	the same thread				
MISRA25_22.17	22.17 No thread shall unlock a mutex or call cnd_wait() or cnd_timedwait() for a mutex it has not locked before	Yes		Required	
MISRA25_22.18	22.18 Non-recursive mutexes shall not be recursively locked	Yes		Required	
MISRA25_22.19	22.19 A condition variable shall be associated with at most one mutex object	Yes		Required	
MISRA25_22.20	22.20 Thread-specific storage pointers shall be created before being accessed	No		Mandatory	
MISRA25_23.1	23.1 A generic selection should only be expanded from a macro	Yes		Advisory	
MISRA25_23.2	23.2 A generic selection that is not expanded from a macro shall not contain	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	potential side effects in the controlling expression				
MISRA25_23.3	23.3 A generic selection should contain at least one non-default association	Yes		Advisory	
MISRA25_23.4	23.4 A generic association shall list an appropriate type	No		Required	
MISRA25_23.5	23.5 A generic selection should not depend on implicit pointer type conversion	No		Advisory	
MISRA25_23.6	23.6 The controlling expression of a generic selection shall have an essential type that matches its standard type	No		Required	
MISRA25_23.7	23.7 A generic selection that is expanded from a macro should evaluate its argument only once	Yes		Advisory	
MISRA25_23.8	23.8 A default association	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
	shall appear as either the first or the last association of a generic selection				
MISRA25_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	
MISRA25_DIR_1.2	Directive 1.2 The use of language extensions should be minimized (Partial)	Yes		Advisory	
MISRA25_DIR_2.1	Directive 2.1 All source files shall compile without any compilation errors	Yes		Required	
MISRA25_DIR_3.1	Directive 3.1 All code shall be traceable to documented requirements	No		Required	
MISRA25_DIR_4.1	Directive 4.1 Run-time failures shall be minimized	No		Required	
MISRA25_DIR_4.2	Directive 4.2	No		Advisory	

Check ID	Check Name	Supported	Automation	Category	Severity
4.2	All usage of assembly language should be documented				
MISRA25_DIR_4.3	Directive 4.3 Assembly language shall be encapsulated and isolated	No		Required	
MISRA25_DIR_4.4	Directive 4.4 Sections of code should not be "commented out"	Yes		Advisory	
MISRA25_DIR_4.5	Directive 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous	Yes		Advisory	
MISRA25_DIR_4.6	Directive 4.6 typedefs that indicate size and signedness should be used in place of the basic integer types	Yes		Advisory	
MISRA25_DIR_4.7	Directive 4.7 If a function returns error information, then that error information shall be tested	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_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	
MISRA25_DIR_4.9	Directive 4.9 A function should be used in preference to a function-like macro where they are interchangeable	Yes		Advisory	
MISRA25_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	
MISRA25_DIR_4.11	Directive 4.11 The validity of values passed to library functions shall be checked	Yes		Required	
MISRA25_DIR_4.12	Directive 4.12 Dynamic memory allocation shall not be used	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MISRA25_DIR_4.13	Directive 4.13 Functions which are designed to provide operations on a resource should be called in an appropriate sequence (Partial)	Yes		Advisory	
MISRA25_DIR_4.14	Directive 4.14 The validity of values received from external sources shall be checked	Yes		Required	
MISRA25_DIR_4.15	Directive 4.15 Evaluation of floating-point expressions shall not lead to the undetected generation of infinities and NaNs	No		Required	
MISRA25_DIR_5.1	Directive 5.1 There shall be no data races between threads	No		Required	
MISRA25_DIR_5.2	Directive 5.2 There shall be no deadlocks between threads	No		Required	
MISRA25_DIR_5.3	Directive 5.3 There shall be no dynamic	Yes		Required	

Check ID	Check Name	Supported	Automation	Category	Severity
MSC30-C	thread creation 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 implemented as a macro	Yes			Low
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
MSC51-CPP	Ensure your random number generator is properly	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
OOP55-CPP	seeded Do not use pointer-to-member operators to access nonexistent members	No			High
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 relinquishment 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	Yes			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
	transferring data between systems				
POS44-C	Do not use signals to terminate threads	Yes			Low
POS47-C	Do not use threads that can be canceled asynchronously	Yes			Medium
POS48-C	Do not unlock or destroy another POSIX thread's mutex	Yes			Medium
POS49-C	When data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed	No			Medium
POS50-C	Declare objects shared between POSIX threads with appropriate storage durations	Yes			Medium
POS51-C	Avoid deadlock with POSIX threads by locking in predefined order	Yes			Low

Check ID	Check Name	Supported	Automation	Category	Severity
POS52-C	Do not perform operations that can block while holding a POSIX lock	Yes			Low
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 and Return Values - Unchecked	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
	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			
RECOMMENDED_01	Definitions in Header Files	Yes			
RECOMMENDED_02	Files too long	Yes			
RECOMMENDED_03	Floating Equality Test	Yes			
RECOMMENDED_04	Functions Too Long	Yes			
RECOMMENDED	Functions shall	Yes			

Check ID	Check Name	Supported	Automation	Category	Severity
D_05	not be declared implicitly				
RECOMMENDED	Goto Statements	Yes			
D_06	Macros shall not be #define'd or #undef'd within a block	Yes			
RECOMMENDED	Magic Numbers	Yes			
D_08	Nested Comments	Yes			
RECOMMENDED	Overly Complex Functions	Yes			
D_09	Trigraphs shall not be used	Yes			
RECOMMENDED	Unreachable Code	Yes			
D_10	Unused Functions	Yes			
RECOMMENDED	Unused C and C++ Local Variables	Yes			
D_11	Unused Static Globals	Yes			
RECOMMENDED	Variables should be commented	Yes			
D_12	Upper limit shall not be modified within the bounds of the loop	Yes			
RECOMMENDED	Comments Indicating Future Fixes	Yes			
D_13	Duplicate Code	Yes			
RECOMMENDED					
D_14					
RECOMMENDED					
D_15					
RECOMMENDED					
D_16					
RECOMMENDED					
D_17					
RECOMMENDED					
D_18					
RECOMMENDED					
D_19					
RECOMMENDED					
D_20					

Check ID	Check Name	Supported	Automation	Category	Severity
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 literals	Yes			Low
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	No			Medium

Check ID	Check Name	Supported	Automation	Category	Severity
STR37-C	larger integer sizes 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 has sufficient space for character data and the null terminator	Yes			High
WIN30-C	Properly pair allocation and deallocation functions	Yes			Low