Axivion Bauhaus Suite – Technical Factsheet MISRA

Version 6.5 upwards

Contents

1. C

1. MISRA C 2004

MISRA -Rule	Severity	Description
1.1	Req	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
1.2	Req	No reliance shall be placed on undefined or unspecified behaviour.
2.1	Req	Assembly language shall be encapsulated and isolated.
2.2	Req	Source code shall only use /* ... */ style comments.
2.3	Req	The character sequence /* shall not be used within a comment.
2.4	Adv	Sections of code should not be “commented out”.
3.1	Req	All usage of implementation-defined behaviour shall be documented.
3.4	Req	All uses of the #pragma directive shall be documented and explained.
3.6	Req	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.
4.1	Req	Only those escape sequences that are defined in the ISO C standard shall be used.
4.2	Req	Trigraphs shall not be used.
5.1	Req	Identifiers (internal and external) shall not rely on the significance of more than 31 characters.
5.2	Req	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.
5.3	Req	A typedef name shall be a unique identifier.
5.4	Req	A tag name shall be a unique identifier.
5.5	Adv	No object or function identifier with static storage duration should be reused.

5.6	Adv	No identifier in one name space should have the same spelling as an identifier in another name space, with the exception of structure member and union member names.
5.7	Adv	No identifier name should be reused.
6.1	Req	The plain char type shall be used only for storage and use of character values.
6.2	Req	signed and unsigned char type shall be used only for the storage and use of numeric values.
6.3	Adv	typedefs that indicate size and signedness should be used in place of the basic numerical types.
6.4	Req	Bit fields shall only be defined to be of type unsigned int or signed int.
6.5	Req	Bit fields of signed type shall be at least 2 bits long.
7.1	Req	Octal constants (other than zero) and octal escape sequences shall not be used.
8.1	Req	Functions shall have prototype declarations and the prototype shall be visible at both the function definition and call.
8.2	Req	Whenever an object or function is declared or defined, its type shall be explicitly stated.
8.3	Req	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical.
8.4	Req	If objects or functions are declared more than once their types shall be compatible.
8.5	Req	There shall be no definitions of objects or functions in a header file.
8.6	Req	Functions shall be declared at file scope.
8.7	Req	Objects shall be defined at block scope if they are only accessed from within a single function.
8.8	Req	An external object or function shall be declared in one and only one file.
8.9	Req	An identifier with external linkage shall have exactly one external definition.

8.10	Req	All declarations and definitions of objects or functions at file scope shall have internal linkage unless external linkage is required.
8.11	Req	The static storage class specifier shall be used in definitions and declarations of objects and functions that have internal linkage.
8.12	Req	When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialisation.
9.1	Req	All automatic variables shall have been assigned a value before being used.
9.2	Req	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures.
9.3	Req	In an enumerator list, the “=” construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised.
10.1	Req	The value of an expression of integer type shall not be implicitly converted to a different underlying type if: (a) it is not a conversion to a wider integer type of the same signedness, or (b) the expression is complex, or (c) the expression is not constant and is a function argument, or (d) the expression is not constant and is a return expression
10.2	Req	The value of an expression of floating type shall not be implicitly converted to a different type if: (a) it is not a conversion to a wider floating type, or (b) the expression is complex, or (c) the expression is a function argument, or (d) the expression is a return expression
10.3	Req	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.
10.4	Req	The value of a complex expression of floating type shall only be cast to a floating type that is narrower or of the same size.
10.5	Req	If the bitwise operators ~ and << are applied to an operand of underlying type unsigned char or unsigned short, the result shall be immediately cast to the underlying type of the operand.
10.6	Req	A “U” suffix shall be applied to all constants of unsigned type.
11.1	Req	Conversions shall not be performed between a pointer to a function and any type other than an integral type.
11.2	Req	Conversions shall not be performed between a pointer to object and any type other than an integral type, another pointer to object type or a pointer to void.

11.3	Adv	A cast should not be performed between a pointer type and an integral type.
11.4	Adv	A cast should not be performed between a pointer to object type and a different pointer to object type.
11.5	Req	A cast shall not be performed that removes any const or volatile qualification from the type addressed by a pointer.
12.1	Adv	Limited dependence should be placed on C's operator precedence rules in expressions.
12.2	Req	The value of an expression shall be the same under any order of evaluation that the standard permits.
12.3	Req	The sizeof operator shall not be used on expressions that contain side effects.
12.4	Req	The right-hand operand of a logical && or || operator shall not contain side effects.
12.5	Req	The operands of a logical && or || shall be primary-expressions.
12.6	Adv	The operands of logical operators (&&, || and !) should be effectively Boolean. Expressions that are effectively Boolean should not be used as operands to operators other than (&&, ||, !, =, ==, != and ?:).
12.7	Req	Bitwise operators shall not be applied to operands whose underlying type is signed.
12.8	Req	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.
12.9	Req	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
12.10	Req	The comma operator shall not be used.
12.11	Adv	Evaluation of constant unsigned integer expressions should not lead to wrap-around.
12.12	Req	The underlying bit representations of floating-point values shall not be used.
12.13	Adv	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.
13.1	Req	Assignment operators shall not be used in expressions that yield a Boolean value.

13.2	Adv	Tests of a value against zero should be made explicit, unless the operand is effectively Boolean.
13.3	Req	Floating-point expressions shall not be tested for equality or inequality.
13.4	Req	The controlling expression of a for statement shall not contain any objects of floating type.
13.5	Req	The three expressions of a for statement shall be concerned only with loop control.
13.6	Req	Numeric variables being used within a for loop for iteration counting shall not be modified in the body of the loop.
13.7	Req	Boolean operations whose results are invariant shall not be permitted.
14.1	Req	There shall be no unreachable code.
14.2	Req	All non-null statements shall either (a) have at least one side-effect however executed, or (b) cause control flow to change.
14.3	Req	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.
14.4	Req	The goto statement shall not be used.
14.5	Req	The continue statement shall not be used.
14.6	Req	For any iteration statement there shall be at most one break statement used for loop termination.
14.7	Req	A function shall have a single point of exit at the end of the function.
14.8	Req	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.
14.9	Req	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.
14.10	Req	All if ... else if constructs shall be terminated with an else clause.
15.0	Req	The MISRA C switch syntax shall be used.
15.1	Req	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.

15.2	Req	An unconditional break statement shall terminate every non-empty switch clause.
15.3	Req	The final clause of a switch statement shall be the default clause.
15.4	Req	A switch expression shall not represent a value that is effectively Boolean.
15.5	Req	Every switch statement shall have at least one case clause.
16.1	Req	Functions shall not be defined with variable numbers of arguments.
16.2	Req	Functions shall not call themselves, either directly or indirectly.
16.3	Req	Identifiers shall be given for all of the parameters in a function prototype declaration.
16.4	Req	The identifiers used in the declaration and definition of a function shall be identical.
16.5	Req	Functions with no parameters shall be declared and defined with the parameter list void.
16.6	Req	The number of arguments passed to a function shall match the number of parameters.
16.7	Adv	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.
16.8	Req	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.
16.9	Req	A function identifier shall only be used with either a preceding &, or with a parenthesised parameter list, which may be empty.
16.10	Req	If a function returns error information, then that error information shall be tested.
17.1	Req	Pointer arithmetic shall only be applied to pointers that address an array or array element.
17.2	Req	Pointer subtraction shall only be applied to pointers that address elements of the same array.
17.3	Req	>, >=, <, <= shall not be applied to pointer types except where they point to the same array.
17.4	Req	Array indexing shall be the only allowed form of pointer arithmetic.

17.5	Adv	The declaration of objects should contain no more than 2 levels of pointer indirection.
17.6	Req	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.
18.1	Req	All structure or union types shall be complete at the end of a translation unit.
18.2	Req	An object shall not be assigned to an overlapping object.
18.4	Req	Unions shall not be used.
19.1	Adv	#include statements in a file should only be preceded by other preprocessor directives or comments.
19.2	Adv	Non-standard characters should not occur in header file names in #include directives.
19.3	Req	The #include directive shall be followed by either a <filename> or "filename" sequence
19.4	Req	C macros shall only expand to a braced initialiser, a constant, a string literal, a parenthesised expression, a type qualifier, a storage class class specifier, or a do-while-zero construct.
19.5	Req	Macros shall not be #define'd or #undef'd within a block.
19.6	Req	#undef shall not be used.
19.7	Adv	A function should be used in preference to a function-like macro.
19.8	Req	A function-like macro shall not be invoked without all of its arguments.
19.9	Req	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
19.10	Req	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 ##.
19.11	Req	All macro identifiers in preprocessor directives shall be defined before use, except in #ifdef and #ifndef preprocessor directives and the defined() operator.
19.12	Req	There shall be at most one occurrence of the # or ## preprocessor operators in a single macro definition.

19.13	Adv	The # and ## preprocessor operators should not be used.
19.14	Req	The defined preprocessor operator shall only be used in one of the two standard forms.
19.15	Req	Precautions shall be taken in order to prevent the contents of a header file being included twice.
19.16	Req	Preprocessing directives shall be syntactically meaningful even when excluded by the preprocessor.
19.17	Req	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
20.1	Req	Reserved identifiers, macros and functions in the standard library, shall not be defined, redefined or undefined.
20.2	Req	The names of standard library macros, objects and functions shall not be reused.
20.3	Req	The validity of values passed to library functions shall be checked.
20.4	Req	Dynamic heap memory allocation shall not be used.
20.5	Req	The error indicator errno shall not be used.
20.6	Req	The macro offsetof, in library <stddef.h>, shall not be used.
20.7	Req	The setjmp macro and the longjmp function shall not be used.
20.8	Req	The signal handling facilities of <signal.h> shall not be used.
20.9	Req	The input/output library <stdio.h> shall not be used in production code.
20.10	Req	The library functions atof, atoi and atol from library <stdlib.h> shall not be used.
20.11	Req	The library functions abort, exit, getenv and system from library <stdlib.h> shall not be used.
20.12	Req	The time handling functions of library <time.h> shall not be used.
21.1	Req	Minimisation of run-time failures shall be ensured by the use of at least one of (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.

2. MISRA C 2012 (incl. Amendment 1)

MISRA-Directive/Rule	Severity	Description
D1.1	Req	Any implementation-defined behaviour on which the output of the program depends shall be documented and understood
D2.1	Req	All source files shall compile without any compilation errors
D4.1	Req	Run-time failures shall be minimized
D4.2	Adv	All usage of assembly language should be documented
D4.3	Req	Assembly language shall be encapsulated and isolated
D4.4	Adv	Sections of code should not be “commented out”
D4.5	Adv	Identifiers in the same name space with overlapping visibility should be typographically unambiguous
D4.6	Adv	typedefs that indicate size and signedness should be used in place of the basic numerical types
D4.7	Req	If a function returns error information, then that error information shall be tested
D4.8	Adv	If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden
D4.9	Adv	A function should be used in preference to a function-like macro where they are interchangeable
D4.10	Req	Precautions shall be taken in order to prevent the contents of a header file being included more than once
D4.11	Req	The validity of values passed to library functions shall be checked
D4.12	Req	Dynamic memory allocation shall not be used
D4.13	Adv	Functions which are designed to provide operations on a resource should be called in an appropriate sequence
1.1	Req	The program shall contain no violations of the standard C syntax and constraints, and shall not exceed the implementation’s translation limits
1.2	Adv	Language extensions should not be used

1.3	Req	There shall be no occurrence of undefined or critical unspecified behaviour
2.1	Req	A project shall not contain unreachable code
2.2	Req	There shall be no dead code
2.3	Adv	A project should not contain unused type declarations
2.4	Adv	A project should not contain unused tag declarations
2.5	Adv	A project should not contain unused macro declarations
2.6	Adv	A function should not contain unused label declarations
2.7	Adv	There should be no unused parameters in functions
3.1	Req	The character sequences /* and // shall not be used within a comment
3.2	Req	Line-splicing shall not be used in // comments
4.1	Req	Octal and hexadecimal escape sequences shall be terminated
4.2	Adv	Trigraphs should not be used
5.1	Req	External identifiers shall be distinct
5.2	Req	Identifiers declared in the same scope and name space shall be distinct
5.3	Req	An identifier declared in an inner scope shall not hide an identifier declared in an outer scope
5.4	Req	Macro identifiers shall be distinct
5.5	Req	Identifiers shall be distinct from macro names
5.6	Req	A typedef name shall be a unique identifier
5.7	Req	A tag name shall be a unique identifier
5.8	Req	Identifiers that define objects or functions with external linkage shall be unique
5.9	Adv	Identifiers that define objects or functions with internal linkage should be unique
6.1	Req	Bit-fields shall only be declared with an appropriate type

6.2	Req	Single-bit named bit fields shall not be of a signed type
7.1	Req	Octal constants shall not be used
7.2	Req	A “u” or “U” suffix shall be applied to all integer constants that are represented in an unsigned type
7.3	Req	The lowercase character “l” shall not be used in a literal suffix
7.4	Req	A string literal shall not be assigned to an object unless the object’s type is “pointer to const-qualified char”
8.1	Req	Types shall be explicitly specified
8.2	Req	Function types shall be in prototype form with named parameters
8.3	Req	All declarations of an object or function shall use the same names and type qualifiers
8.4	Req	A compatible declaration shall be visible when an object or function with external linkage is defined
8.5	Req	An external object or function shall be declared once in one and only one file
8.6	Req	An identifier with external linkage shall have exactly one external definition
8.7	Adv	Functions and objects should not be defined with external linkage if they are referenced in only one translation unit
8.8	Req	The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage
8.9	Adv	An object should be defined at block scope if its identifier only appears in a single function
8.10	Req	An inline function shall be declared with the static storage class
8.11	Adv	When an array with external linkage is declared, its size should be explicitly specified
8.12	Req	Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique
8.13	Adv	A pointer should point to a const-qualified type whenever possible
8.14	Req	The restrict type qualifier shall not be used

9.1	Man	The value of an object with automatic storage duration shall not be read before it has been set
9.2	Req	The initializer for an aggregate or union shall be enclosed in braces
9.3	Req	Arrays shall not be partially initialized
9.4	Req	An element of an object shall not be initialized more than once
9.5	Req	Where designated initializers are used to initialize an array object the size of the array shall be specified explicitly
10.1	Req	Operands shall not be of an inappropriate essential type
10.2	Req	Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations
10.3	Req	The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category
10.4	Req	Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category
10.5	Adv	The value of an expression should not be cast to an inappropriate essential type
10.6	Req	The value of a composite expression shall not be assigned to an object with wider essential type
10.7	Req	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
10.8	Req	The value of a composite expression shall not be cast to a different essential type category or a wider essential type
11.1	Req	Conversions shall not be performed between a pointer to a function and any other type
11.2	Req	Conversions shall not be performed between a pointer to an incomplete type and any other type
11.3	Req	A cast shall not be performed between a pointer to object type and a pointer to a different object type
11.4	Adv	A conversion should not be performed between a pointer to object and an integer type

11.5	Adv	A conversion should not be performed from pointer to void into pointer to object
11.6	Req	A cast shall not be performed between pointer to void and an arithmetic type
11.7	Req	A cast shall not be performed between pointer to object and a non-integer arithmetic type
11.8	Req	A cast shall not remove any const or volatile qualification from the type pointed to by a pointer
11.9	Req	The macro NULL shall be the only permitted form of integer null pointer constant
12.1	Adv	The precedence of operators within expressions should be made explicit
12.2	Req	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
12.3	Adv	The comma operator should not be used
12.4	Adv	Evaluation of constant expressions should not lead to unsigned integer wrap-around
12.5	Man	The sizeof operator shall not have an operand which is a function parameter declared as "array of type"
13.1	Req	Initializer lists shall not contain persistent side effects
13.2	Req	The value of an expression and its persistent side effects shall be the same under all permitted evaluation orders
13.3	Adv	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
13.4	Adv	The result of an assignment operator should not be used
13.5	Req	The right hand operand of a logical && or || operator shall not contain persistent side effects
13.6	Man	The operand of the sizeof operator shall not contain any expression which has potential side effects
14.1	Req	A loop counter shall not have essentially floating type
14.2	Req	A for loop shall be well-formed

14.3	Req	Controlling expressions shall not be invariant
14.4	Req	The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type
15.1	Adv	The goto statement should not be used
15.2	Req	The goto statement shall jump to a label declared later in the same function
15.3	Req	Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement
15.4	Adv	There should be no more than one break or goto statement used to terminate any iteration statement
15.5	Adv	A function should have a single point of exit at the end
15.6	Req	The body of an iteration-statement or a selection-statement shall be a compound-statement
15.7	Req	All if ... else if constructs shall be terminated with an else statement
16.1	Req	All switch statements shall be well-formed
16.2	Req	A switch label shall only be used when the most closely-enclosing compound statement is the body of a switch statement
16.3	Req	An unconditional break statement shall terminate every switch-clause
16.4	Req	Every switch statement shall have a default label
16.5	Req	A default label shall appear as either the first or the last switch label of a switch statement
16.6	Req	Every switch statement shall have at least two switch-clauses
16.7	Req	A switch-expression shall not have essentially Boolean type
17.1	Req	The features of <stdarg.h> shall not be used
17.2	Req	Functions shall not call themselves, either directly or indirectly
17.3	Man	A function shall not be declared implicitly
17.4	Man	All exit paths from a function with non-void return type shall have an explicit return statement with an expression

17.5	Adv	The function argument corresponding to a parameter declared to have an array type shall have an appropriate number of elements
17.6	Man	The declaration of an array parameter shall not contain the static keyword between the [ ]
17.7	Req	The value returned by a function having non-void return type shall be used
17.8	Adv	A function parameter should not be modified
18.1	Req	A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand
18.2	Req	Subtraction between pointers shall only be applied to pointers that address elements of the same array
18.3	Req	The relational operators >, >=, < and <= shall not be applied to objects of pointer type except where they point into the same object
18.4	Adv	The +, -, += and -= operators should not be applied to an expression of pointer type
18.5	Adv	Declarations should contain no more than two levels of pointer nesting
18.6	Req	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
18.7	Req	Flexible array members shall not be declared
18.8	Req	Variable-length array types shall not be used
19.1	Man	An object shall not be assigned or copied to an overlapping object
19.2	Adv	The union keyword should not be used
20.1	Adv	#include directives should only be preceded by preprocessor directives or comments
20.2	Req	The ', " or \ characters and the /* or // character sequences shall not occur in a header file name
20.3	Req	The #include directive shall be followed by either a <filename> or "filename" sequence
20.4	Req	A macro shall not be defined with the same name as a keyword
20.5	Adv	#undef should not be used

20.6	Req	Tokens that look like a preprocessing directive shall not occur within a macro argument
20.7	Req	Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses
20.8	Req	The controlling expression of a #if or #elif preprocessing directive shall evaluate to 0 or 1
20.9	Req	All identifiers used in the controlling expression of #if or #elif preprocessing directives shall be #define'd before evaluation
20.10	Adv	The # and ## preprocessor operators should not be used
20.11	Req	A macro parameter immediately following a # operator shall not immediately be followed by a ## operator
20.12	Req	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
20.13	Req	A line whose first token is # shall be a valid preprocessing directive
20.14	Req	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
21.1	Req	#define and #undef shall not be used on a reserved identifier or reserved macro name
21.2	Req	A reserved identifier or macro name shall not be declared
21.3	Req	The memory allocation and deallocation functions of <stdlib.h> shall not be used
21.4	Req	The standard header file <setjmp.h> shall not be used
21.5	Req	The standard header file <signal.h> shall not be used
21.6	Req	The Standard Library input/output functions shall not be used
21.7	Req	The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used
21.8	Req	The library functions abort, exit, getenv and system of <stdlib.h> shall not be used
21.9	Req	The library functions bsearch and qsort of <stdlib.h> shall not be used
21.10	Req	The Standard Library time and date functions shall not be used

21.11	Req	The standard header file <tgmath.h> shall not be used
21.12	Adv	The exception handling features of <fenv.h> should not be used
21.14	Req	The Standard Library function memcmp shall not be used to compare null terminated strings
21.15	Req	The pointer arguments to the Standard Library functions memcpy, memmove and memcmp shall be pointers to qualified or unqualified versions of compatible types
21.16	Req	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 or an essentially enum type
21.19	Man	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
21.20	Man	The pointer returned by the Standard Library functions asctime, ctime, gmtime, localtime, localeconv, getenv, setlocale or strerror shall not be used following a subsequent call to the same function
22.1	Req	All resources obtained dynamically by means of Standard Library functions shall be explicitly released
22.2	Man	A block of memory shall only be freed if it was allocated by means of a Standard Library function
22.3	Req	The same file shall not be open for read and write access at the same time on different streams
22.4	Man	There shall be no attempt to write to a stream which has been opened as read-only
22.5	Man	A pointer to a FILE object shall not be dereferenced
22.6	Man	The value of a pointer to a FILE shall not be used after the associated stream has been closed
22.7	Req	The macro EOF shall only be compared with the unmodified return value from any Standard Library function capable of returning EOF
22.8	Req	The value of errno shall be set to zero prior to a call to an errno-setting-function
22.9	Req	The value of errno shall be tested against zero after calling an errno-setting-function

22.10	Req	The value of errno shall only be tested when the last function to be called was an errno-setting-function

2. C++

1. MISRA C++ 2008

MISRA -Rule	Severity	Description
0.1.1	Req	A project shall not contain unreachable code.
0.1.2	Req	A project shall not contain infeasible paths.
0.1.3	Req	A project shall not contain unused variables.
0.1.4	Req	A project shall not contain non-volatile POD variables having only one use.
0.1.5	Req	A project shall not contain unused type declarations.
0.1.6	Req	A project shall not contain instances of non-volatile variables being given values that are never subsequently used.
0.1.7	Req	The value returned by a function having a non-void return type that is not an overloaded operator shall always be used.
0.1.8	Req	All functions with void return type shall have external side effect(s).
0.1.9	Req	There shall be no dead code.
0.1.10	Req	Every defined function shall be called at least once.
0.1.11	Req	There shall be no unused parameters (named or unnamed) in non-virtual functions.
0.1.12	Req	There shall be no unused parameters (named or unnamed) in the set of parameters for a virtual function and all the functions that override it.
0.2.1	Req	An object shall not be assigned to an overlapping object.
0.3.1	Doc	Minimization of run-time failures shall be ensured by the use of at least one of: (a) static analysis tools/techniques; (b) dynamic analysis tools/techniques; (c) explicit coding of checks to handle run-time faults.

0.3.2	Req	If a function generates error information, then that error information shall be tested.
0.4.2	Doc	Use of floating-point arithmetic shall be documented.
1.0.1	Req	All code shall conform to ISO/IEC 14882:2003 "The C++ Standard Incorporating Technical Corrigendum 1".
2.3.1	Req	Trigraphs shall not be used.
2.5.1	Adv	Digraphs should not be used.
2.7.1	Req	The character sequence /* shall not be used within a C-style comment.
2.7.2	Req	Sections of code shall not be “commented out” using C-style comments.
2.7.3	Adv	Sections of code should not be “commented out” using C++ comments.
2.10.1	Req	Different identifiers shall be typographically unambiguous.
2.10.2	Req	Identifiers declared in an inner scope shall not hide an identifier declared in an outer scope.
2.10.3	Req	A typedef name (including qualification, if any) shall be a unique identifier.
2.10.4	Req	A class, union or enum name (including qualification, if any) shall be a unique identifier.
2.10.5	Adv	The identifier name of a non-member object or function with static storage duration should not be reused.
2.10.6	Req	If an identifier refers to a type, it shall not also refer to an object or a function in the same scope.
2.13.1	Req	Only those escape sequences that are defined in ISO/IEC 14882:2003 shall be used.
2.13.2	Req	Octal constants (other than zero) and octal escape sequences (other than “\0”) shall not be used.
2.13.3	Req	A “U ” suffix shall be applied to all octal or hexadecimal integer literals of unsigned type.
2.13.4	Req	Literal suffixes shall be upper case.
2.13.5	Req	Narrow and wide string literals shall not be concatenated.

3.1.1	Req	It shall be possible to include any header file in multiple translation units without violating the One Definition Rule.
3.1.2	Req	Functions shall not be declared at block scope.
3.1.3	Req	When an array is declared, its size shall either be stated explicitly or defined implicitly by initialization.
3.2.1	Req	All declarations of an object or function shall have compatible types.
3.2.2	Req	The One Definition Rule shall not be violated.
3.2.3	Req	A type, object or function that is used in multiple translation units shall be declared in one and only one file.
3.2.4	Req	An identifier with external linkage shall have exactly one definition.
3.3.1	Req	Objects or functions with external linkage shall be declared in a header file.
3.3.2	Req	If a function has internal linkage then all re-declarations shall include the static storage class specifier.
3.4.1	Req	An identifier declared to be an object or type shall be defined in a block that minimizes its visibility.
3.9.1	Req	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.
3.9.2	Adv	typedefs that indicate size and signedness should be used in place of the basic numerical types.
3.9.3	Req	The underlying bit representations of floating-point values shall not be used.
4.5.1	Req	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.
4.5.2	Req	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 <, <=, >, >=.
4.5.3	Req	Expressions with type (plain) char and wchar_t shall not be used as operands to built-in operators other than the assignment operator =, the equality operators == and !=, and the unary & operator.

4.10.1	Req	NULL shall not be used as an integer value.
4.10.2	Req	Literal zero (0) shall not be used as the null-pointer-constant.
5.0.1	Req	The value of an expression shall be the same under any order of evaluation that the standard permits.
5.0.2	Adv	Limited dependence should be placed on C++ operator precedence rules in expressions.
5.0.3	Req	A cvalue expression shall not be implicitly converted to a different underlying type.
5.0.4	Req	An implicit integral conversion shall not change the signedness of the underlying type.
5.0.5	Req	There shall be no implicit floating-integral conversions.
5.0.6	Req	An implicit integral or floating-point conversion shall not reduce the size of the underlying type.
5.0.7	Req	There shall be no explicit floating-integral conversions of a cvalue expression.
5.0.8	Req	An explicit integral or floating-point conversion shall not increase the size of the underlying type of a cvalue expression.
5.0.9	Req	An explicit integral conversion shall not change the signedness of the underlying type of a cvalue expression.
5.0.10	Req	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.
5.0.11	Req	The plain char type shall only be used for the storage and use of character values.
5.0.12	Req	signed char and unsigned char type shall only be used for the storage and use of numeric values.
5.0.13	Req	The condition of an if-statement and the condition of an iteration-statement shall have type bool.
5.0.14	Req	The first operand of a conditional-operator shall have type bool.
5.0.15	Req	Array indexing shall be the only form of pointer arithmetic.
5.0.16	Req	A pointer operand and any pointer resulting from pointer arithmetic using that operand shall both address elements of the same array.

5.0.17	Req	Subtraction between pointers shall only be applied to pointers that address elements of the same array.
5.0.18	Req	>, >=, <, <= shall not be applied to objects of pointer type, except where they point to the same array.
5.0.19	Req	The declaration of objects shall contain no more than two levels of pointer indirection.
5.0.20	Req	Non-constant operands to a binary bitwise operator shall have the same underlying type.
5.0.21	Req	Bitwise operators shall only be applied to operands of unsigned underlying type.
5.2.1	Req	Each operand of a logical && or || shall be a postfix-expression.
5.2.2	Req	A pointer to a virtual base class shall only be cast to a pointer to a derived class by means of dynamic_cast.
5.2.3	Adv	Casts from a base class to a derived class should not be performed on polymorphic types.
5.2.4	Req	C-style casts (other than void casts) and functional notation casts (other than explicit constructor calls) shall not be used.
5.2.5	Req	A cast shall not remove any const or volatile qualification from the type of a pointer or reference.
5.2.6	Req	A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type.
5.2.7	Req	An object with pointer type shall not be converted to an unrelated pointer type, either directly or indirectly.
5.2.8	Req	An object with integer type or pointer to void type shall not be converted to an object with pointer type.
5.2.9	Adv	A cast should not convert a pointer type to an integral type.
5.2.10	Adv	The increment (++) and decrement (--) operators should not be mixed with other operators in an expression.
5.2.11	Req	The comma operator, && operator and the || operator shall not be overloaded.
5.2.12	Req	An identifier with array type passed as a function argument shall not decay to a pointer.

5.3.1	Req	Each operand of the ! operator, the logical && or the logical || operators shall have type bool.
5.3.2	Req	The unary minus operator shall not be applied to an expression whose underlying type is unsigned.
5.3.3	Req	The unary & operator shall not be overloaded.
5.3.4	Req	Evaluation of the operand to the sizeof operator shall not contain side effects.
5.8.1	Req	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.
5.14.1	Req	The right hand operand of a logical && or || operator shall not contain side effects.
5.17.1	Req	The semantic equivalence between a binary operator and its assignment operator form shall be preserved.
5.18.1	Req	The comma operator shall not be used.
5.19.1	Adv	Evaluation of constant unsigned integer expressions should not lead to wrap-around.
6.2.1	Req	Assignment operators shall not be used in sub-expressions.
6.2.2	Req	Floating-point expressions shall not be directly or indirectly tested for equality or inequality.
6.2.3	Req	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.
6.3.1	Req	The statement forming the body of a switch, while, do ... while or for statement shall be a compound statement.
6.4.1	Req	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.
6.4.2	Req	All if … else if constructs shall be terminated with an else clause.
6.4.3	Req	A switch statement shall be a well-formed switch statement.
6.4.4	Req	A switch-label shall only be used when the most closely-enclosing compound statement is the body of a switch statement.

6.4.5	Req	An unconditional throw or break statement shall terminate every non-empty switch-clause.
6.4.6	Req	The final clause of a switch statement shall be the default-clause.
6.4.7	Req	The condition of a switch statement shall not have bool type.
6.4.8	Req	Every switch statement shall have at least one case-clause.
6.5.1	Req	A for loop shall contain a single loop-counter which shall not have floating type.
6.5.2	Req	If loop-counter is not modified by -- or ++, then, within condition, the loop-counter shall only be used as an operand to <=, <, > or >=.
6.5.3	Req	The loop-counter shall not be modified within condition or statement.
6.5.4	Req	The loop-counter shall be modified by one of: --, ++, -=n, or +=n; where n remains constant for the duration of the loop.
6.5.5	Req	A loop-control-variable other than the loop-counter shall not be modified within condition or expression.
6.5.6	Req	A loop-control-variable other than the loop-counter which is modified in statement shall have type bool.
6.6.1	Req	Any label referenced by a goto statement shall be declared in the same block, or in a block enclosing the goto statement.
6.6.2	Req	The goto statement shall jump to a label declared later in the same function body.
6.6.3	Req	The continue statement shall only be used within a well-formed for loop.
6.6.4	Req	For any iteration statement there shall be no more than one break or goto statement used for loop termination.
6.6.5	Req	A function shall have a single point of exit at the end of the function.
7.1.1	Req	A variable which is not modified shall be const qualified.
7.1.2	Req	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.
7.2.1	Req	An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration.

7.3.1	Req	The global namespace shall only contain main, namespace declarations and extern "C" declarations.
7.3.2	Req	The identifier main shall not be used for a function other than the global function main.
7.3.3	Req	There shall be no unnamed namespaces in header files.
7.3.4	Req	using-directives shall not be used.
7.3.5	Req	Multiple declarations for an identifier in the same namespace shall not straddle a using-declaration for that identifier.
7.3.6	Req	using-directives and using-declarations (excluding class scope or function scope using-declarations) shall not be used in header files.
7.4.1	Doc	All usage of assembler shall be documented.
7.4.2	Req	Assembler instructions shall only be introduced using the asm declaration.
7.4.3	Req	Assembly language shall be encapsulated and isolated.
7.5.1	Req	A function shall not return a reference or a pointer to an automatic variable (including parameters), defined within the function.
7.5.2	Req	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.
7.5.3	Req	A function shall not return a reference or a pointer to a parameter that is passed by reference or const reference.
7.5.4	Adv	Functions should not call themselves, either directly or indirectly.
8.0.1	Req	An init-declarator-list or a member-declarator-list shall consist of a single init-declarator or member-declarator respectively.
8.3.1	Req	Parameters in an overriding virtual function shall either use the same default arguments as the function they override, or else shall not specify any default arguments.
8.4.1	Req	Functions shall not be defined using the ellipsis notation.
8.4.2	Req	The identifiers used for the parameters in a re-declaration of a function shall be identical to those in the declaration.
8.4.3	Req	All exit paths from a function with non-void return type shall have an explicit return statement with an expression.

8.4.4	Req	A function identifier shall either be used to call the function or it shall be preceded by &.
8.5.1	Req	All variables shall have a defined value before they are used.
8.5.2	Req	Braces shall be used to indicate and match the structure in the non-zero initialization of arrays and structures.
8.5.3	Req	In an enumerator list, the = construct shall not be used to explicitly initialize members other than the first, unless all items are explicitly initialized.
9.3.1	Req	const member functions shall not return non-const pointers or references to class-data.
9.3.2	Req	Member functions shall not return non-const handles to class-data.
9.3.3	Req	If a member function can be made static then it shall be made static, otherwise if it can be made const then it shall be made const.
9.5.1	Req	Unions shall not be used.
9.6.1	Doc	When the absolute positioning of bits representing a bit-field is required, then the behaviour and packing of bit-fields shall be documented.
9.6.2	Req	Bit-fields shall be either bool type or an explicitly unsigned or signed integral type.
9.6.3	Req	Bit-fields shall not have enum type.
9.6.4	Req	Named bit-fields with signed integer type shall have a length of more than one bit.
10.1.1	Adv	Classes should not be derived from virtual bases.
10.1.2	Req	A base class shall only be declared virtual if it is used in a diamond hierarchy.
10.1.3	Req	An accessible base class shall not be both virtual and non-virtual in the same hierarchy.
10.2.1	Adv	All accessible entity names within a multiple inheritance hierarchy should be unique.
10.3.1	Req	There shall be no more than one definition of each virtual function on each path through the inheritance hierarchy.
10.3.2	Req	Each overriding virtual function shall be declared with the virtual keyword.

10.3.3	Req	A virtual function shall only be overridden by a pure virtual function if it is itself declared as pure virtual.
11.0.1	Req	Member data in non-POD class types shall be private.
12.1.1	Req	An object’s dynamic type shall not be used from the body of its constructor or destructor.
12.1.2	Adv	All constructors of a class should explicitly call a constructor for all of its immediate base classes and all virtual base classes.
12.1.3	Req	All constructors that are callable with a single argument of fundamental type shall be declared explicit.
12.8.1	Req	A copy constructor shall only initialize its base classes and the non-static members of the class of which it is a member.
12.8.2	Req	The copy assignment operator shall be declared protected or private in an abstract class.
14.5.1	Req	A non-member generic function shall only be declared in a namespace that is not an associated namespace.
14.5.2	Req	A copy constructor shall be declared when there is a template constructor with a single parameter that is a generic parameter.
14.5.3	Req	A copy assignment operator shall be declared when there is a template assignment operator with a parameter that is a generic parameter.
14.6.1	Req	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->
14.6.2	Req	The function chosen by overload resolution shall resolve to a function declared previously in the translation unit.
14.7.1	Req	All class templates, function templates, class template member functions and class template static members shall be instantiated at least once.
14.7.2	Req	For any given template specialization, an explicit instantiation of the template with the template-arguments used in the specialization shall not render the program ill-formed.
14.7.3	Req	All partial and explicit specializations for a template shall be declared in the same file as the declaration of their primary template.
14.8.1	Req	Overloaded function templates shall not be explicitly specialized.

14.8.2	Adv	The viable function set for a function call should either contain no function specializations, or only contain function specializations.
15.0.1	Doc	Exceptions shall only be used for error handling.
15.0.2	Adv	An exception object should not have pointer type.
15.0.3	Req	Control shall not be transferred into a try or catch block using a goto or a switch statement.
15.1.1	Req	The assignment-expression of a throw statement shall not itself cause an exception to be thrown.
15.1.2	Req	NULL shall not be thrown explicitly.
15.1.3	Req	An empty throw (throw;) shall only be used in the compound-statement of a catch handler.
15.3.1	Req	Exceptions shall be raised only after start-up and before termination of the program.
15.3.2	Adv	There should be at least one exception handler to catch all otherwise unhandled exceptions
15.3.3	Req	Handlers of a function-try-block implementation of a class constructor or destructor shall not reference non-static members from this class or its bases.
15.3.4	Req	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.
15.3.5	Req	A class type exception shall always be caught by reference.
15.3.6	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block for a derived class and some or all of its bases, the handlers shall be ordered most-derived to base class.
15.3.7	Req	Where multiple handlers are provided in a single try-catch statement or function-try-block, any ellipsis (catch-all) handler shall occur last.
15.4.1	Req	If a function is declared with an exception-specification, then all declarations of the same function (in other translation units) shall be declared with the same set of type-ids.
15.5.1	Req	A class destructor shall not exit with an exception.
15.5.2	Req	Where a function’s declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s).

15.5.3	Req	The terminate() function shall not be called implicitly.
16.0.1	Req	#include directives in a file shall only be preceded by other preprocessor directives or comments.
16.0.2	Req	Macros shall only be #define'd or #undef'd in the global namespace.
16.0.3	Req	#undef shall not be used.
16.0.4	Req	Function-like macros shall not be defined.
16.0.5	Req	Arguments to a function-like macro shall not contain tokens that look like preprocessing directives.
16.0.6	Req	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 ##.
16.0.7	Req	Undefined macro identifiers shall not be used in #if or #elif preprocessor directives, except as operands to the defined operator.
16.0.8	Req	If the # token appears as the first token on a line, then it shall be immediately followed by a preprocessing token.
16.1.1	Req	The defined preprocessor operator shall only be used in one of the two standard forms.
16.1.2	Req	All #else, #elif and #endif preprocessor directives shall reside in the same file as the #if or #ifdef directive to which they are related.
16.2.1	Req	The pre-processor shall only be used for file inclusion and include guards.
16.2.2	Req	C++ macros shall only be used for: include guards, type qualifiers, or storage class specifiers.
16.2.3	Req	Include guards shall be provided.
16.2.4	Req	The ', ", /* or // characters shall not occur in a header file name.
16.2.5	Adv	The \ character should not occur in a header file name.
16.2.6	Req	The #include directive shall be followed by either a <filename> or "filename" sequence.
16.3.1	Req	There shall be at most one occurrence of the # or ## operators in a single macro definition.
16.3.2	Adv	The # and ## operators should not be used.

16.6.1	Doc	All uses of the #pragma directive shall be documented.
17.0.1	Req	Reserved identifiers, macros and functions in the standard library shall not be defined, redefined or undefined.
17.0.2	Req	The names of standard library macros and objects shall not be reused.
17.0.3	Req	The names of standard library functions shall not be overridden.
17.0.4	Doc	All library code shall conform to MISRA C++.
17.0.5	Req	The setjmp macro and the longjmp function shall not be used.
18.0.1	Req	The C library shall not be used.
18.0.2	Req	The library functions atof, atoi and atol from library <cstdlib> shall not be used.
18.0.3	Req	The library functions abort, exit, getenv and system from library <cstdlib> shall not be used.
18.0.4	Req	The time handling functions of library <ctime> shall not be used.
18.0.5	Req	The unbounded functions of library <cstring> shall not be used.
18.2.1	Req	The macro offsetof shall not be used.
18.4.1	Req	Dynamic heap memory allocation shall not be used.
18.7.1	Req	The signal handling facilities of <csignal> shall not be used.
19.3.1	Req	The error indicator errno shall not be used.
27.0.1	Req	The stream input/output library <cstdio> shall not be used.