# MISRA C 2012 Mapping to CodeSonar®

| Category ID | Category Name | CodeSonar Class Mnemonic | CodeSonar Class Name | Relationship Type (category to class) |
|---|---|---|---|---|
| Misra2012:1.2 | Language extensions should not be used | LANG.COMM.CPP | C++ Comment in C | closely mapped |
| Misra2012:1.2 | Language extensions should not be used | LANG.EXT.GNU | GNU Extension | closely mapped |
| Misra2012:1.2 | Language extensions should not be used | LANG.EXT.TYPEOF | GNU Typeof | closely mapped |
| Misra2012:1.2 | Language extensions should not be used | LANG.EXT.MS | Microsoft Extension | closely mapped |
| Misra2012:2.1 | A project shall not contain unreachable code | LANG.STRUCT.UC | Unreachable Call | closely mapped |
| Misra2012:2.1 | A project shall not contain unreachable code | LANG.STRUCT.UC | Unreachable Computation | closely mapped |
| Misra2012:2.1 | A project shall not contain unreachable code | LANG.STRUCT.UC | Unreachable Conditional | closely mapped |
| Misra2012:2.1 | A project shall not contain unreachable code | LANG.STRUCT.UC | Unreachable Control Flow | closely mapped |
| Misra2012:2.1 | A project shall not contain unreachable code | LANG.STRUCT.UC | Unreachable Data Flow | closely mapped |
| Misra2012:2.2 | There shall be no dead code | MISC.NOEFFECT | Function Call Has No Effect | closely mapped |
| Misra2012:2.2 | There shall be no dead code | LANG.STRUCT.UUVAL | Unused Value | closely mapped |
| Misra2012:2.2 | There shall be no dead code | LANG.STRUCT.UA | Useless Assignment | closely mapped |
| Misra2012:2.3 | A project should not contain unused type declarations | LANG.STRUCT.UUTYPE | Unused Type | closely mapped |
| Misra2012:2.4 | A project should not contain unused tag declarations | LANG.STRUCT.UUTAG | Unused Tag | closely mapped |
| Misra2012:2.5 | A project should not contain unused macro declarations | LANG.STRUCT.UUMACRO | Unused Macro | closely mapped |
| Misra2012:2.6 | A function should not contain unused label declarations | LANG.STRUCT.UULABEL | Unused Label | closely mapped |
| Misra2012:2.7 | There should be no unused parameters in functions | LANG.STRUCT.UUPARAM | Unused Parameter | closely mapped |
| Misra2012:3.1 | The character sequences /* and // shall not be used within a comment | LANG.COMM.NEST.CSTYLE | /* in Comment | closely mapped |
| Misra2012:3.1 | The character sequences /* and // shall not be used within a comment | LANG.COMM.NEST.CPPSTYLE | // in Comment | closely mapped |
| Misra2012:3.2 | Line-splicing shall not be used in // comments | LANG.COMM.SPLICE | Line Splicing in Comment | closely mapped |
| Misra2012:4.2 | Trigraphs should not be used | LANG.STRUCT.TRIGRAPH | Trigraph | closely mapped |
| Misra2012:5.1 | External identifiers shall be distinct | LANG.ID.ND.EXT | Non-distinct Identifiers: External Names | closely mapped |
| Misra2012:5.2 | Identifiers declared in the same scope and name space shall be distinct | LANG.ID.ND.SS | Non-distinct Identifiers: Same Scope | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:5.3 | An identifier declared in an inner scope shall not hide an identifier declared in an outer scope | LANG.ID.ND.NEST | Non-distinct Identifiers: Nested Scope | closely mapped |
| Misra2012:5.4 | Macro identifiers shall be distinct | LANG.ID.ND.MM | Non-distinct Identifiers: Macro/Macro | closely mapped |
| Misra2012:5.5 | Identifiers shall be distinct from macro names | LANG.ID.ND.MO | Non-distinct Identifiers: Macro/Other | closely mapped |
| Misra2012:5.6 | A typedef name shall be a unique identifier | LANG.ID.NU.TYPE | Non-unique Identifiers: Typedef | closely mapped |
| Misra2012:5.7 | A tag name shall be a unique identifier | LANG.ID.NU.TAG | Non-unique Identifiers: Tag | closely mapped |
| Misra2012:5.8 | Identifiers that define objects or functions with external linkage shall be unique | LANG.ID.NU.EXT | Non-unique Identifiers: External Name | closely mapped |
| Misra2012:5.9 | Identifiers that define objects or functions with internal linkage should be unique | LANG.ID.NU.INT | Non-unique Identifiers: Internal Name | closely mapped |
| Misra2012:6.1 | Bit-fields shall only be declared with an appropriate type | LANG.TYPE.BFSIGN | Bit-field Signedness Not Explicit | closely mapped |
| Misra2012:6.1 | Bit-fields shall only be declared with an appropriate type | LANG.TYPE.BFINT | Inappropriate Bit-field Type | closely mapped |
| Misra2012:6.2 | Single-bit named bit fields shall not be of a signed type | LANG.TYPE.BFSHORT | Bit-field Too Short | closely mapped |
| Misra2012:7.1 | Octal constants shall not be used | LANG.TYPE.OC | Octal Constant | closely mapped |
| Misra2012:7.2 | A "u" or "U" suffix shall be applied to all integer constants that are represented in an unsigned type | LANG.TYPE.MSUF | Missing Literal Suffix | closely mapped |
| Misra2012:7.3 | The lowercase character "l" shall not be used in a literal suffix | LANG.TYPE.CSUF | Confusing Literal Suffix | closely mapped |
| Misra2012:7.4 | A string literal shall not be assigned to an object unless the object's type is "pointer to const-qualified char" | LANG.TYPE.NCS | Non-const String Literal | closely mapped |
| Misra2012:8.2 | Function types shall be in prototype form with named parameters | LANG.FUNCS.PROT | Incomplete Function Prototype | closely mapped |
| Misra2012:8.3 | All declarations of an object or function shall use the same names and type qualifiers | LANG.STRUCT.DECL.MGT | Global Variable Declared with Different Types | closely mapped |
| Misra2012:8.3 | All declarations of an object or function shall use the same names and type qualifiers | LANG.STRUCT.DECL.IF | Inconsistent Function Declarations | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:8.3 | All declarations of an object or function shall use the same names and type qualifiers | LANG.STRUCT.DECL.IO | Inconsistent Object Declarations | closely mapped |
| Misra2012:8.5 | An external object or function shall be declared once in one and only one file | LANG.STRUCT.DECL.NOEXT | Missing External Declaration | closely mapped |
| Misra2012:8.5 | An external object or function shall be declared once in one and only one file | LANG.STRUCT.DECL.MG | Multiple Declarations of a Global | closely mapped |
| Misra2012:8.5 | An external object or function shall be declared once in one and only one file | LANG.STRUCT.DECL.MULTIEXT | Multiple External Declarations | closely mapped |
| Misra2012:8.6 | An identifier with external linkage shall have exactly one external definition | LANG.STRUCT.DEF.NOEXT | Missing External Definition | closely mapped |
| Misra2012:8.6 | An identifier with external linkage shall have exactly one external definition | LANG.STRUCT.DEF.MULTIEXT | Multiple External Definitions | closely mapped |
| Misra2012:8.7 | Functions and objects should not be defined with external linkage if they are referenced in only one translation unit | LANG.STRUCT.SCOPE.FILE | Scope Could Be File Static | closely mapped |
| Misra2012:8.7 | Functions and objects should not be defined with external linkage if they are referenced in only one translation unit | LANG.STRUCT.SCOPE.LOCAL | Scope Could Be Local Static | closely mapped |
| Misra2012:8.8 | The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage | LANG.STRUCT.SCOPE.FILE | Scope Could Be File Static | closely mapped |
| Misra2012:8.8 | The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage | LANG.STRUCT.SCOPE.LOCAL | Scope Could Be Local Static | closely mapped |
| Misra2012:8.9 | An object should be defined at block scope if its identifier only appears in a single function | LANG.STRUCT.SCOPE.LOCAL | Scope Could Be Local Static | closely mapped |
| Misra2012:8.10 | An inline function shall be declared with the static storage class | LANG.TYPE.INS | Inline Function Not static | closely mapped |
| Misra2012:8.11 | When an array with external linkage is declared, its size should be explicitly specified | LANG.STRUCT.DECL.EAWS | Extern Array Without Size | closely mapped |
| Misra2012:8.12 | Within an enumerator list, the value of an implicitly-specified enumeration constant shall be unique | LANG.STRUCT.INIT.ENUM | Inconsistent Enumerator Initialization | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:8.14 | The restrict type qualifier shall not be used | LANG.TYPE.RESTRICT | Restrict Qualifier Used | closely mapped |
| Misra2012:9.1 | The value of an object with automatic storage duration shall not be read before it has been set | LANG.MEM.UVAR | Uninitialized Variable | closely mapped |
| Misra2012:9.2 | The initializer for an aggregate or union shall be enclosed in braces | LANG.STRUCT.INIT.MBI | Missing Braces in Initialization | closely mapped |
| Misra2012:9.3 | Arrays shall not be partially initialized | LANG.STRUCT.INIT.PIARR | Partially Uninitialized Array | closely mapped |
| Misra2012:9.4 | An element of an object shall not be initialized more than once | LANG.STRUCT.INIT.OIE | Over-initialized Element | closely mapped |
| Misra2012:10.1 | Operands shall not be of an inappropriate essential type | LANG.TYPE.IOT | Inappropriate Operand Type | closely mapped |
| Misra2012:10.2 | Expressions of essentially character type shall not be used inappropriately in addition and subtraction operations | LANG.TYPE.ICA | Inappropriate Character Arithmetic | closely mapped |
| Misra2012:10.3 | The value of an expression shall not be assigned to an object with a narrower essential type or of a different essential type category | LANG.TYPE.IAT | Inappropriate Assignment Type | closely mapped |
| Misra2012:10.4 | Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category | LANG.TYPE.MOT | Mismatched Operand Types | closely mapped |
| Misra2012:10.5 | The value of an expression should not be cast to an inappropriate essential type | LANG.TYPE.ICT | Inappropriate Cast Type | closely mapped |
| Misra2012:10.6 | The value of a composite expression shall not be assigned to an object with wider essential type | LANG.TYPE.AWID | Expression Value Widened by Assignment | closely mapped |
| Misra2012:10.7 | 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 | LANG.TYPE.OWID | Expression Value Widened by Other Operand | closely mapped |
| Misra2012:10.8 | The value of a composite expression shall not be cast to a different essential type category or a wider essential type | LANG.TYPE.ICTE | Inappropriate Cast Type: Expression | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:11.1 | Conversions shall not be performed between a pointer to a function and any other type | LANG.CAST.PC.FN2DATA | Conversion from Function Pointer | closely mapped |
| Misra2012:11.1 | Conversions shall not be performed between a pointer to a function and any other type | LANG.CAST.PC.DATA2FN | Conversion to Function Pointer | closely mapped |
| Misra2012:11.1 | Conversions shall not be performed between a pointer to a function and any other type | LANG.CAST.FN | Dangerous Function Cast | closely mapped |
| Misra2012:11.1 | Conversions shall not be performed between a pointer to a function and any other type | LANG.STRUCT.FUNCPTR.CONVERT | Function Pointer Conversion | closely mapped |
| Misra2012:11.2 | Conversions shall not be performed between a pointer to an incomplete type and any other type | LANG.CAST.PC.INC | Conversion: Pointer to Incomplete | closely mapped |
| Misra2012:11.3 | A cast shall not be performed between a pointer to object type and a pointer to a different object type | LANG.CAST.PC.OBJ | Cast: Object Pointers | closely mapped |
| Misra2012:11.4 | A conversion should not be performed between a pointer to object and an integer type | LANG.CAST.PC.INT | Conversion: Pointer/Integer | closely mapped |
| Misra2012:11.5 | A conversion should not be performed from pointer to void into pointer to object | LANG.CAST.PC.PV | Conversion: Void Pointer to Object Pointer | closely mapped |
| Misra2012:11.6 | A cast shall not be performed between pointer to void and an arithmetic type | LANG.CAST.PC.AV | Cast: Arithmetic Type/Void Pointer | closely mapped |
| Misra2012:11.7 | A cast shall not be performed between pointer to object and a non-integer arithmetic type | LANG.CAST.PC.AO | Cast: Non-integer Arithmetic Type/Object Pointer | closely mapped |
| Misra2012:11.8 | A cast shall not remove any const or volatile qualification from the type pointed to by a pointer | LANG.CAST.PC.CRCQ | Cast Removes const Qualifier | closely mapped |
| Misra2012:11.8 | A cast shall not remove any const or volatile qualification from the type pointed to by a pointer | LANG.CAST.PC.CRVQ | Cast Removes volatile Qualifier | closely mapped |
| Misra2012:12.2 | 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 | LANG.ARITH.NEGSHIFT | Negative Shift Amount | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:12.2 | 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 | LANG.ARITH.BIGSHIFT | Shift Amount Exceeds Bit Width | closely mapped |
| Misra2012:12.3 | The comma operator should not be used | LANG.STRUCT.COMMA | Use of Comma Operator | closely mapped |
| Misra2012:12.4 | Evaluation of constant expressions should not lead to unsigned integer wrap-around | LANG.CAST.COERCE | Coercion Alters Value | closely mapped |
| Misra2012:13.3 | 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 | LANG.STRUCT.SE.DEC | Side Effects in Expression with Decrement | closely mapped |
| Misra2012:13.3 | 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 | LANG.STRUCT.SE.INC | Side Effects in Expression with Increment | closely mapped |
| Misra2012:13.6 | The operand of the sizeof operator shall not contain any expression which has potential side effects | LANG.STRUCT.SE.SIZEOF | Side Effects in sizeof | closely mapped |
| Misra2012:14.1 | A loop counter shall not have essentially floating type | LANG.STRUCT.LOOP.FPC | Float-typed Loop Counter | closely mapped |
| Misra2012:14.2 | A for loop shall be well-formed | LANG.STRUCT.LOOP.MFTERM | Malformed for-loop Condition | closely mapped |
| Misra2012:14.2 | A for loop shall be well-formed | LANG.STRUCT.LOOP.MFINIT | Malformed for-loop Initialization | closely mapped |
| Misra2012:14.2 | A for loop shall be well-formed | LANG.STRUCT.LOOP.MFSTEP | Malformed for-loop Step | closely mapped |
| Misra2012:14.2 | A for loop shall be well-formed | LANG.STRUCT.LOOP.NOSTEP | Missing for-loop Step | closely mapped |
| Misra2012:14.2 | A for loop shall be well-formed | LANG.STRUCT.LOOP.NOTERM | Missing for-loop Termination | closely mapped |
| Misra2012:14.3 | Controlling expressions shall not be invariant | LANG.STRUCT.RC | Redundant Condition | closely mapped |
| Misra2012:14.4 | The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type | LANG.STRUCT.NBC | Condition Is Not Boolean | closely mapped |
| Misra2012:15.1 | The goto statement should not be used | LANG.STRUCT.GOTO | Goto Statement | closely mapped |
| Misra2012:15.2 | The goto statement shall jump to a label declared later in the same function | LANG.STRUCT.BGOTO | Backwards goto | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:15.3 | Any label referenced by a goto statement shall be declared in the same block, or in any block enclosing the goto statement | LANG.STRUCT.GLABEL | Label Not In Enclosing Block | closely mapped |
| Misra2012:15.4 | There should be no more than one break or goto statement used to terminate any iteration statement | LANG.STRUCT.LOOP.MAE | Multiple Abnormal Loop Exits | closely mapped |
| Misra2012:15.5 | A function should have a single point of exit at the end | LANG.STRUCT.MISRS | Misplaced Return Statement | closely mapped |
| Misra2012:15.5 | A function should have a single point of exit at the end | LANG.STRUCT.MULRS | Multiple Return Statements | closely mapped |
| Misra2012:15.6 | The body of an iteration-statement or a selection-statement shall be a compound-statement | LANG.STRUCT.BNC | Body Is Not Compound Statement | closely mapped |
| Misra2012:15.7 | All if ... else if constructs shall be terminated with an else statement | LANG.STRUCT.NOELSE | Missing Final else | closely mapped |
| Misra2012:16.1 | All switch statements shall be well-formed | LANG.STRUCT.SW.BAD | Malformed switch Statement | closely mapped |
| Misra2012:16.3 | An unconditional break statement shall terminate every switch-clause | LANG.STRUCT.SW.MB | Missing break | closely mapped |
| Misra2012:16.4 | Every switch statement shall have a default label | LANG.STRUCT.SW.MD | Missing default | closely mapped |
| Misra2012:16.5 | A default label shall appear as either the first or the last switch label of a switch statement | LANG.STRUCT.SW.MPD | Misplaced default | closely mapped |
| Misra2012:16.6 | Every switch statement shall have at least two switch-clauses | LANG.STRUCT.SW.IF | Too Few Cases in switch | closely mapped |
| Misra2012:16.7 | A switch-expression shall not have essentially Boolean type | LANG.STRUCT.SW.BOOL | Boolean switch Expression | closely mapped |
| Misra2012:17.2 | Functions shall not call themselves, either directly or indirectly | LANG.FUNCS.RECURSION | Recursion | closely mapped |
| Misra2012:17.3 | A function shall not be declared implicitly | LANG.STRUCT.DECL.IMPFN | Implicit Function Declaration | closely mapped |
| Misra2012:17.4 | All exit paths from a function with non-void return type shall have an explicit return statement with an expression | LANG.STRUCT.MRS | Missing Return Statement | closely mapped |
| Misra2012:17.6 | The declaration of an array parameter shall not contain the static keyword between the [ ] | LANG.FUNCS.SAP | Static Array Parameter | closely mapped |
| Misra2012:17.7 | The value returned by a function having non-void return type shall be used | LANG.FUNCS.IRV | Ignored Return Value | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:17.8 | A function parameter should not be modified | LANG.FUNCS.MODP | Modified Parameter | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.MEM.BO | Buffer Overrun | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.MEM.BU | Buffer Underrun | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.STRUCT.PBB | Pointer Before Beginning of Object | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.STRUCT.PPE | Pointer Past End of Object | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.MEM.TBA | Tainted Buffer Access | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.MEM.TO | Type Overrun | closely mapped |
| Misra2012:18.1 | A pointer resulting from arithmetic on a pointer operand shall address an element of the same array as that pointer operand | LANG.MEM.TU | Type Underrun | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.MEM.BO | Buffer Overrun | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.MEM.BU | Buffer Underrun | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.STRUCT.PBB | Pointer Before Beginning of Object | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.STRUCT.PPE | Pointer Past End of Object | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.MEM.TBA | Tainted Buffer Access | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.MEM.TO | Type Overrun | closely mapped |
| Misra2012:18.2 | Subtraction between pointers shall only be applied to pointers that address elements of the same array | LANG.MEM.TU | Type Underrun | closely mapped |
| Misra2012:18.5 | Declarations should contain no more than two levels of pointer nesting | LANG.STRUCT.TMID | Too Much Indirection in Declaration | closely mapped |
| Misra2012:18.7 | Flexible array members shall not be declared | LANG.STRUCT.DECL.FAM | Declaration of Flexible Array Member | closely mapped |
| Misra2012:18.8 | Variable-length array types shall not be used | LANG.STRUCT.DECL.VLA | Declaration of Variable Length Array | closely mapped |
| Misra2012:19.1 | An object shall not be assigned or copied to an overlapping object | MISC.MEM.OR | Overlapping Memory Regions | closely mapped |
| Misra2012:19.2 | The union keyword should not be used | LANG.TYPE.UNION | Union Type | closely mapped |
| Misra2012:20.1 | #include directives should only be preceded by preprocessor directives or comments | LANG.PREPROC.CBI | Code Before #include | closely mapped |
| Misra2012:20.2 | The ', " or \ characters and the /* or // character sequences shall not occur in a header file name | LANG.PREPROC.INCL.FNAME | Dangerous Include File Name | closely mapped |
| Misra2012:20.3 | The #include directive shall be followed by either a <filename> or "filename" sequence | LANG.PREPROC.INCL.MF | Malformed #include | closely mapped |
| Misra2012:20.4 | A macro shall not be defined with the same name as a keyword | LANG.ID.NU.MK | Macro Name is C Keyword | closely mapped |
| Misra2012:20.5 | #undef should not be used | LANG.PREPROC.RUNDEF | Macro Undefinition of Reserved Name | closely mapped |
| Misra2012:20.5 | #undef should not be used | LANG.PREPROC.UNDEF | Use of #undef | closely mapped |
| Misra2012:20.6 | Tokens that look like a preprocessing directive shall not occur within a macro argument | LANG.PREPROC.MACROARG | Preprocessing Directives in Macro Argument | closely mapped |
| Misra2012:20.7 | Expressions resulting from the expansion of macro parameters shall be enclosed in parentheses | LANG.PREPROC.NOFPAREN | Macro Parameter Not Parenthesized | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:20.10 | The # and ## preprocessor operators should not be used | LANG.PREPROC.PASTE | Macro Uses ## Operator | closely mapped |
| Misra2012:20.13 | A line whose first token is # shall be a valid preprocessing directive | LANG.PREPROC.INVALID | Invalid Preprocessor Directive | closely mapped |
| Misra2012:20.14 | 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 | LANG.PREPROC.NOENDIF | No Matching #endif | closely mapped |
| Misra2012:20.14 | 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 | LANG.PREPROC.NOIF | No Matching #if | closely mapped |
| Misra2012:21.1 | #define and #undef shall not be used on a reserved identifier or reserved macro name | LANG.PREPROC.RDEF | Macro Definition of Reserved Name | closely mapped |
| Misra2012:21.1 | #define and #undef shall not be used on a reserved identifier or reserved macro name | LANG.PREPROC.RUNDEF | Macro Undefinition of Reserved Name | closely mapped |
| Misra2012:21.3 | The memory allocation and deallocation functions of <stdlib.h> shall not be used | ALLOC.POSTINIT | Dynamic Allocation After Initialization | closely mapped |
| Misra2012:21.4 | The standard header file <setjmp.h> shall not be used | LANG.PREPROC.INCL.SETJMP_H | Use of <setjmp.h> | closely mapped |
| Misra2012:21.4 | The standard header file <setjmp.h> shall not be used | BADFUNC.LONGJMP | Use of longjmp | closely mapped |
| Misra2012:21.4 | The standard header file <setjmp.h> shall not be used | BADFUNC.SETJMP | Use of setjmp | closely mapped |
| Misra2012:21.5 | The standard header file <signal.h> shall not be used | LANG.PREPROC.INCL.SIGNAL_H | Use of <signal.h> | closely mapped |
| Misra2012:21.6 | The Standard Library input/output functions shall not be used | BADFUNC.STDIO_H | Use of <stdio.h> Input/Output | closely mapped |
| Misra2012:21.6 | The Standard Library input/output functions shall not be used | BADFUNC.WCHAR_H | Use of <wchar.h> Input/Output | closely mapped |
| Misra2012:21.7 | The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used | BADFUNC.ATOF | Use of atof | closely mapped |
| Misra2012:21.7 | The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used | BADFUNC.ATOI | Use of atoi | closely mapped |
| Misra2012:21.7 | The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used | BADFUNC.ATOL | Use of atol | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:21.7 | The atof, atoi, atol and atoll functions of <stdlib.h> shall not be used | BADFUNC.ATOLL | Use of atoll | closely mapped |
| Misra2012:21.8 | The library functions abort, exit, getenv and system of <stdlib.h> shall not be used | BADFUNC.ABORT | Use of abort | closely mapped |
| Misra2012:21.8 | The library functions abort, exit, getenv and system of <stdlib.h> shall not be used | BADFUNC.EXIT | Use of exit | closely mapped |
| Misra2012:21.8 | The library functions abort, exit, getenv and system of <stdlib.h> shall not be used | BADFUNC.GETENV | Use of getenv | closely mapped |
| Misra2012:21.8 | The library functions abort, exit, getenv and system of <stdlib.h> shall not be used | BADFUNC.PATH.SYSTEM | Use of system | closely mapped |
| Misra2012:21.9 | The library functions bsearch and qsort of <stdlib.h> shall not be used | BADFUNC.BSEARCH | Use of bsearch | closely mapped |
| Misra2012:21.9 | The library functions bsearch and qsort of <stdlib.h> shall not be used | BADFUNC.QSORT | Use of qsort | closely mapped |
| Misra2012:21.10 | The Standard Library time and date functions shall not be used | BADFUNC.TIME_H | Use of <time.h> Time/Date Function | closely mapped |
| Misra2012:21.11 | The standard header file <tgmath.h> shall not be used | LANG.PREPROC.INCL.TGMATH_H | Use of <tgmath.h> | closely mapped |
| Misra2012:21.12 | The exception handling features of <fenv.h> should not be used | BADFUNC.FENV_H | Use of <fenv.h> Exception Handling Function | closely mapped |
| Misra2012:22.1 | All resources obtained dynamically by means of Standard Library functions shall be explicitly released | ALLOC.LEAK | Leak | closely mapped |
| Misra2012:22.2 | A block of memory shall only be freed if it was allocated by means of a Standard Library function | ALLOC.FNH | Free Non-Heap Variable | closely mapped |
| Misra2012:22.4 | There shall be no attempt to write to a stream which has been opened as read-only | IO.WRITERO | Write to Read Only File | closely mapped |
| Misra2012:22.5 | A pointer to a FILE object shall not be dereferenced | IO.FILEDEREF | FILE* Dereference | closely mapped |
| Misra2012:22.6 | The value of a pointer to a FILE shall not be used after the associated stream has been closed | IO.UAC | Use After Close | closely mapped |
| Misra2012:D.4.3 | Assembly language shall be encapsulated and isolated | LANG.ASM.MIXED | Mixed Assembly and Code | closely mapped |

| | | | | |
|---|---|---|---|---|
| Misra2012:D.4.4 | Sections of code should not be "commented out" | LANG.COMM.CODE | Commented-out Code | closely mapped |
| Misra2012:D.4.5 | Identifiers in the same name space with overlapping visibility should be typographically unambiguous | LANG.ID.AMBIG | Typographically Ambiguous Identifiers | closely mapped |
| Misra2012:D.4.6 | typedefs that indicate size and signedness should be used in place of the basic numerical types | LANG.TYPE.BASIC | Basic Numerical Type Used | closely mapped |
| Misra2012:D.4.7 | If a function returns error information, then that error information shall be tested | LANG.FUNCS.IRV | Ignored Return Value | closely mapped |
| Misra2012:D.4.12 | Dynamic memory allocation shall not be used | ALLOC.POSTINIT | Dynamic Allocation After Initialization | closely mapped |