HIS Standards Model Summary for C / C++

The LDRA tool suite® is developed and certified to BS EN ISO 9001:2015, TÜV SÜD and SGS-TÜV Saar.

This information is applicable to version 10.2.0 of the LDRA tool suite®.
It is correct as of 11th September 2023.

Compliance is measured against
"Gemeinsames Subset der MISRA C Guidelines. Version 1.0.3"
April 2004

Further information is available at http://www.automotive-his.de

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Required	0	86	2	0	2	90
Advisory	0	22	1	1	1	25
Total	0	108	3	1	3	115

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
1	Required	All code shall conform to ISO 9899 standard C, with no extensions permitted.	To check for ANSI compliance the Testbed needs to be configured for the particular compiler, i.e., LDRA needs details of the specific extensions. This is because the LDRA Testbed uses an adaptive grammar which handles language extensions, e.g. for real-time or embedded processes. Therefore the technique for detecting deviation from ANSI is explicit rather than exclusive. S22 is only enabled if modifier 536 is set to 1. The modifier also means that =- etc: are treated as compound assignment operators, rather than as two seperate operators = and -.	17 S	Code insert found.
				22 S	Use of obsolete language feature (use = -). Modifiers :536 = 1 (Default)
				110 S	Use of single line comment //. Modifiers :350 = 0 (Default)
				293 S	Non ANSI/ISO construct used.
				387 S	Enum init not integer-constant-expression.
				632 S	Use of // comment in pre-processor directive or macro defn. Modifiers :350 = 0 (Default)
				646 S	Struct initialisation has too many items.
2	Advisory	Code written in languages other than C should only be used if there is a defined interface standard for object code to which the compilers/assemblers for both languages conform.	In general this is not an issue for the LDRA Testbed but code inserts are indicated with standard S 17.		
5	Required	Only those characters and escape sequences which are defined in the ISO C standard shall be used	The set of characters used by LDRA Testbed is defined in an input data file. Any others are discarded.	113 S	Non standard character in source. Modifiers :408 = 0 (Default) 449 = 0 (Default)
				176 S	Non standard escape sequence in source.
7	Required	Trigraphs shall not be used	Trigraphs are reported with standard S 81	81 S	Use of trigraph.
8	Required	Multibyte characters and wide string literals shall not be used	Multibyte characters and wide string literals are reported with standard S 82.	82 S	Use of wide string literal.
9	Required	Comments shall not be nested	A standards message 119 S is issued if /* appears within either a /* or a // style comment.	119 S	Nested comment found. Modifiers :413 = 1 434 = 0 (Default)
10	Advisory	Sections of code should not be 'commented out'	If comments appear to contain code, a standards message S 302 is issued.	302 S	Comment possibly contains code.
11	Required	Identifiers (internal & external) shall not rely on significance of more than 31 characters. Furthermore the compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifier	LDRA Testbed reports identifiers that are not unique in the number of characters specified in 17 D and 61 X.	17 D	Identifier not unique within *** characters.
				61 X	Identifier match in *** chars.
13	Advisory	The basic types of char. int. short, long, float, and double should not be used, but specific-length equivalents should be typedef'd for the specific compiler, and these type names used in the code	Standard S 90 reports the use of basic types. Note that signed and unsigned used alone are not covered by the standard but probably should be and hence are covered by the Testbed. The report on basic types in typedefs can be included or excluded by the configuration switch. Configuration is via the following entry in the file ctbend.dat: T Flag to apply basic types standard in typedefs	90 S	Basic type declaration used. Modifiers :219 = 0 (Default) 462 = 0 (Default)

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
14	Required	The type char shall always be declared as unsigned char or signed char	89 S does not allow the use of the plain char type and can be disabled if a project allows the use of plain char. The other standards check that objects of plain char type are used in an appropriate way.	89 S	char type not signed or unsigned.
				329 S	Operation not appropriate to plain char. Modifiers :191 = 0 (Default) 331 = 0 (Default) 391 = 0 (Default)
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
16	Required	The underlying bit representations of floating point numbers shall not be used in any way by the programmer.		345 S	Bit operator with floating point operand.
17	Required	Typedef names shall not be reused		112 S	Typedef name redeclared. Modifiers :417 = 0 (Default)
18	Advisory	Numeric constants should be suffixed to indicate type, where an appropriate suffix is available		331 S	Literal value requires a U suffix. Modifiers :309 = 0 (Default) 358 = 0 (Default) 516 = 0 (Default)
				425 S	float literal with no F suffix.
				466 S	long literal with no L suffix.
19	Required	Octal constants (other than zero) shall not be used	Octal constants are identified with standard S 83.	83 S	Octal number found. Modifiers :469 = 0 (Default)
20	Required	All object and function identifiers shall be declared before use	Dataflow standards D 19 and D 20 reports all objects and functions used before declaration	19 D	Procedure called before being defined.
				20 D	No declaration for variable found before use.
21	Required	Identifiers in an inner scope shall not use the same name as an identifier in an outer scope, and therefore hide that identifier.		18 D	Identifier name reused.
				1 S	Procedure name reused.
				128 S	Parameter has same name as global variable.
				131 S	Name reused in inner scope.
22	Advisory	Declarations of objects should be at function scope unless a wider scope is necessary		25 D	Scope of variable could be reduced.
23	Advisory	All declarations at file scope should be static where possible	standard D 27 detects declarations which should be restricted to file scope.	27 D	Variable should be declared static.
24	Required	Identifiers shall not simultaneously have both internal and external linkage in the same translation unit		461 S	Identifier with ambiguous linkage.
				575 S	Linkage differs from previous declaration.
				2 X	Ambiguous declaration of variable.
25	Required	An identifiers with external linkage shall have exactly one external definition		26 D	Variable should be defined once in only one file.
				33 D	No real declaration for external variable.
				34 D	Procedure name re-used in different files.
				172 S	Variable declared multiply. Modifiers :297 = 0 (Default)
				2 X	Ambiguous declaration of variable.
26	Required	If objects or functions are declared more than once they shall have compatible declarations	Standards S 102, S 103 ensure that functions and prototypes have compatible declarations. Checks across file boundaries to ensure full type compatibility for objects and procedures are implemented by standards X 1 and X 2. LDRA Associated MISRA-C:1998 26, 72	102 S	Function and prototype return inconsistent (MR). Modifiers :331 = 0 (Default)
				103 S	Function and prototype param inconsistent (MR).
				1 X	Declaration types do not match across a system.
				2 X	Ambiguous declaration of variable.
27	Advisory	External objects should not be declared in more than one file		60 D	External object should be declared only once.
29	Required	The use of a tag shall agree with its declaration	Standard S 104 reports instances where the initial value of a structure field is non-conformant.	104 S	Struct field initialisation incorrect. Modifiers :358 = 0 (Default) 427 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 430 = 0 (Default) 431 = 0 (Default) 432 = 0 (Default) 453 = 0 (Default) 470 = 0 (Default)
30	Required	All automatic variables shall have been assigned a value before being used	Dataflow analysis reports this with UR anomalies.	69 D	UR anomaly, variable used before assignment.

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
31	Required	Braces shall be used to indicate and match the structure in the non-zero initialisation of arrays and structures		105 S	Initialisation brace { } fault.
32	Required	In an enumerator list, the `=' construct shall not be used to explicitly initialise members other than the first, unless all items are explicitly initialised	Mixed use and non use of initialisers is reported with standard S 85.	85 S	Incomplete initialisation of enumerator.
33	Required	The right hand operands of a && or || operator shall not contain side effects	Side effects are reported in a number of ways. Functions which can change parameters or global variables are reported in Static Dataflow Analysis. If these functions are subsequently used in the right hand operand of the logical conjunction operators && or || then standard Q 1 is raised. Static Analyis reports similarly for all the assignment operators in the same context.LDRA Associated MISRA-C:1998 33, 35, 40, 46	1 Q	Call has execution order dependant side effects.
				9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				30 S	Deprecated usage of ++ or -- operators found. Modifiers :122 = 1 (Default)
				133 S	Assignment operator in RHS of && or ||.
				406 S	Use of ++ or -- on RHS of && or || operator.
				408 S	Volatile variable accessed on RHS of && or ||.
34	Required	The operands of a logical && or || shall be primary expressions		49 S	Logical conjunctions need brackets. Modifiers :206 = 0 (Default)
35	Required	Assignment operators shall not be used in expressions which return Boolean values	LDRA Associated MISRA-C:1998 33, 35, 36, 40, 46, 49	9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				114 S	Expression is not Boolean. Modifiers :386 = 1 (Default) 528 = 0 (Default)
				132 S	Assignment operator in boolean expression.
36	Advisory	Logical operators should not be confused with bitwise operators	If the compiler implements a Boolean type then the normal strong typing rules are applied in all contexts and this will detect such cases. If the compiler does not have an explicit type then either a special type BOOL say can be created and an entry put in the cvals.dat table, the C LDRA Testbed will attempt to detect the use of Boolean values and provide the normal standards checks for strong typing. LDRA Associated MISRA-C:1998 35, 36, 49	114 S	Expression is not Boolean. Modifiers :386 = 1 (Default) 528 = 0 (Default)
				136 S	Bit operator with boolean operand.
37	Required	Bitwise operations shall not be performed on signed integer types	Bitwise operations on signed integers are detected by S 120. standard S 50. Other bit operations are detected by S 120.	50 S	Use of shift operator on signed type.
				120 S	Use of bit operator on signed type. Modifiers :457 = 0 (Default)
38	Required	The right hand operand of a shift operator shall lie between zero and one less than the width in bits of the left hand operand (inclusive)	The size of the right hand operand of the shift operators is detected by standard S 51.Negative (or potentially negative) shift operators are detected by standard S 403.	51 S	Shifting value too far. Modifiers :520 = 0 (Default)
				403 S	Negative (or potentially negative) shift.
39	Required	The unary minus operator shall not be applied to an unsigned expression	The use of the unary minus with unsigned expressions is reported by standard S 52.	52 S	Unsigned expression negated.
40	Advisory	The sizeof operator should not be used on expressions that contain side effects	LDRA Associated MISRA-C:1998 33, 35, 36, 40, 46	9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				54 S	Sizeof operator with side effects.
41	Advisory	The implementation of integer division in the chosen compiler should be determined documented and taken into account		373 S	Use of integer division.
42	Required	The comma operator shall not be used, except in the control expression of a for loop	Comma operators (except in for loop control and parameter lists) are reported with standard S 53.	53 S	Use of comma operator. Modifiers :263 = 0 (Default)
43	Required	Implicit conversions which may result in a loss of information shall not be used	LDRA Associated MISRA-C:1998 43, 48	433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 0 (Default) 431 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 470 = 0 (Default) 386 = 1 (Default) 549 = 0 (Default)
				434 S	Signed/unsigned conversion without cast. Modifiers :374 = 0 (Default) 358 = 0 (Default) 427 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 429 = 0 (Default) 430 = 0 (Default) 470 = 0 (Default)
				435 S	Float/integer conversion without cast. Modifiers :374 = 0 (Default) 432 = 0 (Default) 453 = 0 (Default)
				446 S	Narrower int conversion without cast. Modifiers :374 = 0 (Default)

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
45	Required	Type casting from any type to or from pointers shall not be used	LDRA Associated MISRA-C:1998 45, 48	439 S	Cast from pointer to integral type. Modifiers :548 = 0 (Default)
				440 S	Cast from integral type to pointer. Modifiers :397 = 0 (Default) 548 = 0 (Default)
				540 S	Cast from pointer to void to pointer. Modifiers :381 = 0 (Default)
				554 S	Cast to an unrelated type. Modifiers :439 = 0 (Default) 440 = 0 (Default)
				606 S	Cast involving function pointer. Modifiers :398 = 0 (Default) 433 = 0 (Default) 439 = 0 (Default)
				635 S	Cast from pointer to float type.
				636 S	Cast from float type to pointer.
				701 S	Pointer cast to pointer to void.
46	Required	The value of an expression shall be the same under any order of evaluation that the standard permits	LDRA Associated MISRA-C:1998 33, 35, 40, 46	35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				9 S	Assignment operation in expression. Modifiers :259 = 0 (Default) 266 = 0 (Default)
				30 S	Deprecated usage of ++ or -- operators found Modifiers :122 = 1 (Default)
				55 S	Expression with more than one function.
				134 S	Volatile variable in complex expression. Modifiers :460 = 0 (Default) 461 = 1
47	Advisory	No dependence should be placed on C's operator precedence rules in expressions		361 S	Expression needs brackets. Modifiers :119 = 0 (Default) 264 = 0 (Default) 414 = 0 (Default) 420 = 0 (Default) 421 = 0 (Default)
48	Advisory	Mixed precision arithmetic should be placed on C's operator precedence rules in expressions	LDRA Associated MISRA-C:1998 18, 43, 44, 45, 48	97 S	Use of redundant cast.
				107 S	Type mismatch in ternary expression. Modifiers :428 = c : 0 (Default), c++ : 1 (Default) 429 = 0 (Default) 470 = 0 (Default) 525 = 0 (Default)
				434 S	Signed/unsigned conversion without cast. Modifiers :374 = 0 (Default) 358 = 0 (Default) 427 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 429 = 0 (Default) 430 = 0 (Default) 470 = 0 (Def
				435 S	Float/integer conversion without cast. Modifiers :374 = 0 (Default) 432 = 0 (Default) 453 = 0 (Default)
50	Required	Floating point variables shall not be tested for exact equality or inequality	The testing of floating point values for equality/inequality is reported by standard S 56.	56 S	Equality comparison of floating point. Modifiers :223 = 0 (Default)
51	Advisory	Evaluation of constant unsigned integer expressions should not lead to wrap-around		493 S	Numeric overflow.
52	Required	There shall be no unreachable code		1 J	Unreachable Code found.
				139 S	Construct leads to infeasible code.
				140 S	Infeasible loop condition found.
				631 S	Declaration not reachable.
53	Required	All non-null statements shall have a side-effect		57 S	Statement with no side effect. Modifiers :458 = 0 (Default)
54	Required	A null statement shall only occur on a line by itself, and shall not have any other text on the same line	Standard S 58 reports the use of the NULL statement. An exception is made for those cases where the semi-colon is on a line by itself. The C LDRA Testbed inserts null statements automatically in some contexts, e.g. if(); goes to if(){;}	58 S	Null statement found.
55	Advisory	Labels should not be used, except in switch statements		111 S	Label is not part of switch statement (MR). Modifiers :331 = 0 (Default)
56	Required	The goto statement shall not be used	Goto's are detected by standard S 13	13 S	goto detected.
57	Required	The continue statement shall not be used	The continue statement is detected by standard S 32	32 S	Use of continue statement. Modifiers :331 = 0 (Default)

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
58	Required	The break statement shall not be used (except to terminate the cases of a switch statement)	The use of break (except in switches) is reported by standard S 31	31 S	Use of break statement in loop.
59	Required	The statements forming the body of an if, else, if while, do or for statement shall always be enclosed in braces	Standards S 11 and S 12 report on absence of braces. LDRA Testbed enforces the rule strictly because of the need to insert probes for Dynamic Coverage Analysis. Thus braces are Mandatory between the else and following if statement.	11 S	No brackets to loop body (added by Testbed).
				12 S	No brackets to then/else (added by Testbed). Modifiers :tbend F
60	Advisory	All if, else if constructs should contain a final else clause	Standard S 59 reports the absence of an else clause. Note that this standard can be configured to permit the simple if...then statement.	59 S	Else alternative missing in if. Modifiers :530 = 0 (Default)
61	Required	Every non- empty case clause in a switch statement shall be terminated with a break statement	The absence of break in a non empty case clause in a switch is reported by standard S 62.	62 S	Switch case not terminated with break. Modifiers :283 = 0 (Default) 384 = c++ : 0 (Default) 551 = 0 (Default)
62	Required	All switch statements should contain a final default clause	The absence of a default clause in a switch is reported by standard S 48.	48 S	No default case in switch statement. Modifiers :252 = 0 (Default) 233 = 0 (Default)
63	Advisory	A switch expression should not represent a Boolean value	The presence of Boolean expressions in switches is detected by standard S 121	121 S	Use of boolean expression in switch.
64	Required	Every switch statement shall have at least one case	Switches with no cases are reported by standard S 60 or if only a default is present by standard S 61	60 S	Empty switch statement.
				61 S	Switch contains default only. Modifiers :437 = 1
65	Required	Floating point variables shall not be used as loop counters	The use of floating point variables in loop control is reported by standard S 39.	39 S	Unsuitable type for loop variable.
66	Advisory	Only expressions concerned with loop control should appear within a for statement	Empty increment expressions are reported by standard S 269. Non-simple loop initialisations are reported by standard S 270. Non-simple loop increments are reported by standard S 271. Empty middle expressions are reported by standard S 429. Inconsistent usage of loop control variable is reported by standard S 430.	269 S	Empty increment exprsn in for loop. Modifiers :424 = 0 (Default)
				270 S	For loop initialisation is not simple. Modifiers :249 = 0 (Default)
				271 S	For loop incrementation is not simple. Modifiers :331 = 0 (Default)
				429 S	Empty middle expression in for loop.
				430 S	Inconsistent usage of loop control variable.
67	Advisory	Numeric variables being used within a for loop for iteration counting should not be modified in the body of the loop	Dataflow analysis reports D 55 when modification of the loop counter takes place in the loop body.	55 D	Modification of loop counter in loop body.
68	Required	Functions shall always be declared at file scope		296 S	Function declared at block scope.
69	Required	Functions with variable numbers of arguments shall not be used	Ellipsis is reported by standard S 41 and the other options are banned names checked by standard S 44.LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	41 S	Ellipsis used in procedure parameter list. Modifiers :522 = 0 (Default)
				44 S	Use of banned function, type or variable.
70	Required	Functions shall not call themselves, either directly or indirectly	Recursion either at single procedure level or via a group of procedures is detected by standard D 6. U 1 reports Inter-file recursion	6 D	Recursion in procedure calls found.
				1 U	Inter-file recursion found.
71	Required	Functions shall always have prototype declarations and the prototype shall be visible at both the function definition and call	LDRA Associated MISRA-C:1998 20, 71	19 D	Procedure called before being defined.
				24 D	Procedure definition has no associated prototype.
				41 D	Procedure call has no prototype declared.
				20 S	Parameter not declared explicitly.
72	Required	For each function parameter the type given in the declaration and definition shall be identical, and the return types shall also be identical	LDRA Associated MISRA-C:1998 26, 72	102 S	Function and prototype return inconsistent (MR). Modifiers :331 = 0 (Default)
				103 S	Function and prototype param inconsistent (MR).
73	Required	Identifiers shall either be given for all of the parameters in a function prototype declaration, or for none		142 S	Parameter list declarations are inconsistent.
74	Required	If identifiers are given for any of the parameters, then the identifiers used in the declaration and definition shall be identical		36 D	Prototype and definition name mismatch.
75	Required	Every function shall have an explicit return type		2 D	Function does not return a value on all paths.
				36 S	Function has no return statement. Modifiers :553 = 0 (Default)
				148 S	No return type for function/procedure.
76	Required	Functions with no parameters shall be declared with parameter type void		63 S	Empty parameter list to procedure/function.

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
77	Required	The unqualified type of parameters passed to a function shall be compatible with the unqualified expected types defined in the function prototype	Mismatches between the basic types of formal and actual parameters are reported by standard S 458. LDRA Associated MISRA-C:1998 77, 105	458 S	Implicit conversion: actual to formal param (MR).
78	Required	The number of parameters passed to a function shall match the function prototype		21 S	Number of parameters does not match.
79	Required	The values returned by void functions shall not be used		64 S	Void procedure used in expression. Modifiers :218 = 1
80	Required	Void expressions shall not be passed as function parameters		65 S	Void variable passed as parameter.
81	Advisory	const qualification should be used on function parameters which are passed by reference, where it is intended that the function will not modify the parameter		120 D	Pointer param should be declared pointer to const.
83	Required	For functions with non-void return types i) there shall be one return statement for every exit branch (including the end of the program), ii) each return shall have an expression, iii) the return expression shall match the declared return type.		66 S	Function with empty return expression.
				101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 0 (Default) 428 = c : 0 (Default), c++ : 1 (Default) 430 = 0 (Default) 431 = 0 (Default) 453 = 0 (Default) 470 = 0 (Default)
84	Required	For functions with void return type, return statements shall not have an expression	Void procedures with return expressions are reported with standard	27 S	Void procedure with return statement.
85	Advisory	Functions called with no parameters should have empty parentheses		99 S	Function use is not a call. Modifiers :498 = 0 (Default)
86	Advisory	If a function returns error information, then that error information should be tested	Not Automatically Checkable.		
87	Required	#include statements in a file shall only be preceded by other pre-processor directives or comments		75 S	Executable code before an included file.
88	Required	Non-standard characters shall not occur in header file names in #include directives	Standard S 100 reports banned characters in #include filenames. Note the characters checked for are user definable.LDRA Associated MISRA-C:1998 88, 89	100 S	#include filename is non conformant. Modifiers :120 = '\, (Default)
89	Required	The #include directive shall be followed by either a <filename> or "filename" sequence	LDRA Associated MISRA-C:1998 88, 89	100 S	#include filename is non conformant. Modifiers :120 = '\, (Default)
90	Required	C macros shall only be used for symbolic constants, function-like macros, type qualifiers and storage class specifiers		79 S	Macro contains unacceptable items.
91	Required	Macros shall not be #define'd and #undef'd within a block		67 S	#define used in a block.
				426 S	#undef used in a block.
93	Advisory	A function should be used in preference to function-like macro		340 S	Use of function like macro.
94	Required	A function- like macro shall not be 'called' without all of its arguments		324 S	Macro call has wrong number of parameters.
95	Required	Arguments to a function-like macro shall not contain tokens that look like pre- processing directives		341 S	Preprocessor construct as macro parameter.
96	Required	In the definition of a function-like macro the whole definition, and each instance of a parameter, shall be enclosed in parentheses	Standard S 77 checks that the macro body is enclosed in parentheses and standard S 78 checks that the parameters are enclosed by parentheses.	77 S	Macro replacement list needs parentheses.
				78 S	Macro parameter not in brackets. Modifiers :454 = 0 (Default)
97	Advisory	Identifiers in pre-processor directives should be defined before use		337 S	Undefined macro variable in #if.
98	Required	There shall be at most one occurrence of the # or ## pre-processor operators in a single macro definition		76 S	More than one of # or ## in a macro.
99	Required	All uses of the #pragma directive shall be documented and explained		69 S	#pragma used.
100	Required	The defined pre-processor operator shall only be used in one of the two standard forms	Standard S 335 reports the use of the defined pre-processor operator with any expression other than a macro identifier. Standard S 336 checks for the use of defined through macro expansion substitution in a #if pre-processor directive.	335 S	Operator defined contains illegal items.
				336 S	#if expansion contains define operator.
101	Advisory	Pointer arithmetic should not be used	LDRA Associated MISRA-C:1998 101, 104	87 S	Use of pointer arithmetic. Modifiers :tbend = <Default>
102	Advisory	No more than 2 levels of pointer indirection should be used		80 S	Pointer indirection exceeds 2 levels.
103	Required	Relational operation shall not be applied to pointer types except where both operands are of the same array, structure or union	Standard S 70 reports the use of a subset of the logical operators (<=, >=, < and >) with pointer variables.	70 S	Logical comparison of pointers.
104	Required	Non-constant pointers to functions shall not be used	The use of non constant function pointers is detected by LDRA Associated MISRA-C:1998 101, 104	87 S	Use of pointer arithmetic. Modifiers :tbend = <Default>

HIS Standards Model Compliance for C / C++

Rule	Classification	Rule Description		LDRA Standard	LDRA Standard Description
105	Required	All the functions pointed to by a single pointer to a function shall be identical in the number and type of parameters and the return type	The LDRA Testbed follows the de-references of function pointers and will report calls where the parameters do not match with standard S 98. LDRA Associated MISRA-C:1998 77, 105	98 S	Actual and formal parameters inconsistent (MR).
				576 S	Function pointer is of wrong type.
106	Required	The address of an object with automatic storage shall not be assigned to an object which may persist after the object has ceased to exist		42 D	Local pointer returned in function result.
				71 S	Pointer assignment to wider scope. Modifiers :222 = 1
107	Required	The null pointer shall not be de-referenced		45 D	Pointer not checked for null before use.
				123 D	File pointer not checked for null before use.
				128 D	Global pointer not checked within this procedure.
				129 D	Global file pointer not checked within this procedure.
108	Required	In the specification of a structure or union type, all members of the structure or union shall be fully specified	Incomplete struct declarations of the form struct atagname; are identified by S 141. LDRA Associated MISRA-C:1998 108, 113	141 S	Incomplete structure or class declaration.
109	Required	Overlapping variable storage shall not be used	LDRA Associated MISRA-C:1998 109, 110	74 S	Union declared.
112	Required	Bit fields of type signed int shall be at least 2 bits long		72 S	Signed bit field less than 2 bits wide. Modifiers :416 = 0 (Default)
113	Required	All the members of a structure (or union) shall be named and shall only be accessed via their name	LDRA Associated MISRA-C:1998 108, 113	138 S	Anonymous bit field used in structure. Modifiers :248 = 0 (Default)
114	Required	Reserved words and standard library function names shall not be redefined or undefined	Standard S 86 reports the macro definition of language keywords. A general facility for banning any name is also available.	86 S	Attempt to define reserved word.
115	Required	Standard library function names shall not be reused		218 S	Name is used in standard libraries.
116	Required	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	Not Automatically Checkable.		
117	Required	The validity of values passed to library functions shall be checked	Not Automatically Checkable.		
118	Required	Dynamic heap memory allocation shall not be reused	LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
119	Required	The error indicator errno shall not be used	Preset as banned function or variable S 44. LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
120	Required	The macro offsetof, in library <stddef.h>, shall not be used	Preset as banned function or variable S 44. LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
121	Required	<locale.h> and the setlocale function shall not be used	Preset as banned filename S 130. LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
				130 S	Included file is not permitted.
122	Required	The setjmp macro and the longjmp function shall not be used	Setjmp/longjmp preset as banned S 43.	43 S	Use of setjmp/longjmp.
123	Required	The signal handling facilities of <signal.h> shall not be used	Preset as banned filename S 130. LDRA Associated MISRA-C:1998 121, 123, 124, 127	130 S	Included file is not permitted.
124	Required	The input/output library <stdio.h> shall not be used in production code	Preset as banned filename S 130. LDRA Associated MISRA-C:1998 121, 123, 124, 127	130 S	Included file is not permitted.
125	Required	The library functions atof, atoi and atol from library<stdlib.h> shall not be used	Preset as banned function or variable S 44. LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
126	Required	The library functions abort, exit, getenv and system from library<stdlib.h> shall not be used	Abort, exit, etc. Preset as banned S 122. LDRA Associated MISRA-C:1998 69, 118, 119, 120, 121, 125, 126	44 S	Use of banned function, type or variable.
				122 S	Use of abort, exit, etc.
127	Required	The time handling functions of library <time.h> shall not be used	Preset as banned filename S 130. LDRA Associated MISRA-C:1998 121, 123, 124, 127	130 S	Included file is not permitted.

General Compliance Notes

Enhanced Enforcement: LDRA checks additional cases to those specified by the mapped rule for enhanced safety and security.
Fully Implemented: LDRA checks all statically checkable aspects of the mapped rule.
Partially Implemented: LDRA checks certain aspects of the rule.

The assessment of whether a rule is fully or partially implemented is based on whether the mapped LDRA standards cover all statically checkable aspects of the rule with a high level of coverage or only cover certain statically checkable aspects of the rule. If a rule is undecidable then this assessment is based on what it is deemed reasonable for a static analysis tool to check.