BARR-C:2018 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
"BARR-C:2018 Embedded C Coding Standard"
2018

Further information is available at https://barrgroup.com

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Mandatory	2	22	7	1	5	37
Checking	7	37	15	16	19	94
Optional	1	3	2	2	2	10
Informational	0	0	0	1	1	2
Total	10	62	24	20	27	143

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
1.1.a	Checking	All programs shall be written to comply with the C99 version of the ISO Standard for the C Programming Language.	The analysis should be performed after compilation by the user's compiler. The standards mapped to this rule protect from the following constraint errors, which have been known to go undetected by some compilers.	21 S	Number of parameters does not match.
				143 S	Curly brackets used in expression.
				145 S	#if has invalid expression.
				293 S	Non ANSI/ISO construct used.
				323 S	Switch has more than one default case.
				345 S	Bit operator with floating point operand.
				387 S	Enum init not integer-constant-expression.
				404 S	Array initialisation has too many items. Modifiers :399 = 1
				481 S	Array with no bounds in struct.
				580 S	Macro redefinition without using #undef.
				582 S	const object reassigned.
				615 S	Conditional operator has incompatible types.
				646 S	Struct initialisation has too many items.
1.1.b	Checking	Whenever a C++ compiler is used, appropriate compiler options shall be set to restrict the language to the selected version of ISO C.	To enable this penalty the user must indicate to what extent the check should be performed. Information must be added to the vals.dat file as described in the documentation for 293 S.	293 S	Non ANSI/ISO construct used.
1.1.c	Checking	The use of proprietary compiler language keyword extensions, #pragmas, and inline assembly shall be kept to the minimum necessary to get the job done and be localized to a small number of device driver modules that interface directly to hardware.		69 S	#pragma used.
				88 S	Procedure is not pure assembler.
				293 S	Non ANSI/ISO construct used.
1.1.d	Checking	Preprocessor directive #define shall not be used to alter or rename any keyword or other aspects of the programming language.	The analysis interprets the phrase other aspects of the programming language as in MISRA-C:2004 rule 19.4.	79 S	Macro contains unacceptable items. Modifiers :298 = 0 (Default) 331 = 0 (Default)
				86 S	Attempt to define reserved word.
				626 S	#define of keyword. Modifiers :391 = 1
1.2.a	Checking	The width of all lines in a program shall be limited to a maximum of 80 characters.		189 S	Input line exceeds limit. Modifiers :177 = 80
1.3.a	Mandatory	Braces shall always surround the blocks of code (a.k.a., compound statements), following if, else, switch, while, do, and for statements; single statements and empty statements following these keywords shall also always be surrounded by braces.		11 S	No brackets to loop body (added by Testbed).
				12 S	No brackets to then/else (added by Testbed). Modifiers :tbend F
				428 S	No {} for switch (added by Testbed).

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
1.3.b	Checking	Each left brace ({) shall appear by itself on the line below the start of the block it opens. The corresponding right brace (}) shall appear by itself in the same position the appropriate number of lines later in the file.		188 S	{ or } not on line by itself.
				301 S	} not aligned vertically below {.
1.4.a	Mandatory	Do not rely on Cs operator precedence rules, as they may not be obvious to those who maintain the code. To aid clarity, use parentheses (and/or break long statements into multiple lines of code) to ensure proper execution order within a sequence of operations.		361 S	Expression needs brackets.
1.4.b	Checking	Unless it is a single identifier or constant, each operand of the logical AND(&&) and OR(||) operators shall be surrounded by parentheses.		49 S	Logical conjunctions need brackets. Modifiers :206 = 0 (Default)
1.5.a	Optional	Abbreviations and acronyms should generally be avoided unless their meanings are widely and consistently understood in the engineering community.			
1.5.b	Checking	A table of project-specific abbreviations and acronyms shall be maintained in a version controlled document.			
1.6.a	Checking	Each cast shall feature an associated comment describing how the code ensures proper behavior across the range of possible values on the right side.			
1.7.a	Checking	The auto keyword shall not be used.		464 S	Use of auto specifier. Modifiers :331 = 0 (Default)
1.7.b	Checking	The register keyword shall not be used.		84 S	Register variable declared.
1.7.c	Checking	It is a preferred practice to avoid all use of the goto keyword.		13 S	goto detected.
				509 S	goto label is backwards.
				511 S	Jump into nested block.
1.7.d	Optional	It is a preferred practice to avoid all use of the continue keyword.		32 S	Use of continue statement.
1.8.a	Mandatory	The static keyword shall be used to declare all functions and variables that do not need to be visible outside of the module in which they are declared.		27 D	Variable should be declared static.
				61 D	Procedure should be declared static.
1.8.b	Mandatory	The const keyword shall be used whenever appropriate.	The analysis does not highlight non-const fields of struct and unions, which are not modified.	93 D	Local variable should be declared const.
				120 D	Pointer param should be declared pointer to const.
				168 S	Call by value parameter not const.
				200 S	Define used for numeric constant.
				298 S	Non const pointer to function.
1.8.c	Mandatory	The volatile keyword shall be used whenever appropriate.			
2.1.a	Informational	Single-line comments in the C++ style (i.e., preceded by //) are a useful and acceptable alternative to traditional C style comments (i.e., /* .. */).			
2.1.b	Mandatory	Comments shall never contain the preprocessor tokens /*, //, or \	The analysis permits the use of within comments and line splices within /* . . */ comments.	119 S	Nested comment found. Modifiers :413 = 0 (Default) 434 = 0 (Default)
				611 S	Line splice used in // comment.
2.1.c	Mandatory	Code shall never be commented out, even temporarily.		302 S	Comment possibly contains code.
2.2.a	Checking	All comments shall be written in clear and complete sentences, with proper spelling and grammar and appropriate punctuation.			
2.2.b	Checking	The most useful comments generally precede a block of code that performs one step of a larger algorithm. A blank line shall follow each such code block. The comments in front of the block should be at the same indentation level.			
2.2.c	Optional	Avoid explaining the obvious. Assume the reader knows the C programming language.			
2.2.d	Checking	The number and length of individual comment blocks shall be proportional to the complexity of the code they describe.	The Quality Review Report gives metrics on the density of comments per procedure		
2.2.e	Checking	Whenever an algorithm or technical detail is defined in an external reference, a comment shall include a sufficient reference to the original source to allow a reader of the code to locate the document			
2.2.f	Checking	Whenever a flow-chart or other diagram is needed to sufficiently document the code, the drawing shall be maintained with the source code under version control and the comments should reference the diagram by file name or title.			
2.2.g	Mandatory	All assumptions shall be spelled out in comments.			
2.2.h	Checking	Each module and function shall be commented in a manner suitable for automatic documentation generation, e.g. via Doxygen			
2.2.i	Checking	Use the following capitalized comment markers to highlight important issues: i. "WARNING:" ... ii. ""NOTE:"" .... iii. ""TODO:""			

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
3.1.a	Checking	Each of the keywords if, while, for, switch, and return shall be followed by one space when there is additional program text on the same line.	The analysis checks for spaces after if, while, for and switch. There is no check that there is a space after the return statement.	181 S	No space between if, while, for and expresn.
3.1.b	Checking	Each of the assignment operators =, +=, -=, *=, /=,%=, &=, |=, ^=, ~=, and != shall always be preceded and followed by one space.		186 S	Space missing before or after binary operator.
3.1.c	Checking	Each of the binary operators +, -, *, /, %, <, <=, >, >=, ==, !=, <<, >>, &, |, ^, &&, and || shall always be preceded and followed by one space.		186 S	Space missing before or after binary operator.
3.1.d	Checking	Each of the unary operators +, -, ++, --, !, and ~, shall be written without a space on the operand side.		185 S	Space between unary operator and operand.
3.1.e	Checking	The pointer operators * and & shall be written with white space on each side within declarations but otherwise without a space on the operand side.	3.1.b and 3.1.c check that there is a space before a unary operator. The analysis check sthat there is a space after the *(or &) in a declaration, but not whether there is a space before the *(or &).	180 S	No space between * or & and name in declaration.
				185 S	Space between unary operator and operand.
3.1.f	Checking	The ? and : characters that comprise the ternary operator shall each always be preceded and followed by one space.			
3.1.g	Checking	The structure pointer and structure member operators (-> and ., respectively) shall always be without surrounding spaces.		184 S	Spaces round -> or [] operators. Modifiers :240 = 0 (Default
				315 S	Blank before/after . operator.
3.1.h	Checking	The left and right brackets of the array subscript operator ([and]) shall be without surrounding spaces, except as required by another white space rule.	The analysis checks for the presence of spaces before the left hand bracket ([) of an array subscript or declaration.	184 S	Spaces round -> or [] operators. Modifiers :240 = 0 (Default
3.1.i	Checking	Expressions within parentheses shall always have no spaces adjacent to the left and right parenthesis characters.			
3.1.j	Checking	The left and right parentheses of the function call operator shall always be without surrounding spaces, except that the function declaration shall feature one space between the function name and the left parenthesis to allow that one particular mention of the function name to be easily located.			
3.1.k	Checking	Except when at the end of a line, each comma separating function parameters shall always be followed by one space.	The analysis highlights missing spaces in function calls, but not function declarations	371 S	No space after comma in parameter list.
3.1.l	Checking	Each semicolon separating the elements of a for statement shall always be followed by one space.		182 S	No space after semi colon in for expression.
3.1.m	Checking	Each semicolon shall follow the statement it terminates without a preceding space.			
3.2.a	Checking	The names of variables within a series of declarations shall have their first characters aligned.			
3.2.b	Checking	The names of struct and union members shall have their first characters aligned.			
3.2.c	Checking	The assignment operators within a block of adjacent assignment statements shall be aligned.			
3.2.d	Checking	The # in a preprocessor directive shall always be located at the start of a line, though the directives themselves may be indented within a #if or #ifdef sequence.	The analysis highlights all cases where the # of a preprocessor directive is indented. The user must then manually check whether the indentation of the directive it-self is compliant with this rule.	209 S	Preprocessor command indented.
3.3.a	Checking	No line of code shall contain more than one statement.		183 S	No newline after semi colon.
3.3.b	Checking	There shall be a blank line before and after each natural block of code. Examples of natural blocks of code are loops, if-else and switch statements, and consecutive declarations.			
3.3.c	Checking	Each source file shall terminate with a comment marking the end of file followed by a blank line.		5 Q	File does not end with new line.
3.4.a	Optional	Each indentation level should align at a multiple of 4 characters from the start of the line.	The analysis checks that each level of indentation increases by 4 spaces, but does not check that all the contents are at the same level of indentation.	190 S	{ ... } contents not indented by *** spaces. Modifiers :324 = 0 (Default) 518 = 0 (Default)
3.4.b	Optional	Within a switch statement, the case labels shall be aligned; the contents of each case block shall be indented once from there.		190 S	{ ... } contents not indented by *** spaces. Modifiers :324 = 0 (Default) 518 = 0 (Default)
3.4.c	Informational	Whenever a line of code is too long to fit within the maximum line width, indent the second and any subsequent lines in the most readable manner possible.			
3.5.a	Checking	The tab character shall never appear within any source code file.		187 S	Tab character in source.
3.6.a	Checking	Whenever possible, all source code lines shall end only with the single character LF, not with the pair CR-LF.			
3.6.b	Checking	The only other non-printable character permitted in a source code file is the form feed character FF.	The analysis checks for the use of non standard characters within string literals.	113 S	Non standard character in source. Modifiers : 408 = 1 449 = 0 (Default)
				176 S	Non standard escape sequence in source.

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
4.1.a	Mandatory	All module names shall consist entirely of lowercase letters, numbers, and underscores. No spaces shall appear within the module's header and source file names.	The analysis does not check for the presence of spaces in the filename	3 Q	Filename contains upper case letters.
4.1.b	Checking	All module names shall be unique in their first 8 characters and end with suffixes .h and .c for the header and source file names, respectively.	The analysis does not check that filenames are unique within their first 8 characters. This standard is not enabled by default.	225 S	Inappropriate file extension. Modifiers :527 = 0 (Default)
4.1.c	Mandatory	No module's header file name shall share the name of a header file from the C Standard Library or C++ Standard Library.		568 S	#include "filename" uses standard library name. Modifiers :344 = 1
4.1.d	Checking	Any module containing a main() function shall have the word "main"as part of its source file name.			
4.2.a	Checking	There shall always be precisely one header file for each source file and they shall always have the same root name.			
4.2.b	Checking	Each header file shall contain a preprocessor guard against multiple inclusion.		243 S	Included file not protected with #define.
4.2.c	Checking	The header file shall identify only the procedures, constants, and data types (via prototypes or macros, #define, typedefs respectively) about which it is strictly necessary for other modules to be informed. i. It is a preferred practise that no variable ever be declared (via extern) in a header file. ii. No storage for any variable shall be allocated in a header file.		46 S	extern not in nominated include file.
				286 S	Functions defined in header file. Modifiers :238 = 0 (Default)
				287 S	Variable definition in header file. Modifiers :250 = 1
4.2.d	Checking	No public header file shall contain a #include of any private header file.	The analysis highlights all uses of nested header files. The user must the then manually check which files are considered to be public or private.	154 S	Nested header files found.
4.3.a	Checking	Each source file shall include only the behaviors appropriate to control one entity.			
4.3.b	Checking	Each source file shall be comprised of some or all of the following sections, in the order listed: comment block; include statements; data type, constant, and macro definitions; static data declarations; private function prototypes; public function bodies; then private function bodies.	The analysis checks that #include statements are only preceded by comments.	75 S	Executable code before an included file.
				338 S	#include preceded by non preproc directives.
4.3.c	Mandatory	Each source file shall always #include the header file of the same name, to allow the compiler to confirm that each public function and its prototype match.	The analysis checks that each public function has a visible prototype and that they match. The analysis does not check that the prototype is present in a header file or that the header file has the same name.	106 D	No prototype for non-static function.
				102 S	Function and prototype return inconsistent (MR). Modifiers :331 = 0 (Default)
				103 S	Function and prototype param inconsistent (MR).
				62 X	Function prototype/defn return type mismatch (MR).
				63 X	Function prototype/defn param type mismatch (MR).
4.3.d	Checking	Absolute paths shall not be used in include file names.	The analysis highlights the use of both absolute and relative paths in #include file names.	4 Q	Included file has path.
4.3.e	Checking	Each source file shall be free of unused include files.	The analysis checks for type declarations that are not used, regardless of the file in which they are declared.	413 S	User type declared but not used in code analysed. Modifiers :284 = 1
4.3.f	Checking	No source file shall #include another source file.		288 S	Header file is not .h. Modifiers :301 = 0 (Default)
4.4.a	Checking	A set of templates for header files and source files shall be maintained at the project level.			
5.1.a	Checking	The names of all new data types, including structures, unions, and enumerations, shall consist only of lowercase characters and internal underscores and end with _t.	The analysis permits the first character to be an uppercase character.	318 S	Upper case letter found after first character. Modifiers :204 = 0 (Default) 256 = 1
5.1.b	Checking	All new structures, unions, and enumerations shall be named via a typedef.		381 S	Enum, struct or union not typedeffed.
5.1.c	Checking	The name of all public data types shall be prefixed with their module name and an underscore			
5.2.a	Mandatory	Whenever the width, in bits or bytes, of an integer value matters in the program, one of the fixed width data types shall be used in place of char, short, int, long, or long long.	The analysis permits the use of either declarations from stdint.h or the use of typedefs. However, the analysis can not statically determine whether the use of a basic numerical type occurs when the width … matters in the program and so all uses of the basic numerical types are flagged.	90 S	Basic type declaration used. Modifiers :219 = 1 462 = 0 (Default)
				495 S	Typedef name has no size indication. Modifiers :392 = 1
5.2.b	Mandatory	The keywords short and long shall not be used.	The analysis highlights all uses of basic numerical types except in the definition of a typedef which is permitted by the coding standard. The analysis highlights the use of int which is permitted by this rule, providing rule 5.2.a is not violated.	90 S	Basic type declaration used. Modifiers :219 = 1 462 = 0 (Default)
				90 S	Basic type declaration used. Modifiers :219 = 1 462 = 0 (Default)

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
5.2.c	Mandatory	Use of the keyword char shall be restricted to the declaration of and operations concerning strings.	The analysis permits the use of unsigned char and signed char in the definition of a typedef which is permitted by the coding standard.	329 S	Operation not appropriate to plain char. Modifiers :191 = 1 331 = 0 (Default) 391 = 1
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 1 431 = 1 428 = c : 0 (Default), c++ : 1 (Default) 470 = 0 (Default) 386 = 1 549 = 0 (Default)
5.3.a	Checking	Bit-fields shall not be defined within signed integer types.		316 S	Bit field is not unsigned integral. Modifiers :234 = 0 (Default) 331 = 0 (Default) 350 = 0 (Default)
5.3.b	Mandatory	None of the bit-wise operators (i.e., &, |, ~, ^, <<, and >>) shall be used to manipulate signed integer data.		50 S	Use of shift operator on signed type.
				120 S	Use of bit operator on signed type. Modifiers :457 = 0 (Default)
5.3.c	Mandatory	Signed integers shall not be combined with unsigned integers in comparisons or expressions. In support of this, decimal constants meant to be unsigned should be declared with a u at the end.		101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c : 0 (Default), c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 0 (Default) 470 = 0 (Default)
				107 S	Type mismatch in ternary expression. Modifiers :428 = c : 0 (Default), c++ : 1 (Default) 429 = 1 470 = 0 (Default) 525 = 0 (Default)
				331 S	Literal value requires a U suffix. Modifiers :309 = 0 (Default) 358 = 0 (Default) 516 = 0 (Default)
				434 S	Signed/unsigned conversion without cast. Modifiers :374 = 0 (Default) 358 = 0 (Default) 427 = 1 428 = c : 0 (Default), c++ : 1 (Default) 429 = 1 430 = 0 (Default) 470 = 0 (Default)
				458 S	Implicit conversion: actual to formal param (MR).
				488 S	Value outside range of underlying type. Modifiers :535 = 0 (Default)
5.4.a	Optional	Avoid the use of floating point constants and variables whenever possible. Fixed-point math may be an alternative.	The analysis highlights all uses of floating point types.	144 S	Floating point not permitted.
5.4.b	Mandatory	When floating point calculations are necessary: i. Use the [C99] type names float32_t, float64_t, and float128_t. ii. Append an f to all single-precision constants (e.g., pi = 3.1415927f). iii. Ensure that the compiler supports doubleprecision, if your math depends on it. iv. Never test for equality or inequality of floating point values. v. Always invoke the isfinite() macro to check that prior calculations have resulted in neither INFINITY nor NAN.	The analysis checks i, ii and iv.	56 S	Equality comparison of floating point.
				101 S	Function return type inconsistent. Modifiers :374 = 0 (Default) 427 = 1 428 = c : 0 (Default), c++ : 1 (Default) 430 = 0 (Default) 431 = 1 453 = 0 (Default) 470 = 0 (Default)
				425 S	float literal with no F suffix.
				433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 1 431 = 1 428 = c : 0 (Default), c++ : 1 (Default) 470 = 0 (Default) 386 = 1 549 = 0 (Default)
				445 S	Narrower float conversion without cast. Modifiers :374 = 0 (Default)
				451 S	No cast for widening complex float expression (MR). Modifiers :191 = 1 529 = 0 (Default)
				456 S	Implicit float widening for function return (MR). Modifiers :191 = 1
				458 S	Implicit conversion: actual to formal param (MR).
				490 S	No cast for widening float parameter.
				495 S	Typedef name has no size indication. Modifiers :392 = 1
5.5.a	Mandatory	Appropriate care shall be taken to prevent the compiler from inserting padding bytes within struct or union types used to communicate to or from a peripheral or over a bus or network to another processor.			
5.5.b	Mandatory	Appropriate care shall be taken to prevent the compiler from altering the intended order of the bits within bit-fields.			

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
5.6.a	Checking	Boolean variables shall be declared as type bool.	The analysis checks for the use of non-boolean types in boolean contexts. The analysis permits the boolean types listed in cvals.dat. 114 S also checks that boolean objects are not used in non-boolean contexts.	114 S	Expression is not Boolean. Modifiers :386 = 0 528 = 0 (Default)
5.6.b	Checking	Non-Boolean values shall be converted to Boolean via use of relational operators (e.g., < or !=), not via casts.	The analysis highlights implicit conversions.	433 S	Type conversion without cast. Modifiers :374 = 0 (Default) 391 = 1 431 = 1 428 = c : 0 (Default), c++ : 1 (Default) 470 = 0 (Default) 386 = 1 549 = 0 (Default)
6.1.a	Mandatory	No procedure shall have a name that is a keyword of of any standard version of the C or C++ programming language. Restricted names include interrupt, inline, class, true, false, public, private, friend, protected, and many others.	This standard checks for C++ keywords used as C identifiers. Further keywords can be added by following the instructions in the documentation of 45 S.	45 S	Use of C++ keyword.
6.1.b	Mandatory	No procedure shall have a name that overlaps a function in the C Standard Library.		218 S	Name is used in standard libraries.
6.1.c	Checking	No procedure shall have a name that begins with an underscore.		219 S	User name starts with underscore. Modifiers :412 = 0 (Default)
6.1.d	Checking	No procedure name shall be longer than 31 characters.	This standard is currently not enabled for procedure names in this release.	369 S	Name found with length greater than ***.
6.1.e	Mandatory	No function name shall contain any uppercase letters.		312 S	Function name is not all lower case.
6.1.f	Checking	No macro name shall contain any lowercase letters.		210 S	Macro name is not upper case.
6.1.g	Checking	Underscores shall be used to separate words in procedure names			
6.1.h	Checking	Each procedures name shall be descriptive of its purpose.			
6.1.i	Checking	The names of all public functions shall be prefixed with their module name and an underscore.	Instructions for enabling the H standards can be found in H_standards folder of the Testbed installation directory. The code will need to be altered by the user so that the required prefixes can be checked.	8 H	Global Func Name does not conform to style <file>_<name>.
6.2.a	Checking	All reasonable effort shall be taken to keep the length of each function limited to one printed page, or a maximum of 100 lines.		256 S	Procedure exceeds *** source lines of code. Modifiers :203 = 1
6.2.b	Checking	Whenever possible, all functions shall be made to start at the top of a printed page, except when several small functions can fit onto a single page.			
6.2.c	Checking	sIt is a preferred practice that all functions shall have just one exit point and it shall be via a return at the bottom of the function.	The analysis highlights all functions which have more than one exit point from the function. This includes calls of exit and abort. A deviation must be used if there is a requirement for the use of exit or abort.	7 C	Procedure has more than one exit point.
				2 D	Function does not return a value on all paths.
				36 S	Function has no return statement. Modifiers :553 = 0 (Default)
				66 S	Function with empty return expression.
6.2.d	Mandatory	A prototype shall be declared for each public function in the module header file.	The analysis requires that each function has one and not more than one prototype. This forces the user to write the prototype in a header file if it is used by more than one translation unit.	106 D	No prototype for non-static function.
				110 D	More than one prototype for same function.
				496 S	Function call with no prior declaration.
6.2.e	Mandatory	All private functions shall be declared static.		61 D	Procedure should be declared static.
				553 S	Function and proto should both be static.
6.2.f	Checking	Each parameter shall be explicitly declared and meaningfully named.	Static analysis cannot determine whether a parameter is meaningfully named. However, names less than 3 characters long are not likely to be meaningful.	20 S	Parameter not declared explicitly.
				37 S	Procedure parameter has a type but no identifier.
				135 S	Parameter list is KR.
				274 S	Name found with length less than ***.
6.3.a	Mandatory	Parameterized macros shall not be used if an inline function can be written to accomplish the same behavior.		340 S	Use of function like macro.
6.3.b	Mandatory	If parameterized macros are used for some reason, these rules apply: i. Surround the entire macro body with parentheses. ii. Surround each use of a parameter with parentheses. iii. Use each parameter no more than once, to avoid unintended side effects. iv. Never include a transfer of control (e.g., return keyword)	The analysis checks for side effects in all expressions not just as a result of macro expansion	35 D	Expression has side effects.
				1 Q	Call has execution order dependant side effects.
				77 S	Macro replacement list needs parentheses.
				78 S	Macro parameter not in brackets. Modifiers :454 = 0 (Default)
				79 S	Macro contains unacceptable items. Modifiers :298 = 0 (Default) 331 = 0 (Default)
6.4.a	Checking	All functions that encapsulate threads of execution (a.k.a., tasks, processes) shall be given names ending with "_thread" (or "_task", "_process")			
6.5.a	Checking	Interrupt service routines (ISRs) are not ordinary functions. The compiler must be informed that the function is an ISR by way of a #pragma or compilerspecific keyword, such as "__interrupt".			
6.5.b	Checking	All functions that implement ISRs shall be given names ending with "_isr".			
6.5.c	Mandatory	To ensure that ISRs are not inadvertently called from other parts of the software (they may corrupt the CPU and call stack if this happens), each ISR function shall be declared static, and/or be located at the end of the associated driver module, as permitted by the target platform.			

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
6.5.d	Checking	A stub or default ISR shall be installed in the vector table at the location of all unexpected or otherwise unhandled interrupt sources. Each such stub could attempt to disable future interrupts of the same type, say at the interrupt controller, and assert().			
7.1.a	Mandatory	No variable shall have a name that is a keyword of C, C++, or any other well-known extension of the C programming language, including specifically K&R C and C99.		45 S	Use of C++ keyword.
7.1.b	Mandatory	No variable shall have a name that overlaps a variable name from the C Standard Library.		218 S	Name is used in standard libraries.
7.1.c	Checking	No variable shall have a name that begins with an underscore.		219 S	User name starts with underscore. Modifiers :412 = 0 (Default)
7.1.d	Checking	No variable name shall be longer than 31 characters.		369 S	Name found with length greater than ***.
7.1.e	Checking	No variable name shall be shorter than 3 characters, including loop counters.		274 S	Name found with length less than ***.
7.1.f	Mandatory	No variable name shall contain any uppercase letters.		313 S	Variable name is not all lower case.
7.1.g	Checking	No variable name shall contain any numeric value that is called out elsewhere, such as the number of elements in an array or the number of bits in the underlying type.			
7.1.h	Checking	Underscores shall be used to separate words in variable names.			
7.1.i	Checking	Each variables name shall be descriptive of its purpose.			
7.1.j	Mandatory	The names of any global variables shall begin with the letter g.	Instructions for enabling the H standards can be found in H_standards folder of the Testbed installation directory. The code will need to be altered by the user so that the required prefixes can be checked.	1 H	Global Variable does not conform to style g_<name>.
7.1.k	Mandatory	The names of any pointer variables shall begin with the letter p.	Instructions for enabling the H standards can be found in H_standards folder of the Testbed installation directory. The code will need to be altered by the user so that the required prefixes can be checked.	6 H	Pointer does not conform to style p_<name>.
7.1.l	Checking	The names of any pointer-to-pointer variables shall begin with the letter pp.			
7.1.m	Mandatory	The names of all integer variables containing "Boolean" information (including 0 vs. nonzero) shall begin with the letter b and phrased as the question they answer.			
7.1.n	Checking	The names of any variables representing non-pointer handles for objects, e.g., file handles, shall begin with the letter 'h'.			
7.1.o	Checking	In the case of a variable name requiring multiple of the above prefixes, the order of their inclusion before the first underscore shall be [g][p|pp][b|h].			
7.2.a	Mandatory	All variables shall be initialized before use.		69 D	UR anomaly, variable used before assignment.
7.2.b	Optional	It is preferable to define local variables as you need them, rather than all at the top of a function.	The analysis checks that file scope variables are declared in a function if only used in one function.	25 D	Scope of variable could be reduced.
7.2.c	Optional	If project- or file-global variables are used, their definitions shall be grouped together and placed at the top of a source code file.			
7.2.d	Optional	Any pointer variable lacking an initial address shall be initialized to NULL.		53 D	Attempt to use uninitialised pointer.
				652 S	Object created by malloc used before initialisation.
8.1.a	Mandatory	The comma operator (,) shall not be used within variable declarations.		579 S	More than one variable per declaration.
8.2.a	Optional	It is a preferred practice that the shortest (measured in lines of code) of the if and else if clauses should be placed first.			
8.2.b	Checking	Nested if-else statements shall not be deeper than two levels. Use function calls or switch statements to reduce complexity and aid understanding.		370 S	If nest depth greater than ***.
8.2.c	Mandatory	Assignments shall not be made within an if or else if test.		132 S	Assignment operator in boolean expression.
8.2.d	Checking	Any if statement with an else if clause shall end with an else clause.		59 S	Else alternative missing in if. Modifiers :530 = 0 (Default)
				477 S	Empty else clause following else if.
8.3.a	Mandatory	The break for each case shall be indented to align with the associated case, rather than with the contents of the case code block.		190 S	{ ... } contents not indented by *** spaces. Modifiers :324 = 0 (Default) 518 = 0 (Default)
8.3.b	Checking	All switch statements shall contain a default block.		48 S	No default case in switch statement. Modifiers :252 = 0 (Default) 233 = 0 (Default)
8.3.c	Checking	Any case designed to fall through to the next shall be commented to clearly explain the absence of the corresponding break.	The analysis highlights all fall through case labels. The user must then manually check whether an appropriate comment is present.	25 S	Null case in switch statement.

BARR-C:2018 Standards Model Compliance for C / C++

Rule	Classification	Rule Description	Additional information	LDRA Standard	LDRA Standard Description
8.4.a	Checking	Magic numbers shall not be used as the initial value or in the endpoint test of a while, do-while, or for loop.	The analysis higlights of uses of literals in expressions. The user must then manually check whether they are compliant with this rule. The values of 0 and 1 are permitted.	201 S	Use of numeric literal in expression. Modifiers :217 = 1 163 = 1 237 = 0 (Default) 207 = 0 (Default)
8.4.b	Checking	With the exception of the initialization of a loop counter in the first clause of a for statement and the change to the same variable in the third, no assignment shall be made in any loop's controlling expression.		114 S	Expression is not Boolean. Modifiers :386 = 0 528 = 0 (Default)
				132 S	Assignment operator in boolean expression.
				430 S	Inconsistent usage of loop control variable.
8.4.c	Checking	Infinite loops shall be implemented via controlling expression "for (;;)".		26 S	Loop control expression may not terminate loop. Modifiers :425 = 1
8.4.d	Checking	Each loop with an empty body shall feature a set of braces enclosing a comment to explain why nothing needs to be done until after the loop terminates.	The analysis all empty blocks with no comments. Empty blocks include blocks that only include a null statement ;. The analysis includes a check for empty blocks in if statements that are permitted by this rule.	634 S	Empty block found.
8.5.a	Checking	The use of goto statements shall be restricted as per Rule 1.7.c.		13 S	goto detected.
				509 S	goto label is backwards.
				511 S	Jump into nested block.
8.5.b	Checking	C Standard Library functions abort(), exit(), setjmp(), and longjmp() shall not be used.		43 S	Use of setjmp/longjmp.
				122 S	Use of abort, exit, etc.
8.6.a	Checking	When evaluating the equality of a variable against a constant, the constant shall always be placed to the left of the equal-to operator (==).	The analysis checks for the use of the assignment operator in test for equality as required by the enforcement text for this rule.	114 S	Expression is not Boolean. Modifiers :386 = 0 528 = 0 (Default)
				132 S	Assignment operator in boolean expression.

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.