NETRINO Standards Model Summary

The LDRA tool suite® is developed and certified to BS EN ISO 9001:2000.

This information is applicable to version 9.1.1 of the LDRA tool suite®.
It is correct as of 15th June 2012.

Compliance is measured against
"Embedded C Coding Standard"
2009
Copyright © Netrino

Further information is available at http://www.netrino.com

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Mandatory	0	31	9	28	29	**97**
Checking	0	16	4	0	0	**20**
Optional	0	9	7	0	0	**16**
Total	**0**	**56**	**20**	**28**	**29**	**133**

NETRINO Standards Model Compliance

Rule	Classification	Rule Description	LDRA Standard	LDRA Standard Description
1.1.a	Mandatory	All programs shall be written to comply with the latest available ISO Standard for the C Programming Language, which is currently C99.	293 S	Non ANSI/ISO construct used.
1.1.b	Mandatory	Whenever a C++ compiler is used, appropriate compiler options shall be set to restrict the language to the latest standard C subset supported by the compiler.	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.
1.2.a	Checking	The length of all lines in a program shall be limited to a maximum of 80 characters.	189 S	Input line exceeds limit.
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).
			428 S	No {} for switch. (added by Testbed)
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	188 S	{ or } not on line by itself.
			301 S	} not aligned vertically below {.
		Do not rely on Cs operator precedence rules, as they may not be obvious to those who maintain	96 S	Use of mixed mode arithmetic.

1.4.a	Mandatory	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	Mandatory	Unless it is a single identifier or constant, each operand of the logical && and || operators shall be surrounded by parentheses.	49 S	Logical conjunctions need brackets.
1.5.a	Mandatory	Abbreviations and acronyms should generally be avoided unless their meanings are widely and consistently understood in the engineering community.		
1.5.b	Mandatory	A table of additional project-specific abbreviations and acronyms shall be maintained in a version controlled document.		
1.6.a	Mandatory	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	Mandatory	The auto keyword shall not be used.	464 S	Use of auto specifier.
1.7.b	Optional	The register keyword shall not be used.	84 S	Register variable declared.
1.7.c	Mandatory	The goto keyword shall not be used.	13 S	goto detected.
1.7.d	Mandatory	The continue keyword shall not be used.	32 S	Use of continue statement.
1.7.e	Mandatory	The break keyword shall not be used outside of a switch statement.	31 S	Use of break statement in loop.
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
			461 S	Identifier with ambiguous linkage.
			553 S	Function and proto should both be static.

1.8.b	Mandatory	The const keyword shall be used whenever appropriate.	59 D	Parameter should be declared const
			62 D	Pointer parameter should be declared 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	Mandatory	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 be nested.	119 S	Nested comment found
2.1.c	Mandatory	Comments shall never be used to disable a block of code even temporarily. i. To temporarily disable a block of code, use the preprocessors conditional compilation feature (e.g., #if 0 .. #endif). No block of temporarily disabled code shall remain in the source code of a release candidate.ii. Any line or block of code that exists specifically to increase the level of debugging output information shall be surrounded by #ifndef NDEBUG .. #endif. In this way, useful debug code may be maintained in production code, as the ability to gather additional information is often desirable long after development is done.	302 S	Comment possibly contains code
2.2.a	Mandatory	All comments shall be written in clear and complete sentences, with proper spelling and grammar and appropriate punctuation.		

2.2.b	Mandatory	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	Mandatory	Avoid explaining the obvious. Assume the reader knows the C programming language. For example, end-of-line comments should only be used in exceptional circumstances, where the meaning of that one line of code may be unclear from the variable and function names and operations alone but where a short comment makes it clear. Avoid writing unhelpful and redundant comments such as "shift left 2 bits".		
2.2.d	Mandatory	The number and length of individual comment blocks shall be proportional to the complexity of the code they describe.		
2.2.e	Mandatory	Whenever an algorithm or technical detail has come from a published source, the comment shall include a sufficient reference to the original source (via book title, website URL, or other details) to allow a reader of the code to find the cited reference material.		
2.2.f	Mandatory	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	Mandatory	Each module and function shall be commented in a manner suitable for automatic documentation generation via Doxygen		

2.2.i	Mandatory	Use the following capitalized comment markers to highlight important issues: i. "WARNING:" ... ii. ""NOTE:"" .... iii. ""TODO:""		
3.1.a	Optional	Each of the keywords if, else, while, for, switch, and return shall always be followed by one space.	181 S	No space between if, while, for and expresn.
3.1.b	Optional	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	Optional	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	Optional	Each of the unary operators +, -, ++, --, !, and ~, shall always be written without a space on the operand side and with one space on the other 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.	180 S	No space between * or & and name in declaration.
			185 S	Space between unary operator and operand.
3.1.f	Mandatory	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.
			315 S	Blank before/after . operator.
3.1.h	Mandatory	The left and right brackets of the array subscript operator ([and]) shall always be without surrounding spaces.		
3.1.i	Mandatory	Expressions within parentheses shall always have no spaces adjacent to the left and right parenthesis characters.		

3.1.j	Mandatory	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 particularmention of the function name to be easily located.		
3.1.k	Mandatory	Each comma separating function parameters shall always be followed by one space.	371 S	No space after comma in parameter list
3.1.l	Optional	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	Mandatory	Each semicolon shall follow the statement it terminates without a preceding space.		
3.2.a	Mandatory	The names of variables within a series of declarations shall have their first characters aligned.		
3.2.b	Mandatory	The names of struct and union members shall have their first characters aligned.		
3.2.c	Mandatory	The assignment operators within a block of adjacent assignment statements shall be aligned.		
3.2.d	Optional	The # in a preprocessor directive shall always be located in column 1, except when indenting within a #if or #ifdef .sequence.	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	Mandatory	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	Mandatory	Each source file shall have a blank line at the end.		
3.4.a	Checking	Each indentation level within a module should consist of 4 spaces.	190 S	{ ... } contents not indented by *** spaces.

3.4.b	Checking	Within a switch statement, each case statement should be indented; the contents of the case block should be indented once more.	190 S	{ ... } contents not indented by *** spaces.
3.4.c	Mandatory	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	Optional	The tab character shall never appear within any module.	187 S	Tab character in source.
3.6.a	Mandatory	Whenever possible, all source code lines shall end only with the single character LF (0x0A).		
4.1.a	Mandatory	All module names shall consist entirely of lowercase letters, numbers, and underscores. No spaces shall appear within the file name.	3 Q	Filename contains upper case letters.
4.1.b	Checking	All module names shall be unique in their first eight characters, with .h and .c used for the suffix for header and source files respectively.	288 S	Header file is not .h
4.1.c	Mandatory	No module name shall share the name of a standard library header file. For example, modules shall not be named "stdio" or "math".		
4.1.d	Mandatory	Any module containing a main() function shall have the word "main" in its filename.		
4.2.a	Mandatory	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	Mandatory	The header file shall identify only the procedures, constants, and data types (via prototypes or macros, #define, and struct/union/enum typedefs, respectively) about which it is strictly necessary for other modules to know about. i. It is recommended that no variable be defined (via extern) in a header file. ii. No storage for any variable shall be declared in a header file.	286 S	Functions defined in header file.
			287 S	Variable definitions in header file.
4.2.d	Mandatory	No header file shall contain a #include statement.	154 S	Nested header files found.
4.3.a	Mandatory	Each source file shall include only the behaviors appropriate to control one entity.		
4.3.b	Mandatory	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.	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.	102 S	Function and prototype return inconsistent (MR).
			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	Mandatory	Absolute paths shall not be used in include file names.	4 Q	Included file has path.

4.3.e	Optional	Each source file shall be free of unused include files.	413 S	User type declared but not used in code analysed.
4.3.f	Checking	No source file shall #include another source file.	288 S	Header file is not .h
4.4.a	Mandatory	A set of templates for header files and source files shall be maintained at the project level.		
5.1.a	Mandatory	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.		
5.1.b	Mandatory	All new structures, unions, and enumerations shall be named via a typedef.	381 S	Enum, struct or union not typedeffed.
5.2.a	Optional	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 signed and unsigned fixed width integer types shall be as shown in the table below.	90 S	Basic type declaration used.
			495 S	Typedef name has no size indication.
5.2.b	Mandatory	The keywords short and long shall not be used.		
5.2.c	Mandatory	Use of the keyword char shall be restricted to the declaration of and operations concerning strings.	93 S	Value is not of appropriate type.
			329 S	Operation not appropriate to plain char.
5.3.a	Mandatory	Bit-fields shall not be defined within signed integer types.	316 S	Bit field is not unsigned integral.
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.
			93 S	Value is not of appropriate type.
			96 S	Use of mixed mode arithmetic.

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.
			107 S	Type mismatch in ternary expression.
			330 S	Implicit conversion of underlying type.
			433 S	Type conversion without cast.
			434 S	Signed/unsigned conversion without cast.
			458 S	Implicit conversion: actual to formal param.
			488 S	Value outside range of underlying type.
5.4.a	Mandatory	Avoid the use of floating point constants and variables whenever possible. Fixed-point math may be an alternative.	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.	56 S	Equality comparison of floating point.
			93 S	Value is not of appropriate type.
			96 S	Use of mixed mode arithmetic.
			101 S	Function return type inconsistent.
			330 S	Implicit conversion of underlying type.
			433 S	Type conversion without cast.
			445 S	Narrower float conversion without cast.
			451 S	No cast for widening complex float expression.

			456 S	Implicit float widening for function return.
			458 S	Implicit conversion: actual to formal param.
			490 S	No cast for widening float parameter.
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.		
6.1.a	Mandatory	No procedure 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. Restricted names include interrupt, inline, class, true, false, public, private, friend, protected, and many others.		
6.1.b	Optional	No procedure shall have a name that overlaps a function in the C standard library. Examples of such names include strlen, atoi, and memset.	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.
6.1.d	Checking	No procedure name shall be longer than 31 characters.	369 S	Name found with length greater than ***.
6.1.e	Optional	No function name shall contain any uppercase letters.	312 S	Function name is not all lower case.
6.1.f	Optional	No macro name shall contain any lowercase letters.	210 S	Macro name is not upper case.
6.1.g	Mandatory	Underscores shall be used to separate words in procedure names		

6.1.h	Mandatory	Each procedures name shall be descriptive of its purpose.		
6.1.i	Mandatory	The names of all public functions shall be prefixed with their module name and an underscore.		
6.2.a	Checking	All reasonable effort shall be taken to keep the length of each function limited to one printed page, or about 50-100 lines.	256 S	Procedure exceeds *** source lines of code.
6.2.b	Mandatory	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	All functions shall have just one exit point and it shall be at the bottom of the function. That is, the keyword return shall appear a maximum of once.	7 C	Procedure has more than one exit point.
6.2.d	Mandatory	A prototype shall be defined for each public function in the module header file.	24 D	Procedure definition has no associated prototype
			60 D	External object should be declared only once
			496 S	Function call with no prior declaration.
6.2.e	Mandatory	All private functions shall be defined static.	61 D	Procedure should be declared static
			553 S	Function and proto should both be static.
6.2.f	Mandatory	Each parameter shall be explicitly declared and meaningfully named.	37 S	Procedure Parameter has a type but no identifier.
			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 task.	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 argument no more than once, to avoid unintended side effects.	77 S	Macro replacement list needs parentheses.
			78 S	Macro parameter not in brackets.
6.4.a	Mandatory	All functions that represent tasks (a.k.a., threads) shall be given names ending with "_task" (or "_thread").		
6.5.a	Mandatory	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	Mandatory	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 lack a prototype, be declared static, and be located at the end of the associated driver module.		
6.5.d	Mandatory	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	Checking	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	Optional	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.
7.1.d	Checking	No variable name shall be longer than 31 characters.	369 S	Name found with length greater than ***.
7.1.e	Mandatory	No variable name shall be shorter than 3 characters, including loop counters.	274 S	Name found with length less than ***.
7.1.f	Optional	No variable name shall contain any uppercase letters.	210 S	Macro name is not upper case.
7.1.g	Mandatory	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	Mandatory	Underscores shall be used to separate words in variable names.		
7.1.i	Mandatory	Each variables name shall be descriptive of its purpose.		
7.1.j	Mandatory	The names of all global variables shall begin with the letter g.		
7.1.k	Mandatory	The names of all pointer variables shall begin with the letter p.		
7.1.l	Mandatory	The names of all pointer-to-pointer variables shallbegin with the letter pp.		
7.1.m	Mandatory	The names of all integer variables containing "effectively Boolean" information (i.e., 0 vs. nonzero) shall begin with the letter b and phrased as the question they answer.		
7.2.a	Mandatory	All variables shall be initialized before use.	57 D	Global not initialised at declaration.
			64 D	Local not initialised at declaration.
7.2.b	Optional	It is preferable to create variables as you need them, rather than all at the top of a function.	25 D	Scope of variable could be reduced
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	Mandatory	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 expression.	132 S	Assignment operator in boolean expression.
8.2.d	Mandatory	Any if statement with an else if clause shall end with an else clause.	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.		
8.3.b	Mandatory	All switch statements shall contain a default block.	48 S	No default case in switch statement.
8.4.a	Checking	Magic numbers shall not be used as the initial value or in the endpoint test of a while or for loop.	201 S	Use of numeric literal in expression.
8.4.b	Mandatory	Except for a single loop counter initialization in the first clause of a for statement, assignments shall not be made in any loops controlling expression.	430 S	Inconsistent usage of loop control variable.
8.4.c	Mandatory	Infinite loops shall be implemented via the controlling expression "for (;;)".		
8.4.d	Mandatory	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.		
8.5.a	Mandatory	The keywords goto, continue, and break shall not be used to create unconditional jumps.	13 S	goto detected.
			31 S	Use of break statement in loop.
			32 S	Use of continue statement.

8.6.a	Mandatory	When evaluating the equality or inequality of a variable with a constant value, always place the constant value on the left side of the comparison operator.	9 S	Assignment operation in expression.
			132 S	Assignment operator in boolean expression.